NXP Semiconductors
Reference Manual

Document identifier: MCXNx4xSRM
Rev. 4, 09/2024

MCX Nx4x Security Reference Manual

Supports MCXN54x and MCXN94x

Contents

Chapter 1
About this manual

1.1 About This Document

1.1.1 Audience

This manual is intended for the board-level product designers and product software developers. This manual assumes that the reader has a background in computer engineering and/or software engineering and understands the concepts of the digital system design, microprocessor architecture, Input/Output (I/O) devices, industry standard communication, and device interface protocols.

1.1.2 Organization

This document is organized into two main sets of chapters.

1. First set covers the chip at a system level and provides an architectural overview. It also covers the system memory map, system-level interrupt events, clock, reset and system controller.
2. Chapters in the second set provides a technical description of individual modules along with chip-specific contents at the beginning of each chapter.

1.1.3 Suggested Reading

This section lists the additional resources that provide background for the information in this manual, as well as general information about the architecture.

1.1.3.1 General Information

The following documentation provides useful background information about the Arm® Cortex® processor.

For information about the Arm Cortex processor see:

- **http://infocenter.arm.com**

Refer Security Primitives: Requirements in (I)IoT Systems document to familiarize with NXP's security nomenclature.

1.1.3.2 Related Documentation

For a current list of documentation, refer to **http://www.nxp.com**.

1.1.4 Conventions

This document uses the following notational conventions:

cleared / set	When a bit has a value of zero, it is said to be cleared; when it has a value of one, it is said to be set.
mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. The book titles in the text are set in italics.
15	An integer in decimal.
0x	the prefix to denote a hexadecimal number.
0b	The prefix to denote a binary number. Binary values of 0 and 1 are written without a prefix.
n'H4000CA00	The n-bit hexadecimal number.

BLK_REG_NAME	The register names are all uppercase. The block mnemonic is prepended with an underscore delimiter (_).
BLK_REG[FIELD]	The fields within registers appear in brackets. For example, ESR[RLS] refers to the Receive Last Slot field of the ESAI Status Register.
BLK_REG[*n*]	The bit number *n* within the BLK.REG register.
BLK_REG[*l:r*]	The register bit ranges. The ranges are indicated by the left-most bit number *l* and the right-most bit number *r*, separated by a colon (:). For example, ESR[15:0] refers to the lower half word in the ESAI Status Register.
x, U	In some contexts, such as signal encodings, an unitalicized x indicates a "don't care" or "uninitialized". The binary value can be 1 or 0.
x	An italicized *x* indicates an alphanumeric variable.
n, m	Italicized *n or m* represent integer variables.
!	Binary logic operator NOT.
&&	Binary logic operator AND.
||	Binary logic operator OR.
^ or <O+>	Binary logic operator XOR. For example, A <O+> B.
|	Bit-wise OR. For example, 0b0001 | 0b1000 yields the value of 0b1001.
&	Bit-wise AND. For example, 0b0001 and 0b1000 yields the value of 0b0000.
{A,B}	Concatenation, where the *n*-bit value A is prepended to the *m*-bit value B to form an (*n*+*m*)-bit value. For example, {0, REG*m* [14:0]} yields a 16-bit value with 0 in the most significant bit.
- or grey fill	Indicates a reserved bit field in a register. Although these bits can be written to with ones or zeros, they always read zeros.
>>	Shift right logical one position.
<<	Shift left logical one position.
<=	Assignment.
==	Compare equal.
!=	Compare not equal.
>	Greater than.
<	Less than.

1.1.5 Register Access

1.1.5.1 Register Diagram Field Access Type Legend

This figure provides the interpretation of the notation used in the register diagrams for a number of common field access types:

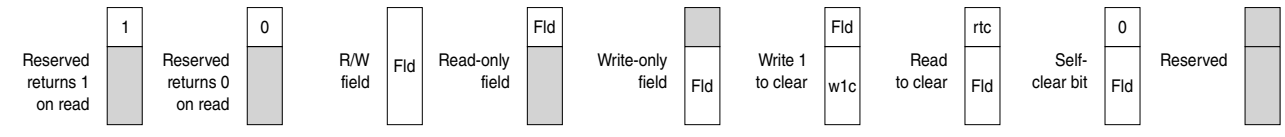


Figure 1. Register Field Conventions

NOTE

For reserved register fields, the software should mask off the data in the field after a read (the software can't rely on the contents of data read from a reserved field) and always write all zeros.

1.1.5.2 Register Macro Usage

A common operation is to update one field without disturbing the contents of the remaining fields in the register. Normally, this requires a read-modify-write (RMW) operation, where the CPU reads the register, modifies the target field, then writes the results back to the register. This is an expensive operation in terms of CPU cycles, because of the initial register read.

To address this issue, some hardware registers are implemented as a group, including registers that can be used to either set, clear, or toggle (SCT) individual bits of the primary register. When writing to an SCT register, all the bits set to 1 perform the associated operation on the primary register, while the bits set to 0 are not affected. The SCT registers always read back 0 and should be considered write-only. The SCT registers are not implemented if the primary register is read-only.

With this architecture, it is possible to update one or more fields using only register writes. First, all bits of the target fields are cleared by a write to the associated clear register, then the desired value of the target fields is written to the set register. This sequence of two writes is referred to as a clear-set (CS) operation.

A CS operation does have one potential drawback. Whenever a field is modified, the hardware sees a value of 0 before the final value is written. For most fields, passing through the 0 state is not a problem. Nonetheless, this behavior is something to consider when using a CS operation.

Also, a CS operation is not required for fields that are one-bit wide. While the CS operation works in this case, it is more efficient to simply set or clear the target bit (that is, one write instead of two). A simple set or clear operation is also atomic, while a CS operation is not.

Note that not all macros for set, clear, or toggle (SCT) are atomic. For registers that do not provide hardware support for this functionality, these macros are implemented as a sequence of read-modify-write operations. When an atomic operation is required, the developer should pay attention to this detail, because unexpected behavior might result if an interrupt occurs in the middle of the critical section comprising the update sequence.

A set of SCT registers is offered for registers in many modules on this device, as described in this manual.
In a module memory map table, the suffix _SET, _CLR, or _TOG is added to the base name of the register.
For example, the CCM_ANALOG_PLL_ARM register has three other registers called CCM_ANALOG_PLL_ARM_SET, CCM_ANALOG_PLL_ARM_CLR, and CCM_ANALOG_PLL_ARM_TOG.

In the sub-section that describes one of these sets of registers, a short-hand convention is used to denote that a register has the SCT register set. There is an italicized *n* appended to the end of the short register name. Using the above example, the name used for this register is CCM_ANALOG_PLL_ARM*n*. When you see this designation, there is a SCT register set associated with the register, and you can verify this by checking it in the memory map table. The address offset for each of these registers is given in the form of the following example:

Address: 20C_8000h base + 0h offset + (4d × i), where i=0d to 3d

In this example, the address for each of the base registers and their three SCT registers can be calculated as:

Register	Address
CCM_ANALOG_PLL_ARM	20C_8000h
CCM_ANALOG_PLL_ARM_SET	20C_8004h
CCM_ANALOG_PLL_ARM_CLR	20c_8008h
CCM_ANALOG_PLL_ARM_TOG	20C_800Ch

Chapter 2
Introduction

2.1 Overview

The MCX Nx4x series microcontrollers combine the Arm Cortex-M33 TrustZone® core with a CoolFlux BSP32, a PowerQuad DSP Co-processor, and multiple high-speed connectivity options running at 150 MHz. To support a wide variety of applications, the MCX N-series includes advanced serial peripherals, timers, high-precision analog, and state-of-the-art security features like secure user code, data, and communications. All MCX Nx4x products include dual-bank flash which supports read while write operation from internal flash. The MCX Nx4x series also supports large external serial memory configurations.

The MCX Nx4x are as follows:

- **N54x**: Mainstream MCU with a second M33 core, advanced timers, analog and high-speed connectivity (including Hi-Speed USB), 10/100 Ethernet, and FlexIO which can be programmed as an LCD controller.
- **N94x**: Integration CPU and DSP of serial connectivity, advanced timers, high precision analog, and high-speed connectivity including Hi-Speed USB, CAN 2.0, 10/100 Ethernet, and FlexIO which can be programmed as an LCD controller.

The chip includes these key features:

Table 1. Key features

Function	Features
Security	• TrustZone for Armv8M • Secure boot, firmware update, and debug authentication using ROM • EdgeLock® secure subsystem (ELS) S50 • Public-key cryptography (PKC) • External and internal flash memory interface with on-the-fly PRINCE decryption and encryption • Physically Unclonable Function (PUF) hardware options • Factory Root of Trust (RoT) programming • Tamper detection • Analog and digital glitch detection • Code Watchdog (CDOG) • Intrusion and Tamper Response Controller (ITRC)
Industrial strength	• Industrial temperature rating • Industrial communication protocol support (CAN-FD) • High-resolution mixed signal analog • BLDC and PMSM motor control support • Integrated sensor interfaces: I3C, I2C, SPI, and UART • 15-year longevity • Configurable RAM ECC • Ethernet (ENET) with QoS

Table continues on the next page...

Table 1. Key features (continued)

Function	Features
Power-efficient operating modes	• Down to 57 µA/MHz (3.3 V, @25 ℃) • 170 µA in Deep Sleep mode (full 512 KB SRAM retention, 3.3 V, @25 ℃) • 5.2 µA in Power Down mode (full 512 KB SRAM retention, 3.3 V, @25 ℃) • Down to 2.0 µA in Deep Power Down mode, 5.3 ms wake-up (RTC enabled 8 KB RAM and Reset pin enabled, @25 ℃)

2.2 Target applications

The MCX Nx4x MCUs are ideal solutions for:

- Industrial
  - Energy Storage and Management System
  - Smart Metering
  - Power Line Communication
  - Factory Automation
  - Industrial HMI
  - Mobile Robotics Ecosystem
  - Motion Control and Robotics
  - Motor Drives
  - Brushless DC Motor (BLDC) Control
  - Permanent Magnet Synchronous Motor (PMSM)
  - Edge AI/ML Anomaly Detection and Predictive Maintainence
- Smart Home
  - Home Control Panel
  - Home Security and Surveillance
  - Major Home Appliances
  - Robotic Appliance
  - Smart Speaker
  - Soundbar
  - Gaming Accessories
  - Smart Lighting
  - Smart Power Socket and Light Switch

2.3 Block diagram

Figure 2 shows a top-level view of the modules within the chip, organized by functional category.

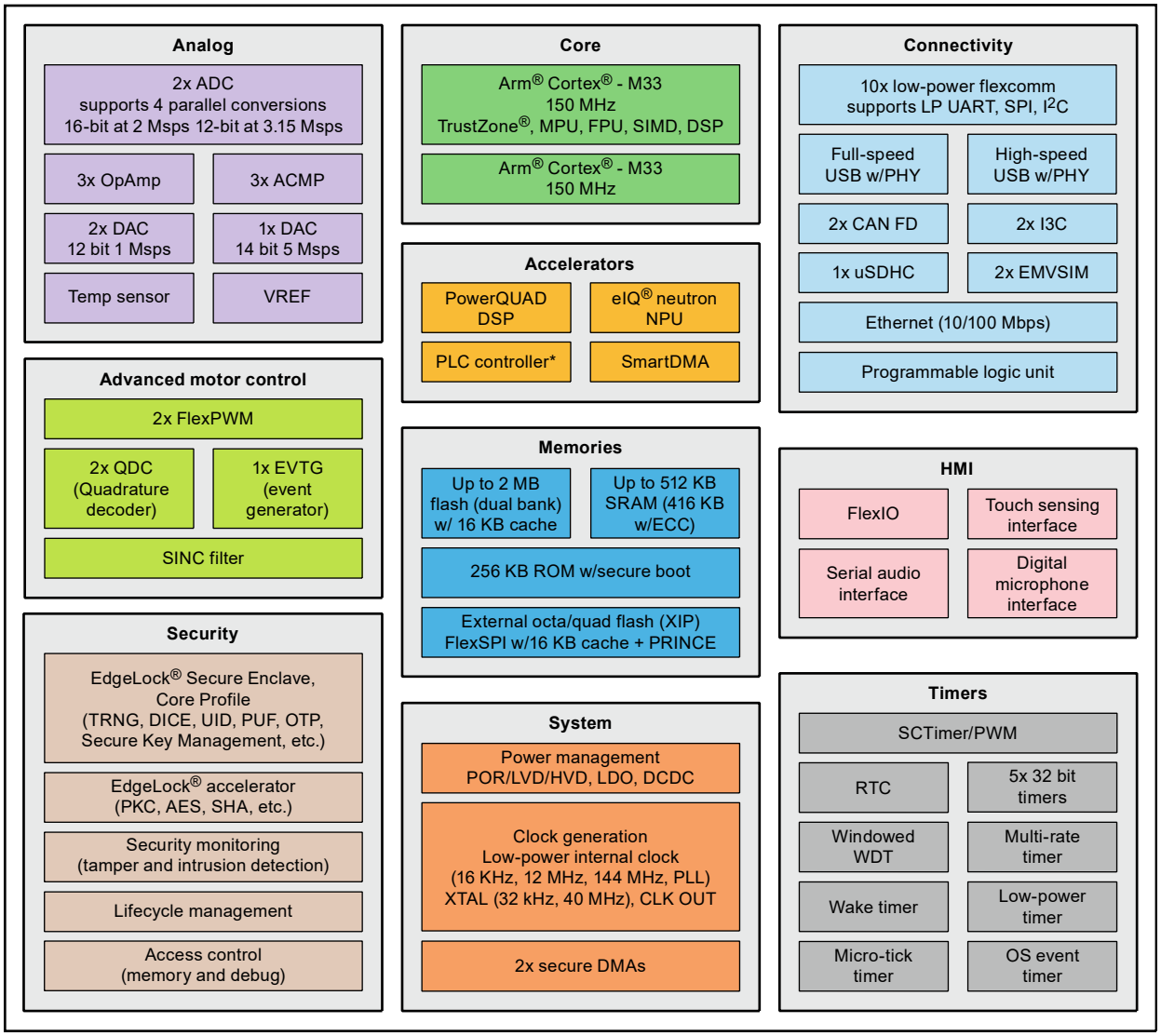


Figure 2. Features block diagram

Figure 3 shows the chip's block diagram, including bus connections.

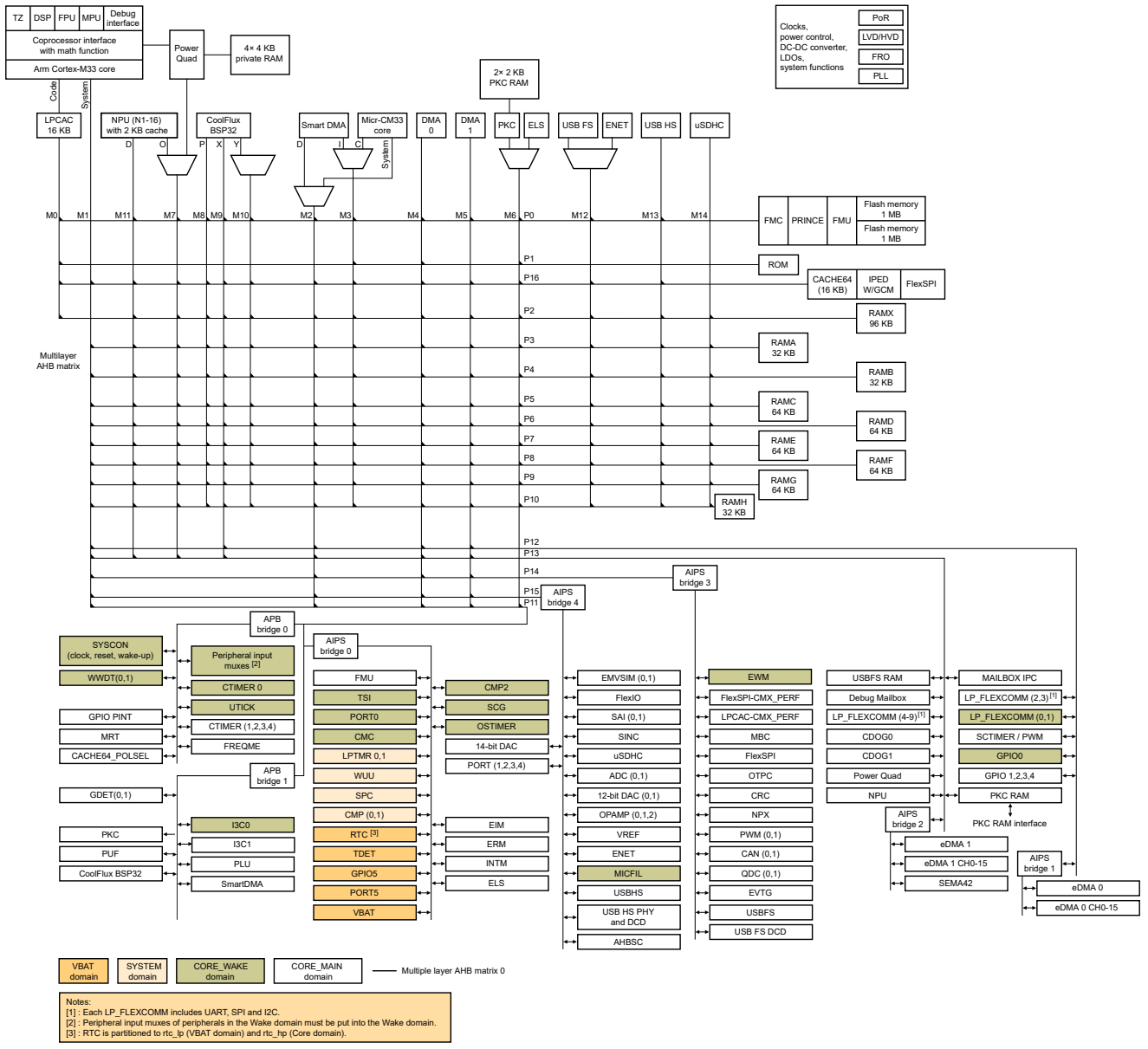


Figure 3. Bus matrix block diagram

2.4 System bus priority and arbitration

As shown in Figure 3 , this device includes a number of masters that can access the various slave ports. The multilayer AHB matrix allows for concurrent accesses when multiple masters are attempting to access different slave ports. If multiple masters attempt an access to the same slave port at the same time then arbitration is required.

The SYSCON_AHBMATPRIO register is where the programmable priorities for each of the master ports can be configured. Masters are assigned a priority value between zero and three with three being the highest priority. If two ports have the same priority, then the lowest port number is given priority.

There are some master ports that are shared between two masters. Where the port is shared, only one of the masters can have an active access at a time. The priority between two masters sharing a port uses a fixed arbitration scheme. The table below lists the master or masters for each of the ports. In the case where the port is shared, the high-priority master is specified:

Table 2. AHB bus matrix ports

Master port	Master	Accessible slave ports	Inaccessible slave ports	AHB ID
M0	CPU0 (CM33) code bus	P0 - P2 and P16	P3 - P15	0
M1	CPU0 (CM33) system bus	P3 - P16	P0 - P2	1
M2	CPU1 (Micr-CM33) code bus (low-priority)	P0 and P2 - P16	P1	2
	SmartDMA data bus (high-priority)			3
M3	CPU1 (Micr-CM33) system bus (low-priority)	P0 - P10 and P16	P11 - P15	4
	SmartDMA instruction bus (high-priority)			5
M4	DMA0	P0 - P16	None	6
M5	DMA1	P0 and P2 - P16	P1	7
M6	PKC (low-priority)	P0 - P11 and P16	P12 - P15	8
	ELS (high-priority)			9
M7	PowerQuad (low-priority)	P0, P2 - P10, P13, and P16	P1, P11 - P12, and P14 - P15	10
	NPU operand bus (high-priority)			11
M8	CoolFlux BSP32 program memory bus	P0, P2 - P10, and P16	P1 and P11 - P15	12
M9	CoolFlux BSP32 X-data memory bus	P0, P2 - P10, P12 - P13, and P16	P1, P11, and P14 - P15	13
M10	CoolFlux BSP32 Y-data memory bus	P0, P2 - P10, and P16	P1 and P11 - P15	14
M11	NPU data bus	P0, P2 - P10, P13, and P16	P1, P11 - P12, and P14 - P15	16
M12	USBFS (low-priority)	P0, P2 - P10, and P16	P1 and P11 - P15	17
	Ethernet (high-priority)			18
M13	USBHS	P0, P2 - P10, and P16	P1 and P11 - P15	19
M14	uSDHC	P2 - P10	P0 - P1 and P11 - P16	20

Chapter 3
Security Overview

3.1 Disclaimer

As system security requirements and the attack surface evolves, it is important for customers to understand the types of attacks (especially advanced physical attacks) which NXP does not claim to protect against, or strongly mitigate, so that appropriate mitigation can be taken by the customer at the system level if necessary.

- This SoC has built-in security event detection features. However, NXP does not guarantee against advanced tamper attempts, including operation of the device beyond the defined specification limits. NXP does not guarantee the protection against semi-invasive and invasive attacks.
- This SoC has several built-in features addressing side channel attacks. However, there is no claim to be completely resistant. The effectiveness of these features has not been independently evaluated. Therefore NXP does not guarantee that the result will meet specific customer requirements.
- This SoC's security trust architecture relies on the strength of cryptographic algorithms and digital signatures. If these are subsequently determined to have inherent flaws, then the impact for each flaw must be evaluated and, in this case, NXP does not guarantee the underlying trust architecture claims.
- This SoC has some built-in access control mechanisms to support the logical separation of executed code. However, NXP does not guarantee that the device completely ensures logical separation by itself. Any vulnerabilities identified in Trusted Execution Environments or Hypervisor software may impact this separation and data integrity, and may require additional mitigations.

NXP recommends customers to implement appropriate design and operating safeguards based on defined threat models, to minimize the security risks associated with their applications and products.

3.2 Overview

This chapter provides an overview of Platform Security features the device implements using following on-chip components.

- ELS security subsystem provides key isolation. It also includes random number generation functionality.
- Physically Unclonable Function (PUF) SRAM controller - PUF
- PUF subsystem that stores the applications keys, which are provisioned to ELS S50 key store, securely.
- Public-key cryptography accelerator (PKC) - PKC.
- OTP controller (OTPC) which supports loading and housing of EFUSE content into shadow registers.
- Code Watch Dog (CDOG) to ensure software integrity by detecting unexpected changes in the code execution flow.
- Intrusion and Tamper Response Controller (ITRC) to configure the response action for an intrusion event detected by an on-chip security sensors.
- Memory Block Checker (MBC) that provides read, write, and execute access control per block of internal flash memory.
- Digital tamper (TDET) to support tamper detection.
- Secure AHB bus and AHB Controller to support secure trusted execution at the system level - AHBSC.
- PRINCE in CTR mode of operation to provide confidentiality protection of the content stored on internal flash.
- FlexSPI providing support of inline Prince Encryption Decryption (IPED) in GCM mode of operation.
- Digital and analog Glitch Detect (GDET) to provide power supply glitch detection.
- Cyclic Redundancy Check (CRC) that provides error detection for multi-bit errors. Refer the MCX Nx4x Reference Manual to see this chapter.

NOTE

The EdgeLock Secure Subsystem (ELS) is also known as EdgeLock Secure Enclave, Core Profile (ELE). This document uses the ELS name, but other materials might refer to this module as EdgeLock Secure Enclave, Core Profile or ELE.

The following figure shows the MCX Nx4x security system diagram:

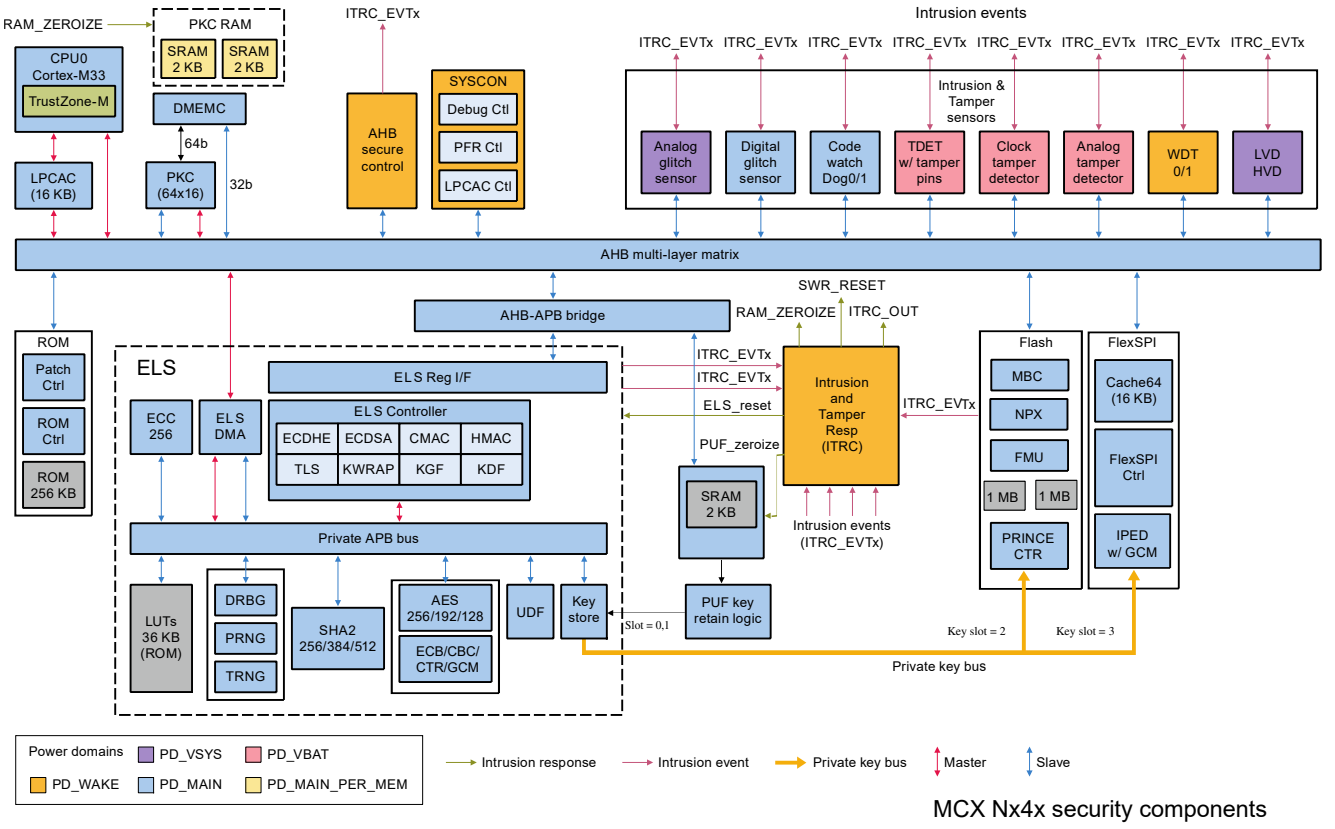


Figure 4. Security system

3.3 Immutable Root of Trust

As defined by Trusted Computing Group, “an Immutable Root of Trust (RoT) is expected to remain identical across all devices within a set of device models based on a defined threat model. It is also expected not to change across time and, therefore, will behave the same during each device’s lifespan.” It consists of truly immutable hardware logic, including analog and digital logic, read-only memory and one-time programmable memory. Immutable RoT is essential for guaranteeing any security feature, including Secure Boot, Secure Debug, Life-cycle Management, and a number of others. In this device, Immutable RoT is embedded in the Boot ROM (immutable bootloader). It uses the device hardware cryptographic functionality for its function.

3.4 Life-cycle management

During its lifespan, a typical device finds itself in various places around the world. It is manufactured in a semiconductor factory, tested and packaged in silicon manufacturer facilities, sold to various distributors, sold further to the Original Equipment Manufacturer (OEM), assembled, tested and provisioned by their Contract Manufacturers and, finally, delivered to the end-customer. In the case of failure, a device is returned to OEM or even back to the silicon manufacturer for further failure analysis.

Device life-cycle state is used to reflect the actual state of the device, which is further used to instruct the device on how exactly to protect the assets a device hosts during specific time. For example, when a device is being tested at a silicon manufacturer facility and no OEM or end-customer assets have been provisioned on it, then access to the device, in terms of debug or test, is less restrictive than when a device is with the end-customer.

Life-cycle state is monotonic, meaning it can only always be increased. Immutable RoT is in charge of life-cycle management and it enforces device access policies accordingly. Refer Life cycle states chapter in MCX Nx4x Security Reference Manual for details.

3.5 Secure boot

Secure Boot ensures authenticity, integrity and confidentiality of the device bootloader, firmware, and other software during the boot process and ensures that the intended secure life-cycle state is reached. ECDSA P-256 with SHA-256 or ECDSA P-384 with SHA-384 are there to guarantee authenticity and integrity of the firmware image. PRINCE-based memory encryption using a device-unique key derived from PUF can be used to provide code/data confidentiality while the firmware image is stored in internal or external FLASH. Immutable RoT is in charge of enforcing Secure Boot and it does so according to the policies defined by the life-cycle state.

Refer Overview chapter for details. Also see "NPX submodule" section in the Flash controller chapter and PRINCE IP GCM function support section in the FlexSPI chapter.

3.6 Secure update

Secure Update is the process used to securely update the firmware image in the field. The firmware image is encrypted using AES-128 or AES-256 and signed using ECDSA P-256 or ECDSA P-384, following the SB3.1 firmware image format. Secure Update guarantees authenticity and confidentiality of the new image. It also ensures that the new image is up-to-date, preventing the rollback to an older image. Running firmware is in charge of receiving and verifying the new firmware image. The follow-up Secure Boot verifies the new firmware image again, making sure the Immutable RoT is still in charge of ensuring authenticity of the latest firmware.

Refer Overview chapter for details.

3.7 Secure debug

Secure Debug is the process used to securely access debug, following the policy defined by the life-cycle state. When a debugger wants to debug a device, they indicate that through a so-called debug mailbox and initiate reset. During boot, a device is checking the debug mailbox and starting a full asymmetric-key crypto based challenge-response protocol. A 128-bit random challenge is issued by the device and signed by a debugger. The response is then verified by the device and, if successful, debug is allowed. Immutable RoT is in charge of the whole process.

Refer Debug mailbox chapter for details.

3.8 IP protection

IP Protection is a set of mechanisms used to protect confidentiality and integrity of valuable code and data stored on the device. During Secure Update, the firmware image is encrypted using AES. While at rest and during Secure Boot, the firmware image is encrypted using PRINCE. Data or code stored in internal or external memory is encrypted using PRINCE as well. PRINCE encryption is done using a device-unique key, which is derived from PUF by the Immutable RoT. Up to 4 independent memory regions of internal FLASH and up to 4 independent memory regions of external FLASH can be used for protecting IP content from various vendors.

3.9 Secure isolation

Arm TrustZone enables Secure Isolation during run-time by providing four distinct levels of privilege: secure-privilege, secure-user, non-secure-privilege, non-secure-user. Every peripheral is equipped with Peripheral Protection Checker (PPC) that can be programmed to control access to that peripheral, following the Arm TrustZone philosophy. Every memory is equipped with Memory Protection Checker (MPC) that can also be programmed in the same way as the PPC. Secure AHB Controller is in charge of programming all PPC and MPC blocks and only the highest level of privilege, which is secure-privilege, is allowed to do that.

As highlighted by the mechanisms we use for IP Protection, PRINCE-based memory encryption also ensures Secure Isolation between multiple IP vendors. Initial Vector (IV) is derived by secure-privilege and a different value is used for every independent memory region, ensuring the isolation between each other.

3.10 Secure attestation

Secure Attestation is a set of mechanisms used to provide evidence to a remote party on the device's genuine identity, its software and firmware versions, as well as its integrity and lifecycle state. Device Identity Composition Engine (DICE), as defined by Trusted Computing Group, uses Immutable RoT during boot time to create a unique Device Identity which takes into account Unique Device Secret (UDS), hardware state of the device and its firmware. The DICE feature is implemented using the ELS Runtime Fingerprint (RTF) in this device. RTF is the NXP-proprietary attestation mechanism, which measures the device's state during boot-time and run-time as well.

3.11 Secure storage

Immutable RoT, including PUF as the source of device-unique master key, is in charge of key derivation and key management. RFC3394 using device-unique key wrapping key is used as a method for securely storing keys. AES GCM using device-unique keys is used for securely storing valuable data. PRINCE memory encryption using device-unique key is used for storing valuable code in internal and external FLASH.

3.12 Trust provisioning

Trust provisioning is a process used for creation of initial Device Identity keys. Its major objective is to provide a cryptographic proof of the device's origin and to offer a set of tools to OEM for secure provisioning of their own assets. In a nutshell, a device-unique private-public key pair is created on every device, the public portion of which is collected and signed by NXP. That signed public key is installed back onto every device in a form of device-unique certificate, which serves as the actual proof of the device's origin. The corresponding private key, together with other pre-installed key material, is then used for authentication and secure connection to the device, enabling secure provisioning of OEM assets even in a manufacturing environment OEM may not fully trust.

3.13 Secure key management

Secure key management is a process of securing valuable keys and various key material which are essential in maintaining security of the end-user, OEM and NXP assets. The process strongly relies on the Immutable RoT consisting of PUF, hardware logic and ROM. A device-unique master key is provided by PUF and only used for further key derivation. Part of the derivation data is supplied by the Immutable RoT, accurately reflecting the device's state, making sure different keys are derived in different states. For example, different life-cycle state will often yield a different derived key. Similarly, when the debug port is open a different key will be derived than when the debug port is closed. All the platform keys reside within a security subsystem, hidden from the application core at all times.

Refer Overview chapter for details.

3.14 Anomaly Detection and Reaction

Anomaly Detection and Reaction describes the processes or algorithms that analyze the device input and output such as sensor data, as well as the software integrity and application operation for abnormal events and, if required, trigger and execute an action. Typically these actions encompass logging the anomaly, issuing a message to the cloud backend, resetting the device, and/or changing a life cycle state.

Chapter 4
Core Overview

4.1 Introduction

This section covers the core modules included in this chip.

NOTE

This chapter is also repeated in the MCX Nx4x Reference Manual, with differences only in the Peripheral Bridge memory map tables in Peripheral Bridge (PBRG). The sections in the Reference Manual does not show the security relevant information in it.

NOTE

IFR and PFR refer to "Protected Flash Regions" in this document.

4.2 CPU0 and CPU1 Cortex-M33 Code and System buses

CPU0 is the primary Cortex-M33 (ver r0p4-00rel0) processor, which supports TrustZone-M, Floating Point Unit (FPU), and Memory Protection Unit (MPU).

The MCX Nx4x device also includes a second instance of Cortex-M33, CPU1, which is the secondary Mirco-CM33 intended to offload the main processor to support special dedicated applications. The configuration of this instance does not include MPU, FPU, DSP, ETM, Trustzone-M, Secure Attribution Unit (SAU) or co-processor interface. SYSTICK is supported on both cores.

Cortex-M33 implements a modified Harvard memory architecture using two 32-bit bus interfaces: the Code and System buses. The bus interfaces are activated by address range and can include both instruction fetches and operand data references on a given bus port. (A traditional Harvard architecture strictly separates instruction fetches and operand data references onto specific bus ports regardless of access address.) The Code bus is typically used for instruction fetching and data accesses of PC-relative data, while the system bus is typically used for operand data references to the on-chip and off-chip memories and peripheral accesses. The bus structure fully supports concurrent instruction fetch and data accesses, but the Cortex-M33 implementations can generate both types of references on each bus.

NOTE

It is recommended that performance critical code be located such that it fetches from the Code bus interface as defined by addresses < 0x2000_0000.

4.2.1 Code Bus access

CPU0 and CPU1 code bus can access memory as shown in the "Bus matrix block diagram", shown in the MCX Nx4x Reference Manual. CPU0 code bus access is routed to Low Power Cache (LPCAC) controller. This controller then processes the cacheable accesses as needed, while bypassing the non-cacheable accesses or forwarding the cache write-through and cache miss accesses to the downstream memories through the master port of this cache controller.

4.2.2 System bus access

All System bus accesses are routed to the target address in destination memories through multilayer AHB matrix slave port. See "Bus matrix block diagram" in the MCX Nx4x Reference Manual.

4.2.3 Access control

CPU0 Code and System Bus accesses are checked by the core access control logic, that is, IDAU/SAU and MPU. All requests that miss or bypass the cache are checked by downstream secure AHB bus logic. The caches include protection control signals (HPROT[3:0], which is defined in Arm AHB protocol) and processing domain bits as part of the tags. If a fetch address hits the cache but the protection control and/or domain bits are different, the cache controller forces a miss with the allocate location the

same as the address hit location in the cache. This policy allows all the downstream checks to take place, and this new miss is loaded in the cache with the updated protection control and domain bits overwriting the line with the same address. This keeps the cache coherent while always checking accesses that need to see the downstream checks.

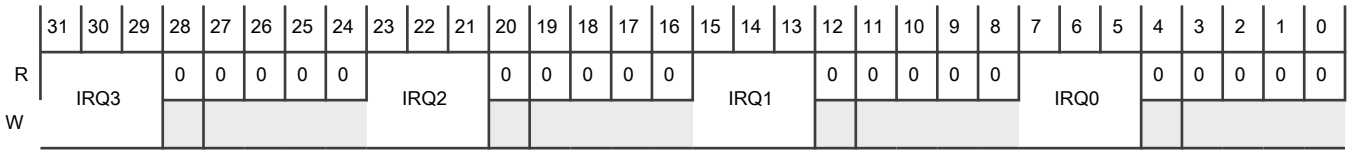
CPU1 access is controlled by secure AHB control.

Read, write, and execute access control to on-chip flash is controlled by Memory Block Checker (MBC). See MBC chapter in MCX Nx4x Security Reference Manual.

4.3 Nested Vectored Interrupt Controller (NVIC)

4.3.1 Interrupt priority levels

This device supports 8 priority levels for interrupts. Therefore, in the NVIC each source in the IPR registers contains 3 bits. For example, IPR0 is shown below:



4.3.2 Non-Maskable Interrupt (NMI) configuration

The Non-Maskable Interrupt (NMI) enable bit and source selection bits are implemented for each core in SYSCON NMI Source Select register.

4.3.3 Interrupt channel assignments

The interrupt source assignments are defined in the following table.

- Vector number - the value stored on the stack when an interrupt is serviced.
- IRQ number - non-core interrupt source count, which is the vector number minus 16.

The IRQ number is used within Arm's NVIC documentation.

Table 4. Interrupt Vector Assignments

Address	Vector	IRQ	NVIC non-IPR register number	NVIC IPR register number	Alias	Source Description
0000_0000	0	-	-	-	Cortex-M33	Initial Stack Pointer
0000_0004	1	-	-	-	Cortex-M33	Initial Program Counter
0000_0008	2	-	-	-	Cortex-M33	Non-Maskable Interrupt (NMI)
0000_000C	3	-	-	-	Cortex-M33	Hard Fault
0000_0010	4	-	-	-	Cortex-M33	MemManage Fault
0000_0014	5	-	-	-	Cortex-M33	Bus Fault
0000_0018	6	-	-	-	Cortex-M33	Usage Fault
0000_001C	7	-	-	-	Cortex-M33	Secure fault

Table continues on the next page...

Table 4. Interrupt Vector Assignments (continued)

Address	Vector	IRQ	NVIC non-IPR register number	NVIC IPR register number	Alias	Source Description
0000_0020	8	-	-	-	Reserved	
0000_0024	9	-	-	-	Reserved	
0000_0028	10	-	-	-	Reserved	
0000_002C	11	-	-	-	Cortex-M33	Supervisor Call (SVCall)
0000_0030	12	-	-	-	Cortex-M33	Debug Monitor
0000_0034	13	-	-	-	Reserved	
0000_0038	14	-	-	-	Cortex-M33	Pendable request for system service (Pendable SrvReq)
0000_003C	15	-	-	-	Cortex-M33	System Tick Timer
0000_0040	16	0	0	0		ORs IRQs 1 - 155 (OR IRQ[155:1])
0000_0044	17	1	0	0	eDMA_0	eDMA_0_CH0 error or transfer complete
0000_0048	18	2	0	0	eDMA_0	eDMA_0_CH1 error or transfer complete
0000_004C	19	3	0	0	eDMA_0	eDMA_0_CH2 error or transfer complete
0000_0050	20	4	0	1	eDMA_0	eDMA_0_CH3 error or transfer complete
0000_0054	21	5	0	1	eDMA_0	eDMA_0_CH4 error or transfer complete
0000_0058	22	6	0	1	eDMA_0	eDMA_0_CH5 error or transfer complete
0000_005C	23	7	0	1	eDMA_0	eDMA_0_CH6 error or transfer complete
0000_0060	24	8	0	2	eDMA_0	eDMA_0_CH7 error or transfer complete
0000_0064	25	9	0	2	eDMA_0	eDMA_0_CH8 error or transfer complete
0000_0068	26	10	0	2	eDMA_0	eDMA_0_CH9 error or transfer complete
0000_006C	27	11	0	2	eDMA_0	eDMA_0_CH10 error or transfer complete
0000_0070	28	12	0	3	eDMA_0	eDMA_0_CH11 error or transfer complete
0000_0074	29	13	0	3	eDMA_0	eDMA_0_CH12 error or transfer complete
0000_0078	30	14	0	3	eDMA_0	eDMA_0_CH13 error or transfer complete
0000_007C	31	15	0	3	eDMA_0	eDMA_0_CH14 error or transfer complete
0000_0080	32	16	0	4	eDMA_0	eDMA_0_CH15 error or transfer complete
0000_0084	33	17	0	4	GPIO0	GPIO0 interrupt 0
0000_0088	34	18	0	4	GPIO0	GPIO0 interrupt 1
0000_008C	35	19	0	4	GPIO1	GPIO1 interrupt 0
0000_0090	36	20	0	5	GPIO1	GPIO1 interrupt 1
0000_0094	37	21	0	5	GPIO2	GPIO2 interrupt 0

Table continues on the next page...

Table 4. Interrupt Vector Assignments (continued)

Address	Vector	IRQ	NVIC non-IPR register number	NVIC IPR register number	Alias	Source Description
0000_0098	38	22	0	5	GPIO2	GPIO2 interrupt 1
0000_009C	39	23	0	5	GPIO3	GPIO3 interrupt 0
0000_00A0	40	24	0	6	GPIO3	GPIO3 interrupt 1
0000_00A4	41	25	0	6	GPIO4	GPIO4 interrupt 0
0000_00A8	42	26	0	6	GPIO4	GPIO4 interrupt 1
0000_00AC	43	27	0	6	GPIO5	GPIO5 interrupt 0
0000_00B0	44	28	0	7	GPIO5	GPIO5 interrupt 1
0000_00B4	45	29	0	7	UTICK0	Micro-Tick Timer interrupt
0000_00B8	46	30	0	7	MRT0	Multi-Rate Timer interrupt
0000_00BC	47	31	0	7	CTIMER0	Standard counter/timer 0 interrupt
0000_00C0	48	32	1	8	CTIMER1	Standard counter/timer 1 interrupt
0000_00C4	49	33	1	8	SCT0	SCTIMER/PWM interrupt
0000_00C8	50	34	1	8	CTIMER2	Standard counter/timer 2 interrupt
0000_00CC	51	35	1	8	LP_FLEXCOMM0	LP_FLEXCOMM0 (LPSPI interrupt or LPI2C interrupt or LPUART Receive/Transmit interrupt)
0000_00D0	52	36	1	9	LP_FLEXCOMM1	LP_FLEXCOMM1 (LPSPI interrupt or LPI2C interrupt or LPUART Receive/Transmit interrupt)
0000_00D4	53	37	1	9	LP_FLEXCOMM2	LP_FLEXCOMM2 (LPSPI interrupt or LPI2C interrupt or LPUART Receive/Transmit interrupt)
0000_00D8	54	38	1	9	LP_FLEXCOMM3	LP_FLEXCOMM3 (LPSPI interrupt or LPI2C interrupt or LPUART Receive/Transmit interrupt)
0000_00DC	55	39	1	9	LP_FLEXCOMM4	LP_FLEXCOMM4 (LPSPI interrupt or LPI2C interrupt or LPUART Receive/Transmit interrupt)
0000_00E0	56	40	1	10	LP_FLEXCOMM5	LP_FLEXCOMM5 (LPSPI interrupt or LPI2C interrupt or LPUART Receive/Transmit interrupt)
0000_00E4	57	41	1	10	LP_FLEXCOMM6	LP_FLEXCOMM6 (LPSPI interrupt or LPI2C interrupt or LPUART Receive/Transmit interrupt)
0000_00E8	58	42	1	10	LP_FLEXCOMM7	LP_FLEXCOMM7 (LPSPI interrupt or LPI2C interrupt or LPUART Receive/Transmit interrupt)
0000_00EC	59	43	1	10	LP_FLEXCOMM8	LP_FLEXCOMM8 (LPSPI interrupt or LPI2C interrupt or LPUART Receive/Transmit interrupt)
0000_00F0	60	44	1	11	LP_FLEXCOMM8	LP_FLEXCOMM8 (LPSPI interrupt or LPI2C interrupt or LPUART Receive/Transmit interrupt)

Table continues on the next page...

Table 4. Interrupt Vector Assignments (continued)

Address	Vector	IRQ	NVIC non-IPR register number	NVIC IPR register number	Alias	Source Description
0000_00F4	61	45	1	11	ADC0	Analog-to-Digital Converter 0 - General Purpose interrupt
0000_00F8	62	46	1	11	ADC1	Analog-to-Digital Converter 1 - General Purpose interrupt
0000_00FC	63	47	1	11	PINT0	Pin Interrupt Pattern Match Interrupt
0000_0100	64	48	1	12	MICFIL0	Microphone Interface interrupt
0000_0104	65	49	1	12	Reserved	Reserved
0000_0108	66	50	1	12	USBFS0	Universal Serial Bus - Full Speed interrupt
0000_010C	67	51	1	12	USBDCD0	Universal Serial Bus - Device Charge Detect interrupt
0000_0110	68	52	1	13	RTC0	RTC Subsystem interrupt (RTC interrupt or Wake timer interrupt)
0000_0114	69	53	1	13	SmartDMA	SmartDMA_IRQ
0000_0118	70	54	1	13	MAILBOX0	Inter-CPU Mailbox interrupt0 for CPU0 Inter-CPU Mailbox interrupt1 for CPU1
0000_011C	71	55	1	13	CTIMER3	Standard counter/timer 3 interrupt
0000_0120	72	56	1	14	CTIMER4	Standard counter/timer 4 interrupt
0000_0124	73	57	1	14	OSTIMER0	OS event timer interrupt
0000_0128	74	58	1	14	FlexSPI0	Flexible Serial Peripheral Interface interrupt
0000_012C	75	59	1	14	SAI0	Serial Audio Interface 0 interrupt
0000_0130	76	60	1	15	SAI1	Serial Audio Interface 1 interrupt
0000_0134	77	61	1	15	uSDHC0	Ultra Secured Digital Host Controller interrupt
0000_0138	78	62	1	15	CAN0	Controller Area Network 0 interrupt
0000_013C	79	63	1	15	CAN1	Controller Area Network 1 interrupt
0000_0140	80	64	2	16	Reserved	—
0000_0144	81	65	2	16	Reserved	—
0000_0148	82	66	2	16	USBHS1_PHY	USBHS DCD or USBHS Phy interrupt
0000_014C	83	67	2	16	USBHS1	USB High Speed OTG Controller interrupt
0000_0150	84	68	2	17	SEC_HYPERVISOR_CALL	AHB Secure Controller hypervisor call interrupt
0000_0154	85	69	2	17	Reserved	—
0000_0158	86	70	2	17	PLU0	Programmable Logic Unit interrupt

Table continues on the next page...

Table 4. Interrupt Vector Assignments (continued)

Address	Vector	IRQ	NVIC non-IPR register number	NVIC IPR register number	Alias	Source Description
0000_015C	87	71	2	17	FREQME0	Frequency Measurement interrupt
0000_0160	88	72	2	18	SEC_Violation	Secure violation interrupt (Memory Block Checker (MBC) interrupt or Secure AHB Bus and AHB Controller (AHBSC) matrix violation interrupt)
0000_0164	89	73	2	18	ELS	ELS interrupt
0000_0168	90	74	2	18	PKC	PKC interrupt
0000_016C	91	75	2	18	PUF	Physical Unclonable Function interrupt
0000_0170	92	76	2	19	POWERQUAD0	PowerQuad interrupt
0000_0174	93	77	2	19	eDMA_1	eDMA_1_CH0 error or transfer complete
0000_0178	94	78	2	19	eDMA_1	eDMA_1_CH1 error or transfer complete
0000_017C	95	79	2	19	eDMA_1	eDMA_1_CH2 error or transfer complete
0000_0180	96	80	2	20	eDMA_1	eDMA_1_CH3 error or transfer complete
0000_0184	97	81	2	20	eDMA_1	eDMA_1_CH4 error or transfer complete
0000_0188	98	82	2	20	eDMA_1	eDMA_1_CH5 error or transfer complete
0000_018C	99	83	2	20	eDMA_1	eDMA_1_CH6 error or transfer complete
0000_0190	100	84	2	21	eDMA_1	eDMA_1_CH7 error or transfer complete
0000_0194	101	85	2	21	eDMA_1	eDMA_1_CH8 error or transfer complete
0000_0198	102	86	2	21	eDMA_1	eDMA_1_CH9 error or transfer complete
0000_019C	103	87	2	21	eDMA_1	eDMA_1_CH10 error or transfer complete
0000_01A0	104	88	2	22	eDMA_1	eDMA_1_CH11 error or transfer complete
0000_01A4	105	89	2	22	eDMA_1	eDMA_1_CH12 error or transfer complete
0000_01A8	106	90	2	22	eDMA_1	eDMA_1_CH13 error or transfer complete
0000_01AC	107	91	2	22	eDMA_1	eDMA_1_CH14 error or transfer complete
0000_01B0	108	92	2	23	eDMA_1	eDMA_1_CH15 error or transfer complete
0000_01B4	109	93	2	23	CDOG0	Code Watchdog Timer 0 interrupt
0000_01B8	110	94	2	23	CDOG1	Code Watchdog Timer 1 interrupt
0000_01BC	111	95	2	23	I3C0	Improved Inter Integrated Circuit interrupt 0
0000_01C0	112	96	3	24	I3C1	Improved Inter Integrated Circuit interrupt 1
0000_01C4	113	97	3	24	NPU	NPU interrupt
0000_01C8	114	98	3	24	GDET0 & GDET1	Digital Glitch Detect 0 interrupt or Digital Glitch Detect 1 interrupt

Table continues on the next page...

Table 4. Interrupt Vector Assignments (continued)

Address	Vector	IRQ	NVIC non-IPR register number	NVIC IPR register number	Alias	Source Description
						NOTE The SYSCON module has the fields that are used to enable and clear the GDET interrupts.
0000_01CC	115	99	3	24	VBAT0	VBAT interrupt (VBAT interrupt or digital tamper interrupt)
0000_01D0	116	100	3	25	EWM0	External Watchdog Monitor interrupt
0000_01D4	117	101	3	25	TSI0	TSI End of Scan interrupt
0000_01D8	118	102	3	25	TSI0	TSI End of Scan interrupt
0000_01DC	119	103	3	25	EMVSIM0	EMVSIM0 interrupt
0000_01E0	120	104	3	26	EMVSIM1	EMVSIM1 interrupt
0000_01E4	121	105	3	26	FlexIO0	Flexible Input/Output interrupt
0000_01E8	122	106	3	26	DAC0	12-bit Digital-to-Analog Converter 0 - General Purpose interrupt
0000_01EC	123	107	3	26	DAC1	12-bit Digital-to-Analog Converter 1 - General Purpose interrupt
0000_01F0	124	108	3	27	DAC2	14-bit Digital-to-Analog Converter interrupt
0000_01F4	125	109	3	27	CMP0	Comparator0 interrupt
0000_01F8	126	110	3	27	CMP1	Comparator1 interrupt
0000_01FC	127	111	3	27	CMP2	Comparator2 interrupt
0000_0200	128	112	3	28	PWM0	FlexPWM0_reload_error interrupt
0000_0204	129	113	3	28	PWM0	FlexPWM0_fault interrupt
0000_0208	130	114	3	28	PWM0	FlexPWM0 Submodule 0 capture/compare/ reload interrupt
0000_020C	131	115	3	28	PWM0	FlexPWM0 Submodule 1 capture/compare/ reload interrupt
0000_0210	132	116	3	29	PWM0	FlexPWM0 Submodule 2 capture/compare/ reload interrupt
0000_0214	133	117	3	29	PWM0	FlexPWM0 Submodule 3 capture/compare/ reload interrupt
0000_0218	134	118	3	29	PWM1	FlexPWM1_reload_error interrupt
0000_021C	135	119	3	29	PWM1	FlexPWM1_fault interrupt

Table continues on the next page...

Table 4. Interrupt Vector Assignments (continued)

Address	Vector	IRQ	NVIC non-IPR register number	NVIC IPR register number	Alias	Source Description
0000_0220	136	120	3	30	PWM1	FlexPWM1 Submodule 0 capture/compare/ reload interrupt
0000_0224	137	121	3	30	PWM1	FlexPWM1 Submodule 1 capture/compare/ reload interrupt
0000_0228	138	122	3	30	PWM1	FlexPWM1 Submodule 2 capture/compare/ reload interrupt
0000_022C	139	123	3	30	PWM1	FlexPWM1 Submodule 3 capture/compare/ reload interrupt
0000_0230	140	124	3	31	QDC0	QDC0_Compare interrupt
0000_0234	141	125	3	31	QDC0	QDC0_Home interrupt
0000_0238	142	126	3	31	QDC0	QDC0_WDG_IRQ/SAB interrupt
0000_023C	143	127	3	31	QDC0	QDC0_IDX interrupt
0000_0240	144	128	4	32	QDC1	QDC1_Compare interrupt
0000_0244	145	129	4	32	QDC1	QDC1_Home interrupt
0000_0248	146	130	4	32	QDC1	QDC1_WDG_IRQ/SAB interrupt
0000_024C	147	131	4	32	QDC1	QDC1_IDX interrupt
0000_0250	148	132	4	33	ITRC0	Intrusion and Tamper Response Controller interrupt
0000_0254	149	133	4	33	BSP32	CoolFlux BSP32 interrupt
0000_0258	150	134	4	33	ELS	ELS error interrupt
0000_025C	151	135	4	33	PKC	PKC error interrupt
0000_0260	152	136	4	34	ERM0	ERM Single Bit error interrupt
0000_0264	153	137	4	34	ERM0	ERM Multi Bit error interrupt
0000_0268	154	138	4	34	FMU0	Flash Management Unit interrupt
0000_026C	155	139	4	34	ENET0	Ethernet QoS interrupt
0000_0270	156	140	4	35	ENET0	Ethernet QoS power management interrupt
0000_0274	157	141	4	35	ENET0	Ethernet QoS MAC LPI interrupt
0000_0278	158	142	4	35	SINC0	SINC Filter interrupt
0000_027C	159	143	4	35	LPTMR0	Low Power Timer 0 interrupt
0000_0280	160	144	4	36	LPTMR1	Low Power Timer 1 interrupt
0000_0284	161	145	4	36	SCG0	System Clock Generator interrupt
0000_0288	162	146	4	36	SPC0	System Power Controller interrupt
0000_028C	163	147	4	36	WUU0	Wake Up Unit interrupt

Table continues on the next page...

Table 4. Interrupt Vector Assignments (continued)

Address	Vector	IRQ	NVIC non-IPR register number	NVIC IPR register number	Alias	Source Description
0000_0290	164	148	4	37	PORT	PORT0~5 EFT interrupt
0000_0294	165	149	4	37	ETB0	ETB counter expires interrupt
0000_0298	166	150	4	37	Reserved	—
0000_029C	167	151	4	37	Reserved	—
0000_02A0	168	152	4	38	WWDT0	Windowed Watchdog Timer 0 interrupt
0000_02A4	169	153	4	38	WWDT1	Windowed Watchdog Timer 1 interrupt
0000_02A8	170	154	4	38	CMC0	Core Mode Controller interrupt
0000_02AC	171	155	4	38	CTI0	Cross Trigger Interface (CTI) interrupt **Note:** CTI interrupt is generated by its own CPU. CPU0 at slot 155 is CPU0 CTI interrupt. CPU1 at slot 155 is CPU1 CTI interrupt.

4.4 Implementation Defined Attribution Unit (IDAU)

MCX Nx4x implements a simple attribution unit that divides whole memory map into secure or non-secure regions. All peripherals and memories are aliased at two locations.

- Address 0x0000_0000 to 0x1FFF_FFFF
  - Non-Secure always
- Address 0x2000_0000 to 0xFFFF_FFFF
  - If Address Bit_28 = 0 Non-Secure
  - If Address Bit_28 = 1 Secure

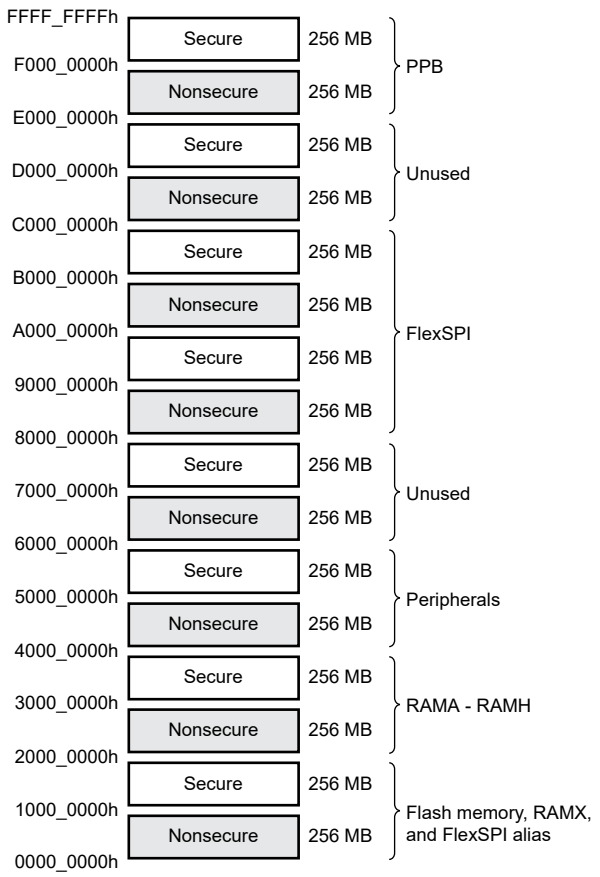


Figure 5. IDAU implementation

4.5 System memory map

The following table shows the chip's high-level memory map.

Additionally, the various sections are split into "Secure" and "Non-Secure" apertures for each IP. Although these apertures address the same IP, access through the "Secure" aperture can only be performed by code with the appropriate security settings.

NOTE

The secure space is the alias of the non-secure region.

Table 5. System memory map

Start address	End address	TZ-M State[1]	Description	Size	Cached
0000_0000	001F_FFFF	Non-Secure	Program flash **Note:** Program flash has two flash memory arrays/ banks of up to 1 MB each.	2 MB	LPCAC
0020_0000	00FF_FFFF		Reserved	—	—
0100_0000	0100_7FFF		Flash Bank 0 IFR 0	32 KB	No
0100_8000	0100_FFFF		Flash Bank 1 IFR 0	32 KB	No

Table continues on the next page...

Table 5. System memory map (continued)

Start address	End address	TZ-M State[1]	Description	Size	Cached
0101_0000	010F_FFFF		Reserved	—	—
0110_0000	0110_1FFF		Flash Bank 0 IFR 1	8 KB	No
0110_2000	0110_3FFF		Flash Bank 1 IFR 1	8 KB	No
0110_4000	02FF_FFFF		Reserved	—	—
0300_0000	0303_FFFF		ROM-BOOT	256 KB	No
0304_0000	03FF_FFFF		Reserved	—	—
0400_0000	0401_7FFF		RAMX	96 KB	No
0401_8000	07FF_FFFF		Reserved	—	—
0800_0000	0FFF_FFFF		Flexible Serial Peripheral Interface (FlexSPI) **Note:** The 128 MB FlexSPI at address 0x0800_0000 is aliased to the system memory map FlexSPI region at address 0x8000_0000 - 0x87FF_FFFF.	128 MB	LPCAC/ CACHE64
1000_0000	101F_FFFF	Secure	Program Flash **Note:** Program flash has two flash memory arrays/ banks of up to 1 MB each.	2 MB	LPCAC
1020_0000	10FF_FFFF		Reserved	—	—
1100_0000	1100_7FFF		Flash Bank 0 IFR 0	32 KB	No
1100_8000	1100_FFFF		Flash Bank 1 IFR 0	32 KB	No
1101_0000	110F_FFFF		Reserved	—	—
1110_0000	1110_1FFF		Flash Bank 0 IFR 1	8 KB	No
1110_2000	1110_3FFF		Flash Bank 1 IFR 1	8 KB	No
1110_4000	12FF_FFFF		Reserved	—	—
1300_0000	1303_FFFF		ROM-BOOT	256 KB	No
1304_0000	13FF_FFFF		Reserved	—	—
1400_0000	1401_7FFF		RAMX	96 KB	No
1401_8000	17FF_FFFF		Reserved	—	—
1800_0000	1FFF_FFFF		Flexible Serial Peripheral Interface (FlexSPI) **Note:** The 128 MB FlexSPI at address 0x1800_0000 is aliased to the system memory map FlexSPI region at address 0x9000_0000-0x97FF_FFFF.	128 MB	LPCAC/ CACHE64
2000_0000	2000_7FFF	Non-Secure	RAMA	32 KB	No
2000_8000	2000_FFFF		RAMB	32 KB	No

Table continues on the next page...

Table 5. System memory map (continued)

Start address	End address	TZ-M State[1]	Description	Size	Cached
2001_0000	2001_FFFF		RAMC	64 KB	No
2002_0000	2002_FFFF		RAMD	64 KB	No
2003_0000	2003_FFFF		RAME	64 KB	No
2004_0000	2004_FFFF		RAMF	64 KB	No
2005_0000	2005_FFFF		RAMG[2]	64 KB	No
2006_0000	2006_7FFF		RAMH[3,2]	32 KB	No
2006_8000	27FF_FFFF		Reserved	—	—
2800_0000	2FFF_FFFF		Reserved	—	—
3000_0000	3000_7FFF	Secure	RAMA	32 KB	No
3000_8000	3000_FFFF		RAMB	32 KB	No
3001_0000	3001_FFFF		RAMC	64 KB	No
3002_0000	3002_FFFF		RAMD	64 KB	No
3003_0000	3003_FFFF		RAME	64 KB	No
3004_0000	3004_FFFF		RAMF	64 KB	No
3005_0000	3005_FFFF		RAMG[2]	64 KB	No
3006_0000	3006_7FFF		RAMH[2,3]	32 KB	No
3006_8000	37FF_FFFF		Reserved	—	—
3800_0000	3FFF_FFFF	Secure	Reserved	—	—
4000_0000	4005_FFFF	Non-Secure	Peripheral Bridge 0 - PBRG0	384 KB	No
4006_0000	4007_FFFF		Reserved	—	—
4008_0000	4009_FFFF	Non-Secure	Peripheral Bridge 1 - PBRG1	128 KB	No
400A_0000	400B_FFFF		Peripheral Bridge 2 - PBRG2	128 KB	No
400C_0000	400D_FFFF		Peripheral Bridge 3 - PBRG3	128 KB	No
400E_0000	400F_FFFF		Reserved	—	—
4010_0000	4013_FFFF	Non-Secure	Peripheral Bridge 4 - PBRG4	256 KB	No
4014_0000	4FFF_FFFF		Reserved	—	—
5000_0000	5005_FFFF	Secure	Peripheral Bridge 0 - PBRG0	384 KB	No
5006_0000	5007_FFFF		Reserved	—	—
5008_0000	5009_FFFF	Secure	Peripheral Bridge 1 - PBRG1	128 KB	No
500A_0000	500B_FFFF		Peripheral Bridge 2 - PBRG2	128 KB	No
500C_0000	500D_FFFF		Peripheral Bridge 3 - PBRG3	128 KB	No
500E_0000	500F_FFFF		Reserved	—	—

Table continues on the next page...

Table 5. System memory map (continued)

Start address	End address	TZ-M State[1]	Description	Size	Cached
5010_0000	5013_FFFF	Secure	Peripheral Bridge 4 - PBRG4	256 KB	No
5014_0000	5FFF_FFFF		Reserved	—	—
6000_0000	6FFF_FFFF		Reserved	—	—
7000_0000	7FFF_FFFF		Reserved	—	—
8000_0000	8FFF_FFFF	Non-Secure	Flexible Serial Peripheral Interface (FlexSPI)	256 MB	CACHE64
9000_0000	9FFF_FFFF	Secure	Flexible Serial Peripheral Interface (FlexSPI)	256 MB	CACHE64
A000_0000	AFFF_FFFF	Non-Secure	Flexible Serial Peripheral Interface (FlexSPI)	256 MB	CACHE64
B000_0000	BFFF_FFFF	Secure	Flexible Serial Peripheral Interface (FlexSPI)	256 MB	CACHE64
C000_0000	CFFF_FFFF		Reserved	—	—
D000_0000	DFFF_FFFF		Reserved	—	—
E000_0000	E003_FFFF		Private peripheral bus (internal)	256 KB	No
E004_0000	E00F_FFFF		Private peripheral bus (external) (includes NVIC and SYSTICK timer)	768 KB	No
E010_0000	FFFF_FFFF		Reserved	—	—

1. This is the default Targeted Security Attribute for the memory region at reset. It is set by the IDAU and the SAU configuration. The attribute listed is the intended use case for users of TrustZone-M (TZM) security function.
2. On device part numbers that support optional ECC, RAMG and RAMH can be used to provide ECC data for other RAM blocks. RAMG and RAMH are not accessible as regular memory when used for ECC, so their availability in the memory map depends on the selected ECC configuration.
3. On device part numbers that support optional ECC, RAMB and RAMX ECC is enabled by default. This means that RAMH is used to provide ECC and is not accessible as regular RAM by default.

NOTE

For detailed connection between masters and slaves, see Bus matrix block diagram.

4.6 Peripheral Bridge (PBRG)

The Peripheral Bridge (PBRG) is the portion of the bus fabric that connects the peripherals to the processor elements. Each peripheral has a base address where the processor elements can access them. The following sections provide the memory map of the peripherals connected to each of the Peripheral Bridges.

4.6.1 Peripheral Bridge 0 (PBRG0) memory map

Table 6. Peripheral bridge 0

Base address	Slot	Module	Alias
4000_0000	**APB 0**		
4000_0000	0	System controller	SYSCON
4000_1000	1	Reserved	—
4000_2000	2	Reserved	—

Table continues on the next page...

Table 6. Peripheral bridge 0 (continued)

Base address	Slot	Module	Alias
4000_3000	3	Reserved	—
4000_4000	4	Pin Interrupt and Pattern Match	PINT0
4000_5000	5	Reserved	—
4000_6000	6	Input multiplexing	INPUTMUX0
4000_7000	7	Reserved	—
4000_8000	8	Reserved	—
4000_9000	9	Reserved	—
4000_A000	10	Reserved	—
4000_B000	11	Reserved	—
4000_C000	12	Standard counter/timers	CTIMER0
4000_D000	13	Standard counter/timers	CTIMER1
4000_E000	14	Standard counter/timers	CTIMER2
4000_F000	15	Standard counter/timers	CTIMER3
4001_0000	16	Standard counter/timers	CTIMER4
4001_1000	17	Frequency Measurement Unit	FREQME0
4001_2000	18	Micro-Tick Timer	UTICK0
4001_3000	19	Multi-Rate Timer	MRT0
4001_4000	20	Reserved	—
4001_5000	21	Reserved	—
4001_6000	22	Windowed Watchdog Timer	WWDT0
4001_7000	23	Windowed Watchdog Timer	WWDT1
4001_8000	24	Reserved	—
4001_9000	25	Reserved	—
4001_A000	26	Reserved	—
4001_B000	27	CACHE64 Policy Select	CACHE64_POLSEL0
4001_C000	28	Reserved	—
4001_D000	29	Reserved	—
4001_E000	30	Reserved	—
4001_F000	31	Reserved	—
4002_0000	APB 1		
4002_0000	0	Reserved	—

Table continues on the next page...

Table 6. Peripheral bridge 0 (continued)

Base address	Slot	Module	Alias
4002_1000	1	Improved Inter-Integrated Circuit	I3C0
4002_2000	2	Improved Inter-Integrated Circuit	I3C1
4002_3000	3	Reserved	—
4002_4000	4	Digital Glitch Detector	GDET0
4002_5000	5	Digital Glitch Detector	GDET1
4002_6000	6	Intrusion and Tamper Response Controller	ITRC0
4002_7000	7	Reserved	—
4002_8000	8	Reserved	—
4002_9000	9	Reserved	—
4002_A000	10	Reserved	—
4002_B000	11	Public-key Cryptography Accelerator	PKC0
4002_C000	12	Physically Unclonable Function	PUF
4002_D000	13	Physically Unclonable Function (alias 1) - alias of PUF	PUF_alias1
4002_E000	14	Physically Unclonable Function (alias 2) - alias of PUF	PUF_alias2
4002_F000	15	Physically Unclonable Function (alias 3) - alias of PUF	PUF_alias3
4003_0000	16	Reserved	—
4003_2000	18	CoolFlux BSP32	BSP32
4003_3000	19	SmartDMA	SmartDMA
4003_4000	20	Programable Logic Unit	PLU0
4003_5000	21	Reserved	—
4003_6000	22	Reserved	—
4003_7000	23	Reserved	
4003_8000	24	Reserved	—
4003_9000	25	Reserved	—
4003_A000	26	Reserved	—
4003_B000	27	Reserved	—
4003_C000	28	Reserved	—
4003_D000	29	Reserved	—
4003_E000	30	Reserved	—
4003_F000	31	Reserved	—
4004_0000	**AIPS 0**		
4004_0000	0	General Purpose Input/Output	GPIO5

Table continues on the next page...

Table 6. Peripheral bridge 0 (continued)

Base address	Slot	Module	Alias
4004_1000	1	General Purpose Input/Output (alias 1) - alias of GPIO 5	GPIO5_alias1
4004_2000	2	Port5	PORT5
4004_3000	3	Flash Management Unit	FMU0
4004_4000	4	System Clock Generator	SCG0
4004_5000	5	System Power Controller	SPC0
4004_6000	6	Wake-Up Unit	WUU0
4004_7000	7	Reserved	—
4004_8000	8	Core Mode Controller	CMC0
4004_9000	9	OS Event Timer	OSTIMER0
4004_A000	10	Low-Power Timer	LPTMR0
4004_B000	11	Low-Power Timer	LPTMR1
4004_C000	12	Real Time Clock Subsystem and Real Time Clock	RTC_SUBSYSTEM0 and RTC0
4004_D000	13	Reserved	—
4004_E000	14	Reserved	—
4004_F000	15	Reserved	—
4005_0000	16	Touch Sensing Interface	TSI0
4005_1000	17	Low Power Comparator	CMP0
4005_2000	18	Low Power Comparator	CMP1
4005_3000	19	Low Power Comparator	CMP2
4005_4000	20	Edge Lock Secure Subsystem	ELS
4005_5000	21	Edge Lock Secure Subsystem - alias of ELS	ELS_alias1
4005_6000	22	Edge Lock Secure Subsystem - alias of ELS	ELS_alias2
4005_7000	23	Edge Lock Secure Subsystem - alias of ELS	ELS_alias3
4005_8000	24	Digital tamper	TDET0
4005_9000	25	VBAT	VBAT0
4005_A000	26	Reserved	—
4005_B000	27	Error Injection Module	EIM0
4005_C000	28	Error Recording Module	ERM0
4005_D000	29	Interrupt Monitor	INTM0
4005_E000	30	Reserved	—
4005_F000	31	Reserved	—

4.6.2 Peripheral Bridge 1 (PBRG1) memory map

Table 7. Peripheral bridge 1

Base	Slot	Module	Alias
4008_0000	**AIPS 1**		
4008_0000	0	eDMA 0 Management Page	eDMA_0_MP
4008_1000	1	eDMA 0 CH0 Control and Configuration	eDMA_0_CH0
4008_2000	2	eDMA 0 CH1 Control and Configuration	eDMA_0_CH1
4008_3000	3	eDMA 0 CH2 Control and Configuration	eDMA_0_CH2
4008_4000	4	eDMA 0 CH3 Control and Configuration	eDMA_0_CH3
4008_5000	5	eDMA 0 CH4 Control and Configuration	eDMA_0_CH4
4008_6000	6	eDMA 0 CH5 Control and Configuration	eDMA_0_CH5
4008_7000	7	eDMA 0 CH6 Control and Configuration	eDMA_0_CH6
4008_8000	8	eDMA 0 CH7 Control and Configuration	eDMA_0_CH7
4008_9000	9	eDMA 0 CH8 Control and Configuration	eDMA_0_CH8
4008_A000	10	eDMA 0 CH9 Control and Configuration	eDMA_0_CH9
4008_B000	11	eDMA 0 CH10 Control and Configuration	eDMA_0_CH10
4008_C000	12	eDMA 0 CH11 Control and Configuration	eDMA_0_CH11
4008_D000	13	eDMA 0 CH12 Control and Configuration	eDMA_0_CH12
4008_E000	14	eDMA 0 CH13 Control and Configuration	eDMA_0_CH13
4008_F000	15	eDMA 0 CH14 Control and Configuration	eDMA_0_CH14
4009_0000	16	eDMA 0 CH15 Control and Configuration	eDMA_0_CH15
4009_1000	**AHB Peripherals**		
4009_1000	17	SCTIMER	SCT0
4009_2000	18	Low-power Flexible Communications Interface	LP_FLEXCOMM0
4009_3000	19	Low-power Flexible Communications Interface	LP_FLEXCOMM1
4009_4000	20	Low-power Flexible Communications Interface	LP_FLEXCOMM2
4009_5000	21	Low-power Flexible Communications Interface	LP_FLEXCOMM3
4009_6000	22	General Purpose Input/Output	GPIO0
4009_7000	23	GPIO 0 (alias 1)	GPIO0_alias1
4009_8000	24	General Purpose Input/Output	GPIO1
4009_9000	25	GPIO 1 (alias 1)	GPIO1_alias1

Table continues on the next page...

Table 7. Peripheral bridge 1 (continued)

Base	Slot	Module	Alias
4009_A000	26	General Purpose Input/Output	GPIO2
4009_B000	27	GPIO 2 (alias 1)	GPIO2_alias1
4009_C000	28	General Purpose Input/Output	GPIO3
4009_D000	29	GPIO 3 (alias 1)	GPIO3_alias1
4009_E000	30	General Purpose Input/Output	GPIO4
4009_F000	31	GPIO 4 (alias 1)	GPIO4_alias1

4.6.3 Peripheral Bridge 2 (PBRG2) memory map

Table 8. Peripheral bridge 2

Base	Slot	Module	Alias
400A_0000	**AIPS 2**		
400A_0000	0	eDMA 1 Management Page	eDMA_1_MP
400A_1000	1	eDMA 1 CH0 Control and Configuration	eDMA_1_CH0
400A_2000	2	eDMA 1 CH1 Control and Configuration	eDMA_1_CH1
400A_3000	3	eDMA 1 CH2 Control and Configuration	eDMA_1_CH2
400A_4000	4	eDMA 1 CH3 Control and Configuration	eDMA_1_CH3
400A_5000	5	eDMA 1 CH4 Control and Configuration	eDMA_1_CH4
400A_6000	6	eDMA 1 CH5 Control and Configuration	eDMA_1_CH5
400A_7000	7	eDMA 1 CH6 Control and Configuration	eDMA_1_CH6
400A_8000	8	eDMA 1 CH7 Control and Configuration	eDMA_1_CH7
400A_9000	9	eDMA 1 CH8 Control and Configuration	eDMA_1_CH8
400A_A000	10	eDMA 1 CH9 Control and Configuration	eDMA_1_CH9
400A_B000	11	eDMA 1 CH10 Control and Configuration	eDMA_1_CH10
400A_C000	12	eDMA 1 CH11 Control and Configuration	eDMA_1_CH11
400A_D000	13	eDMA 1 CH12 Control and Configuration	eDMA_1_CH12
400A_E000	14	eDMA 1 CH13 Control and Configuration	eDMA_1_CH13
400A_F000	15	eDMA 1 CH14 Control and Configuration	eDMA_1_CH14
400B_0000	16	eDMA 1 CH15 Control and Configuration	eDMA_1_CH15
400B_1000	17	Semaphore 2	SEMA42_0
400B_2000	**AHB Peripherals**		
400B_2000	18	Inter-CPU Mailbox	MAILBOX0
400B_3000	19	Public-key Cryptography Accelerator RAM	PKC RAM

Table continues on the next page...

Table 8. Peripheral bridge 2 (continued)

Base	Slot	Module	Alias
400B_4000	20	Low-power Flexible Communications Interface	LP_FLEXCOMM4
400B_5000	21	Low-power Flexible Communications Interface	LP_FLEXCOMM5
400B_6000	22	Low-power Flexible Communications Interface	LP_FLEXCOMM6
400B_7000	23	Low-power Flexible Communications Interface	LP_FLEXCOMM7
400B_8000	24	Low-power Flexible Communications Interface	LP_FLEXCOMM8
400B_9000	25	Low-power Flexible Communications Interface	LP_FLEXCOMM9
400B_A000	26	USB FS RAM	USB0_FS_RAM
400B_B000	27	Code Watchdog	CDOG0
400B_C000	28	Code Watchdog	CDOG1
400B_D000	29	Debug Mailbox	DM0
400B_E000	30	NPU	NPU0
400B_F000	31	PowerQuad	POWERQUAD0

4.6.4 Peripheral Bridge 3 (PBRG3) memory map

Table 9. Peripheral bridge 3

Base	Slot	Module	Alias
400C_0000	AIPS 3		
400C_0000	0	External Watchdog Monitor	EWM0
400C_1000	1	Performance Monitor - LPCAC	CMX_PERFMON0
400C_2000	2	Performance Monitor - FlexSPI	CMX_PERFMON1
400C_3000	3	Reserved	—
400C_4000	4	Reserved	—
400C_5000	5	Reserved	—
400C_6000	6	Reserved	—
400C_7000	7	Memory Block Checker	MBC0
400C_8000	8	Flexible Serial Peripheral Interface	FLEXSPI0
400C_9000	9	OTP Controller	OTPC0

Table continues on the next page...

Table 9. Peripheral bridge 3 (continued)

Base	Slot	Module	Alias
400C_A000	10	Reserved	—
400C_B000	11	Cyclic Redundancy Check	CRC0
400C_C000	12	FMC with NVM PRINCE Encryption and Decryption	NPX0
400C_D000	13	Reserved	—
400C_E000	14	Enhanced Flex Pulse Width Modulator	PWM0
400C_F000	15	Quadrature Decoder	QDC0
400D_0000	16	Enhanced Flex Pulse Width Modulator	PWM1
400D_1000	17	Quadrature Decoder	QDC1
400D_2000	18	Event Generator	EVTG0
400D_3000	19	Reserved	—
400D_4000	20	FlexCAN with FD	CAN0
400D_8000	21	FlexCAN with FD	CAN1
400D_C000	22	USB FS DCD	USBDCD0
400D_D000	23	USB FS Controller	USBFS0
400D_E000	24	Reserved	—
400D_F000	25	Reserved	—

4.6.5 Peripheral Bridge 4 (PBRG4) memory map

Table 10. Peripheral bridge 4

Base	Slot	Module	Alias
4010_0000	AIPS 4		
4010_0000	0	Ethernet with QoS	ENET0
4010_2000	1	Reserved	—
4010_3000	2	EMVSIM0	EMVSIM0
4010_4000	3	EMVSIM1	EMVSIM1
4010_5000	4	Flexible Input/Output	FLEXIO0
4010_6000	5	Serial Audio Interface	SAI0
4010_7000	6	Serial Audio Interface	SAI1
4010_8000	7	SINC Filter	SINC0
4010_9000	8	Ultra Secured Digital Host Controller	uSDHC0
4010_A000	9	USB HS PHY & DCD	USBHS1_PHY
4010_B000	10	USB HS Controller	USBHS1

Table continues on the next page...

Table 10. Peripheral bridge 4 (continued)

Base	Slot	Module	Alias
4010_C000	11	Microphone Interface	MICFIL0
4010_D000	12	Analog-to-Digital Converter	ADC0
4010_E000	13	Analog-to-Digital Converter	ADC1
4010_F000	14	12-bit Digital-to-Analog Converter	DAC0
4011_0000	15	Operational Amplifier	OPAMP0
4011_1000	16	Voltage Reference	VREF0
4011_2000	17	12-bit Digital-to-Analog Converter	DAC1
4011_3000	18	Operational Amplifier	OPAMP1
4011_4000	19	14-bit Digital-to-Analog Converter	DAC2
4011_5000	20	Operational Amplifier	OPAMP2
4011_6000	21	Port Control	PORT0
4011_7000	22	Port Control	PORT1
4011_8000	23	Port Control	PORT2
4011_9000	24	Port Control	PORT3
4011_A000	25	Port Control	PORT4
4011_B000	26	Reserved	—
4011_C000	27	Reserved	—
4011_D000	28	Reserved	—
4011_E000	29	Reserved	—
4011_F000	30	Reserved	—
4012_0000	AHB Peripherals		
4012_0000	32	Secure AHB Controller	AHBSC
4012_1000	33	Secure AHB Controller Alias 1	AHBSC_alias1
4012_2000	34	Secure AHB Controller Alias 2	AHBSC_alias2
4012_3000	35	Secure AHB Controller Alias 3	AHBSC_alias3
4012_4000		Reserved	—

Chapter 5
Life Cycle States

5.1 Life cycle states for secure devices

5.1.1 Overview

This chip supports a security life cycle state model. The current life cycle state determines the device functionality, debug and test port availability, and asset accessibility. The life cycle state is controlled by the LC_STATE fuse value, and state values are selected so that additional fuse bits are burned to advance the state. Because fuses control the life cycle state, moving to a more advanced state is an irreversible and permanent process. The life cycle can only be advanced and cannot return to a previous state.

The Boot ROM is responsible for checking the life cycle state. Based on the life cycle state, the ROM determines what boot flow is used, including if control is passed to application code or not. The ROM also handles the opening of test and debug ports based on the life cycle state. If the part is in the Bricked state or any invalid life cycle state, then the ROM locks the part.

NOTE

IFR and PFR refer to "Protected Flash Regions" in this document.

5.1.2 Life cycle state transitions

The diagram below shows the lifecycle states and transitions, with the SoC state shown provisioned and/or configured as it is expected to be in the respective LC state (for example, when entering the respective state).

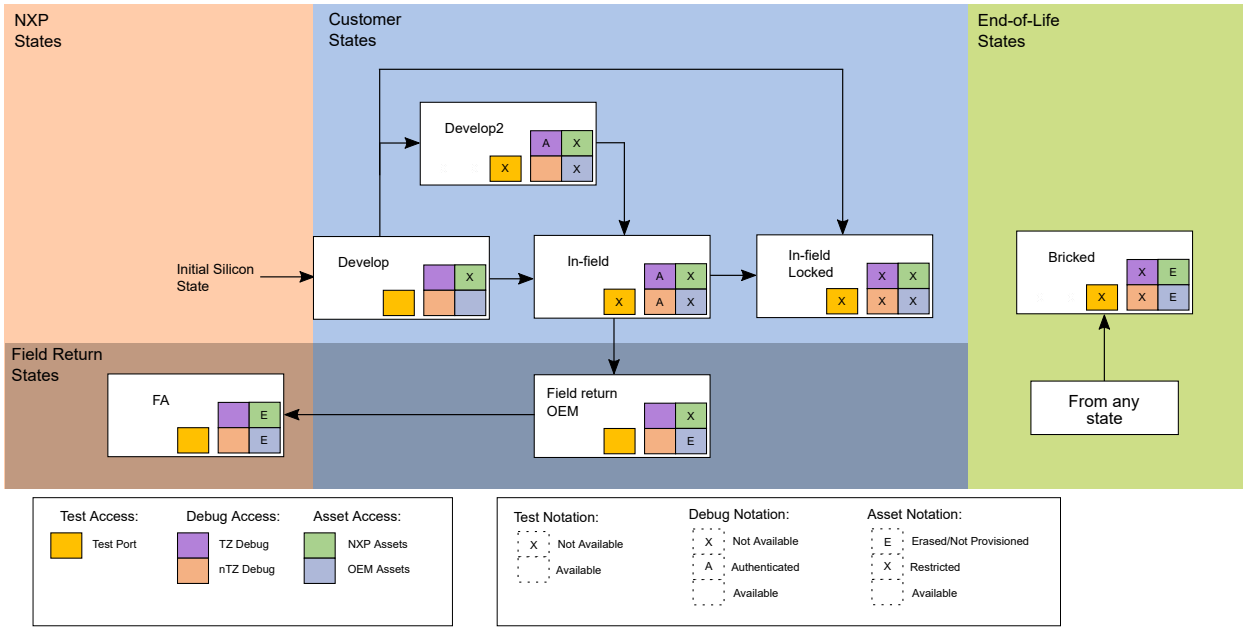


Figure 6. Security life cycle states

NOTE

While additional state transitions to advance the life cycle state are possible, the transitions shown in this diagram and in the following sections are the only ones that are recommended and supported.

NOTE

The Cortex-M33 with TrustZone technology provides secure and non-secure processing domains within a single core. Debugging of these processing domains are isolated and denoted as "TZ debug" for the secure domain and "nTZ debug" for the non-secure domain in the diagram above and throughout this chapter.

5.1.3 Life cycle states

The table below lists the life cycle states and shows the debug and test port availability and asset access for each state. Refer the attached IFR and Fusemap for more information.

Table 11. Life Cycle States

Life Cycle	Life Cycle Type	LC_STATE[7:0] fuses	Test Port	TZ Debug Port	nTZ Debug Port	ISP Cmds	NXP Assets	OEM Assets	Description
Develop	Customer	0000_0011	Yes	Yes	Yes	All	Restricted	Available	Initial customer development state after leaving NXP manufacturing.
Develop2	Customer	0000_0111	No	Authenticated	Yes	Limited	Restricted	Restricted	Optional customer development state. Used for development of NS world code.
In-field	Customer	0000_1111	No	Authenticated	Authenticated	Limited	Restricted	Restricted	In-field application state for end-customer use.
In-field Locked	Customer	1100_1111	No	No	No	Limited	Restricted	Restricted	Alternative in-field application state that prevents use of field return/failure analysis states. The rest of the behavior of the device is the same as the In-field state.
Field Return OEM	Field Return	0001_1111	Yes	Yes	Yes	None	Restricted	Erased	Field return state.
Failure analysis (FA)	Field Return	0111_1111	Yes	Yes	Yes	None	Erased	Erased	NXP field return state (CQC).
Bricked	End-of-Life	1111_1111	No	No	No	None	Erased	Erased	Bricked state to prevent device use.

5.1.4 Customer life cycle states

The following sections give descriptions of the customer life cycle states--Develop, Develop2, In-field, and In-field Locked. These states are intended for regular customer use, but some of the life cycle states are optional (for example, Develop2, In-field Locked, and Field Return OEM). The customer life cycle states are described in detail along with information on all the valid state transitions with the exception of transitions to the Bricked state which are covered in the End-of-Life Life Cycle States section.

5.1.4.1 Develop life cycle state (LC_STATE = 0x03)

The devices delivered from NXP are in the Develop life cycle state. In this state, all customer chip functionality is accessible. This mode is recommended for early software development.

In the Develop state:

- Test and debug ports are open.
- NMPA region is frozen and can't be changed. Any mismatch in NMPA CMAC or CRC32 is considered an invalid state and the part is locked.
- CMPA region is writable. Use CMPA region to test boot configuration settings before programming corresponding fields in fuses.

  NOTE

  If the value of a field in CMPA mismatches with the fuse field then the most restrictive option between the value is used. In most cases, the value in fuse supersedes the value in CMPA region.

- Secure boot is disabled by default. The SEC_BOOT_EN fuses or IFR determine if the secure boot flow is enabled or not.
- The Debug Mailbox mass erase/bulk erase command is only available while in the Develop life cycle state.

From the Develop life cycle, the device can be advanced to Develop2, In-field, In-field Locked, or Bricked states.

5.1.4.1.1 Transitioning from Develop to Develop2 life cycle state

Devices can be moved from Develop to the Develop2 life cycle state to allow for debug and development of non-secure world code (NS) while the secure world code and debug is protected.

Before transitioning from Develop to Develop2 the following features must be configured:

- Secure boot
  - Root of trust key hash - ROTKH[383:0]/ROTKH[255:0]
  - BOOT_CFG
  - SECURE_BOOT_CFG
- TrustZone
  - TrustZone configuration can be done as part of the secure application code.
  - TrustZone preset data can be included as part of the image manifest area.

Because the CMPA area is locked after exiting the Develop state, any optional features that are enabled or controlled by CMPA fields must be configured before transitioning from Develop to Develop2. This includes:

- Debug authentication settings
  - CC_SOCU_PIN
  - CC_SOCU_DFLT
  - VENDOR_USAGE
- PRINCE encryption/decryption for on-chip flash
  - NPX_CTXn_WD0

  - NPX_CTXn_WD1
  - NPX_LOCK_CTXn
- Inline Prince encryption/decryption for external FlexSPI NOR flash
  - IPEDn_START
  - IPEDn_END
- CMPA_CMAC_WORD0-CMPA_CMAC_WORD3 - Used by secure parts to check integrity of the CMPA area during cold and warm boots.

  The ROM writes the CMPA_CMAC value to the flash CMPA area when the CMAC_UPD[CMPA_UPD] field in the CFPA area is set to a non-zero value. The 0b011 value can be used to write the CMPA_CMAC to flash, and then advance to the Develop2 state (burns the LC_STATE fuses to 0b0000_0111). The 0b011 value can be used to write the CMPA_CMAC to flash and then advance to the Develop2 state (burns the LC_STATE fuses to 0b0000_0111). The 0b010 value just updates the CMPA_CMAC value in the flash CMPA area. It can be used to calculate the CMAC without transitioning the life cycle at the same time.

  NOTE

  The CMAC key used to compute the MAC is only accessible to the ROM, so the CMAC_UPD field in the CFPA must be used to obtain the CMAC value.
- CMPA_CRC - Used by secure parts to check integrity of the CMPA on deep power-down wakeup.

To transition from Develop to Develop2 state these fuses must be blown:

- SECBOOT_CFG - Secure boot options
- LC_STATE - If the CMPA_UPD value of 0b011 is used, then the LC_STATE fuses are changed to 0b0000_0111 automatically. If the CMPA_UPD value of 0b010 is used, then software must call the ROM's otp_program API to set the LC_STATE fuses to 0b0000_0111.

5.1.4.1.2 Transitioning from Develop to In-field cycle state

Devices can be moved from Develop to the In-field life cycle state when development is complete and product is ready to be deployed in the field.

Before transitioning from Develop to In-field the following features must be configured:

- Secure boot
  - Root of trust key hash - ROTKH[383:0]/ROTKH[255:0]
  - BOOT_CFG
  - SECURE_BOOT_CFG

Because the CMPA area is locked after exiting the Develop state, any optional features that are enabled or controlled by CMPA fields must be configured before transitioning from Develop to In-field. This includes:

- TrustZone
- Debug authentication settings
  - CC_SOCU_PIN
  - CC_SOCU_DFLT
  - VENDOR_USAGE
- PRINCE encryption/decryption for on-chip flash
  - NPX_CTXn_WD0
  - NPX_CTXn_WD1

- NPX_LOCK_CTXn

- Inline Prince encryption/decryption for external FlexSPI NOR flash
  - IPEDn_START
  - IPEDn_END

- CMPA_CMAC_WORD0-CMPA_CMAC_WORD3 - Used by secure parts to check integrity of the CMPA area during cold and warm boots. The ROM writes the CMPA_CMAC value to the flash CMPA area when the CMAC_UPD[CMPA_UPD] field in the CFPA area is set to a non-zero value. The 0b011 value can be used to write the CMPA_CMAC to flash and then advance to the Develop2 state (burns the LC_STATE fuses to 0b0000_0111). The 0b010 value just updates the CMPA_CMAC value in the flash CMPA area. It can be used to calculate the CMAC without transitioning the life cycle at the same time.

  NOTE

  The CMAC key used to compute the MAC is only accessible to the ROM, so the CMAC_UPD field in the CFPA must be used to obtain the CMAC value.

- CMPA_CRC - Used by secure parts to check integrity of the CMPA on deep power-down wakeup.

To transition from Develop to the In-field state these fuses must be blown:

- SECBOOT_CFG - Secure boot options
- LC_STATE - If the CMPA_UPD value of 0b101 is used, then the LC_STATE fuses are changed to 0b0000_1111 automatically. If the CMPA_UPD value of 0b010 is used, then software must call the ROM's otp_program API to set the LC_STATE fuses to 0b0000_1111.

5.1.4.1.3 Transitioning from Develop to In-field Locked cycle state

Devices can be moved from Develop to the In-field Locked life cycle state when development is complete and product is ready to be deployed in the field and field return functionality is not needed.

Before transitioning from Develop to In-field Locked the following features must be configured:

- Secure boot
  - Root of trust key hash - ROTKH[383:0]/ROTKH[255:0]
  - BOOT_CFG
  - SECURE_BOOT_CFG

Because the CMPA area is locked after exiting the Develop state, any optional features that are enabled or controlled by CMPA fields must be configured before transitioning from Develop to In-field Locked. This includes:

- TrustZone
- Debug authentication settings
  - CC_SOCU_PIN
  - CC_SOCU_DFLT
  - VENDOR_USAGE
- PRINCE encryption/decryption for on-chip flash
  - NPX_CTXn_WD0
  - NPX_CTXn_WD1
  - NPX_LOCK_CTXn
- Inline Prince encryption/decryption for external FlexSPI NOR flash
  - IPEDn_START
  - IPEDn_END

- CMPA_CMAC_WORD0-CMPA_CMAC_WORD3 - Used by secure parts to check integrity of the CMPA area during cold and warm boots. The ROM writes the CMPA_CMAC value to the flash CMPA area when the CMAC_UPD[CMPA_UPD] field in the CFPA area is set to a non-zero value. The 0b011 value can be used to write the CMPA_CMAC to flash and then advance to the Develop2 state (burns the LC_STATE fuses to 0b0000_0111). The 0b010 value just updates the CMPA_CMAC value in the flash CMPA area. It can be used to calculate the CMAC without transitioning the life cycle at the same time.

  NOTE

  The CMAC key used to compute the MAC is only accessible to the ROM, so the CMAC_UPD field in the CFPA must be used to obtain the CMAC value.

- CMPA_CRC - Used by secure parts to check integrity of the CMPA on deep power-down wakeup.

To transition from Develop to In-field Locked state these fuses must be blown:

- SECBOOT_CFG - Secure boot options
- LC_STATE - If the CMPA_UPD value of 0b110 is used, then the LC_STATE fuses are changed to 0b1100_1111 automatically. If the CMPA_UPD value of 0b010 is used, then software must call the ROM's otp_program API to set the LC_STATE fuses to 0b1100_1111.

5.1.4.2 Develop2 life cycle state (LC_STATE = 0x07)

Develop2 is an optional life cycle that can be used to allow for development of NS world code while providing protection for S-world code.

In the Develop2 state:

- The test ports and S-world debug port are closed. The S-world debug port can optionally be opened using the debug authentication mechanism (which must have been enabled while the device was in the Develop life cycle state).
- The NS-world debug port is open by default. If the debug authentication field (CC_SOCU_NS) is programmed, then it is used to determine debug access.
- The NMPA and CMPA regions are both frozen and can't be changed. Any mismatches in NMPA CMAC or CRC32 or the CMPA CMAC or CRC32 are considered an invalid state and the part will be locked.
- Secure boot flow is used.
- The CFPA fields are programmable by the ROM's ffr_infield_page_write API. This API might optionally be exposed by the application to the NS-world.
- The primary image is responsible for configuring the TZ-M settings. Use of the TrustZone preset data to configure the TZ settings is recommended.
- Limited ISP commands are allowed (GetProperty, Reset, SetProperty, ReceiveSbFile, and TrustProvisioning).

From the Develop2 life cycle, the device can be advanced to the In-field or Bricked states.

5.1.4.2.1 Transitioning from Develop2 to In-field cycle state

Devices can be moved from Develop2 to the In-field life cycle state when development is complete and product is ready to be deployed in the field.

Before transitioning from Develop to In-field the following optional features must be configured:

- NS-world debug authentication settings
  - CC_SOCU_PIN_NS
  - CC_SOCU_DFLT_NS
  - VENDOR_USAGE

To transition from Develop2 to the In-field state these fuses must be blown:

- LC_STATE - Software must call the ROM's otp_program API to set the LC_STATE fuses to 0b0000_1111.

5.1.4.3 In-field life cycle state (LC_STATE = 0x0F)

The In-field state is the primary state intended to be used for deployment of products to end-customers in the field.

In the In-field state:

- Test and debug ports are closed by default. If debug authentication is enabled, then the authentication process can be used to reopen debug ports.
- The NMPA and CMPA regions are both frozen and cannot be changed. Any mismatches in NMPA CMAC or CRC32 or the CMPA CMA or CRC32 are considered an invalid state and the part will be locked.
- The CFPA fields are programmable through the ROM's ffr_infield_page_write API.
- Secure boot flow is used.
- Use of TZ-M is optional.
- Limited ISP commands are allowed (GetProperty, Reset, SetProperty, ReceiveSbFile, and TrustProvisioning).

From the In-field life cycle, the device can be advanced to In-field Locked, Field Return OEM, or Bricked states.

5.1.4.3.1 Transitioning from In-field to In-field Locked cycle state

Devices can be moved from In-field to the In-field Locked life cycle state if it is determined that field return functionality is not needed.

To transition out of In-field mode these fuses must be blown:

- LC_STATE - Software must call the ROM's otp_program API to set the LC_STATE fuses to 0b1100_1111.

5.1.4.3.2 Transitioning from In-field to Field Return OEM cycle state

If a device deployed to the field is returned for a reported failure, then the device can be moved from In-field to the Field Return OEM state to disable normal device operation and re-enable testing of the device.

To transition from In-field to Field Return OEM NXP recommends:

1. Use debug authentication to enable permission to send the Set FA mode command through the debug mailbox.
2. Send Set FA mode debug mailbox command. The Set FA mode command calls a ROM handler that:
    a. Erases the entire user flash area
    b. Erases the CFPA, CMPA, and key store areas (this is done on every subsequent boot while in the Field Return OEM or Failure Analysis states)
    c. Sets the LC_STATE fuses to 0b0001_1111
    d. Resets the device

5.1.4.4 In-field Locked life cycle state (LC_STATE = 0xCF)

The In-field Locked state is an optional state intended to be used for deployment of products to end-customers in the field for products that do not support field return or failure analysis. Most of the behavior in this mode is the same as the In-field state, but the debug and test ports can never be fully opened again.

In the In-field Locked state:

- Test and debug ports are closed. The debug authentication mechanism cannot be used to re-enable the debug port.
- The NMPA and CMPA regions are both frozen and can't be changed. Any mismatches in NMPA CMAC or CRC32 or the CMPA CMAC or CRC32 are considered an invalid state and the part will be locked.
- The CFPA fields are programmable through the ROM's ffr_infield_page_write API.
- Secure boot flow is used.

- Use of TZ-M is optional.
- Limited ISP commands are allowed (GetProperty, Reset, SetProperty, ReceiveSbFile, and TrustProvisioning).

From the In-field life cycle, the device can only be advanced to the Bricked state.

5.1.5 Field Return states

The following sections describe the field return life cycle states--Field Return OEM and Failure Analysis (FA). These states are used to support various stages of field return debugging.

- Stage 1: Initial investigation and failure duplication phase where the device remains in the In-Field state. The debug authentication mechanism can be used to re-open the debug port(s) and gain additional information on the reported failure.
- Stage 2: If the issue needs further control of the device for investigation beyond what the debug authentication process allows, then the device can be moved to the Field Return OEM life cycle state for additional debugging and testing.
- Stage 3: If a silicon structural or manufacturing issue is suspected after stage one and stage two investigations, then the device can be moved to the Failure Analysis (FA) state before returning the device to NXP for additional testing.

5.1.5.1 Field Return OEM life cycle state (LC_STATE = 0x1F)

The Field Return OEM state can be used to debug products returned by end customers.

In the Field Return OEM state:

- On every boot ROM verifies that the CMPA area (0x0100_4000 - 0x0100_5FFF) and CFPA area (0x0100_0000 - 0x0100_3FFF) are blank. If not, ROM erases the CMPA and CFPA areas before opening debug access. The PUF and ELS modules are put in FA mode which will re-key all application usage keys including memory encryption Prince keys. This mechanism protects leakage of any residue data left during life-cycle state transition.
- Test and debug ports are open.
- ROM stays in a while(1) loop and does not pass execution to application code. The debug port can be used to load and then execute diagnostic firmware.

From the Field Return OEM life cycle, the device can be advanced to Failure Analysis (FA) or Bricked states.

5.1.5.1.1 Transitioning from Field Return OEM to FA life cycle state

Devices can be moved from Field Return OEM to the FA life cycle state if the device is being returned to NXP for testing and failure analysis.

To transition from Field Return OEM to FA:

1. Call the ROM's otp_program API to set the LC_STATE fuses to 0b0111_1111.
2. Erase System SRAM.
3. Erase any other sensitive information.

5.1.5.2 Failure Analysis (FA) life cycle state (LC_STATE = 0x7F)

The FA state is used for product that is being returned to NXP for testing. In this mode on every boot, the ROM:

- Verifies that main (user) flash area is blank, if not it erases this area
- On every boot, ROM checks and erases user flash, CMPA and CFPA.

Ensures that the PUF key store is empty and also block PUF key unwrapping before enabling the debug ports (note: the debug ports are no longer authenticated as OEM assets are erased). In this mode, the ROM stays in a while(1) loop and does not pass execution to the application code. The debug or test port must be used to load and then execute diagnostic firmware.

From the FA life cycle, the device can only be advanced to the Bricked state.

5.1.6 End-of-Life life cycle states

The following sections describe the End-of-Life security life cycle state--Bricked. This state can be used by customers or NXP to remove a chip from regular use and erase/block access to secrets inside the chip.

5.1.6.1 Bricked life cycle state (LC_STATE = 0xFF)

The Bricked state is the end-of-life state for the device to be used when permanently removing a part from service. In this mode, the ROM locks up the device immediately on any reset. The debug and test mode ports are also disabled.

Bricked is the final life cycle state, so the device cannot be advanced to any other states from Bricked.

5.1.6.1.1 Transitioning from any state to Bricked life cycle state

Devices can be moved from any other life cycle state to the Bricked life cycle state to permanently prevent any use of the device.

To transition to the Bricked state:

- Erase the flash - This step is recommended to ensure that sensitive information is removed from the device.
- LC_STATE - Software must call the ROM's otp_program API to set the LC_STATE fuses to 0b1111_1111.

Chapter 6
ROM API

6.1 Overview

This section provides an introduction to the ROM API in this device.

NOTE

This chapter is also available in the MCX Nx4x Reference Manual, with differences related to security functionality. The chapter in the Reference Manual does not show the security relevant information in it.

6.1.1 Features

- The ROM API supports programming both the Programmable FLASH region (store application code and data) and the CFPA and CMPA Regions (store device configurations, boot configuration, in-field programmable data).
- The ROM API enables the Serial NOR FLASH programming via the FlexSPI Flash Driver API.
- The ROM API supports OTP-eFuse read and programming.
- The ROM API provides native support for Nboot API in the user application.
- The ROM API supports In-application-programming functionality by the IAP API, via the unified memory interface or the Sbloader API.

6.2 Functional description

This section introduces the ROM API structure first, then describes the FLASH API, OTP API, FLEXSPI NOR API, IAP API and NBOOT APIs. This section also demonstrates the typical usage of each API.

- The primary purpose of the FLASH API is to update the programmable FLASH area and specify the parameters in the CMPA and the CFPA.
- The FLEXSPI FLASH API eases the support of various Serial NOR devices in the market.
- OTP API provides the functionality to advance lifecycle; program/read critical one-time programmable parameters and some general-purpose one-time programmable FUSE field.
- NBOOT APIs generate random numbers and authenticate the application image. NBOOT SB3.1 API functions can also verify signature of SB3.1 header (manifest) and decrypt individual data blocks without needing to process the entire SB file, like SBloader APIs need.
- IAP API enables the flexibility of doing in-application-programming by the memory interface or the dedicated SBloader API.

NOTE

All ROM APIs are callable from secure world only. Therefore, all function pointers use secure address aliases. Applications should switch to secure mode and then call ROM API or create NSC veneers for ROM APIs.

6.2.1 ROM API structure

The ROM API table locates at address 0x1303fc00. See Figure 7.

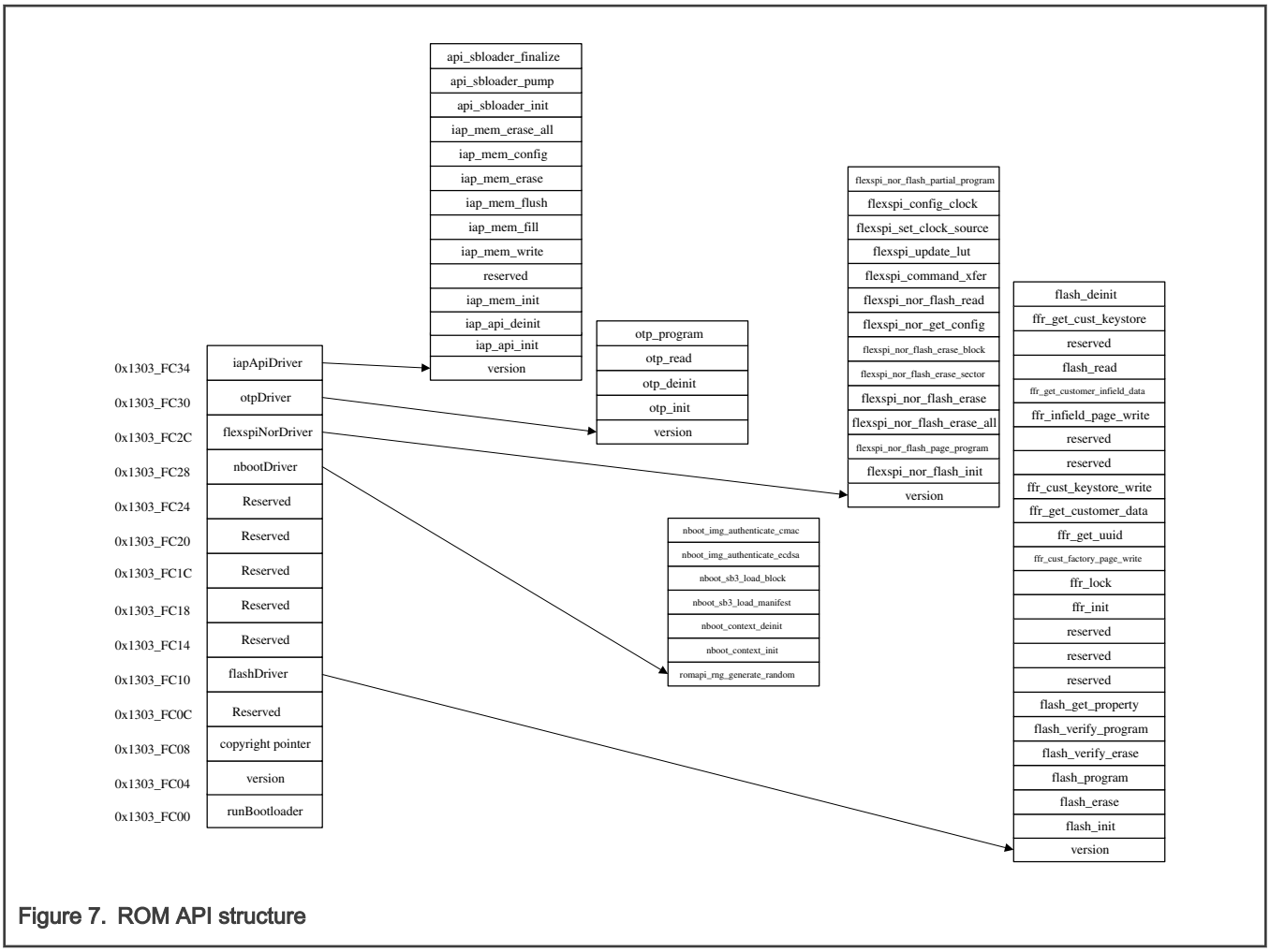


Figure 7. ROM API structure

6.2.2 FLASH APIs

The FLASH API set enables the following features:

- Initialize FLASH controller
- Erase and verify the specified FLASH area
- Program and verify the specified FLASH page
- Retrieve FLASH properties
- Initialize or Lock the CMPA and CFPA Regions
- Program and Read CMPA
- Program and Read CFPA

The FLASH APIs are organized in the FLASH Driver API Interface structure. See the API layout and API prototypes from the following data structure. The whole FLASH API wrapper is available in the SDK release package.

The bootloader API prototypes are:

```
typedef struct FLASHDriverInterface
{
    standard_version_t version; /* flash driver API version number */
```

```
    /* FLASH driver */
    status_t (*flash_init)(flash_config_t *config);
    status_t (*flash_erase)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes, uint32_t
key);
    status_t (*flash_program)(flash_config_t *config, uint32_t start, uint8_t *src, uint32_t
lengthInBytes);
    status_t (*flash_verify_erase)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes);
    status_t (*flash_verify_program)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes,
const uint8_t *expectedData, uint32_t *failedAddress, uint32_t *failedData);
    status_t (*flash_get_property)(flash_config_t *config, flash_property_tag_t whichProperty,
uint32_t *value);
    uint32_t reserved0[3];

    /* FLASH FFR driver */
    status_t (*ffr_init)(flash_config_t *config);
    status_t (*ffr_lock)(flash_config_t *config);
    status_t (*ffr_cust_factory_page_write)(flash_config_t *config, uint8_t *page_data, bool
seal_part);
    status_t (*ffr_get_uuid)(flash_config_t *config, uint8_t *uuid);
    status_t (*ffr_get_customer_data)(flash_config_t *config, uint8_t *pData, uint32_t offset,
uint32_t len);
    status_t (*ffr_cust_keystore_write)(flash_config_t *config, ffr_key_store_t *pKeyStore);
    status_t reserved1;
    status_t reserved2;
    status_t (*ffr_infield_page_write)(flash_config_t *config, uint8_t *page_data, uint32_t
valid_len);
    status_t (*ffr_get_customer_infield_data)(flash_config_t *config, uint8_t *pData, uint32_t
offset, uint32_t len);
    status_t (*flash_read)(flash_config_t *config, uint32_t start, uint8_t *dest, uint32_t
lengthInBytes);
    status_t reserved3;
    status_t (*flash_get_cust_keystore)(flash_config_t *config, uint8_t *pData, uint32_t offset,
uint32_t len);
    status_t (*flash_deinit)(flash_config_t *config);
}
flash_driver_interface_t;
```

Each FLASH API depends on a common FLASH context structure named *flash_config_t* to perform the proper FLASH operation. See the structure details below.

```

/*! @brief FLASH driver state information.
*
* An instance of this structure is allocated by the user of the flash driver and
* passed into each of the driver APIs.
*/
typedef struct
{
    uint32_t PFlashBlockBase;  /*!< A base address of the first PFlash block */
    uint32_t PFlashTotalSize;  /*!< The size of the combined PFlash block. */
    uint32_t PFlashBlockCount; /*!< A number of PFlash blocks. */
    uint32_t PFlashPageSize;   /*!< The size in bytes of a page of PFlash. */
    uint32_t PFlashSectorSize; /*!< The size in bytes of a sector of PFlash. */
    flash_ffr_config_t ffrConfig;
    uint32_t reserved0[5];
    uint32_t *nbootCtx;
    bool useAhbRead;
}
flash_config_t;
```

The flash_ffr_config_t is defined as below:

```
/*! @brief FLASH controller paramter config. */

typedef struct
{
    uint32_t ffrBlockBase;
    uint32_t ffrTotalSize;
    uint32_t ffrPageSize;
    uint32_t sectorSize;
    uint32_t cfpaPageVersion;
    uint32_t cfpaPageOffset;
}
flash_ffr_config_t;
```

flash_ffr_config_t is managed by the FFR_Init API call, the user application doesn't need to touch it.

The ffr_key_store_t and flash_ffr_config_t definitions are:

```
#define FLASH_FFR_KETBLOB_SIZE (0x30u)
typedef struct
{
    uint8_t reserved[1][FLASH_FFR_KETBLOB_SIZE];
}
ffr_key_store_t;
typedef enum
{
    kFLASH_PropertyPflashSectorSize = 0x00U, /*!< Pflash sector size property.*/
    kFLASH_PropertyPflashTotalSize = 0x01U, /*!< Pflash total size property.*/
    kFLASH_PropertyPflashBlockSize = 0x02U, /*!< Pflash block size property.*/
    kFLASH_PropertyPflashBlockCount = 0x03U, /*!< Pflash block count property.*/
    kFLASH_PropertyPflashBlockBaseAddr = 0x04U, /*!< Pflash block base address property.*/
    kFLASH_PropertyPflashPageSize = 0x30U, /*!< Pflash page size property.*/
    kFLASH_PropertyPflashSystemFreq = 0x31U, /*!< System Frequency property.*/
    kFLASH_PropertyFfrSectorSize = 0x40U, /*!< FFR sector size property.*/
    kFLASH_PropertyFfrTotalSize = 0x41U, /*!< FFR total size property.*/
    kFLASH_PropertyFfrBlockBaseAddr = 0x42U, /*!< FFR block base address property.*/
    kFLASH_PropertyFfrPageSize = 0x43U, /*!< FFR page size property.*/
}
flash_property_tag_t;
```

6.2.2.1 Version

The "version" field in the FLASH API table indicates the current FLASH API version in the ROM bootloader.

Prototype :

```
standard_version_t version;
```

Table 12. Definition of the version field

Parameter	Description
standard_version_t	Pointer to version structure to store current FLASH driver version information uint8_t bugfix; /* bugfix version [7:0] */ uint8_t minor; /* minor version [15:8] */ uint8_t major; /* major version [23:16] */ char name; /* character name is "F", where "F" stands for Flash and character version cannot be changed, it is fixed */

Example:

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define FLASH_API_TREE ((flash_driver_interface_t*)ROM_API_TREE[4])
uint32_t FlashDriverVersion = FLASH_API_TREE-> version;
```

In this SoC, the FLASH Driver version is "0x46010100", it means the FLASH driver version is 1.1.0.

6.2.2.2 flash_init

This API initializes the FLASH default parameters and related FLASH clock for the FLASH and FMC. The *flash_init* API should be called before all the other FLASH APIs.

Prototype :

```
status_t (*flash_init)(flash_config_t *config);
```

Table 13. Parameters used for flash_init API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.

Example :

```
flash_config_t flashConfig;
uint32_t status = FLASH_API_TREE->flash_init (&flashConfig);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means flash has been inited successfully.

6.2.2.3 flash_erase

This API erases the internal FLASH region specified by the parameters. This API must be called after the flash_init.

Prototype :

```
status_t (*flash_erase)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes, uint32_t key);
```

Table 14. Parameters used in flash_erase API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
start	The start address of the required flash memory to be erased. The start address must be sector size aligned (that is, a multiple of 8 KB).
lengthInBytes	The length, given in bytes (not words or long words) to be erased. Must be sector size aligned.
key	Key is used to validate erase operation. Must be set to “kFLASH_ApiEraseKey”. kFLASH_ApiEraseKey = (((‘k’) << 24) | ((‘e’) << 16) | ((‘f’) << 8) | ((‘l’)))

Example :

```
uint32_t start = 0x0 ;
uint32_t lengthInBytes = 0x2000 ;
#define ERASE_KEY 0x6b65666b
uint32_t status = FLASH_API_TREE->flash_erase (&flashConfig, start, lengthInBytes, ERASE_KEY);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the FLASH region 0x0 – 0x2000 has been erased.

6.2.2.4 flash_program

This API programs the data into the internal flash page specified by input parameters. The required "start" and "lengthInBytes" parameters must be page-size aligned. This API must be called after the flash_init. Flash must be erased before programming.

Prototype :

```
status_t (*flash_program)(flash_config_t *config, uint32_t start, uint8_t *src, uint32_t
lengthInBytes);
```

Table 15. Parameters used in flash_program API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
start	The start address of the required flash memory to be programmed. The start address must be 128 bytes-aligned.
src	Pointer to the source buffer of data that is to be programmed into flash.
lengthInBytes	The length in bytes (not words or long words) to be programmed; the length must also be 128 bytes-aligned.

Example:

```
uint8_t programBuffer[512];
for (uint32_t i=0; i<sizeof(programBuffer); i++)
{
```

```
    programBuffer[i] = (uint8_t)(i & 0xFF);
}status = FLASH_API_TREE->flash_program(&flashConfig, 0x0, programBuffer, sizeof(programBuffer);)
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the specified data has been programmed into the FLASH region.

6.2.2.5 flash_verify_erase

This API checks whether the specified FLASH area is in the erasure state.

This API must be called after the flash_init.

Prototype :

```
status_t (*flash_verify_erase)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes);
```

Table 16. Parameters used in flash_verify_erase API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
start	The start address of the required flash memory to be verified. Must be page-aligned.
lengthInBytes	The length, given in bytes (not words or long words) to be verified. Must be page-aligned.

Example:

```
uint32_t start = 0x0 ;
uint32_t lengthInBytes = 0x1000 ;
uint32_t status = FLASH_API_TREE->flash_verify_erase (&flashConfig, start, lengthInBytes);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the internal FLASH region 0x0 – 0x1000 is erased.

6.2.2.6 flash_verify_program

This API checks whether the data in the specified area is identical to the data supposed to be programmed. This API must be called after the flash_init.

Prototype :

```
status_t (*flash_verify_program)(flash_config_t *config,
uint32_t start,
uint32_t lengthInBytes,
const uint8_t *expectedData,
uint32_t *failedAddress,
uint32_t *failedData);
```

Table 17. Parameters used in flash_verify_program API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
start	The start address of the required flash memory to be verified.
lengthInBytes	The length, given in bytes (not words or long words) to be verified.
expectedData	Pointer to the expected data that is to be verified against.
failedAddress	Pointer to returned first failing address.
failedData	Pointer to return failing data.

Example:

```
uint32_t start = 0x0 ;
uint32_t lengthInBytes = sizeof(expected_buffer) ;
uint32_t failedAddress, failedData;
uint32_t status = FLASH_API_TREE->flash_verify_erase (&flashConfig, start, lengthInBytes, (const
uint8_t *) expected_buffer, & failedAddress, & failedData);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the expected data same as the programed data in the specific FLASH region.

6.2.2.7 flash_get_property

This API returns the requested FLASH property. This API must be called after the flash_init and ffr_init.

Prototype :

```
status_t (*flash_get_property)(flash_config_t *config, flash_property_tag_t whichProperty, uint32_t
*value);
```

Table 18. Parameters used in flash_get_property API

Parameter	whichProperty	Description
config		Pointer to flash_config_t data structure in memory to store driver runtime state.
whichProperty		Pointer to the which flash property need to get.
value		Pointer to the returned flash property value.
Property definition		
kFLASH_PropertyPflashTotalSize	0x01	Pflash total size property.
kFLASH_PropertyPflashBlockSize	0x02	Pflash Block size property.

Table continues on the next page...

Table 18. Parameters used in flash_get_property API (continued)

Parameter	whichProperty	Description
kFLASH_PropertyPflashBlockCount	0x03	Pflash Block Count size property.
kFLASH_PropertyPflashBlockBaseAddr	0x04	flash block base address.
kFLASH_PropertyPflashPageSize	0x30	Pflash page size property.
kFLASH_PropertyFfrTotalSize	0x41	FFR total size property.
kFLASH_PropertyFfrBlockBaseAddr	0x42	FFR block base address property.
kFLASH_PropertyFfrPageSize	0x43	FFR page size property.

Example:

```
flash_property_tag_t whichProperty = kFLASH_PropertyPflashTotalSize;
uint32_t value = 0;
uint32_t status = FLASH_API_TREE->flash_get_property(&flashConfig, whichProperty, &value);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the flash property get successfully, and will return the property value if needed.

6.2.2.8 ffr_init

This API is used for initializing the FFR controller and the *flash_ffr_config* context. It must be called before calling other FFR APIs.

NOTE

flash_init must be called before calling the ffr_init API. Before calling other flash_driver_interface_t APIs, especially flash_verify_erase and flash_verify_program, it is necessary to call the flash_init and ffr_init APIs.

Prototype :

```
status_t (*ffr_init)(flash_config_t *config);
```

Table 19. Parameters used in ffr_init API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.

Example:

```
uint32_t status = FLASH_API_TREE-> ffr_init (&flashConfig);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the FFR parameters has been configured.

6.2.2.9 ffr_lock

This API is used to enable the firewall for all FLASH banks, including enabling FLASH protection for three IFR banks and disabling write access to the FLASHBENKENABLE register. ffr_lock unconditionally locks writing to the CFPA and CMPA flash areas. Subsequent writes are inhibited unless a power-on reset (POR) or brown-out detect (BOD) reset occurs.

Prototype :

```
status_t (*ffr_lock)(flash_config_t *config);
```

Table 20. Parameters used in ffr_lock

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.

Example:

```
uint32_t status = FLASH_API_TREE-> ffr_lock (&flashConfig);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the FFR regions has been locked.

6.2.2.10 ffr_cust_factory_page_write

The API is used for writing the CMPA data into the CMPA region, and the API should be called after the flash_init and ffr_init.

Prototype :

```
status_t (*ffr_cust_factory_page_write)(flash_config_t *config, uint8_t *page_data, bool seal_part);
```

Table 21. Parameters used in ffr_cust_factory_page_write API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
page_data	Pointer to value address that will be written to the destination address.
seal_part	Reserved – This parameter is not used on this device. It should always be set to zero (0b'0).”

Example:

```
uint32_t cmpa_buffer [ffr_page_size] = {0, 2, 3, 4};
uint32_t status = FLASH_API_TREE-> ffr_cust_factory_page_write (&flashConfig, (uint8_t *)cmpa_buffer,
false);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the cmpa_buffer data has been programed into the CMPA region.

6.2.2.11 ffr_get_uuid

The API is used for getting the UUID data of the device, and the API should be called after the flash_init and ffr_init.

Prototype :

```
status_t (*ffr_get_uuid)(flash_config_t *config, uint8_t *uuid);
```

Table 22. Parameters used in ffr_get_uuid API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
uuid	Pointer to value address, the value is read back from the Bank0_IFR1 0x0110_0000 – 0x0110_0010.

Example:

```
uint32_t uuid_buffer [4];
uint32_t status = FLASH_API_TREE-> ffr_get_uuid(&flashConfig, (uint8_t *)uuid_buffer);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the uuid data has been got from the UUID of the device.

6.2.2.12 ffr_get_customer_data

This API is used to read data stored in CMPA, and the API should be called after the flash_init and ffr_init.

Prototype :

```
status_t (*ffr_get_customer_data)(flash_config_t *config, uint8_t *pData, uint32_t offset, uint32_t
len);
```

Table 23. Parameters used in ffr_get_customer_data API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
pData	Point to the destination buffer that stores data read from CMPA.
offset	Point to the offset value based on the CMPA start address(0x0100_4000) of the device.
len	The length in bytes to be read back. The offset + len <= 512B.

Example:

```
uint32_t cmpa_buffer[4];
uint32_t offset = 0;
uint32_t status = FLASH_API_TREE->ffr_get_customer_data(&flashConfig, (uint8_t *)cmpa_buffer, offset,
sizeof(cmpa_buffer));
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the CMPA data has been got from the CMPA region and stored into the cmpa_buffer.

6.2.2.13 ffr_cust_keystore_write

The API is used for programing the customer key store data into the customer key store region (0x0100_4160 - 0x0100_418f), and the API should be called after the flash_init and ffr_init.

Prototype :

```
status_t (*ffr_cust_keystore_write)(flash_config_t *config, ffr_key_store_t *pKeyStore);
```

Table 24. . Parameters used in ffr_cust_keystore_write API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
pKeyStore	Pointer to the customer key store data buffer, which will be programed into the customer key store region.

Example:

```
uint32_t status = FLASH_API_TREE->ffr_cust_keystore_write(&flashConfig, (ffr_key_store_t *)
cust_keystore_buffer);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means customer key store data has been programed into the customer key store region.

6.2.2.14 ffr_infield_page_write

The API is used for programing the CFPA data into the CFPA inactive page and the API should be called after the flash_init and ffr_init.

Prototype :

```
status_t (*ffr_infield_page_write)(flash_config_t *config, uint8_t *page_data, uint32_t valid_len);
```

Table 25. . Parameters used in ffr_infield_page_write API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
page_data	Pointer to the source buffer of data that is to be programed into the in-field page.
valid_len	The length in bytes to be programmed, the length must equal the page size.

Example:

```
uint32_t cfpa_buffer [128];
uint32_t status = FLASH_API_TREE-> ffr_infield_page_write (&flashConfig, (uint8_t *) cfpa_buffer,
sizeof(cfpa_buffer));
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the CFPA data has been programed into the CFPA Scratch region.

6.2.2.15 ffr_get_customer_infield_data

The API is used for getting the CFPA data from the FFR CFPA region, and the API should be called after the flash_init and ffr_init.

Prototype :

```
status_t (*ffr_get_customer_infield_data)(flash_config_t *config, uint8_t *pData, uint32_t offset,
uint32_t len);
```

Table 26. . Parameters used in ffr_get_customer_infield_data API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
pData	Point to the destination buffer of data that stores data read from customer In-field page.
offset	Pointer to the offset value based on the CFPA address (Ping page: 0x01000000, Pong page: 0x01002000) of the device.
len	Point to the length of data to read, and the offset + len <= 512B.

Example:

```
uint32_t cfpa_buffer[4];
uint32_t offset = 0;
uint32_t status = FLASH_API_TREE->ffr_get_customer_infield_data (&flashConfig, (uint8_t *)
cfpa_buffer, offset, sizeof(cfpa_buffer ));
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the CFPA data has been got from the CFPA region and stored into the cfpa_buffer.

6.2.2.16 flash_read

The API is used for getting internal flash data, including the FLASH and FFR data, and the API should be called after the flash_init.

Prototype :

```
status_t (*flash_read)(flash_config_t *config, uint32_t start, uint8_t *dest, uint32_t lengthInBytes);
```

Table 27. Parameters used in flash_read API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
start	Point to the start address where data will be read.
dest	Pointer to the buffer used for storing the read data.
lengthInBytes	Point to the read data length

Example:

```
uint32_t start_addr = 0x1000;
uint8_t read_buffer[512] = {0};
uint32_t length = 512;
uint32_t status = FLASH_API_TREE-> flash_read(&flashConfig, start_addr, (uint8_t *) read_buffer,
length);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the expected read region has been loaded into the read_buffer.

6.2.2.17 flash_get_cust_keystore

The API is used for getting the customer key store data from the customer key store region (0x0100_4160 - 0x0100_418f), and the API should be called after the flash_init and ffr_init.

Prototype:

```
status_t (*flash_get_cust_keystore)(flash_config_t *config, uint8_t *pData, uint32_t offset, uint32_t
len);
```

Table 28. Parameters used in flash_get_cust_keystore API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.
pData	Pointer to the customer key store data buffer, which got from the customer key store region.
offset	Point to the offset value based on the customer key store address (0x01004160)of the device.
len	Point to the length of the expected get customer key store data, and the offset + len <= 512B.

Example:

```
uint8_t cust_keystore_buffer[512] = {0};
uint32_t offset = 0;
uint32_t length = 512;
uint32_t status = FLASH_API_TREE-> flash_get_cust_keystore (&flashConfig, (uint8_t *)
cust_keystore_buffer, offset, length);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means the customer key store data has been got from the customer key store region and stored into the cust_keystore_buffer.

6.2.2.18 flash_deinit

This API deinitializes the FLASH default parameters and related FLASH clock for the FLASH and FMC. The flash_deinit API should be called after all the other FLASH APIs.

Prototype :

```
status_t (*flash_deinit)(flash_config_t *config);
```

Table 29. Parameters used in flash_deinit API

Parameter	Description
config	Pointer to flash_config_t data structure in memory to store driver runtime state.

Example:

```
flash_config_t flashConfig;
uint32_t status = FLASH_API_TREE->flash_deinit (&flashConfig);
```

See the possible status codes in Table 30. If the status is kStatus_FLASH_Success return value, it means FLASH has been deinited successfully.

6.2.2.19 Possible return codes of the FLASH APIs

Table 30. Return codes for FLASH APIs

Return code	Value	Description
kStatus_FLASH_Success	0	API is executed successfully
kStatus_FLASH_InvalidArgument	4	An invalid argument is provided
kStatus_FlashAlignmentError	101	Address or length does not meet the required alignment.
kStatus_FlashAddressError	102	Address or length is outside addressable memory.
kStatus_FLASH_CommandFailure	105	The FMU_FSTAT[FAIL] bit is set.
kStatus_FlashUnknownProperty	106	Unknown Flash property.
kStatus_FlashEraseKeyError	107	The key provided does not match the programmed flash key.
kStatus_FlashRegionExecuteOnly	108	The area of flash is protected as execute-only.
kStatus_FLASH_CommandNotSupported	111	Flash API is not supported
kStatus_FLASH_ReadOnlyProperty	112	The flash property is read-only
kStatus_FLASH_InvalidPropertyValue	113	The flash property value is out of range
kStatus_FLASH_EccError	116	A correctable or uncorrectable error during command execution
kStatus_FLASH_CompareError	117	Destination and source memory contents do not match
kStatus_FLASH_InvalidWaitStateCycles	119	The wait state cycle set to r/w mode is invalid

Table continues on the next page...

Table 30. Return codes for FLASH APIs (continued)

kStatus_FLASH_OutOfDateCfpaPage	132	CFPA page version is out of date
kStatus_FLASH_BlankIfrPageData	133	Blank page cannot be read
kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce	134	Encrypted flash subregions are not erased at once
kStatus_FLASH_SealedFfrRegion	137	The FFR region is sealed
kStatus_FLASH_FfrRegionWriteBroken	138	The FFR Spec region is not allowed to be written discontinuously
kStatus_FLASH_NmpaAccessNotAllowed	139	The NMPA region is not allowed to be read/ written/erased
kStatus_FLASH_CmpaCfgDirectEraseNotAllowed	140	The CMPA Cfg region is not allowed to be erased directly
kStatus_FLASH_FfrBankIsLocked	141	The FFR bank region is locked
kStatus_FLASH_CfpaScratchPageInvalid	148	CFPA Scratch Page is invalid
kStatus_FLASH_CfpaVersionRollbackDisallowed	149	CFPA version rollback is not allowed

6.2.2.20 Flash APIs Example

The section provides examples to introduce the Flash APIs usage.

Example 1 (program the image data or specific data into internal flash region):

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define FLASH_API_TREE ((flash_driver_interface_t*)ROM_API_TREE[4])
flash_config_t flashConfig;
status_t status = kStatus_Success;
uint32_t load_address = 0x10000;
uint32_t erase_size = 0x800;
uint32_t s_buffer[512];
for (uint32_t i = 0; i < 512; i++)
{
    s_buffer[i] = i;
}
uint32_t FlashDriverVersion = FLASH_API_TREE->version;
debug_printf("Flash version= %x\r\n", FlashDriverVersion);
status = FLASH_API_TREE->flash_init(&flashConfig);
debug_printf("flash_init version= %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
status = FLASH_API_TREE->ffr_init(&flashConfig);
debug_printf("ffr_init version= %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
```

```
status = FLASH_API_TREE->flash_erase(&FlashConfig, load_address, erase_size, kFLASH_ApiEraseKey);
debug_printf("flash_erase status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
/* call the flash_verify_erase api to check whether the flash region has been erased */
status = FLASH_API_TREE->flash_verify_erase(&FlashConfig, load_address, erase_size);
debug_printf("flash_verify_erase status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
/* here s_buffer data can also be replaced by the image data that is used for dual image boot */
status = FLASH_API_TREE->flash_program(&FlashConfig, load_address, (uint8_t *)s_buffer,
sizeof(s_buffer);
debug_printf("flash_program status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
/* call the flash_verify_program api to check whether the s_buffer data has been loaded into the
flash region */
uint32_t failedAddress, failedData;
status = FLASH_API_TREE->flash_verify_program (&FlashConfig, load_address, sizeof(s_buffer), (const
uint8_t *)s_buffer, &failedAddress, &failedData);
debug_printf("flash_verify_program status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
```

Example 2 (update the ffr data and get the data from the ffr region):

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define FLASH_API_TREE ((flash_driver_interface_t*)ROM_API_TREE[4])
flash_config_t flashConfig;
uint32_t pflashBlockBase = 0;
uint32_t cmpa_key_store_address = 0x0100_4000;
uint8_t pData[512];
uint8_t pData1[512] ;
uint8_t uuid[16];
uint32_t ffr_buffer_len = 512 ;
status_t status = kStatus_Success;
status = FLASH_API_TREE->flash_init(&flashConfig);
debug_printf("flash_init version= %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
status = FLASH_API_TREE->ffr_init(&flashConfig);
debug_printf("ffr_init version= %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
status = FLASH_API_TREE->flash_get_property(&FlashConfig, kFLASH_PropertyPflashBlockBaseAddr,
&pflashBlockBase);
```

```
debug_printf("kFLASH_PropertyPflashBlockBaseAddr = %x\r\n", pflashBlockBase);
if (status != kStatus_FLASH_Success)
{
    return status;
}
/* call ffr_infield_page_write to load CfpaData into the CFPA inactive region; CfpaData is provided
by user; the version of CfpaData should be larger than or same as before */
status = FLASH_API_TREE->ffr_infield_page_write(&FlashConfig, (uint8_t *)&CfpaData, FFR_BUFFER_LEN);
debug_printf("ffr_infield_page_write status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
status = FLASH_API_TREE->ffr_get_customer_infield_data(&FlashConfig, (uint8_t *)pData, 0,
ffr_buffer_len);
debug_printf("ffr_get_customer_infield_data status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
/* call ffr_cust_factory_page_write to load CmpaData into the CMPA region; CMpaData is provided by
user */
status = FLASH_API_TREE->ffr_cust_factory_page_write(&FlashConfig, (uint8_t *)CmpaData, false);
debug_printf("ffr_cust_factory_page_write status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
status = FLASH_API_TREE->ffr_get_customer_data(&FlashConfig, (uint8_t *)pData, 0, ffr_buffer_len);
debug_printf("ffr_get_customer_data status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
status = FLASH_API_TREE->ffr_get_uuid(&FlashConfig, uuid);
debug_printf("ffr_get_uuid status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
/* call ffr_keystore_write to load CMPA_KeyStore into the CMPA key_store region; CMPA_KeyStore is
provided by user */
status = FLASH_API_TREE->ffr_keystore_write(&FlashConfig, (ffr_key_store_t *)CMPA_KeyStore);
debug_printf("ffr_keystore_write status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
status = FLASH_API_TREE->flash_get_cust_keystore(&FlashConfig, (uint8_t *)pData, 0, ffr_buffer_len);
debug_printf("flash_get_cust_keystore status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
status = FLASH_API_TREE->flash_read(&FlashConfig, cmpa_key_store_address, (uint8_t *)pData1,
ffr_buffer_len);
debug_printf("flash_read status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
```

```
}
status = memcmp((const void*) pData, (const void*) pData1, ffr_buffer_len);
debug_printf("cmpa key_store read data compare status = %x\r\n", status);
if (status != 0)
{
    return status;
}
/* call ffr_lock to lock the ffr region; the ffr region can not be accessed until the next reset;
call the flash_init  and the ffr_init APIs */
status = FLASH_API_TREE->ffr_lock(&FlashConfig);
debug_printf("ffr_lock status = %x\r\n", status);
if (status != kStatus_FLASH_Success)
{
    return status;
}
```

6.2.3 runBootloader API

The ROM bootloader provides an API for the user application to enter the ISP mode based on the designated ISP interface mode.

Prototype

*void (*runBootloader)(void *arg);*

Table 31. runBootloader API arg fields

Field	Offset	Description	
Tag	[31:24]	Fixed value: 0xEB (Enter Boot)	
Boot Mode	[23:20]	0 - Master boot mode	
		1 - Enter ISP mode	
ISP Interface	[19:16]	**ISP boot mode**	**Master boot mode**
		0 - Auto detection	0 – Internal flash boot
		1 - USB-HID	Reserved
		2 - UART	Reserved
		3 - SPI	Reserved
		4 - I2C	
		5 - CAN	
Reserved	[15:04]		
Image Index	[03:00]	Used for Master boot mode 0	

Example:

In the application image boot process, regardless of whether the ISP pin is connected to the high level, the device will directly enter the ISP mode through the UART interface according to the parameter ISP Interface in arg.

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define RUN_BOOTLOADER_API_TREE ((void (*)(void *)) ROM_API_TREE[0])
uint32_t arg = 0xEB120000;   /*0xEB: represents Enter Boot; 0x12: represents enter ISP mode by UART
only */
RUN_BOOTLOADER_API_TREE->runBootloader(&arg);
```

After the application image, which calls the above runBootloader API, has booted successfully, the device will only allow the UART interface to be connected to transfer the data with the host.

6.2.4 FLEXSPI FLASH Driver APIs

The FLEXSPI NOR Driver API set consists of APIs that can simplify external FLASH support. The API set provides the following features:

- Recognize an external Serial NOR FLASH device based on the JESD216 or above revision
- Probe the Octal Serial NOR FLASH via simplified option
- Program data to or read data from an external Serial NOR FLASH device
- Partially or fully erase an external Serial NOR FLASH device.
- Support for most Serial NOR FLASH devices

6.2.4.1 General description

The FlexSPI FLASH APIs are organized in the flexspi_nor_flash_driver_t structure. See the details in data structure below. See the details of each API in the subsequent sections.

```
typedef struct
{
    uint32_t version;
    status_t (*init)(uint32_t instance, flexspi_nor_config_t *config);
    status_t (*page_program)(uint32_t instance, flexspi_nor_config_t *config, uint32_t dstAddr, const
uint32_t *src);
    status_t (*erase_all)(uint32_t instance, flexspi_nor_config_t *config);
    status_t (*erase)(uint32_t instance, flexspi_nor_config_t *config, uint32_t start, uint32_t
length);
    status_t (*erase_sector)(uint32_t instance, flexspi_nor_config_t *config, uint32_t address);
    status_t (*erase_block)(uint32_t instance, flexspi_nor_config_t *config, uint32_t address);
    status_t (*get_config)(uint32_t instance, flexspi_nor_config_t *config,
serial_nor_config_option_t *option);
    status_t (*read)(uint32_t instance, flexspi_nor_config_t *config, uint32_t *dst, uint32_t start,
uint32_t bytes);
    status_t (*xfer)(uint32_t instance, flexspi_xfer_t *xfer);
    status_t (*update_lut)(uint32_t instance, uint32_t seqIndex, const uint32_t *lutBase, uint32_t
numberOfSeq);
    status_t (*set_clock_source)(uint32_t clockSrc);
    void (*config_clock)(uint32_t instance, uint32_t freqOption, uint32_t sampleClkMode);
    status_t (*partial_program)(uint32_t instance, flexspi_nor_config_t *config, uint32_t dstAddr,
const uint32_t *src, uint32_t length);
} flexspi_nor_flash_driver_t;
```

Each FLEXSPI FLASH API depends on a common *flexspi_nor_config_t* parameter to perform the proper operation. The *flexspi_nor_config_t* data structure is defined as below. See the details of the flexspi_nor_config_t in FlexSPI FLASH Driver APIs.

```
/* Serial NOR configuration block */

typedef struct _flexspi_nor_config
{
    flexspi_mem_config_t memConfig; /* Common memory configuration info via FlexSPI */
    uint32_t pageSize;              /* Page size of Serial NOR */
    uint32_t sectorSize;            /* Sector size of Serial NOR */
    uint8_t ipcmdSerialClkFreq;     /* Clock frequency for IP command */
    uint8_t isUniformBlockSize;     /* Sector/Block size is the same */
    uint8_t isDataOrderSwapped;     /* Data order (D0, D1, D2, D3) is swapped (D1,D0, D3, D2) */
    uint8_t reserved0[1];           /* Reserved for future use */
    uint8_t serialNorType;          /* Serial NOR FLASH type: 0/1/2/3 */
    uint8_t needExitNoCmdMode;      /* Need to exit NoCmd mode before other IP command */
    uint8_t halfClkForNonReadCmd;   /* Half the Serial Clock for non-read command: true/false */
    uint8_t needRestoreNoCmdMode;   /* Need to Restore NoCmd mode after IP commmand execution */
    uint32_t blockSize;             /* Block size */
    uint32_t flashStateCtx;         /* FLASH State Context */
    uint32_t reserved2[10];         /* Reserved for future use */
} flexspi_nor_config_t;
```

6.2.4.2 version

This field indicates the FlexSPI FLASH driver version number.

Prototype :

```
uint32_t version;
```

Example:

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define FLEXSPI_FLASH_API_TREE ((flexspi_nor_flash_driver_t *)ROM_API_TREE[11])
uint32_t FlexSPIFLASHDriverVersion = FLEXSPI_FLASH_API_TREE -> version;
```

In this device, the FlexSPI FLASH Driver version is “0x00010802”, it means the FLEXSPI FLASH API version is 1.8.2.

6.2.4.3 flexspi_nor_flash_init

The API is used for initializing the FlexSPI controller based on the parameters. It requires a full flexspi_nor_config_t data structure to initialize the FLEXSPI controller, clocks, FLASH etc. The data structure can be generated via flexspi_nor_get_config API or in other manners.

Prototype :

```
status_t (*init)(uint32_t instance, flexspi_nor_config_t *config);
```

Table 32. Parameters used in the flexspi_nor_flash_init API

Parameter	Description
instance	Point to the FlexSPI FLASH controller instance. Only instance 0 is supported on this device.
config	This parameter includes the first parameter flexspi_mem_config_t pointing to the flash configuration block(FCB) and related external FLASH device parameters, which consists of arguments related to an external FLASH device, for example, flash size, page size, sector size. See the details of the flexspi_nor_config_t in FlexSPI FLASH Driver APIs.

Example:

```
#define FLEXSPI_INSTANCE (0)
flexspi_nor_config_t flashConfig;
// get the valid flashConfig data here
// Call the API with proper flashConfig parameter
uint32_t status = FLEXSPI_FLASH_API_TREE->init (FLEXSPI_INSTANCE, &flashConfig);
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means FlexSPI FLASH driver initialization is completed successfully.

6.2.4.4 flexspi_nor_flash_page_program

The API is used for programming the page data into the Serial NOR FLASH, and the API should be called after the *flexspi_nor_flash_init* API call.

Prototype :

```
status_t (*page_program)(uint32_t instance, flexspi_nor_config_t *config, uint32_t dstAddr, const
uint32_t *src);
```

Table 33. Parameters used in flexspi_nor_flash_page_program API

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
config	This parameter points to the flash configuration block(FCB), consisting of arguments related to an external FLASH device, such as flash size, page size, and sector size. See the details of the flexspi_nor_config_t in FlexSPI FLASH Driver APIs.
dstAddr	The start address of the required FLASH memory to be programmed.
src	Pointer to the source buffer of data that is to be programmed into FLASH. The src address must be 32-bit aligned

Example:

```
uint32_t dstAddr = 0x1000 ;
uint8_t programBuffer[256];
for (uint32_t i=0; i<sizeof(programBuffer); i++)
{
    programBuffer[i] = (uint8_t)(i & 0xFF);
```

```
}
status = FLEXSPI_FLASH_API_TREE->page_program(FLEXSPI_INSTANCE, &flashConfig, dstAddr,
&programBuffer);
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means the data has been written into the specified region of the FLASH. It cannot guarantee the data has been successfully programmed into the FLASH device.

6.2.4.5 flexspi_nor_flash_erase_all

The API is used for erasing the entire external FLASH region, and the API should be called after the *flexspi_nor_flash_init* API call.

Prototype :

```
status_t (*erase_all)(uint32_t instance, flexspi_nor_config_t *config);
```

Table 34. Parameters used in flexspi_nor_flash_erase_all API

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
config	This parameter points to the flash configuration block(FCB) which consists of arguments related to an external FLASH device, for example, flash size, page size, sector size. See the details of the flexspi_nor_config_t in FlexSPI FLASH Driver APIs.

Example:

```
status = FLEXSPI_FLASH_API_TREE-> erase_all (FLEXSPI_INSTANCE, &flashConfig);
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means that the chip erase operation is completed on the FLASH side.

6.2.4.6 flexspi_nor_flash_erase

The API is used for erasing the specified region of the external FlexSPI FLASH region, and the API should be called after the *flexspi_nor_flash_init* API call.

Prototype :

```
status_t (*erase)(uint32_t instance, flexspi_nor_config_t *config, uint32_t start, uint32_t length);
```

Table 35. Parameters used in flexspi_nor_flash_erase API

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
config	This parameter points to the flash configuration block(FCB) which consists of arguments related to an external FLASH device, for example, flash size, page size, sector size. See the details of the flexspi_nor_config_t in FlexSPI FLASH Driver APIs.

Table continues on the next page...

Table 35. Parameters used in flexspi_nor_flash_erase API (continued)

Parameter	Description
start	The start offset address of the required flash memory to be erased. The start address must be page-aligned (that is, a multiple of page of the specific FlexSPI Flash).
lengthInBytes	The length, given in bytes (not words or long words) to be erased. Must be page-aligned.

Example:

```
uint32_t offset = 0x1000 ;
uint832_t lengthInBytes = 0x1000 ;
status = FLEXSPI_FLASH_API_TREE-> erase (FLEXSPI_INSTANCE, &flashConfig, offset, lengthInBytes);
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means the external FlexSPI FLASH region (0x80001000 – 0x80002000) has been erased successfully.

6.2.4.7 flexspi_nor_flash_erase_sector

The API is used for erasing the specified sector of the external FlexSPI FLASH, and the API should be called after the *flexspi_nor_flash_init* API call.

Prototype :

```
status_t (*erase_sector)(uint32_t instance, flexspi_nor_config_t *config, uint32_t address);
```

Table 36. Parameters used in flexspi_nor_flash_erase_sector

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
config	This parameter points to the flash configuration block(FCB) which consists of arguments related to an external FLASH device, for example, flash size, page size, sector size. See the details of the flexspi_nor_config_t in FlexSPI FLASH Driver APIs.
start	The parameter specifies the offset address of the sector to be erased.

Example:

```
uint32_t offset = 0x0 ;
status = FLEXSPI_FLASH_API_TREE-> erase_sector (FLEXSPI_INSTANCE, &flashConfig, offset);
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means the specified sector region of the external FlexSPI FLASH region has been erased successfully.

6.2.4.8 flexspi_nor_flash_erase_block

The API is used for erasing the specified block region of the external FlexSPI FLASH, and the API should be called after the *flexspi_nor_flash_init* API call.

Prototype :

```
status_t (*erase_block)(uint32_t instance, flexspi_nor_config_t *config, uint32_t address);
```

Table 37. Parameters used in flexspi_nor_flash_erase_block

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
config	This parameter points to the flash configuration block(FCB) which consists of arguments related to an external FLASH device, for example, flash size, page size, sector size. See the details of the flexspi_nor_config_t in FlexSPI FLASH Driver APIs.
start	The parameter specifies the offset address of the block to be erased.

Example:

```
uint32_t offset = 0x0 ;
status = FLEXSPI_FLASH_API_TREE-> erase_block (FLEXSPI_INSTANCE, &flashConfig, offset);
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means the specified block region of the external FlexSPI FLASH has been erased successfully.

6.2.4.9 flexspi_nor_get_config

The API is used for getting the FlexSPI NOR configuration block based on specified option. It can probe the FLASH type and generate the flexspi_nor_config_t for later full initialization. Typically, the flexspi_nor_flash_init will be performed with the flexspi_nor_config_t parameter probed by the flexspi_nor_get_config API.

Note:

This FlexSPI Flash API only support the Quad FlexSPI Flash type.

Prototype :

```
status_t (*get_config)(uint32_t instance, flexspi_nor_config_t *config, serial_nor_config_option_t
*option);
```

Table 38. Parameters used in flexspi_nor_get_config API

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
config	This parameter points to the flash configuration block(FCB) which consists of arguments related to an external FLASH device, for example, flash size, page size, sector size. See the details of the flexspi_nor_config_t in FlexSPI FLASH Driver APIs.
option	This parameter provides a simplified option for the FlexSPI driver to probe the FLASH and get the FLASH parameters later. See the details of the flexspi_nor_config_t in FLEXSPI FLASH Driver APIs. Definitions of this block as below.

Table continues on the next page...

Parameter	Description

```
typedef struct _serial_nor_config_option
{
    union
    {
        struct
        {
            uint32_t max_freq : 4;              //!< Maximum supported Frequency
            uint32_t misc_mode : 4;            //!< miscellaneous mode
            uint32_t quad_mode_setting : 4;   //!< Quad mode setting
            uint32_t cmd_pads : 4;           //!< Command pads
            uint32_t query_pads : 4;        //!< SFDP read pads
            uint32_t device_type : 4;      //!< Device type
            uint32_t option_size : 4;     //!< Option size, in terms of uint32_t,size = (option_size
+ 1)* 4
            uint32_t tag : 4;            //!< Tag, must be 0x0E
        } B;
        uint32_t U;
    } option0;

    union
    {
        struct
        {
            uint32_t dummy_cycles : 8;    //!< Dummy cycles before read
            uint32_t status_override : 8; //!< Override status register value during device mode
configuration
            uint32_t pinmux_group : 4;    //!< The pinmux group selection
            uint32_t dqs_pinmux_group : 4; //!< The DQS Pinmux Group Selection
            uint32_t drive_strength : 4;   //!< The Drive Strength of FlexSPI Pads
            uint32_t flash_connection : 4; //!< FLASH connection option: 0 - Single FLASH connected
to port A, 1 - Reserved, 2 - Single FLASH connected to Port B
        } B;
        uint32_t U;
    } option1;

} serial_nor_config_option_t;
```

Table 39. serial_nor_config_option_t option

Offset	Field	Description
0	Option0	See option0 definition for more details
4	Option1	Optional, effective only if the option Size field in Option0 is non-zero See option1 definition for more details.

Table 40. option0 definition

Field	Bits	Description
tag	31:28	The tag of the config option, fixed to 0x0C

Table continues on the next page...

Table 40. option0 definition (continued)

Field	Bits	Description
option_size	27:24	Size in bytes = (Option Size + 1) * 4 It is 0 if only option0 is required
device_type	23:20	Device Detection Type 0 - Read SFDP for SDR commands 1 - Read SFDP for DDR Read commands 2 - HyperFLASH 1V8 3 - HyperFLASH 3V 4 - Macronix Octal DDR 6 - Micron Octal DDR 8 - Adesto EcoXiP DDR
query_pad	19:16	Data pads during Query command (read SFDP or read MID) 0 - 1 2 - 4 3 – 8
cmd_pad	15:12	Data pads during FLASH access command 0 - 1 2 - 4 3 – 8
quad_mode_setting	11:8	Quad Mode Enable Setting 0 - Not configure 1 - Set bit 6 in Status Register 1 2 - Set bit 1 in Status Register 2 3 - Set bit 7 in Status Register 2 4 - Set bit 1 in Status Register 2 via 0x31 command **NOTE** This field will be effective only if the device is compliant with JESD216 only 5 - (9 longword SDFP table)
misc_mode	7:4	Miscellaneous Mode 0 - Not enabled 1 - Enable 0-4-4 mode for High Random Read performance

Table continues on the next page...

Table 40. option0 definition (continued)

Field	Bits	Description
		3 - Data Order Swapped mode (for MXIC OctaFLASH only) 5 - Select the FlexSPI data sample source as internal loop back; see FlexSPI usage for more details 6 - Configure the FlexSPI NOR flash memory running at stand SPI mode **NOTE** Experimental feature only. Do not use in production: keep it as 0 for normal usage.
max_freq	3:0	Max FLASH Operation speed 0 - Don't change the FlexSPI clock setting

Table 41. option1 definition

Field	Bits	Description
flash_connection	31:28	FLASH connection option: • 0 - Single FLASH connected to port A • 1 - Reserved • 2 - Single FLASH connected to Port B
Reserved	27:24	**NOTE** This field does not have any effect in this device.
dqs_pinmux_group	23:20	The DQS Pinmux Group Selection
pinmux_group	19:16	The pinmux group selection
status_override	15:8	Override status register value during device mode configuration
dummy_cycles	7:0	Dummy cycles for a read command • 0 - Use detected dummy cycle • Others - dummy cycles provided in the flash datasheet

Typical Option configurations are as below:

- QUAD NOR - Quad SDR Read: option0 = 0xc0000004 (75MHz)
- QUAD NOR - Quad DDR Read: option0 = 0xc0100003 (60MHz)

Example:

```
#define FLASH_OPTION_QSPI_SDR 0xc0000001
serial_nor_config_option_t flashConfigOption = { .option0 = { .U = FLASH_OPTION_QSPI_SDR } };
status = FLEXSPI_FLASH_API_TREE-> get_config(FLEXSPI_INSTANCE, &flashConfig, &flashConfigOption);
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means the FlexSPI FLASH config block has been configured based on the provided option successfully.

6.2.4.10 flexspi_nor_flash_read

The API is used for getting the FlexSPI NOR Flash data and the API should be called only after the FLEXSPI IP has been properly configured.

Prototype :

```
status_t (*read)(uint32_t instance, flexspi_nor_config_t *config, uint32_t *dst, uint32_t start,
uint32_t bytes);
```

Table 42. Parameters used in flexspi_nor_flash_read API

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
config	This parameter points to the flash configuration block(FCB) which consists of arguments related to an external FLASH device, for example, flash size, page size, sector size. See the details of the flexspi_nor_config_t in FlexSPI FLASH Driver APIs.
dst	Pointer to the buffer used for storing the read data. The dst address must be 32-bit aligned
start	The offset address of the required flash memory to be read.
lengthInBytes	The length, given in bytes (not words or long words) to be read. Must be page-aligned.

Example:

```
uint32_t start_offset = 0x1000;
uint32_t read_buffer[512] = {0};
status = FLEXSPI_FLASH_API_TREE->read(FLEXSPI_INSTANCE, &flashConfig, & read_buffer, start_offset,
sizeof(read_buffer));
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means the FlexSPI FLASH (0x80001000 – 0x80001800) has been read and stored the data into the read_buffer successfully.

6.2.4.11 flexspi_command_xfer

The API is used for performing the FlexSPI command, this API should be called only when the FLEXSPI IP is properly configured.

Prototype :

```
typedef enum _FlexSPIOperationType
{
    // FlexSPI operation: Only command, both TX and RX buffer are ignored.
    kFlexSpiOperation_Command,
    // FlexSPI operation: Configure device mode, the TX FIFO size is fixed in LUT.
    kFlexSpiOperation_Config,
    // FlexSPI operation: Write,  only TX buffer is effective
    kFlexSpiOperation_Write,
    // FlexSPI operation: Read, only Rx Buffer is effective.
    kFlexSpiOperation_Read,
```

```
    kFlexSpiOperation_End,
} flexspi_operation_t;

//!@brief FlexSPI Transfer Context
typedef struct _FlexSpiXfer
{
    flexspi_operation_t operation; // FlexSPI operation
    uint32_t baseAddress;          // FlexSPI operation base address
    uint32_t seqId;                // Sequence Id
    uint32_t seqNum;               // Sequence Number

    bool Reserved;     // This field does not have any effect in the ROM boot flow.
    const uint32_t *txBuffer;      // Tx buffer
    uint32_t txSize;               // Tx size in bytes
    uint32_t *rxBuffer;            // Rx buffer
    uint32_t rxSize;               // Rx size in bytes
} flexspi_xfer_t;

status_t flexspi_command_xfer(uint32_t instance, flexspi_xfer_t *xfer);
```

Table 43. Parameters used in flexspi_command_xfer API

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
xfer	This parameter pointer to FlexSPI Transfer Context which consists of arguments related to FlexSPI operation, sequence, TX and RX related parameters.

Example:

```
uint32_t start_offset = 0x1000;
uint32_t read_buffer[512] = {0};
flexspi_xfer_t flashXfer;
flashXfer.operation = kFlexSpiOperation_Read;
flashXfer.seqNum = 1;
flashXfer.seqId = NOR_CMD_LUT_SEQ_IDX_READ;
while (sizeof(read_buffer))
{
    uint32_t readLength = bytes > 65535 ? 65535 : bytes;
    flashXfer.baseAddress = start_offset;
    flashXfer.rxBuffer = read_buffer;
    flashXfer.rxSize = readLength;
    status = FLEXSPI_FLASH_API_TREE->flexspi_command_xfer (instance, &flashXfer);
    if (status != kStatus_Success)
    {
        return status;
    }
    bytes -= readLength;
    start_offset += readLength;
    read_buffer += readLength / sizeof(uint32_t);
}
return status;
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means the FlexSPI FLASH command_xfer has executed the read command successfully, and the read data (0x80001000 – 0x80001800) has been load into the read_buffer array.

NOTE

This API relies on the valid LUT to perform the supported operations. If the LUT is invalid, the result is unpredictable.

6.2.4.12 flexspi_update_lut

The API is used for configuring the FlexSPI Lookup table before calling the flexspi_command_xfer API. This API should be called only when the FLEXSPI IP is properly configured.

Prototype :

```
status_t (*update_lut)(uint32_t instance, uint32_t seqIndex, const uint32_t *lutBase, uint32_t
numberOfSeq);
```

Table 44. Parameters used in flexspi_update_lut API

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
seqIndex	This parameter pointer to FlexSPI Dedicated LUT Sequence Index for IP Command.
lutBase	This parameter pointer to FlexSPI Lookup table , which holds FLASH command sequences
numberOfSeq	This parameter points to FlexSPI Sequence Number

Example:

```
// Program Pattern
const uint32_t probe_pattern[4] = { 0x33221100, 0x77665544, 0xbbaa9988, 0xffeeddcc };
flexspi_xfer_t flashXfer;
flashXfer.baseAddress = sizeof(flexspi_nor_config_t);
flashXfer.seqId = NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM;
flashXfer.operation = kFlexSpiOperation_Write;
flashXfer.txBuffer = (uint32_t *)probe_pattern;
flashXfer.txSize = sizeof(probe_pattern);
flexspi_update_lut(instance,NOR_CMD_LUT_FOR_IP_CMD,&flashConfig-
>memConfig.lookupTable[4*flashXfer.seqId], flashXfer.seqNum);
flashXfer.seqId = NOR_CMD_LUT_FOR_IP_CMD;
status = flexspi_command_xfer(instance, &flashXfer);
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means the flexspi_update_lut has execute successfully, and the probe_pattern data has been load into the (baseAddress + 0x80000000) region.

6.2.4.13 flexspi_set_clock_source

The API is used for configuring the FlexSPI clock source. This API should be called before calling the flexspi_not_get_config. This API should be called only when the FlexSPI IP is properly configured. This API only changes the FLEXSPI clock selector in the SYSCON register if the user needs to use it.

Prototype:

```
status_t (*set_clock_source)(uint32_t clockSrc);
```

Parameter	Description
clockSrc	Point to the FlexSPI select clock source Flexspi clock select: • 0000 - No clock • 0001 - PLL0 clock • 0010 - No clock • 0011 - FRO_HF • 0100 - No clock • 0101 - PLL1 clock • 0110 - USB PLL clock

Example:

```
uint32_t flexspi_clock_source = 0x0;
status = FLEXSPI_FLASH_API_TREE-> set_clock_source (flexspi_clock_source);
```

See the possible status codes in Table 94. If the status is kStatus_Success return value, it means the set_clock_source has execute successfully, and the FlexSPI has configure the clock source from the configured clock before.

6.2.4.14 flexspi_config_clock

The API is used for configuring the FlexSPI clock. This API should be called only when the FLEXSPI IP is properly configured. flexspi_set_clock_source only changes the clock selector in SYSCON, flexspi_clock_config can configure the clock frequency of the FLEXSPI root clock and clock source.

Prototype:

```
void (*config_clock)(uint32_t instance, uint32_t freqOption, uint32_t sampleClkMode);
```

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
freqOption	This parameter pointer to FlexSPIFlexSPI flash serial clock frequency: kFlexSpiSerialClk_30MHz (1U) kFlexSpiSerialClk_50MHz (2U) kFlexSpiSerialClk_60MHz (3U) kFlexSpiSerialClk_75MHz (4U) kFlexSpiSerialClk_100MHz (5U)

sampleClkMode	This parameter pointer to configure the FlexSPI clock configuration type.
	kFlexSpiClk_SDR (0U) //!< Clock configure for SDR mode
	kFlexSpiClk_DDR (1U) //!< Clock configure for DDR mode

Example:

```
uint32_t flexspi_freqOption = 0x1;
uint32_t flexspi_sampleClkMode = 0x0;
status = FLEXSPI_FLASH_API_TREE-> config_clock(FLEXSPI_INSTANCE, flexspi_freqOption,
flexspi_sampleClkMode);
```

See the possible status codes in Table 94. If the status is kStatus_Success return value, it means the config_clock has execute successfully, and the FlexSPI flash clock will be configured with the provided parameter.

6.2.4.15 flexspi_nor_flash_partial_program

The API is used for partially programming the data into the specified address of FlexSPI FLASH, and the API should be called after the *flexspi_nor_flash_init* API call.

Prototype :

```
status_t (*partial_program)(uint32_t instance, flexspi_nor_config_t *config, uint32_t dstAddr, const
uint32_t *src, uint32_t length);
```

Table 45. Parameters used in the flexspi_nor_flash_partial_program API

Parameter	Description
instance	Point to the FlexSPI controller instance, only support 0
config	This parameter points to the flash configuration block(FCB) which consists of arguments related to an external FLASH device, for example, flash size, page size, sector size. See the details of the flexspi_nor_config_t in FlexSPI FLASH Driver APIs.
dstAddr	The start address of the required FlexSPI flash memory to be programmed.
src	Pointer to the source buffer of data that is to be programmed into FlexSPI flash.
length	The program size of the source buffer.

Example:

```
uint32_t flashAhbAddr = 0x80001000UL;
uint32_t flashAhbBase = 0x80000000UL;
uint32_t flashOffset = flashAhbAddr - flashAhbBase;
uint32_t programBuffer[256 / sizeof(uint32_t)];
uint8_t *buf_u8 = (uint8_t*)programBuffer;
for (uint32_t i=0; i<sizeof(programBuffer); i++)
{
    *buf_u8++ = (uint8_t)(i & 0xFFU);
}
status = FLEXSPI_FLASH_API_TREE->partial_program(FLEXSPI_INSTANCE, &flashConfig, flashOffset,
&programBuffer, sizeof(programBuffer));
```

See the possible status codes in Table 46. If the status is kStatus_Success return value, it means the data has been written into the FlexSPI FLASH.

6.2.4.16 Possible return codes for FLEXSPI FLASH driver APIs

Table 46. Return codes for FLEXSPI FLASH APIs

Error Code	Value	Description
kStatus_Success	0	Operation succeeded without error.
kStatus_Fail	1	The operation failed with a generic error.
kStatus_Invalidargument	4	The requested command's argument is undefined.
kStatus_FLEXSPI_SequenceExecutionTimeout	6000	The FLEXSPI Sequence Execution timeout
kStatus_FLEXSPI_InvalidSequence	6001	The FLEXSPI LUT sequence invalid
kStatus_FLEXSPI_DeviceTimeout	6002	The FLEXSPI device timeout
kStatus_FLEXSPINOR_ProgramFail	20100	Page programming failure
kStatus_FLEXSPINOR_EraseSectorFail	20101	Sector Erase failure
kStatus_FLEXSPINOR_EraseAllFail	20102	Chip Erase failure
kStatus_FLEXSPINOR_WaitTimeout	20103	The execution time out
kStatus_FlexSPINOR_WriteAlignmentError	20105	Address alignment error
kStatus_FlexSPINOR_CommandFailure	20106	Erase/Program Verify Error
kStatus_FlexSPINOR_SFDP_NotFound	20107	SFDP table was not found call
kStatus_FLEXSPINOR_Unsupported_SFDP_Version	20108	Unrecognized SFDP version
kStatus_FLEXSPINOR_FLASH_NotFound	20109	Flash detection failure
kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed	20110	DDR Read dummy probe failure

6.2.4.17 FlexSPI Flash APIs Example

The section will provide the example to introduce All the FlexSPI Flash APIs usage and help users understand and use more deeply. Example(Program the image or specific data into the FlexSPI Flash region):

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define FLEXSPI_FLASH_API_TREE ((flexspi_nor_flash_driver_t *)ROM_API_TREE[11])
#define FLEXSPI_INSTANCE 0
#define FLEXSPI_PROGRAM_OFFSET (0x1000u)
#define FLEXSPI_ERASE_SIZE (0x2000u)
#define MAX_PAGE_SIZE 512
flexspi_nor_config_t flashConfig;
```

```
uint32_t FlexSPIFLASHDriverVersion = FLEXSPI_FLASH_API_TREE->version;
debug_printf("FlexSPIFLASHDriverVersion version= %x\r\n", FlexSPIFLASHDriverVersion);

// choose the Quad SDR FlexSPI Flash type as the example

serial_nor_config_option_t flashConfigOption = { .option0 = { .U = 0xc0000001}};
status = FLEXSPI_FLASH_API_TREE->get_config(FLEXSPI_INSTANCE, &flashConfig, &flashConfigOption);
debug_printf("flexspi_nor_get_config API status=%d!\n",status);
if (status != kStatus_Success)
{
    return status;
}

//Init the FlexSPI Flash base on the flashConfig, which is configured from the get_config API
status = FLEXSPI_FLASH_API_TREE->init(FLEXSPI_INSTANCE, &flashConfig);
debug_printf("flexspi_nor_flash_init returned error:%d\n", status);
if (status != kStatus_Success)
{
    return status;
}

status = FLEXSPI_FLASH_API_TREE->erase(FLEXSPI_INSTANCE, &flashConfig, FLEXSPI_PROGRAM_OFFSET,
FLEXSPI_ERASE_SIZE)
debug_printf ("flexspi_nor_flash_erase returned status:%d\n", status);;
if (status != kStatus_Success)
{
    return status;
}

// Create the specific data programBuffer to program the data into the FlexSPI Flash, also the data
can be replaced by image data provided by user
uint32_t programBuffer[MAX_PAGE_SIZE / sizeof(uint32_t)];
uint8_t *pBuffer = (uint8_t *)&programBuffer[0];
for (uint32_t i = 0; i < sizeof(programBuffer); i++)
{
    *pBuffer++ = (uint8_t)(i & 0xFF);
}

status = FLEXSPI_FLASH_API_TREE->page_program(FLEXSPI_INSTANCE, &flashConfig, FLEXSPI_PROGRAM_OFFSET,
programBuffer);
debug_printf("flexspi_nor_flash_page_program returned error:%d\n", status);
if (status != kStatus_Success)
{
    return status;
}

uint32_t readBuffer[MAX_PAGE_SIZE];
status = FLEXSPI_FLASH_API_TREE->read(FLEXSPI_INSTANCE, &flashConfig, readBuffer,
FLEXSPI_PROGRAM_OFFSET,
flashConfig.pageSize);
debug_printf("flexspi_nor_flash_read status = %d\n", status);

if (status != kStatus_Success)
{
    return status;
}
else
{
    if (memcmp(readBuffer, programBuffer, flashConfig.pageSize) == 0)
    {
```

```
    debug_printf ("Verify programming by IP read: PASSED\n");
    }
}

status = FLEXSPI_FLASH_API_TREE->erase(FLEXSPI_INSTANCE, &flashConfig, FLEXSPI_PROGRAM_OFFSET,
FLEXSPI_ERASE_SIZE)
debug_printf ("flexspi_nor_flash_erase returned status:%d\n", status);;
if (status != kStatus_Success)
{
    return status;
}
```

6.2.5 OTP-eFuse APIs

6.2.5.1 Version

This fields indicates the version number of the OTP API in this device.

Prototype :

```
standard_version_t version;
```

Table 47. Definition of the version Field in OTP API tree

Parameter	Description
standard_version_t	Pointer to version structure to store current eFuse driver version information uint8_t bugfix; /* bugfix version [7:0] */ uint8_t minor; /* minor version [15:8] */ uint8_t major; /* major version [23:16] */ char name; /* character name is "E", where "E" stands for eFuse. Character name and version are fixed */

Example:

```
#define ROM_API_TREE ((uint32_t*)0x1303fc 00)
#define OTP_API_TREE ((efuse_driver_t *)ROM_API_TREE[12])
uint32_t otpDriverVersion = OTP_API_TREE-> version;
```

In this device, the OTP driver version is "0x45010100", it means the OTP driver version is 1.1.0.

Following is the efuse_driver_t definition:

```
/* @brief EFUSE driver API Interface */
typedef struct
{
    standard_version_t version; /* efuse driver API version number */
    status_t (*init)(void);
    status_t (*deinit)(void);
    status_t (*p_efuse_read)(uint32_t addr, uint32_t *data);
    status_t (*p_efuse_program)(uint32_t addr, uint32_t data);
```

```
} efuse_driver_t;
```

6.2.5.2 otp_init

The API is used for initializing the OTP clock. The *otp_init* API should be called before all the other OTP APIs.

Prototype :

```
status_t (*init)(void);
```

Example :

```
uint32_t status = OTP_API_TREE->init ();
```

See the possible status codes in Table 50. If the status is kStatus_Success return value, it means OTP driver initialization completed successfully.

6.2.5.3 otp_deinit

This API is used to disable OTP controller APB clock.

Prototype :

```
status_t (*deinit)(void);
```

Example :

```
uint32_t status = OTP_API_TREE->deinit ();
```

See the possible status codes in Table 50. If the status is kStatus_Success return value, it means OTP driver deinitialization completed successfully.

6.2.5.4 otp_read

This API is used to read data from OTP-eFuse controller.

Prototype :

```
status_t (*p_efuse_read)(uint32_t addr, uint32_t *data);
```

Table 48. Parameters used in otp_read API

Parameter	Description
addr	Pointer to the eFuse index that specifies the EFUSE word to be read out. The index 0 read data: 0 - 127 bits one word data (which 32bits word data in the 128bits will be changed to a byte) The index 1 read data: 128 - 255 bits one word data (which 32bits word data in the 128bits will be changed to a byte)

Table continues on the next page...

Table 48. Parameters used in otp_read API (continued)

Parameter	Description
	The index 2 read data: 256 - 383 bits one word data (which 32bits word data in the 128bits will be changed to a byte) The index 3 read data: 800 - 831 bits (one word data) The following index return data will follow the index 3, each word will increase the index by one.
data	Pointer to the buffer used for storing the read 32 bit data.

Example :

```
#define EFUSE_INDEX (0x0)
uint32_t eFuseData;
uint32_t status = OTP_API_TREE-> p_efuse_read (EFUSE_INDEX, & eFuseData);
```

See the possible status codes in Table 50. If the status is kStatus_Success return value, it means the OTP data read completed successfully.

6.2.5.5 otp_program

This API is used to program data to specified OTP Word.

Prototype :

```
status_t (*p_efuse_program)(uint32_t addr, uint32_t data);
```

Table 49. Parameters used in the otp_program API

Parameter	Description
addr	Pointer to the eFuse index that specifies the EFUSE word to be programmed
data	Pointer to the 32-bit source data used for programming into the eFuse region.

Example:

```
#define EFUSE_INDEX (43)  /* represents CUST Efuse Index */
uint32_t eFuseData = 0x01010101;
uint32_t status = OTP_API_TREE-> p_efuse_program (EFUSE_INDEX, & eFuseData);
```

See the possible status codes in Table 50. If the status is kStatus_Success return value, it means the data 0x01010101 has been programmed into the eFuse index 43 region.

6.2.5.6 Possible return codes of OTP API

Table 50. Return codes for OTP APIs

Error Code	Value	Description

Table continues on the next page...

Table 50. Return codes for OTP APIs (continued)

kStatus_Success	0	Operation succeeded without error.
kStatus_Fail	1	The operation failed with a generic error.
kStatus_ReadOnly	2	Requested value cannot be changed because it is read-only.
kStatus_OutOfRange	3	Requested value is out of range.
kStatus_InvalidArgument	4	The requested command's argument is undefined.
kStatus_Timeout	5	A timeout occurred.

6.2.5.7 OTP-eFuse APIs Example

This section provides an example of OTP API usage.

NOTE

The OTP-eFuse program is only-program-once, user need to make sure the program data is valid value.

Example:

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define OTP_API_TREE ((efuse_driver_t *)ROM_API_TREE[12])
#define EFUSE_CUST_TEST_INDEX 43 /* The efuse read will return the efuse bit 2080 - 2111 bits data in
the efusemap file */

#define EFUSE_CMPA_CRC_INDEX 1
uint32_t otpDriverVersion = OTP_API_TREE->version;
debug_printf("otpDriverVersion version= %x\r\n", otpDriverVersion);
uint32_t status = OTP_API_TREE->init();
debug_printf("efuse init status= %x\r\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* The otp efuse value program should not program invalid data into the efuse region, otherwise the
device can boot fail and not behave as expected */

uint32_t fuseLCVal = 0;
status_t status = OTP_API_TREE->efuse_program(EFUSE_CUST_TEST_INDEX, fuseLCVal);
debug_printf("efuse_program LC state status= %x\r\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* Here crc value of the cmpa is just the example data, user needs to set the valid crc data of the
cmpa */

uint32_t fuseCMPACRCVal = 0x12345678;
status_t status = OTP_API_TREE->efuse_program(EFUSE_CMPA_CRC_INDEX, fuseCMPACRCVal);
debug_printf("efuse_program crc value of the CMPA status= %x\r\n", status);
if (status != kStatus_Success)
{
```

```
    return status;
}
uint32_t fuseLCVal1;
uint32_t fuseCMPACRCVal1;
status_t status = OTP_API_TREE->efuse_read(EFUSE_LC_STATE_INDEX, &fuseLCVal1);
debug_printf("efuse_read LC state status = %x \r\n", status);
if (status != kStatus_Success)
{
    return status;
}
status_t status = OTP_API_TREE->efuse_read(EFUSE_CMPA_CRC_INDEX, & fuseCMPACRCVal1);
debug_printf("efuse_program crc value of the CMPA status= %x \r\n", status,);
if (status != kStatus_Success)
{
    return status;
}
status_t status = OTP_API_TREE->deinit();
debug_printf("efuse deinit status= %x\r\n", status);
if (status != kStatus_Success)
{
    return status;
}
```

6.2.6 IAP APIs

6.2.6.1 General description

IAP is in–application–programming. The general IAP APIs call and execution flow is shown in Figure 8.

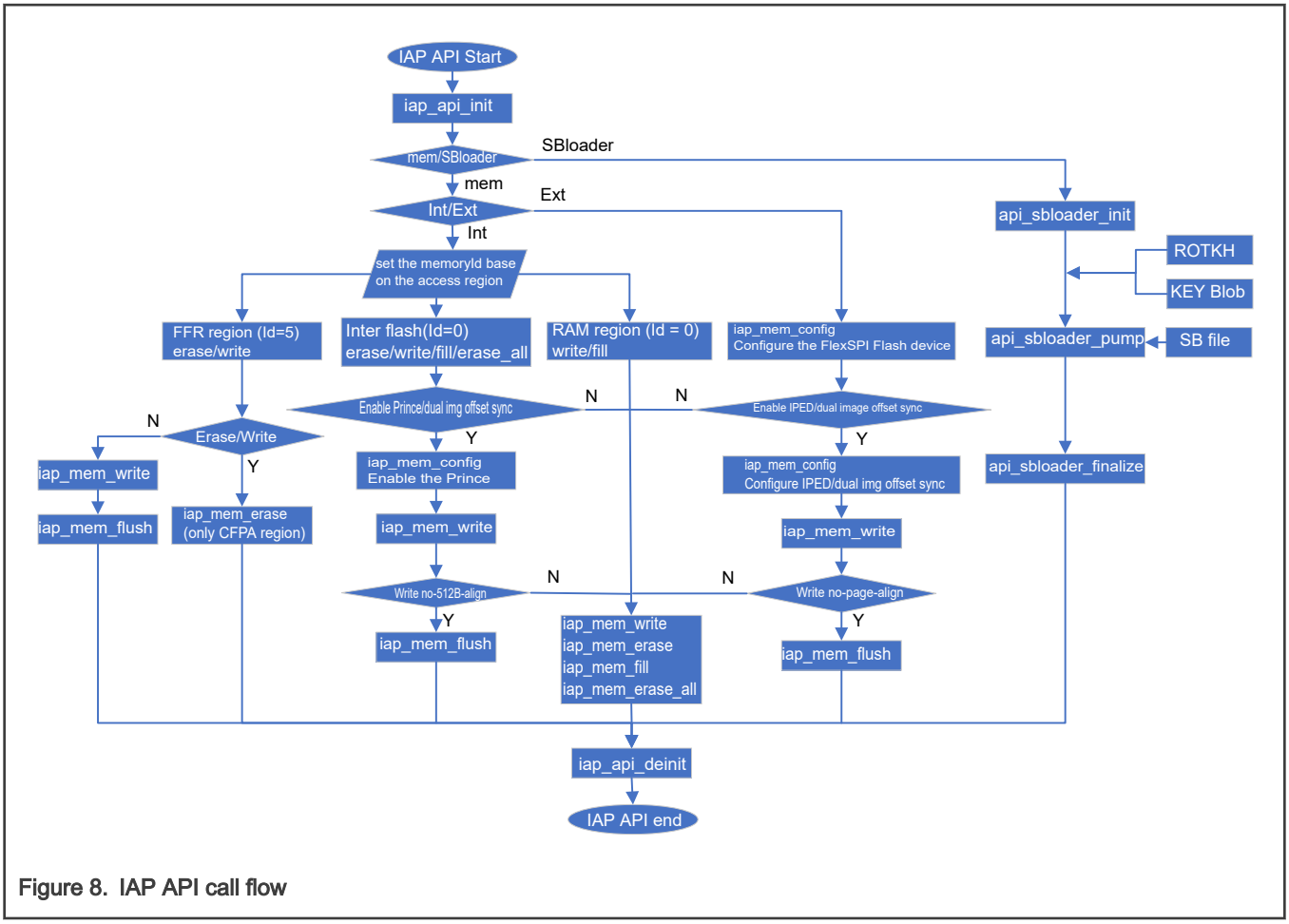


Figure 8. IAP API call flow

The bootloader IAP APIs are organized in the iap_api_interface_t structure, see the details from the below data structure. See the details of each API in the subsequent sections.

```
typedef struct iap_api_interface_struct
{
    standard_version_t version; /* IAP API version number */
    status_t (*api_init)(api_core_context_t *coreCtx, const kp_api_init_param_t *param);
    status_t (*api_deinit)(api_core_context_t *coreCtx);
    status_t (*mem_init)(api_core_context_t *ctx);
    status_t (*mem_read)(api_core_context_t *ctx, uint32_t addr, uint32_t len, uint8_t *buf, uint32_t
memoryId);
    status_t (*mem_write)(api_core_context_t *ctx, uint32_t addr, uint32_t len, const uint8_t *buf,
uint32_t memoryId);
    status_t (*mem_fill)(api_core_context_t *ctx, uint32_t addr, uint32_t len, uint32_t pattern,
uint32_t memoryId);
    status_t (*mem_flush)(api_core_context_t *ctx);
    status_t (*mem_erase)(api_core_context_t *ctx, uint32_t addr, uint32_t len, uint32_t memoryId);
    status_t (*mem_config)(api_core_context_t *ctx, uint32_t *buf, uint32_t memoryId);
    status_t (*mem_erase_all)(api_core_context_t *ctx, uint32_t memoryId);
    status_t (*sbloader_init)(api_core_context_t *ctx);
    status_t (*sbloader_pump)(api_core_context_t *ctx, uint8_t *data, uint32_t length);
    status_t (*sbloader_finalize)(api_core_context_t *ctx);
} iap_api_interface_t;
```

Each IAP API depends on the common context named api_core_context_t. The detail of the context is defined as bellow:

```
/* @brief The API context structure */

typedef struct api_core_context
{
    uint32_t reserved0[10];
    uint32_t reserved1[3];
    flash_config_t flashConfig;
    flexspi_nor_config_t flexspinorCfg;
    uint32_t reserved2[2];
    uint32_t reserved3[1];
    nboot_context_t *nbootCtx;
    uint8_t *sharedBuf;
    uint32_t reserved4[6];
} api_core_context_t;
```

The kp_api_init_param_t is defined as below:

```
typedef struct kb_api_parameter_struct
{
    uint32_t allocStart;
    uint32_t allocSize;
} kp_api_init_param_t;
```

The IAP APIs require a dedicated RAM region as the buffer for each operation. The user application needs to prepare an unused RAM region organized as the *kp_api_init_param_t* structure, and pass this structure to the iap_api_init API. The allocSize in the iap_api_init API call is used to store the different flash context which is used in the api_core_context_t. The minimum size can be set to at least the size of (api_core_context_t).

These fields are managed by api_core_context_t parameter, the user application doesn't need to touch it.

NOTE

Before IAP APIs are used, caller needs to make sure that ELS clocks are enabled. This is required for correct operation of sbloader and iap_mem API on devices with security subsystem.

6.2.6.2 version

The version field indicates the IAP API version number in the device.

Prototype :

```
standard_version_t version;
```

Table 51. Definition of the version in IAP API tree

Parameter	Description

standard_version_t	Pointer to version structure to store current FLASH driver version information
	uint8_t bugfix; //!< bugfix version [7:0]
	uint8_t minor; //!< minor version [15:8]
	uint8_t major; //!< major version [23:16]
	char name; //!< character name is 0x4B = "K".

Example:

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define IAP_API_TREE ((iap_api_interface_t*)ROM_API_TREE[13])
uint32_t IAPVersion = IAP_API_TREE-> version;
```

In this SoC, the version of IAP Driver is “0x4B030100”, it means the API version is 1.0.0.

6.2.6.3 iap_api_init

The API is used for initializing the IAP API context for the other IAP APIs, this API must be called prior to other IAP APIs.

Prototype :

```
status_t (*api_init)(api_core_context_t *coreCtx, const kp_api_init_param_t *param);
```

Table 52. Parameters used in iap_api_init API

Parameter	Description
coreCtx	Pointer to IAP API core context structure.
param	Pointer to IAP API initialization data structure, used for storing the data during execute the IAP APIs.

Example:

```
api_core_context_t apiCoreCtx;
const kp_api_init_param_t apiInitParam = { .allocStart = 0x30010000, .allocSize = 0x6000 };
uint32_t status = IAP_API_TREE->api_init(&apiCoreCtx, &apiInitParam);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the IAP API operating environment has been init successfully.

6.2.6.4 iap_api_deinit

The API is used for deinitializing the IAP API context.

Prototype :

```
status_t (*api_deinit)(api_core_context_t *coreCtx);
```

Table 53. Parameters used in iap_api_deinit API

Parameter	Description
coreCtx	Pointer to IAP API core context structure.

Example:

```
uint32_t status = IAP_API_TREE->api_deinit(&apiCoreCtx);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the IAP API operating environment has been freed and cannot call the other IAP APIs.

6.2.6.5 iap_mem_init

The API is used for initializing the memory interface, including the internal FLASH, RAM, FLEXSPI FLASH.

Prototype :

```
status_t (*iap_mem_init)(api_core_context_t *coreCtx);
```

Table 54. Parameters used in the iap_mem_init API

Parameter	Description
coreCtx	Pointer to IAP API core context structure.

Example:

```
uint32_t status = IAP_API_TREE->iap_mem_init(&apiCoreCtx);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the IAP API memory interfaces have been init successfully.

NOTE

The mem_init will also be called in the api_init API, so the mem_init no need to be called again after calling the api_init.

6.2.6.6 iap_mem_write

The API is used for programing the data into the internal FLASH, RAM or the external FlexSPI FLASH based on the parameters, such as start address and memoryId, and this API must be called after the iap_api_init API.

Prototype :

```
status_t iap_mem_write(api_core_context_t *coreCtx, uint32_t start, uint32_t lengthInBytes, const
uint8_t *buf, uint32_t memoryId)
```

Table 55. Parameters used in iap_mem_write API

Parameter	Description
coreCtx	Pointer to IAP API core context structure.
start	Points to the program address.
lengthInBytes	Program data size in bytes. If the data size is less than 512 bytes or not an even multiple of 512 bytes, then the iap_mem_flush command must be called to write the remaining bytes.
buf	Pointer to source buffer data that will be programed.
memoryId	Points to the memory id of the different memory regions.

Table continues on the next page...

Table 55. Parameters used in iap_mem_write API (continued)

Parameter	Description

	#define kMemoryID_Internal (0x0U) //!< Internal FLASH/RAM ID
	#define kMemoryID_FFR (0x5U) //!< FFR region ID
	#define kMemoryID_FlexspiNor (0x9U) //!< FlexSPI NOR ID

Example1:

```
uint32_t start1 = 0x10000 ;
uint8_t programBuffer[512];
for (uint32_t i=0; i<sizeof(programBuffer); i++)
{
    programBuffer[i] = (uint8_t)(i & 0xFF);
}
uint32_t status = IAP_API_TREE->mem_write(&apiCoreCtx, start1, sizeof(programBuffer), (uint8_t *)
programBuffer, 0);
```

Example2:

```
uint32_t start2 = 0x80001000 ;
uint8_t programBuffer[512];
for (uint32_t i=0; i<sizeof(programBuffer); i++)
{
    programBuffer[i] = (uint8_t)(i & 0xFF);
}
uint32_t status = IAP_API_TREE->mem_write(&apiCoreCtx, start2, sizeof(programBuffer), (uint8_t *)
programBuffer, 9);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the programBuffer data has been programed into the FLASH region in the example1 and programed into the external FlexSPI FLASH in the example2.

NOTE

- Before calling the mem_write API, the internal flash should be erased first by calling the iap_mem_erase.
- Before calling the mem_write API to program the data into the external FlexSPI FLASH, the external flash should be configured first by calling the iap_mem_config.
- If the programBufferis not 512 alignment or program the data into the ffr region, need to call the iap_mem_flush to program the last less than 512 data into the memory region.

NOTE

If the iap_mem_write used for the FlexSPI Flash data program, the program data should be padded page-alignment to use the iap_mem_flush API to load the expect program data.

6.2.6.7 iap_mem_fill

The API is used for filling the specified pattern data into the memory region with the specific data length. This API is mainly for filling the FLASH configuration option related words into the internal RAM. This API must be called after the iap_api_init API.

Prototype :

```
status_t iap_mem_fill(api_core_context_t *coreCtx, uint32_t start, uint32_t lengthInBytes, uint32_t
pattern, uint32_t memoryId)
```

Table 56. Parameters used in iap_mem_fill API

Parameter	Description
coreCtx	Pointer to IAP API core context structure.
start	Points to the program address.
lengthInBytes	Points to the program data size. Date size must be a multiple of 4 because the pattern is 32-bit.
pattern	Points to 32-bit data pattern that will be programed.

memoryId	Points to the memory id of the different memory regions.
	#define kMemoryID_Internal (0x0U) //!< Internal FLASH/RAM ID
	#define kMemoryID_FFR (0x5U) //!< FFR region ID
	#define kMemoryID_FlexspiNor (0x9U) //!< FlexSPI NOR ID

Example:

```
uint32_t start = 0x10000 ;
uint32_t pattern = 0xa5a55a5a;
uint32_t status = IAP_API_TREE->mem_fill(&apiCoreCtx, start, 0x2000, pattern, 0);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the pattern data has been programed into the FLASH region (0x10000 – 0x12000).

6.2.6.8 iap_mem_erase

The API is used for erasing the internal FLASH region, FLEXSPI FLASH region via the memory interface. This API should be called after the iap_api_init API.

Prototype :

```
status_t iap_mem_erase(api_core_context_t *coreCtx, uint32_t start, uint32_t lengthInBytes, uint32_t
memoryId)
```

Table 57. Parameters used in iap_mem_erase API

Parameter	Description
coreCtx	Pointer to IAP API core context structure.
start	Points to the erase address.
lengthInBytes	Points to the erase size. The erase size must be 8KB alignment.

Table continues on the next page...

Table 57. Parameters used in iap_mem_erase API (continued)

Parameter	Description

memoryId	Points to the memory id of the different memory regions.
	#define kMemoryID_Internal (0x0U) //!< Internal FLASH/RAM ID
	#define kMemoryID_FFR (0x5U) //!< FFR region ID
	#define kMemoryID_FlexspiNor (0x9U) //!< FlexSPI NOR ID

Example:

```
uint32_t start = 0x10000 ;
uint32_t lengthInBytes = 0x2000;
uint32_t status = IAP_API_TREE->mem_erase(&apiCoreCtx, start, lengthInBytes, 0);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the FLASH region (0x10000 – 0x12000) has be erased.

6.2.6.9 iap_mem_erase_all

The API is used for erasing the entire internal FLASH or FLEXSPI FLASH based on the memoryId parameter. This API should be called after the iap_api_init API.

Prototype :

```
status_t iap_mem_erase_all(api_core_context_t *coreCtx, uint32_t memoryId)
```

Table 58. Parameters used in iap_mem_erase_all API

Parameter	Description
coreCtx	Pointer to IAP API core context structure.

memoryId	Points to the memory id of the different memory regions.
	#define kMemoryID_Internal (0x0U) //!< Internal FLASH/RAM ID
	#define kMemoryID_FlexspiNor (0x9U) //!< FlexSPI NOR ID

Example:

```
uint32_t status = IAP_API_TREE->mem_erase_all(&apiCoreCtx, 0);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the FLASH region has be erased all.

6.2.6.10 iap_mem_config

The API is used for configuring the PRINCE for internal FLASH, IPED for the FlexSPI FLASH; it is also used for configuring the FLexSPI FLASH and enabling the dual image start address synchronization feature. The SWAP flash feature updates the dual image without considering whether the current active image is running in the bank0 or bank1. This API should be called after the iap_api_init API.

Prototype :

```
status_t iap_mem_config(api_core_context_t *coreCtx, uint32_t *config, uint32_t memoryId)
```

Table 59. Parameters used in iap_mem_config API

Parameter	Description
coreCtx	Pointer to IAP API core context structure.
config	Pointer to the configuration data which will be used for configuring the memory region with the specific encrypt feature based on the memory id. See IPED region configuration parameters and ISP commands to enable PRINCE.

memoryId	Points to the memory id of the different memory regions.
	#define kMemoryID_Internal (0x0U) //!< Internal FLASH/RAM ID
	#define kMemoryID_FlexspiNor (0x9U) //!< FlexSPI NOR ID

Example 1:

```
prince_prot_region_arg_t flashConfigOptionPrince = { .option = { .tag = 0x50 }, .start =
0x00010000, .length = 0x2000 };
uint32_t status = IAP_API_TREE->mem_config(&apiCoreCtx, (uint32_t *)&flashConfigOptionPrince, 0);
```

Example 2:

```
#define FLASH_OPTION_QSPI_SDR 0xc0000001
flexspi_iped_region_arg_t flashConfigOptionIped = { .option = { .tag = 0x49 }, .start =
0x80001000, .end = 0x80001800 };
uint32_t status = IAP_API_TREE->mem_config(&apiCoreCtx, (uint32_t *)& flashConfigOptionIped, 9);
serial_nor_config_option_t flashConfigOption = { .option0 = { .U = FLASH_OPTION_QSPI_SDR } };
status = IAP_API_TREE->mem_config(&apiCoreCtx, (uint32_t *)& flashConfigOption, 9);
```

Example 3:

```
typedef struct
{
    uint32_t reserved : 24;
    uint32_t tag : 8; // Fixed to 0x52 ('R')
} dual_image_update_option_t;
dual_image_update_option_t dualImgUpdateOption = { .tag = 0x52 };
uint32_t status = IAP_API_TREE->mem_config(&apiCoreCtx, (uint32_t *)& dualImgUpdateOption, 0);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means:

- The internal FLASH region 0x10000 – 0x12000 has been enable the PRINCE feature
- The external FlexSPI FLASH has been configured with the QSPI SDR mode and enable the region 0x80001000 – 0x80001800 IPED feature.
- The dual image start address synchronization feature has been enabled when using the dual image boot feature, the access address will be processed with the remap offset to access the actual address.

6.2.6.11 iap_mem_flush

The API is used after iap_mem_write for flushing the data stored in the memory buffer into the destination memory region. This API should be called after the iap_api_init API.

Prototype :

```
status_t iap_mem_flush(api_core_context_t *coreCtx);
```

Table 60. Parameters used in iap_mem_flush API

Parameter	Description
coreCtx	Pointer to IAP API core context structure.

Example:

```
uint32_t cfpastart = 0x0100_0000 ;
uint8_t cfpaBuffer[512];
for (uint32_t i=0; i<sizeof(cfpaBuffer); i++)
{
    cfpaBuffer [i] = (uint8_t)(i & 0xFF);
}
uint32_t status = IAP_API_TREE->mem_write(&apiCoreCtx, cfpastart, sizeof(cfpaBuffer), (uint8_t *)
cfpaBuffer, 5);
if (status != kStatus_Success)
{
    return status;
}
uint32_t status = IAP_API_TREE->mem_flush(&apiCoreCtx);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the cfpaBuffer data has been programed into the CFPA region. In the process of the FFR program and when the data of programed into the internal and external FLASH region is not page-align, user need to call the iap_mem_flush after the iap_mem_write to store the data into the actual memory region.

6.2.6.12 api_sbloader_init

This API is used for initializing the sbloader state machine before calling the api_sbloader_pump. This API should be called after the iap_api_init API.

Prototype :

```
status_t api_sbloader_init(api_core_context_t *ctx);
```

Table 61. Parameters used in api_sbloader_init API

Parameter	Description
ctx	Pointer to IAP API core context structure.

Example:

```
uint32_t status = IAP_API_TREE-> sbloader_init (&apiCoreCtx);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the sbloader runtime environment has been initialized successfully.

6.2.6.13 api_sbloader_pump

This API is used for handling the secure binary (SB3.1 format) data stream, which is used for image update, lifecycle advancing, etc. This API should be called after the iap_api_init and api_sbloader_init APIs.

Prototype :

```
status_t api_sbloader_pump(api_core_context_t *ctx, uint8_t *data, uint32_t length);
```

Table 62. Parameters used in api_sbloader_pump API

Parameter	Description
ctx	Pointer to IAP API core context structure.
data	Pointer to source data that is the sb file buffer data.
length	Points to the size of the process buffer data.

NOTE

If the api_sbloader_pump is used for the FlexSPI Flash data program, the program data should be padded to page-alignment and add additional few data (one Byte) to load the expect program data successfully.

If the api_sbloader_pump is used for internal flash operation, both the flash prefetch buffer and cache should be disabled before calling this API.

Example:

```
/* load the SB file chunk-by-chunk into chunk_processing_buffer */

get_chunk_from_SB_file(uint8_t * chunk_processing_buffer, uint8_t * SBfile_incoming_chunk, uint32_t
chunk_size);
status = IAP_API_TREE-> sbloader_pump(context, & chunk_processing_buffer, chunk_size);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the sb data has been processed successfully.

6.2.6.14 api_sbloader_finalize

The API is used for finalizing the sbloader operations.

Prototype :

```
status_t api_sbloader_finalize(api_core_context_t *ctx)
```

Table 63. Parameters used in api_sbloader_finalize API

Parameter	Description
ctx	Pointer to IAP API core context structure.

Example:

```
status = IAP_API_TREE-> sbloader_finalize (context);
```

See the possible status codes in Table 64. If the status is kStatus_Success return value, it means the last sb data has programed successfully.

6.2.6.15 Possible return codes for IAP driver APIs

Table 64. Return and Error codes for IAP APIs

Error Code	Value	Description
kStatus_IAP_Success	0	IAP API execution succeeded
kStatus_IAP_Fail	1	IAP API execution failed
kStatus_IAP_InvalidArgument	100001	Invalid argument detected during API execution
kStatus_IAP_OutOfMemory	100002	The Heap size is not enough during API execution
kStatus_IAP_ReadDisallowed	100003	The read memory operation is disallowed during API execution
kStatus_IAP_CumulativeWrite	100004	The FLASH region to be programmed is not empty
kStatus_IAP_EraseFailure	100005	Erase operation failed
kStatus_IAP_CommandNotSupported	100006	The specific command is not supported
kStatus_IAP_MemoryAccessDisabled	100007	Memory access is disabled, typically occurred on the FLEXSPI NOR if it is not configured properly
kStatus_Success	0	Operation succeeded without error.
kStatus_Fail	1	The operation failed with a generic error.
kStatus_Invalidargument	4	The requested command's argument is undefined.
kStatusRomLdrSectionOverrun	10100	means reached the end of the sb file processing
kStatusRomLdrSignature	10101	the signature or version are incorrect
kStatusRomLdrSectionLength	10102	the bootOffset/ new section count is out of range
kStatusRomLdrEOFReached	10104	the end of the image file is reached
kStatusRomLdrChecksum	10105	The checksum of a command tag block is invalid.

Table continues on the next page...

Table 64. Return and Error codes for IAP APIs (continued)

Error Code	Value	Description
kStatusRomLdrCrc32Error	10106	The CRC-32 of the data for a load command is incorrect.
kStatusRomLdrUnknownCommand	10107	An unknown command was found in the SB file.
kStatusRomLdrIdNotFound	10108	There was no bootable section found in the SB file.
kStatusRomLdrDataUnderrun	10109	The SB state machine is waiting for more data.
kStatusRomLdrJumpReturned	10110	The function that was jumped to by the SB file has returned.
kStatusRomLdrCallFailed	10111	The call command in the SB file failed.
kStatusRomLdrKeyNotFound	10112	A matching key was not found in the SB file's key dictionary to unencrypt the section.
kStatusRomLdrSecureOnly	10113	The SB file sent is unencrypted, and security on the target is enabled.
kStatusRomLdrResetReturned	10114	The SB file reset operation has unexpectedly returned.
kStatusRomLdrRollbackBlocked	10115	An Image version rollback event detected
kStatusRomLdrInvalidSectionMacCount	10116	Invalid Section MAC count detected in the SB file
kStatusRomLdrUnexpectedCommand	10117	The command tag in the sb-file is unexpected
kStatusRomLdrBadSBKEK	10118	Bad SBKEK detected

6.2.6.16 IAP API examples

The section will provide the example to introduce All the IAP APIs usage and help users understand and use more deeply.

Example1 (program the image or specific data into internal Prince enable region and update ffr region):

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define IAP_API_TREE ((iap_api_interface_t*)ROM_API_TREE[13])
#define FLASH_BUFFER_LEN 0x500
#define CFPA_SCRATCH_ADDRESS 0x3dc00U
#define CMPA_ADDRESS 0x3e200U
#define CMPA_KEY_STORE_ADDRESS 0x3e400U
#define FLASH_PROGRAM_ADDRESS 0x10000;
#define FLASH_ERASE_LEN 0x2000;
#define flashMemId 0
#define ffrMemId 5

uint32_t s_buffer[FLASH_BUFFER_LEN];
for (uint32_t i = 0; i < FLASH_BUFFER_LEN; i++)
{
    s_buffer[i] = i;
}

uint32_t IAPVersion = IAP_API_TREE->version;
```

```
debug_printf("IAPVersion = %x\r\n", IAPVersion);
uint32_t status = kStatus_Success;
api_core_context_t apiCoreCtx;

/* User needs to provide the apiInitParam, which is used as the RAM region for the IAP APIs call */

const kp_api_init_param_t apiInitParam = { .allocStart = 0x30010000, .allocSize = 0x6000 };
status = IAP_API_TREE->api_init(&apiCoreCtx, &apiInitParam);

debug_printf("IAP API api_init return status=%d!\n",status);
if (status != kStatus_Success)
{
    return status;
}

/* config the internal flash region 0x10000 - 0x12000 with the prince feature to encrypt the data in
the region */

prince_prot_region_arg_t flashConfigOptionPrince = { .option = { .tag = 0x50 }, .start =
FLASH_PROGRAM_ADDRESS, .length = FLASH_ERASE_LEN };
status = IAP_API_TREE->mem_config(&apiCoreCtx, (uint32_t *)&flashConfigOptionPrince, flashMemId);
debug_printf("iap_mem_config prince status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

status = IAP_API_TREE->mem_erase(&apiCoreCtx, FLASH_PROGRAM_ADDRESS, FLASH_ERASE_LEN, flashMemId);

debug_printf("iap_mem_erase return status=%d!\n",status);
if (status != kStatus_Success)
{
    return status;
}
status = IAP_API_TREE->mem_write(&apiCoreCtx, FLASH_PROGRAM_ADDRESS, sizeof(s_buffer), (uint8_t
*)s_buffer, flashMemId);
debug_printf("iap_mem_write status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* If load data is not 512B-aligned, user needs to call mem_flush to program the last remaining data
that is not 512B-aligned */

status = IAP_API_TREE->mem_flush(&apiCoreCtx);
debug_printf("iap_mem_flush status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* Here CfpaData is provided by the user and the version should be larger than the current cfpa
version */

status = IAP_API_TREE->mem_write(&apiCoreCtx, CFPA_SCRATCH_ADDRESS, sizeof(CfpaData), (uint8_t
*)&CfpaData, ffrMemId);
debug_printf("iap_mem_write cfpa data return status = %d\n", status);
if (status != kStatus_Success)
{
```

```
    return status;
}

/* User should always call mem_flush after mem_write to program the ffr data */

status = IAP_API_TREE->mem_flush(&apiCoreCtx);
debug_printf("iap_mem_flush cfpa data status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* Here the CmpaData is provided by the user */

status = iap_mem_write(&apiCoreCtx, CMPA_ADDRESS, sizeof(CmpaData), (uint8_t *)&CmpaData, ffrMemId);
debug_printf("iap_mem_write status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* User should always call mem_flush after mem_write to program the ffr data */

status = IAP_API_TREE->mem_flush(&apiCoreCtx);
debug_printf("iap_mem_flush cfpa data status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* Here the CMPA_KeyStore is provided by the user */

status = IAP_API_TREE->mem_write(&apiCoreCtx, CMPA_KEY_STORE_ADDRESS, sizeof(CMPA_KeyStore), (uint8_t
*)CMPA_KeyStore, ffrMemId);
debug_printf("iap_mem_write cmpa key_store status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* User should always call mem_flush after mem_write to program the ffr data */

status = IAP_API_TREE->mem_flush(&apiCoreCtx);
debug_printf("iap_mem_flush cmpa key_store status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}
status = iap_api_deinit(&apiCoreCtx);
debug_printf("iap_api_deinit status = %x!\n", status);
if (status != kStatus_Success)
{
    return status;
}
```

Example2 (FlexSPI Flash with IPED):

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define IAP_API_TREE ((iap_api_interface_t*)ROM_API_TREE[13])
```

```
#define FLASH_OPTION_SEL 0xc0000001     /* Choose the QSPI SDR FlexSPI Flash option */
#define FLEXSPI_BASE_ADDR (0x80000000u) /* Flash Base Addresss for AHB access */
#define FLEXSPI_START_OFFSET (0x1000u)  /* Test address for IP access */
#define FLEXSPI_START_ADDR (FLEXSPI_BASE_ADDR + FLEXSPI_START_OFFSET) /* Test addresss for AHB access
*/
#define FLEXSPI_ERASE_SIZE (0x2000u)   /* Erase size in the demo */

uint32_t status = kStatus_Success;

api_core_context_t apiCoreCtx;

/* User needs to provide apiInitParam, which is used as the RAM region for the IAP APIs call */

const kp_api_init_param_t apiInitParam = { .allocStart = 0x30010000, .allocSize = 0x6000 };
status = IAP_API_TREE->api_init(&apiCoreCtx, &apiInitParam);

debug_printf("IAP API api_init return status=%d!\n",status);
if (status != kStatus_Success)
{
    return status;
}

/* Configure the FlexSPI flash region 0x80001000 - 0x80001800 with IPED feature */

uint32_t flexspiMemId = 9;
flexspi_iped_region_arg_t flashConfigOptionIped = { .option = { .tag = 0x49 }, .start =
0x80001000, .end = 0x80001800 };
status = IAP_API_TREE->mem_config(&apiCoreCtx, (uint32_t *)&flashConfigOptionIped, flexspiMemId);
debug_printf("iap_mem_config iped status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* Configure the FlexSPI flash device with the selected option */

serial_nor_config_option_t flashConfigOption = { .option0 = { .U = FLASH_OPTION_SEL}};
status = IAP_API_TREE->mem_config(&apiCoreCtx, (uint32_t *)&flashConfigOption, flexspiMemId);
debug_printf("iap_mem_config status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

status = IAP_API_TREE->mem_erase(&apiCoreCtx, FLEXSPI_START_ADDR, FLEXSPI_ERASE_SIZE, flexspiMemId);
debug_printf("iap_mem_erase returned status:%d\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* Here the programBuffer data can also be replaced by the image data for app to boot in the FlexSPI
Flash boot mode */
uint32_t programBuffer[FLEXSPI_ERASE_SIZE / sizeof(uint32_t)];
uint8_t *pBuffer = (uint8_t *)&programBuffer[0];
for (uint32_t i = 0; i < sizeof(programBuffer); i++)
{
    *pBuffer++ = (uint8_t)(i & 0xFF);
}
status = IAP_API_TREE->mem_write(&apiCoreCtx, FLEXSPI_START_ADDR, sizeof(programBuffer),
```

```
(const uint8_t *)programBuffer, flexspiMemId);
debug_printf("iap_mem_write returned status:%d\n", status);
if (status != kStatus_Success)
{
    return status;
}
status = iap_api_deinit(&apiCoreCtx);
debug_printf("iap_api_deinit status = %x!\n", status);
if (status != kStatus_Success)
{
    return status;
}
```

Example3 (update the dual-image into the internal flash):

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define IAP_API_TREE ((iap_api_interface_t*)ROM_API_TREE[13])
#define FLASH_PROGRAM_ADDRESS 0x0;
#define FLASH_ERASE_LEN 0x2000;
#define flashMemId 0

typedef struct
{
    uint32_t reserved : 24;
    uint32_t tag : 8; // Fixed to 0x52 ('R')
} dual_image_update_option_t;

uint32_t status = kStatus_Success;

api_core_context_t apiCoreCtx;

/* User needs to provide apiInitParam, which is used as the RAM region for the IAP APIs call */

const kp_api_init_param_t apiInitParam = { .allocStart = 0x30010000, .allocSize = 0x6000 };
status = IAP_API_TREE->api_init(&apiCoreCtx, &apiInitParam);

debug_printf("IAP API api_init return status=%d!\n",status);
if (status != kStatus_Success)
{
    return status;
}

/* Configure the dual-image update into specific lower version image region */

dual_image_update_option_t dualImgUpdateConfigOption = { .tag = 0x52 };
uint32_t flashMemId = 0x0U;
status = iap_mem_config(&apiCoreCtx, (uint32_t *)&dualImgUpdateConfigOption, flashMemId);
debug_printf("iap_mem_config status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* The erase address will be processed with the remap offset to access the actual address; if the
lower version image is stored in the 0x0, the erase address will be 0; if the lower version image is
stored in the remap offset region, the erase address will be 0x0 + remap_offset; User does not need
to set the erase address, which is just processed by API. */
```

```
status = IAP_API_TREE->mem_erase(&apiCoreCtx, FLASH_PROGRAM_ADDRESS, FLASH_ERASE_LEN, flashMemId);

debug_printf("iap_mem_erase return status=%d!\n",status);
if (status != kStatus_Success)
{
    return status;
}

/* The write address will be processed with the remap offset to access the actual address; if the
lower version image is stored in the 0x0, the write address will be 0; if the lower version image is
stored in the remap offset region, the erase address will be 0x0 + remap_offset. User does not need
to set the write address, which is just processed by API. */

status = IAP_API_TREE->mem_write(&apiCoreCtx, FLASH_PROGRAM_ADDRESS, sizeof(s_buffer), (uint8_t
*)s_buffer, flashMemId);
debug_printf("iap_mem_write status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}

/* If the load data is not 512B-aligned, user needs to call mem_flush to program the last remaining
data that is not 512B-aligned */

status = IAP_API_TREE->mem_flush(&apiCoreCtx);
debug_printf("iap_mem_flush status = %d\n", status);
if (status != kStatus_Success)
{
    return status;
}
```

6.2.6.17 SBloader APIs

This section uses the SBloader APIs demo to introduce the SBloader APIs usage and helps the user to know how to use the APIs. In addition, the section uses the practical application scenarios to introduce the convenience, flexibility, and practicality of the SBloader APIs. The flow diagram of the SBloader APIs call demo is as follow:

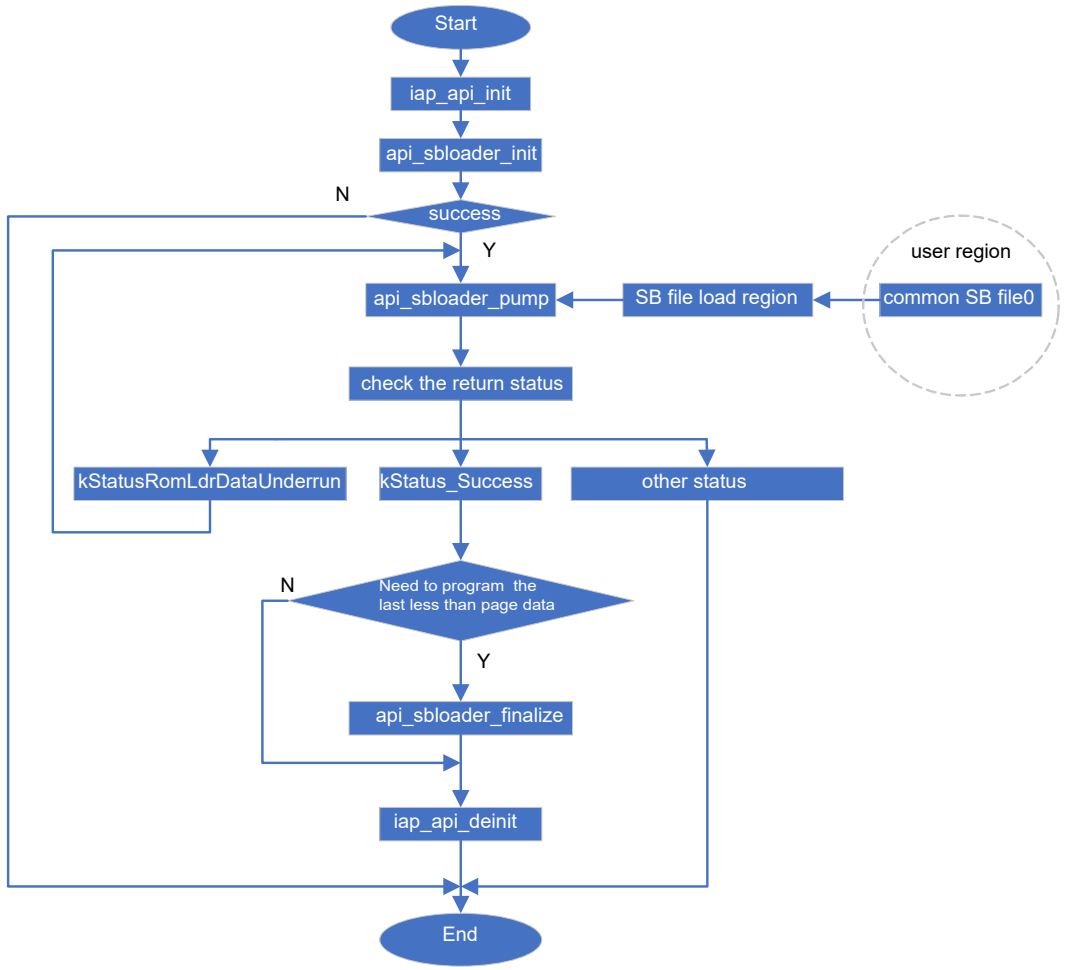


Figure 9. Flow diagram of the SBloader APIs call demo

The above flow diagram mainly introduces the SBloader APIs call flow. When the valid SB3.1 file has been generated and the necessary running environment(ROKTH and KeyBlob) has been setup up, the mainly details can be referred to the SB file chapter. From the flow diagram, the iap_api_init will be called first to initialize and configure the IAP API core context. And then call the api_sbloader_init to initialize the loader state machine. If the api_sbloader_init return status is success, then the spi_sbloader_pump will be called to process the SB file data stream, which is transferred from the user region (internal flash region, external flash region and RAM region apart from the reserved RAM region, list in the following).

1: 0x20008000 - 0x2000ffff

2: 0x30008000 - 0x3000ffff

3: 0x20010000 - 0x20017fff

4: 0x30010000 - 0x30017fff

5: 0x20060000 - 0x20067fff

6: 0x30060000 - 0x30067fff

In the running process of the api_sbloader_pump function, the return status includes three kinds of the results, which can be referred to in Table 64.Return and Error codes for IAP APIs. If the return status is kStatusRomLdrDataUnderrun, it seems that more data in the common SB_file0 should be transferred to the api_sbloader_pump to process, and the process has not reached the end of the SB file data. When the kStatus_Success has been received, it seems that common SB_file0 has been

completely received and processed so as to update the firmware in the SB_file0 successfully. The other status will be thought that api_sbloader_pump API failed to execute. and the iap_api_deinit will be executed to clear the IAP APIs context parameters.

In addition, in the process of executing the SB file in the api_sbloader_pump, only the load address of the SB file data will be transferred to the api_sbloader_pump, the data of the SB file is executed with the packet chunk. Users should get and load the packet chunk of the SB file into the ram region like the following function get_data_from_sbfile(), and then call the api_sbloader_pump to process the ram data repeatedly until to the end of the SB file. In transferring the SB file chunk to the device, the user can use different channels to receive the chunk data like UART, SPI, I2C, USB, OTP interface.

Example:

```
#define ROM_API_TREE ((uint32_t*)0x1303fc00)
#define IAP_API_TREE ((iap_api_interface_t*)ROM_API_TREE[13])
api_core_context_t apiCoreCtx = { .flashConfig.modeConfig.sysFreqInMHz = 96 };
const kp_api_init_param_t apiInitParam = { .allocStart = 0x30010000, .allocSize = 0x6000 };
uint32_t status = IAP_API_TREE->api_init(&apiCoreCtx, &apiInitParam);
if (status != kStatus_Success)
{
    return status;
}
uint32_t status = IAP_API_TREE->sbloader_init(&apiCoreCtx);
if (status != kStatus_Success)
{
    return status;
}
uint8_t user_buf[1024];
uint32_t size;
while ((status = get_data_from_sbfile(&buffer, &size)) == kStatus_Success)
{
    status = IAP_API_TREE->sbloader_pump(context, &buffer, size);
    if (status == kStatusRomLdrDataUnderrun)
    {
        /* continue to receive the data from SB file for processing */
        debug_printf("Warning: Success to execute ROM API erase, but need more data \n");
    }
    else if (status == kStatus_Success)
    {
        debug_printf("Completed: Success to execute ROM API erase\n");
        break;   /* finish the SB file firewarre update and break */
    }
    else if (status != kStatus_Success)
    {
        debug_printf("Error: Failed to run API kb_execute, status is hex(%x) \n", (uint32_t)status);
        return status;   /* return the fail status or other actions */
    }
}
status = IAP_API_TREE->sbloader_finalize (context);
if (status != kStatus_Success)
{
    debug_printf("Error: Failed to sbloader_finalize\n");
    return status;
}
```

6.2.7 NBOOT APIs

6.2.7.1 Functional description

These NBOOT function addresses can be called using function pointers. The main purpose of these APIs is to provide access to functions used and implemented in ROM to generate random number and authenticate the application image. NBOOT SB3.1 API functions also provides possibility to verify signature of SB3.1 header (manifest) and decrypt individual data blocks without need to process entire SB file like in case of using SBloader API described in IAP APIs section.

6.2.7.2 NBOOT data structures description

6.2.7.2.1 nboot_interface_t

```
typedef struct
{
romapi_status_t (*romapi_rng_generate_random)(uint8_t *output, size_t outputByteLen);
nboot_status_t (*nboot_context_init)(nboot_context_t *context);
nboot_status_t (*nboot_context_deinit)(nboot_context_t *context);
nboot_status_protected_t (*nboot_sb3_load_manifest)(nboot_context_t *context,
uint32_t *manifest,
nboot_sb3_load_manifest_parms_t *parms);
nboot_status_protected_t (*nboot_sb3_load_block)(nboot_context_t *context,
uint32_t *block);
nboot_status_protected_t (*nboot_img_authenticate_ecdsa)(nboot_context_t *context,
uint8_t imageStartAddress[],
nboot_bool_t *isSignatureVerified,
nboot_img_auth_ecdsa_parms_t *parms);
nboot_status_protected_t (*nboot_img_authenticate_cmac)(nboot_context_t *context,
uint8_t imageStartAddress[],
nboot_bool_t *isSignatureVerified,
nboot_img_authenticate_cmac_parms_t *parms);
} nboot_interface_t;
```

6.2.7.2.2 nboot_context_t

```
typedef struct _nboot_context
{
uint32_t buffer[0x2e8/sizeof(uint32_t)];
} nboot_context_t;
```

6.2.7.2.3 nboot_root_key_revocation_t

```
#define kNBOOT_RootKey_Enabled (0xAAu)
#define kNBOOT_RootKey_Revoked (0xBBu)
/* any other value means the root key is revoked */
typedef uint32_t nboot_root_key_revocation_t;
```

6.2.7.2.4 nboot_root_key_usage_t

```
/*! @brief NBOOT type for the root key usage. This type defines the NBOOT root key usage */
#define kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey (0x0u)
#define kNBOOT_RootKeyUsage_DebugCA (0x1u)
#define kNBOOT_RootKeyUsage_ImageCA_FwCA (0x2u)
#define kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA (0x3u)
#define kNBOOT_RootKeyUsage_ImageKey_FwKey (0x4u)
#define kNBOOT_RootKeyUsage_ImageKey (0x5u)
#define kNBOOT_RootKeyUsage_FwKey (0x6u)
#define kNBOOT_RootKeyUsage_Unused (0x7u)
/* any other value means the root key is not valid (treat as if revoked) */
typedef uint32_t nboot_root_key_usage_t;
```

6.2.7.2.5 nboot_root_key_type_and_length_t

```
#define kNBOOT_RootKey_Ecdsa_P256 (0x0000FE01u)
#define kNBOOT_RootKey_Ecdsa_P384 (0x0000FD02u)
typedef uint32_t nboot_root_key_type_and_length_t;
```

6.2.7.2.6 nboot_soc_lifecycle_t

```
/*! @brief Enumeration for SoC Lifecycle. */
#define nboot_lc_nxpBlank (0xFFFF0000u)
#define nboot_lc_nxpFab (0xFFFE0001u)
#define nboot_lc_nxpDev (0xFF0300FCu)
#define nboot_lc_nxpProvisioned (0xFFFC0003u)
#define nboot_lc_oemOpen (0xFFFC0003u)
#define nboot_lc_oemSecureWorld (0xFFF80007u)
#define nboot_lc_oemClosed (0xFFF0000Fu)
#define nboot_lc_oemLocked (0xFF3000CFu)
#define nboot_lc_oemFieldReturn (0xFFE0001Fu)
#define nboot_lc_nxpFieldReturn (0xFF80007Fu)
#define nboot_lc_shredded (0xFF0000FFu)
typedef uint32_t nboot_soc_lifecycle_t;
```

6.2.7.2.7 nboot_rot_auth_parms_t

```
typedef struct _nboot_rot_auth_parms
{
/* trusted information originated from CFPA */
nboot_root_key_revocation_t soc_rootKeyRevocation[NBOOT_ROOT_CERT_COUNT]; /*!< Provided by caller
based on NVM information in CFPA: ROTKH_REVOKE */
```

```
uint32_t soc_imageKeyRevocation; /*!< Provided by caller based on NVM information in CFPA:
IMAGE_KEY_REVOKE */
/* trusted information originated from CMPA */
uint32_t soc_rkh[12]; /*!< Provided by caller based on NVM information in CMPA:
ROTKH (hash of hashes)*/
/*!< In case of kNBOOT_RootKey_Ecdsa_P384, sock_rkh[0..11] are used */
/*!< In case of kNBOOT_RootKey_Ecdsa_P256, sock_rkh[0..7] are used */
uint32_t soc_numberOfRootKeys; /* unsigned int, between minimum = 1 and maximum = 4; */
nboot_root_key_usage_t soc_rootKeyUsage[NBOOT_ROOT_CERT_COUNT]; /* CMPA */
nboot_root_key_type_and_length_t soc_rootKeyTypeAndLength; /* static selection between ECDSA P-256 or
ECDSA P-384 based root keys */
/* trusted information originated from OTP fuses */
nboot_soc_lifecycle_t soc_lifecycle;
} nboot_rot_auth_parms_t;
```

6.2.7.2.8 nboot_sb3_load_manifest_parms_t

```
typedef struct _nboot_sb3_load_manifest_parms
{
nboot_rot_auth_parms_t soc_RoTNVM; /*! trusted information originated from CFPA and NMPA */
uint32_t soc_trustedFirmwareVersion; /*!< Provided by caller based on NVM information in CFPA:
Secure_FW_Version */
uint8_t custMkSkBlob[48];
} nboot_sb3_load_manifest_parms_t;
```

6.2.7.2.9 nboot_img_auth_ecdsa_parms_t

```
typedef struct _nboot_img_auth_ecdsa_parms
{
/* trusted information originated from CFPA and NMPA */
nboot_rot_auth_parms_t soc_RoTNVM;
uint32_t soc_trustedFirmwareVersion; /*!< Provided by caller based on NVM information in CFPA:
Secure_FW_Version */
} nboot_img_auth_ecdsa_parms_t;
```

6.2.7.2.10 nboot_img_authenticate_cmac_parms_t

```
typedef struct _nboot_cmac_authenticate_parms
{
uint32_t expectedMAC[4]; /*!< expected MAC result */
} nboot_img_authenticate_cmac_parms_t;
```

6.2.7.3 romapi_rng_generate_random

This ROM API function is used to generate random number with specified length.

Prototype :

```
romapi_status_t (*romapi_rng_generate_random)(uint8_t *output, size_t outputByteLen);
```

Table 65. Parameters used in romapi_rng_generate_random API

Parameter	Description
output [in]	Pointer to random number buffer.
outputByteLen [in]	length of generated random number in bytes. Length has to be in range <1, 2^16>.

Example :

```
uint8_t rndBuffer[32];
romapi_status_t status = NBOOT_API_TREE->romapi_rng_generate_random(&rndBuffer[0], 32));
```

6.2.7.4 nboot_context_init

The function is used for initializing of the nboot context data structure. It should be called prior to any other calls of nboot API.

Prototype :

```
nboot_status_t (*nboot_context_init)(nboot_context_t *context);
```

Table 66. Parameters used in nboot_context_init API

Parameter	Description
nbootCtx [in]	Pointer to nboot_context_t structure

Example :

```
nboot_context_t nbootCtx;
nboot_status_t nbootStatus = NBOOT_API_TREE->nboot_context_init(&nbootCtx);
```

6.2.7.5 nboot_context_deinit

The function is used to de-initialize nboot context data structure. Its contents are overwritten with random data so that any sensitive data does not remain in memory.

Prototype :

```
nboot_status_t (*nboot_context_deinit)(nboot_context_t *context);
```

Table 67. Parameters used in nboot_context_deinit API

Parameter	Description
nbootCtx [in]	Pointer to nboot_context_t structure

Example :

```
nboot_context_t nbootCtx;
nboot_status_t nbootStatus = NBOOT_API_TREE->nboot_context_init(&nbootCtx);
...
nbootStatus = NBOOT_API_TREE->nboot_context_deinit(&nbootCtx);
```

6.2.7.6 nboot_sb3_load_manifest

This function verifies the NBOOT SB3.1 manifest (header message), initializes the context and loads keys into the ELS key store so that they can be used by nboot_sb3_load_block function. The NBOOT context has to be initialized by the function nboot_context_init before calling this function. Please note that this API is intended to be used only by users who needs to split FW update process (loading of SB3.1 file) to partial steps to customize whole operation. For regular SB3.1 processing, please use API described in chapter “SBloader APIs”.

Prototype :

```
nboot_status_protected_t (*nboot_sb3_load_manifest)(nboot_context_t *context,
uint32_t *manifest,
nboot_sb3_load_manifest_parms_t *parms);
```

Table 68. Parameters used in nboot_sb3_load_manifest API

Parameter	Description
nbootCtx [in]	Pointer to nboot_context_t structure
manifest [in]	Pointer to the input manifest buffer
parms [in]	Additional input parameters. Please refer to nboot_sb3_load_manifest_parms_t definition for details.

Example :

```
#define ROM_API_TREE ((uint32_t*)0x1303fc 00)
#define NBOOT_API_TREE ((nboot_interface_t *)ROM_API_TREE[10])
nboot_context_t nbootCtx;
nboot_status_t nbootStatus = NBOOT_API_TREE->nboot_context_init(&nbootCtx);
nboot_sb3_load_manifest_parms_t parms = {0};
parms.soc_trustedFirmwareVersion = 0x0;
parms.soc_RoTNVM.soc_rootKeyRevocation[0] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_rootKeyRevocation[1] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_rootKeyRevocation[2] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_rootKeyRevocation[3] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_imageKeyRevocation = 0x0;
memcpy(parms.soc_RoTNVM.soc_rkh, testRkh384, sizeof(testRkh384));
parms.soc_RoTNVM.soc_rootKeyUsage[0] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_rootKeyUsage[1] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_rootKeyUsage[2] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_rootKeyUsage[3] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
```

```
parms.soc_RoTNVM.soc_numberOfRootKeys = 1;
parms.soc_RoTNVM.soc_rootKeyTypeAndLength = kNBOOT_RootKey_Ecdsa_P256;
parms.soc_RoTNVM.soc_lifecycle = SYSCON->ELS_AS_CFG0 & 0x000000FFu;
parms.soc_RoTNVM.soc_lifecycle = (((~parms.soc_RoTNVM.soc_lifecycle) << 16) & 0xFFFF0000u) |
(parms.soc_RoTNVM.soc_lifecycle & 0x0000FFFFu);
nbootStatus = (nboot_status_t)(NBOOT_API_TREE->nboot_sb3_load_manifest(nbootCtx, &sbManifest[0],
&parms));
if (status != kStatus_NBOOT_Success)
{
PRINTF("Error loading manifest.");
}
else
{
PRINTF ("SB3 manifest authentication PASSED!");
}
```

6.2.7.7 nboot_sb3_load_block

This function verifies and decrypts an NBOOT SB3.1 block. Decryption is performed in-place. The NBOOT context has to be initialized by the function nboot_sb3_load_manifest() before calling this function. Please note that this API is intended to be used only by users who needs to split FW update process (loading of SB3.1 file) to partial steps to customize whole operation. For regular SB3.1 processing, please use API described in chapter “SBloader APIs”.

Prototype :

```
nboot_status_protected_t (*nboot_sb3_load_block)(nboot_context_t *context, uint32_t *block);
```

Table 69. Parameters used in nboot_sb3_load_block API

Parameter	Description
nbootCtx [in]	Pointer to nboot_context_t structure
block [in]	Pointer to the input SB3.1 data block

Example :

```
#define ROM_API_TREE ((uint32_t*)0x1303fc 00)
#define NBOOT_API_TREE ((nboot_interface_t *)ROM_API_TREE[10])
nboot_context_t nbootCtx;
nboot_status_t nbootStatus = NBOOT_API_TREE->nboot_context_init(&nbootCtx);
nboot_sb3_load_manifest_parms_t parms = {0};
parms.soc_trustedFirmwareVersion = 0x0;
parms.soc_RoTNVM.soc_rootKeyRevocation[0] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_rootKeyRevocation[1] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_rootKeyRevocation[2] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_rootKeyRevocation[3] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_imageKeyRevocation = 0x0;
memcpy(parms.soc_RoTNVM.soc_rkh, testRkh384, sizeof(testRkh384));
parms.soc_RoTNVM.soc_rootKeyUsage[0] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_rootKeyUsage[1] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_rootKeyUsage[2] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_rootKeyUsage[3] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_numberOfRootKeys = 1;
parms.soc_RoTNVM.soc_rootKeyTypeAndLength = kNBOOT_RootKey_Ecdsa_P256;
parms.soc_RoTNVM.soc_lifecycle = SYSCON->ELS_AS_CFG0 & 0x000000FFu;
parms.soc_RoTNVM.soc_lifecycle = (((~parms.soc_RoTNVM.soc_lifecycle) << 16) & 0xFFFF0000u) |
(parms.soc_RoTNVM.soc_lifecycle & 0x0000FFFFu);
nbootStatus = (nboot_status_t)(NBOOT_API_TREE->nboot_sb3_load_manifest(nbootCtx, &sbManifest[0],
```

```
&parms));
if (nbootStatus != kStatus_NBOOT_Success)
{
PRINTF("Error loading manifest.");
}
else
{
PRINTF ("SB3 manifest authentication PASSED!");
}
nbootStatus = (nboot_status_t)(nboot_sb3_load_block(ctx, &dataBlock[0]));
if (nbootStatus != kStatus_NBOOT_Success)
{
PRINTF("Error during data block decryption!");
}
else
{
PRINTF("SB3 data block was successfully decrypted!");
}
```

6.2.7.8 nboot_img_authenticate_ecdsa

This function authenticates image with asymmetric cryptography. The NBOOT context has to be initialized by the function nboot_context_init before calling this function.

Prototype :

```
nboot_status_protected_t (*nboot_img_authenticate_ecdsa)(nboot_context_t *context,

uint8_t imageStartAddress[],

nboot_bool_t *isSignatureVerified,

nboot_img_auth_ecdsa_parms_t *parms);
```

Table 70. Parameters used in nboot_img_authenticate_ecdsa API

Parameter	Description
nbootCtx [in]	Pointer to nboot_context_t structure
imageStartAddress [in]	Pointer to start of the image in memory
isSignatureVerified [out]	Pointer to memory holding function call result. After the function returns, the value will be set to kNBOOT_TRUE when the image is authentic. Any other value means the authentication does not pass.
parms [in]	Pointer to a data structure in trusted memory, holding input parameters for the algorithm. The data structure shall be correctly filled before the function call.

Return values :

kStatus_NBOOT_Success - Operation successfully finished

kStatus_NBOOT_Fail - Returned in all other cases. Doesn't always mean invalid image, it could also mean transient error caused by short time environmental conditions.

Example :

```

#define ROM_API_TREE ((uint32_t*)0x1303fc 00)
#define NBOOT_API_TREE ((nboot_interface_t *)ROM_API_TREE[10])
```

```
nboot_context_t nbootCtx;
nboot_status_t nbootStatus = NBOOT_API_TREE->nboot_context_init(&nbootCtx);
nboot_bool_t nres = kNBOOT_FALSE;
nboot_img_auth_ecdsa_parms_t parms;
/* fill in device OTP/NVM data */
parms.soc_trustedFirmwareVersion = 0x0;
parms.soc_RoTNVM.soc_rootKeyRevocation[0] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_rootKeyRevocation[1] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_rootKeyRevocation[2] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_rootKeyRevocation[3] = kNBOOT_RootKey_Enabled;
parms.soc_RoTNVM.soc_imageKeyRevocation = 0x0;
memcpy(parms.soc_RoTNVM.soc_rkh, testRkh, sizeof(testRkh));
parms.soc_RoTNVM.soc_rootKeyUsage[0] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_rootKeyUsage[1] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_rootKeyUsage[2] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_rootKeyUsage[3] = kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey;
parms.soc_RoTNVM.soc_numberOfRootKeys = 4;
parms.soc_RoTNVM.soc_rootKeyTypeAndLength = kNBOOT_RootKey_Ecdsa_P256;
parms.soc_RoTNVM.soc_lifecycle = nboot_lc_nxpProvisioned;
nboot_printf("TEST: nboot_img_authenticate_ecdsa");
nstatus = (nboot_status_t)NBOOT_API_TREE->nboot_img_authenticate_ecdsa(ctx, (uint8_t*)
(uintptr_t)testImage, &nres, &parms);
if (nstatus != kStatus_NBOOT_Success)
{
PRINTF("Image authentication FAILED!\n");
}
else
{
PRINTF ("Image authentication PASSED! \n");
}
if (nres != kNBOOT_TRUE)
{
PRINTF("Image authentication FAILED!\n");
}
```

6.2.7.9 nboot_img_authenticate_cmac

This function calculates the CMAC over the given image and compares it to the expected value. To be more resistant against SPA, it is recommended that imageStartAddress is word aligned. The NBOOT context has to be initialized by the nboot_context_init() before calling this function.

Prototype :

```
nboot_status_protected_t (*nboot_img_authenticate_cmac)(nboot_context_t *context,

uint8_t imageStartAddress[],

nboot_bool_t *isSignatureVerified,

nboot_img_auth_ecdsa_parms_t *parms);
```

Table 71. Parameters used in romapi_rng_generate_random API

Parameter	Description
nbootCtx [in]	Pointer to nboot_context_t structure.
imageStartAddress [in]	Pointer to start of the image in memory.

Table continues on the next page...

Table 71. Parameters used in romapi_rng_generate_random API (continued)

Parameter	Description
isSignatureVerified [out]	Pointer to memory holding function call result. After the function returns, the value will be set to kNBOOT_TRUE when the image is authentic. Any other value means the authentication does not pass.
parms [in]	Pointer to a data structure in trusted memory, holding the reference MAC. The data structure shall be correctly filled before the function call.

Return values :

kStatus_NBOOT_Success - Operation successfully finished

kStatus_NBOOT_Fail - Returned in all other cases. Doesn't always mean invalid image, it could also mean transient error caused by short time environmental conditions.

Example :

```
#define ROM_API_TREE ((uint32_t*)0x1303fc 00)
#define NBOOT_API_TREE ((nboot_interface_t *)ROM_API_TREE[10])
nboot_context_t nbootCtx;
nboot_status_t nbootStatus = NBOOT_API_TREE->nboot_context_init(&nbootCtx);
nboot_bool_t nres = kNBOOT_FALSE;
nboot_img_auth_ecdsa_parms_t parms;
nboot_img_authenticate_cmac_parms_t parms = {0};
memcpy(parms.expectedMAC, &expMac[0], sizeof(parms.expectedMAC));
nbootStatus = (nboot_status_t)NBOOT_API_TREE->nboot_img_authenticate_cmac(ctx, (void*)image, &nres,
&parms);
if ((nbootStatus == kStatus_NBOOT_Success) || (nres == kNBOOT_TRUE))
{
PRINTF("Image authentication PASSED!\n");
}
else
{
PRINTF("Image authentication FAILED!\n");
}
```

6.2.7.10 Possible return codes of NBOOT API

Table 72. Return codes for NBOOT APIs

Error Code	Value	Description
kStatus_NBOOT_InvalidArgument		Invalid input parameters (Input pointers points to NULL or length is invalid)
kStatus_NBOOT_Success		Operation successfully finished
kStatus_NBOOT_Fail		Error occurred during operation

Chapter 7
In-System Programming (ISP)

7.1 Overview

This chip includes ISP functions to support image programming from the serial interface (UART, I^2C, SPI, CAN) and USB HID.

NOTE

This chapter is also available in the *MCX Nx4x* reference manual, with differences related to security functionality. The version of this chapter in the reference manual does not include the security-relevant information.

7.1.1 Features

- Peripheral interfaces:
  - UART
  - I^2C
  - SPI
  - CAN
  - USB
- Automatic detection of the active peripheral
- UART peripheral implements auto-baud detection
- CAN peripheral implements auto-baud detection for predefined baud rates:
  - 1 Mbit/s
  - 500 kbit/s
  - 250 kbit/s
  - 125 kbit/s
- A common packet-based protocol for all peripherals
- Packet error detection and retransmission
- Flash-resident configuration options (in CMPA)
- RAM protection used by the bootloader while it is running
- Retrieval of the device properties, such as flash memory and RAM size
- Multiple options for executing the bootloader, either at system startup or under application control at runtime
- Support for internal flash memory access
- Support for encrypted image downloading
- External flash memory access
- In CMPA, user can use blhost tool command 'write-memory' to program or update the CMPA

7.2 Functional description

7.2.1 Bootloader

ISP stores the boot code in internal ROM. After a reset, the Arm processor starts its code execution from this memory. ISP executes the bootloader code every time the part is powered on, reset, or woken up from a deep power-down, low-power mode.

The bootloader provides a flash memory and OTP-eFuse programming utility that operates over a serial connection on the MCUs. It enables quick and easy programming of MCUs through the entire product lifecycle, including application development, final product manufacturing, and beyond. Host-side command line and GUI tools are available to communicate with the bootloader. Users can use host tools to upload and download application code and to perform manufacturing via the bootloader.

For information on bootloader operation and boot pin, see Overview.

7.2.2 In-system programming (ISP)

Serial booting and other related functions are supported in several different ways:

- ISP protocol
- Bootloader packet types
- Bootloader command set
- UART ISP
- I^2C ISP
- SPI ISP
- USB ISP
- CAN ISP

7.2.3 Memory map after reset

The boot ROM is located in the memory region beginning with the address 1300_0000h. Both the ISP and IAP software use parts of the on-chip RAM. Use of the RAM is described in ISP interrupt and SRAM use.

Based on the DEFAULT_ISP_MODE settings or ISP pin settings, the ROM enters ISP mode and can auto-detect activity on these interfaces:

- CAN
- I^2C
- SPI
- UART
- USB HID

The auto-detect looks for activity on these interfaces and selects the appropriate interface when a properly formed frame is received. If an invalid frame is received, the data is discarded and scanning resumes. Communications for these interfaces are described in In-System programming protocol and Bootloader packet types.

7.2.4 ISP interrupt and SRAM use

7.2.4.1 Interrupts during IAP

When the user application code starts executing, the interrupt vectors from the SRAM are active. Before making an IAP call, you must disable the interrupts. The IAP code does not use or disable interrupts.

7.2.4.2 RAM used by the ISP command handler

ISP reserves the regions designated below for bootloader use when the bootloader is running.

Table 73. RAM used by ROM during ROM execution

RAM region	Purpose
2000_8000h–2001_7FFFh 2001_8000h–2001_9FFFh	• BSS • RW • Stack • Heap 2001_8000h–2001_9FFFh (For the recovery sb loader file handling only)

7.2.5 ISP boot flow

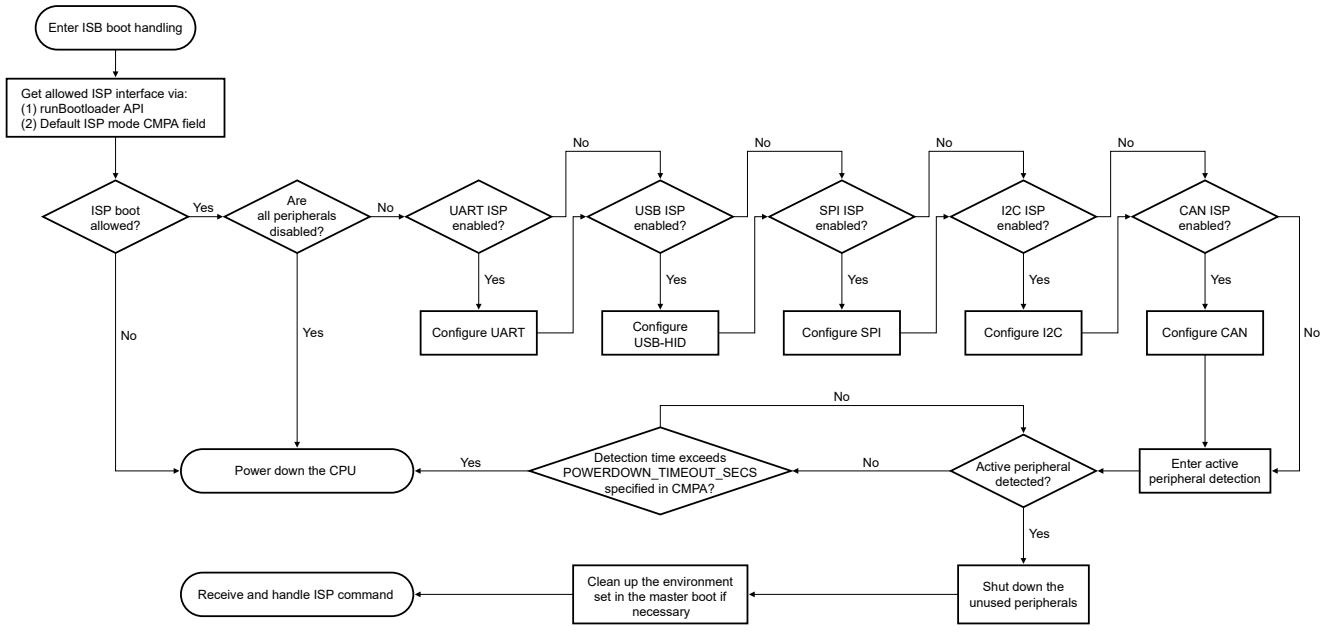


Figure 10. ISP boot flow

7.3 ISP protocol

This section explains the general protocol for packet transfers between the host and the bootloader. The description includes the transfer of packets for different transactions, such as commands with no data phase and commands with an incoming or outgoing data phase. The next section describes various packet types used in a transaction.

Each command sent from the host is replied to with a response command.

Commands can include an optional data phase:

- If the data phase is incoming (from the host to the bootloader), it is part of the original command.
- If the data phase is outgoing (from the bootloader to host), it is part of the response command.

7.3.1 Command with no data phase

The protocol for a command with no data phase contains:

1. Command packet (from the host).

2. GenericResponse command packet (to host).

The ACK sent in response to a command or a data packet can arrive at any time before, during, or after the command or data packet has been processed. See the diagrams in the Bootloader command set section.

7.3.2 Command with incoming data phase

The protocol for a command with incoming data phase contains:

1. Command packet (from host) (kCommandFlag_HasDataPhase set).
2. GenericResponse command packet (to host).
3. Incoming data packets (from the host).
4. GenericResponse command packet to host.

Note:

- The host may not send any further packets while it is waiting for the response to a command.
- The data phase is aborted if the GenericResponse packet, prior to the start of the data phase, does not have a status of kStatus_Success.
- Data phases may be aborted by the receiving side by sending the final GenericResponse early with a status of kStatus_AbortDataPhase. The host may abort the data phase early by sending a zero-length data packet.
- The final GenericResponse packet sent after the data phase includes the status of the entire operation.

7.3.3 Command with outgoing data phase

The protocol for a command with an outgoing data phase contains:

1. Command packet (from the host).
2. ReadMemory Response command packet (to host) (kCommandFlag_HasDataPhase set).
3. Outgoing data packets (to host).
4. GenericResponse command packet (to host).

- The data phase is considered part of the response command for the outgoing data phase sequence.
- The host may not send any further packets while the host is waiting for the response to a command.
- The data phase is aborted if the ReadMemory Response command packet, prior to the start of the data phase, does not contain the kCommandFlag_HasDataPhase flag.
- Data phases may be aborted by the device sending the final GenericResponse early with a status of kStatus_AbortDataPhase. The sending side may abort the data phase early by sending a zero-length data packet.
- The final GenericResponse packet sent after the data phase includes the status of the entire operation.

7.4 Bootloader packet types

7.4.1 Introduction

The bootloader device works in slave mode. All data communications are initiated by a host, which is either a PC or an embedded host. The bootloader device is the target, which receives a command or data packet. All data communications between host and target are packetized.

There are six types of packets used:

- Ping packet
- Ping response packet
- Framing packet

- Command packet
- Data packet
- Response packet

All fields in the packets are in little-endian byte order.

7.4.2 Ping packet

The ping packet is the first packet sent from a host to the target to establish a connection on the selected peripheral in order to run autobaud detection. The ping packet can be sent from host to target at any time that the target is expecting a command packet. If the selected peripheral is UART, a ping packet must be sent before any other communications. For other serial peripherals, it is optional.

In response to a ping packet, the target sends a ping response packet, discussed in the later sections.

Table 74. Ping packet format

Byte #	Value	Name
0	5Ah	Start byte
1	A6h	Ping

7.4.3 Ping response packet

The target sends a ping response packet back to the host after receiving a ping packet. If communication is over a UART peripheral, the target uses the incoming ping packet to determine the baud rate before replying with the ping response packet. After the ping response packet is received by the host, the connection is established, and the host starts sending commands to the target.

Table 75. Ping response packet format

Byte	Value	Parameter
0	5Ah	Start byte
1	A7h	Ping response code
2	00h	Protocol bugfix
3	03h	Protocol minor
4	01h	Protocol major
5	50h	Protocol name = 'P' (50h)
6	00h	Options low
7	00h	Options high
8	fbh	CRC16 low
9	40h	CRC16 high

To run autobaud for the UART peripheral, the ping response packet must be sent by the host when a connection is first established. For other serial peripherals, it is optional but recommended to determine the serial protocol version. The version number is in the same format as the bootloader version number returned by the GetProperty command.

7.4.4 Framing packet

The framing packet is used for flow control and error detection for the communications links that do not have such features built in. The framing packet structure sits between the link layer and the command layer. It wraps command and data packets as well.

Every framing packet containing data sent in one direction results in a synchronizing response framing packet in the opposite direction.

The framing packet described in this section is used for serial peripherals including the UART, CAN, I^2C, and SPI. The USB HID peripheral does not use framing packets. Instead, the packetization inherent in the USB protocol itself is used.

Table 76. Framing packet format

Byte	Value	Parameter	Description
0	5Ah	Start byte	—
1		PacketType	—
2		Length_low	Length is a 16-bit field that specifies the entire command or data packet size in bytes. —
3		Length_high	
4		Crc16_low	This is a 16-bit field. The CRC16 value covers entire framing packet, including the start byte and command or data packets, but does not include the CRC bytes. See CRC16 algorithm.
5		Crc16_high	
6....*n*		Command or Data packet payload	—

A special framing packet that contains only a start byte and a packet type is used for synchronization between the host and target.

Table 77. Special framing packet format

Byte	Value	Parameter
0	5Ah	Start byte
1	A*n*h	packetType

The Packet Type field specifies the type of packet from one of the defined types (below):

Table 78. Packet type field

Packet type	Name	Description
A1h	kFramingPacketType_Ack	The previous packet was received successfully; the sending of more packets is allowed.

Table continues on the next page...

Table 78. Packet type field (continued)

Packet type	Name	Description
A2h	kFramingPacketType_Nak	The previous packet was corrupted and must be re-sent.
A3h	kFramingPacketType_AckAbort	Data phase is being aborted.
A4h	kFramingPacketType_Command	The framing packet contains a command packet payload.
A5h	kFramingPacketType_Data	The framing packet contains a data packet payload.
A6h	kFramingPacketType_Ping	Sent to verify the other side is alive. Also used for UART autobaud.
A7h	kFramingPacketType_PingResponse	A response to ping. It contains the framing protocol version number and options.

7.4.5 CRC16 algorithm

The CRC is computed over each byte in the framing packet header, excluding the CRC16 field itself, and all of the payload bytes. The CRC algorithm is the XMODEM variant of CRC16.

The characteristics of the XMODEM variants are:

Table 79. CRC16 algorithm

Width	16
Polynomial	1021h
Init value	0000h
Reflect in	False
Reflect out	False
Xor out	0000h
Check result	31C3h

The check result is computed by running the ASCII character sequence "123456789" through the algorithm.

```
uint16_t crc16_update(const uint8_t * src, uint32_t lengthInBytes)
{
    uint32_t crc = 0;
    uint32_t j;
    for (j=0; j < lengthInBytes; ++j)
        {
            uint32_t i;
            uint32_t byte = src[j];
            crc ^= byte << 8;
            for (i = 0; i < 8; ++i)
                {
```

```
                    uint32_t temp = crc << 1;
                    if (crc & 8000h)
                        {
                            temp ^= 1021h;
                        }
                    crc = temp;
                }
        }
    return crc;
}
```

7.4.6 Command packet

The command packet carries a 32-bit command header and a list of 32-bit parameters.

Table 80. Command packet format

Command packet format (32 bytes)										
Command header (4 bytes)				28 bytes for Parameters (Max 7 parameters)						
Tag	Flags	Rsvd	Param Count	Param 1 (32-bit)	Param 2 (32-bit)	Param 3 (32-bit)	Param 4 (32-bit)	Param 5 (32-bit)	Param 6 (32-bit)	Param 7 (32-bit)

Table 81. Command header format

Byte #	Command header field	Description
0	Command or Response tag	The command header is 4 bytes long with these fields.
1	Flags	
2	Reserved. Must be 00h.	
3	ParameterCount	

The header is followed by 32-bit parameters up to the value of the ParameterCount field specified in the header. Because a command packet is 32 bytes long, only seven parameters can fit into the command packet.

- **Flags:** Each command packet contains a flag byte. Only bit 0 of the flag byte is used. If bit 0 of the flag byte is set to 1, then data packets follow the command sequence. The number of bytes that are transferred in the data phase is determined by a command-specific parameter in the parameters array.
- **ParameterCount:** The number of parameters included in the command packet.
- **Parameters:** The parameters are word-length (32 bits). With the default maximum packet size of 32 bytes, a command packet can contain up to seven parameters.

Command packets are also used by the target to send responses back to the host. As mentioned previously, command packets and data packets are embedded into framing packets for all transfers.

Table 82. Command tags

Command tag	Name	Description
01h	FlashEraseAll	The command tag specifies which command is supported by the bootloader. The valid command tags for the bootloader are listed here.[1]
02h	FlashEraseRegion	
03h	ReadMemory	
04h	WriteMemory	
05h	FillMemory	
06h	Reserved	
07h	GetProperty	
08h	ReceiveSbFile	
09h	Execute	
0Ah	Call	
0Bh	Reset	
0Ch	SetProperty	
0Dh	Reserved	
0Eh	eFuseProgram/FlashProgramOnce	
0Fh	eFuseRead/FlashReadOnce	
10h	Reserved	
11h	ConfigureMemory	
12h	Reserved	
13h	Reserved	
14h	Reserved	
15h	Reserved	
16h	TrustProvisioning	

1. The GetProperty, Reset, TrustProvisioning, SetProperty, and ReceiveSbFile are allowed in limited ISP mode.

Table 83. Response tags

Response tag	Name	Description
A0h	GenericResponse	The response tag specifies which of the responses the bootloader (target) returns to the host. The valid response tags are listed here.
A3h	ReadMemoryResponse	
A7h	GetPropertyResponse (used for sending responses to GetProperty command only)	
A3h	ReadMemoryResponse (used for sending responses to ReadMemory command only)	
AFh	FlashReadOnceResponse (used for sending responses to FlashReadOnce command only)	
B6h	TrustProvisioningResponse(used for sending response to TrustProvisioning command only)	

7.4.7 Response packet

The responses are carried using the same command packet format wrapped with framing packet data. Types of responses include:

- GenericResponse
- GetPropertyResponse
- ReadMemoryResponse
- FlashReadOnceResponse
- TrustProvisioningResponse

GenericResponse: After the bootloader has processed a command, the bootloader sends a generic response with status and command tag information to the host. GenericResponse is the last packet in the command protocol sequence. The GenericResponse packet contains the framing packet data and the command packet data (with GenericResponse tag = A0h) and a list of parameters (defined in the next section). The parameter count field in the header is always set to 2, for status code and command tag parameters.

Table 84. GenericResponse parameters

Byte #	Parameter	Description
0–3	Status code	Bootloader status error codes are errors encountered during the execution of a command by the target. If a command succeeds, then a kStatus_Success code is returned.
4–7	Command tag	The Command tag parameter identifies the response to the command sent by the host.

GetPropertyResponse: The GetPropertyResponse packet is sent by the target in response to the host query that uses the GetProperty command. The GetPropertyResponse packet contains the framing packet data and the command packet data, with the command or response tag set to a GetPropertyResponse tag value (A7h).

The parameter count field in the header is set to greater than 1, to always include the status code and one or many property values.

Table 85. GetPropertyResponse parameters

Byte #	Value	Parameter
0–3		Status code
4–7		Property value
. . .		. . .
		There can be up to a maximum of 6 property values, limited to the size of the 32-bit command packet and property type.

ReadMemoryResponse: The ReadMemoryResponse packet is sent by the target in response to the host sending a ReadMemory command. The ReadMemoryResponse packet contains the framing packet data and the command packet data, with the command or response tag set to a ReadMemoryResponse tag value (A3h), and the flags field set to kCommandFlag_HasDataPhase (1).

The parameter count is set to 2 for the status code and the data byte count parameters shown below.

Table 86. ReadMemoryResponse parameters

Byte #	Parameter	Description
0–3	Status code	The status of the associated Read Memory command.
4–7	Data byte count	The number of bytes sent in the data phase.

FlashReadOnceResponse: The FlashReadOnceResponse packet is sent by the target in response to the host sending a FlashReadOnce command. The FlashReadOnceResponse packet contains the framing packet data and the command packet data, with the commanresponse tag set to a FlashReadOnceResponse tag value (AFh), and the flags field set to 0. The parameter count is set to 2 plus the number of words requested to be read in the FlashReadOnceCommand.

Table 87. FlashReadOnceResponse parameters

Byte #	Value	Parameter
0–3		Status code
4–7		Byte count to read
. . .		. . .
		Can be up to 20 bytes of requested read data.

TrustProvisioningResponse: The TrustProvisioningResponse packet is sent by the target in response to the host sending a TrustProvisioning command. The TrustProvisioningResponse packet contains the framing packet data and command packet data, with the command or response tag set to a TrustProvisioningResponse tag value (B6h), and the flags field set to 0. The parameter count varies for different operations. See TrustProvisioning command for details.

7.5 Bootloader command set

7.5.1 Introduction

All bootloader commands follow the command packet format wrapped by the framing packet as explained in previous sections.

For a list of status codes returned by bootloader see Bootloader status error codes.

7.5.2 GetProperty command

The GetProperty command is used to query the bootloader about various properties and settings. Each supported property has a unique 32-bit tag associated with it. The tag occupies the first parameter of the command packet. The target returns a GetPropertyResponse packet with the property values for the property identified with the tag in the GetProperty command.

Properties are the defined units of data that can be accessed with the GetProperty or SetProperty commands. Properties may be read-only or read-write. All read-write properties are 32-bit integers, so they can easily be carried in a command parameter.

The 32-bit property tag is the only parameter required for GetProperty command.

Table 88. Parameters for GetProperty Command

Byte #	Parameter
0 - 3	Property tag (which can be received from the GetProperty 0 command)
4 - 7	External Memory Identifier (only applies to GetProperty for external memory, or status identifier if the property tag is equal to 8).

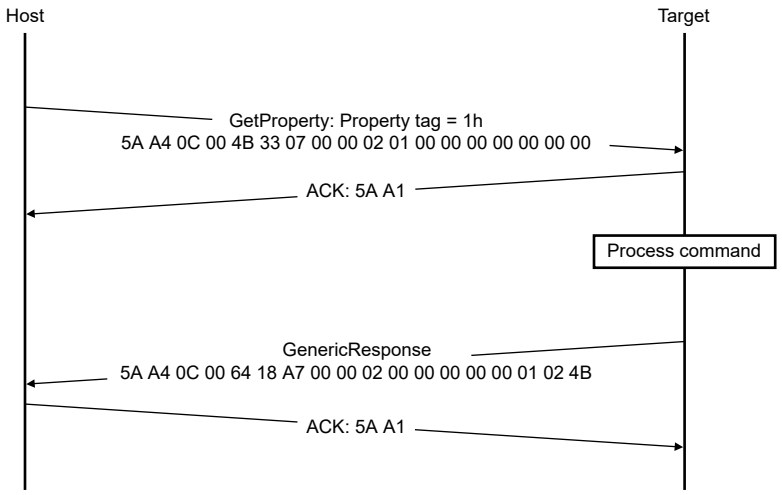


Figure 11. Protocol sequence for GetProperty command

Table 89. GetProperty command packet format (example)

GetProperty	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	0Ch 00h
	Crc16	4Bh 33h

Table 90. GetProperty command packet format (example)

GetProperty	Parameter	Value
Command packet	CommandTag	07h—GetProperty
	Flags	00h
	Reserved	00h
	ParameterCount	02h
	PropertyTag	0000_0001h—CurrentVersion
	Memory ID	0000_0000h—Internal Flash

The GetProperty command has no data phase.

Response: In response to a GetProperty command, the target sends a GetPropertyResponse packet with the response tag set to A7h. The parameter count indicates the number of parameters sent for the property values, with the first parameter showing status code 0, followed by the property value(s). Table 91 shows an example of a GetProperty response packet.

Table 91. GetProperty response packet format (example)

GetPropertyResponse	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	0Ch 00h (12 bytes)
	Crc16	07h 7Ah
Command packet	ResponseTag	A7h
	Flags	00h
	Reserved	00h
	ParameterCount	02h
	Status	0000_0000h
	PropertyValue	0000_014Bh—CurrentVersion

7.5.3 SetProperty command

The SetProperty command is used to change or alter the values of the properties or options of the bootloader. The command accepts the same property tags used with the GetProperty command. However, only some properties are writable. If an attempt to write a read-only property is made, an error is returned indicating the property is read-only and cannot be changed.

The property tag and the new value to set are the two parameters required for the SetProperty command.

Table 92. Parameters for SetProperty command

Byte #	Command
0—3	Property tag
4—7	Property value

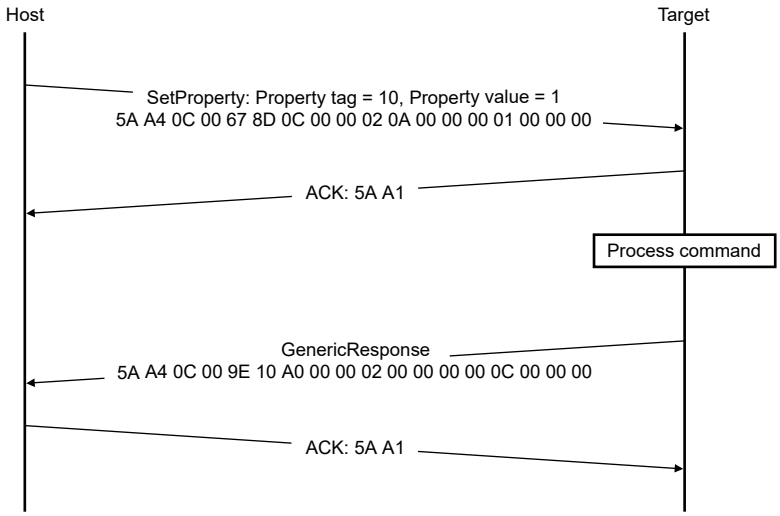


Figure 12. Protocol sequence for SetProperty Command

Table 93. SetProperty command packet format (example)

SetProperty	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	0Ch 00h
	Crc16	67h 8Dh
Command packet	CommandTag	0Ch—SetProperty with property tag 10
	Flags	00h
	Reserved	00h
	ParameterCount	02h
	PropertyTag	0000_000Ah—VerifyWrites
	PropertyValue	0000_0001h

The SetProperty command has no data phase.

Response: The target returns a GenericResponse packet with one of the following status codes:

Table 94. SetProperty response status codes

Status code
kStatus_Success
kStatus_ReadOnly
kStatus_UnknownProperty
kStatus_InvalidArgument

7.5.4 FlashEraseAll command

The FlashEraseAll command performs an erase of the entire flash memory (excluding IFR region). If any flash regions are protected, then the FlashEraseAll command fails and returns an error status code. The command tag for FlashEraseAll command is 01h set in the commandTag field of the command packet.

The FlashEraseAll command requires memory ID. If memory ID is not specified, the internal flash (memory ID = 0) is selected as default.

Table 95. Parameter for FlashEraseAll command

Byte #	Parameter	
0-3	Memory ID	
	000h	Internal Flash
	09h	FlexSPI NOR
	110h	Serial NOR/EEPROM through SPI

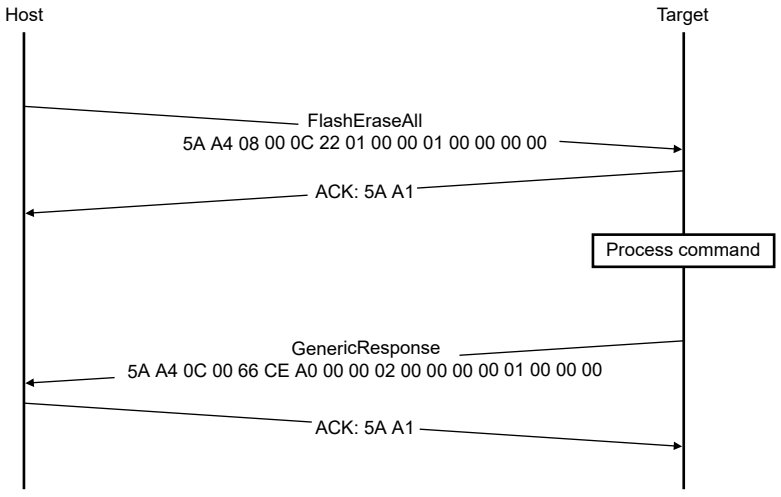


Figure 13. FlashEraseAll command

Table 96. FlashEraseAll command packet format (example)

FlashEraseAll	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	08h 00h
	Crc16	0Ch 22h
Command packet	CommandTag	01h—FlashEraseAll
	Flags	00h
	Reserved	00h
	ParameterCount	01h
	Memory ID	See the above table.

The FlashEraseAll command has no data phase.

Response: The target returns a GenericResponse packet with status code either set to kStatus_Success for successful execution of the command or set to an appropriate error status code.

7.5.5 FlashEraseRegion command

The FlashEraseRegion command performs an erase of one or more sectors of the flash memory.

The start address and number of bytes are the two parameters required for the FlashEraseRegion command. The start address and byte count parameters must be 4-byte aligned ([1:0] = 00), or the FlashEraseRegion command fails and returns kStatus_FlashAlignmentError (101). If the region specified does not fit in the flash memory space, the FlashEraseRegion command fails and returns kStatus_FlashAddressError (102). If any part of the region specified is protected, the FlashEraseRegion command fails and returns kStatus_MemoryRangeInvalid (10200).

Table 97. Parameter for FlashEraseRegion command

Byte #	Parameter
0–3	Start address
4–7	Byte count
8–11	Memory ID

The FlashEraseRegion command has no data phase.

Response: The target returns a GenericResponse packet with one of the following error status codes.

Table 98. FlashEraseRegion response status codes

Status code
kStatus_Success (0).

Table continues on the next page...

Table 98. FlashEraseRegion response status codes (continued)

Status code
kStatus_MemoryRangeInvalid (10200).
kStatus_FlashAlignmentError (101).
kStatus_IFlashAddressError (102).
kStatus_FlashAccessError (103).
kStatus_FlashProtectionViolation (104).
kStatus_FlashCommandFailure (105).

7.5.6 ReadMemory command

The ReadMemory command returns the contents of memory at the given address, for a specified number of bytes. This command can read any region of memory that is accessible by the CPU and not protected by security.

The start address and the number of bytes are the two parameters required for the ReadMemory command. The memory ID is optional. Internal memory is selected as the default if memory ID is not specified.

Table 99. Parameter for read memory command

Byte #	Parameter	Description
0–3	Start address	Start address of memory to read from
4–7	Byte count	Number of bytes to read and return to the caller
8–11	Memory ID	Internal or external memory identifier

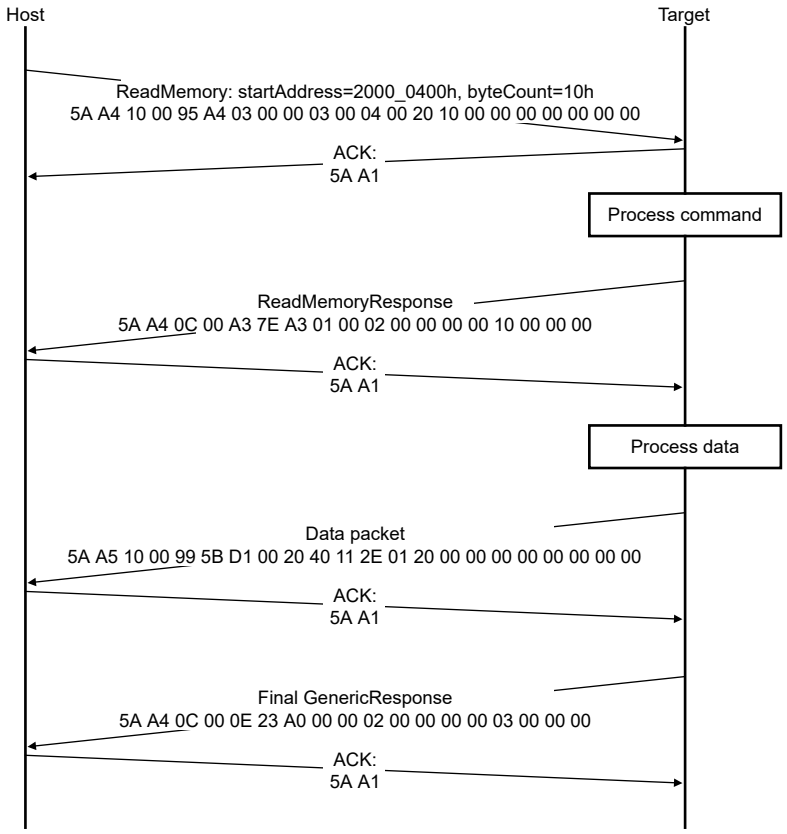


Figure 14. Command sequence for ReadMemory

Table 100. ReadMemory command packet format (example)

ReadMemory	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	10h 00h
	Crc16	F4h 1Bh
Command packet	CommandTag	03h—ReadMemory
	Flags	00h
	Reserved	00h
	ParameterCount	03h
	StartAddress	2000_0400h
	ByteCount	0000_0064h
	Memory ID	0h

Data phase: The ReadMemory command has a data phase. Because the target works in slave mode, the host needs to pull data packets until it has received the number of bytes of data specified in the byteCount parameter of the ReadMemory command.

Response: The target returns a GenericResponse packet with a status code either set to kStatus_Success (upon successful execution of the command) or set to an appropriate error status code.

7.5.7 WriteMemory command

The WriteMemory command writes data that is provided in the data phase to a specified range of bytes in memory (flash memory or RAM). However, if flash memory protection is enabled, then writes to protected sectors fail.

Consider the following information when you write to flash memory:

- First, any flash sector you write to must have been previously erased with a FlashEraseAll or FlashEraseRegion command.
- Writing to flash memory requires the start address to be page-aligned.
- The byte count is rounded up to a page size, and trailing bytes are filled with the flash erase pattern (FFh).
- If the VerifyWrites property is set to true, then writes to flash memory also perform a flash memory verify program operation.

When writing to RAM, the start address does not need to be aligned, and the data is not padded.

The two parameters required for the WriteMemory command are the start address and the number of bytes. The memory ID is optional. Internal memory is selected as the default if memory ID is not specified.

Table 101. Parameters for WriteMemory command

Byte #	Command
0–3	Start address
4–7	Byte count
8–11	Memory ID

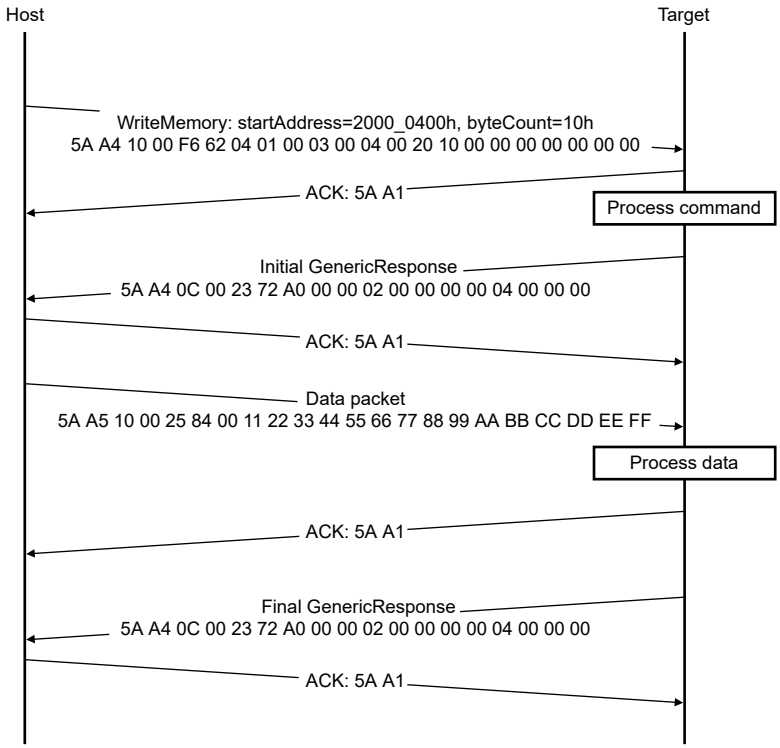


Figure 15. Protocol sequence for WriteMemory command

Table 102. WriteMemory command packet format (example)

WriteMemory	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	10h 00h
	Crc16	97h DDh
Command packet	CommandTag	04—WriteMemory
	Flags	01h
	Reserved	00h
	ParameterCount	03h
	StartAddress	2000_0400h
	ByteCount	0000_0064h
	Memory ID	0h

Data phase: The WriteMemory command has a data phase: the host sends data packets until the target has received the number of bytes of data specified in the byteCount parameter of the WriteMemory command.

Response: The target returns a GenericResponse packet with a status code set to kStatus_Success (upon successful execution of the command), or to an appropriate error status code.

7.5.8 FillMemory command

The FillMemory command fills a range of bytes in memory with a data pattern. It follows the same rules as the WriteMemory command. The difference between FillMemory and WriteMemory is that the FillMemory command parameter includes a data pattern. Also, there is no data phase for the FillMemory command, but WriteMemory does have a data phase.

Table 103. Parameters for FillMemory command

Byte #	Command
0–3	Start address of memory to fill
4–7	Number of bytes to write with the pattern • The start address must be 32-bit aligned. • The number of bytes must be evenly divisible by 4.
8–11	32-bit pattern

- To fill with a byte pattern (8-bit), the byte must be replicated four times in the 32-bit pattern.
- To fill with a short pattern (16-bit), the short value must be replicated two times in the 32-bit pattern.

For example, to fill a byte value with FEh, the word pattern is FEFE_FEFEh; to fill a short value 5AFEh, the word pattern is 5AFE_5AFEh.

Consider the following information when you write to flash memory:

- First, any flash memory sector you write to must have been previously erased with a FlashEraseAll or FlashEraseRegion command.
- Writing to flash memory requires the start address to be 4-byte aligned ([1:0] = 00).
- If the VerifyWrites property is set to true, then writes to flash memory also perform a flash memory verify program operation.

When writing to RAM, the start address does not need to be aligned, and the data is not padded.

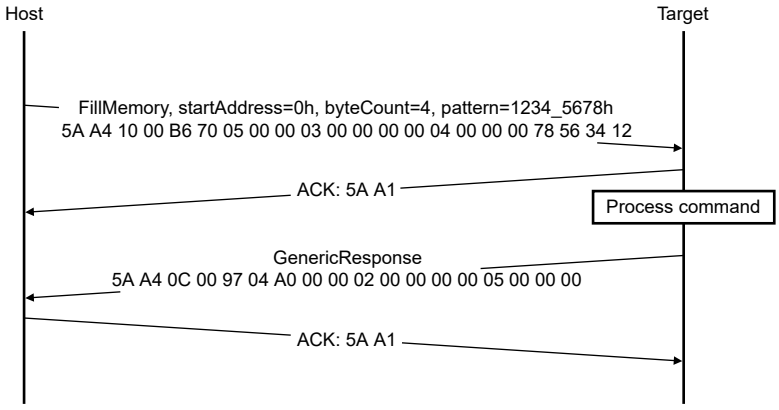


Figure 16. Protocol sequence for FillMemory command

Table 104. FillMemory command packet format (example)

FillMemory	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	10h 00h
	Crc16	E4h 57h
Command packet	CommandTag	05h—FillMemory
	Flags	01h
	Reserved	00h
	ParameterCount	03h
	StartAddress	0000_7000h
	ByteCount	0000_0800h
	PatternWord	1234_5678h

The FillMemory command has no data phase.

Response: The target (bootloader) returns a GenericResponse packet with a status code set to kStatus_Success (upon successful execution of the command), or to an appropriate error status code.

7.5.9 Execute command

The Execute command results in the bootloader setting of each of these items:

- The program counter to the code at the provided jump address
- R0 to the provided argument
- A stack pointer to the provided stack pointer address

Prior to the jump, the system returns to the Reset state.

The Execute command requires these parameters:

- Jump address
- Function argument pointer
- Stack pointer

If the stack pointer is set to zero, the called code is responsible for setting the processor stack pointer before using the stack.

Table 105. Parameters for Execute command

Byte #	Command
0–3	Jump address

Table continues on the next page...

Table 105. Parameters for Execute command (continued)

Byte #	Command
4–7	Argument word
8–11	Stack pointer address

The Execute command has no data phase.

Response: Before executing the Execute command, the target validates the parameters and returns a GenericResponse packet with a status code, set to either kStatus_Success or to an appropriate error status code.

7.5.10 Reset command

The Reset command results in the bootloader resetting the chip.

The Reset command requires no parameters.

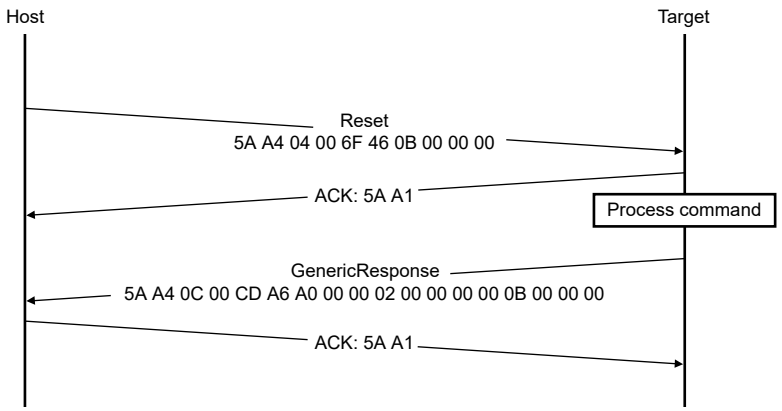


Figure 17. Protocol sequence for Reset command

Table 106. Reset command packet format (example)

Reset	Parameter	Value
Framing packet	Start byte	5Ah
	PacketType	A4h, kFramingPacketType_Command
	Length	04h 00h
	Crc16	6Fh 46h
Command packet	CommandTag	0Bh—reset
	Flags	00h
	Reserved	00h
	ParameterCount	03h

The Reset command has no data phase.

Response: Before resetting the chip, the target returns a GenericResponse packet with status code set to kStatus_Success.

You can also use the Reset command to switch boot from flash memory after successful flash image provisioning via the ROM bootloader. After issuing the Reset command, allow five seconds for the user application to start running from flash memory.

7.5.11 eFuseProgramOnce/FlashProgramOnce command

The FlashProgramOnce command writes data (provided in a command packet) to a specified range of bytes in the program-once field.

Consider the following information when you write to the program-once field:

- The program-once field only supports programming once, so any attempt to reprogram a program-once field generates an error response.
- Writing to the program-once field requires the byte count to be 4-byte aligned or 8-byte aligned.

The FlashProgramOnce command uses three parameters:

- Index 2
- byteCount
- Data

Table 107. Parameters for FlashProgramOnce command

Byte #	Command
0–3	Index of program-once field
4–7	Byte count (must be evenly divisible by 4)
8–11	Data

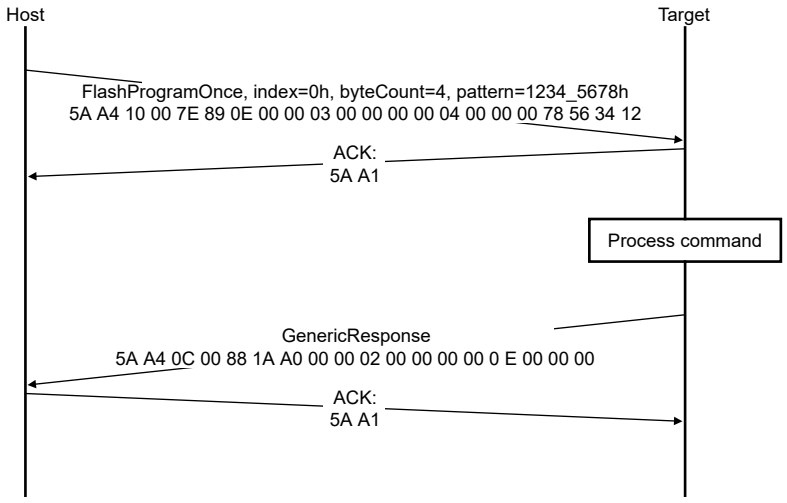


Figure 18. Protocol sequence for FlashProgramOnce command

Table 108. FlashProgramOnce command packet format

FlashProgramOnce	Parameter	Value
Framing packet	start byte	5Ah
	packetType	A4h, kFramingPacketType_Command
	length	10h 00h
	crc16	7E4h 89h
Command packet	commandTag	0Eh—FlashProgramOnce
	flags	0
	reserved	0
	parameterCount	3
	index	0000_0051h (the customer last efuse index)
	byteCount	0000_0004h
	Data	1234_5678h

Response: The target (MCX bootloader) returns a GenericResponse packet with a status code set to kStatus_Success (upon successful execution of the command), or to an appropriate error status code.

7.5.12 eFuseReadOnce / FlashReadOnce command

The FlashReadOnce command returns the contents of the program-once field by providing the given index and byte count.

The FlashReadOnce command uses two parameters: Index and byteCount. The eFuseReadOnce command uses only one parameter: Index.

Table 109. Parameters for FlashReadOnce command

Byte #	Parameter	Description
0–3	Index	Index of the program-once field (to read from)
4–7	byteCount	Number of bytes to read and return to the caller

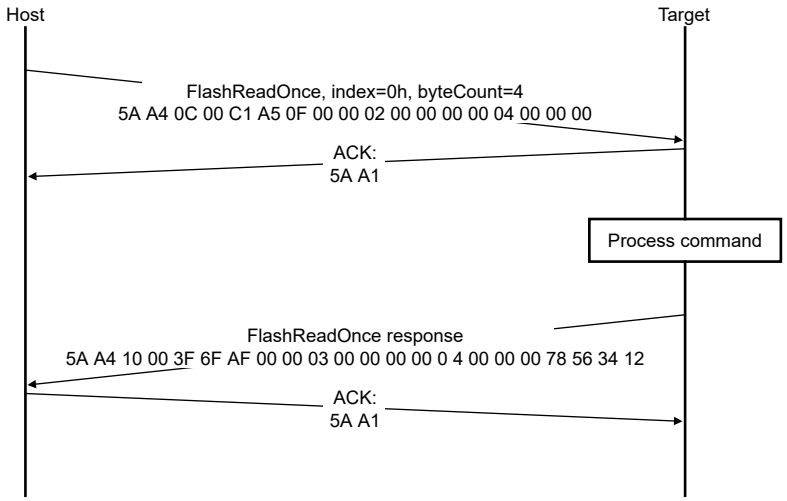


Figure 19. Protocol sequence for FlashReadOnce command

Table 110. FlashReadOnce command packet format

FlashReadOnce	Parameter	Value
Framing packet	start byte	5Ah
	packetType	A4h
	Length	0Ch 00h
	Crc	C1h A5h
Command packet	commandTag	0Fh—FlashReadOnce
	Flags	00h
	Reserved	00h
	parameterCount	02h
	Index	0000_0000h
	byteCount	0000_0004h

Table 111. FlashReadOnce response format

FlashReadOnce response	Parameter	Value
Framing packet	start byte	5Ah
	packetType	A4h
	Length	10h 00h

Table continues on the next page...

Table 111. FlashReadOnce response format (continued)

FlashReadOnce response	Parameter	Value
	CRC	3Fh 6Fh
Command packet	commandTag	AFh
	Flags	00h
	Reserved	00h
	parameterCount	03h
	Status	0000_0000h
	byteCount	0000_0004h
	Data	1234_5678h

Response: Upon successful execution of the command, the target returns:

- A FlashReadOnceResponse packet with a status code set to kStatus_Success
- A byte count
- A corresponding data read from the program-once field

If the execution was not successful, then the target returns a status code set to an appropriate error status and a byte count set to 0.

7.5.13 ConfigureMemory command

The ConfigureMemory command configures an internal or external memory device using a preprogrammed configuration block. The parameters passed in the command are:

- The memory ID
- The memory address from which the configuration data can be loaded

One case for loading the data could be a scenario where the configuration data is written to a RAM or flash memory location. This command would then direct the bootloader to use the data at that location for configuration of the external memory device.

Table 112. Parameters for ConfigureMemory command

Byte #	Command
0–3	Memory ID
4–7	Configuration block address

Response: The target (bootloader) returns a GenericResponse packet with a status code either set to kStatus_Success (upon successful execution of the command) or set to an appropriate error code.

The following table shows the supported memory IDs.

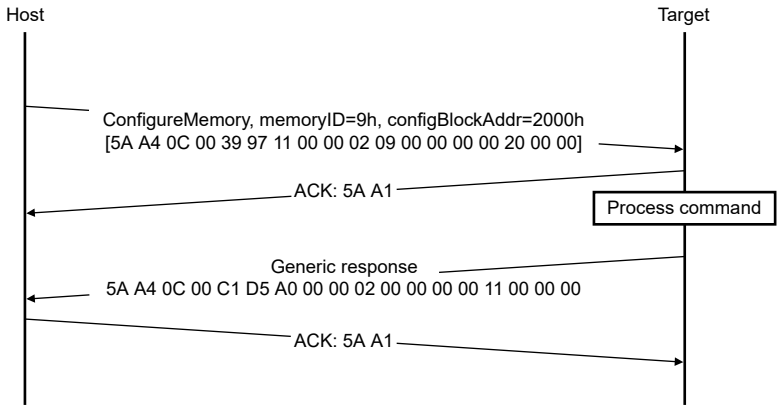


Figure 20. Protocol sequence for ConfigureMemory command

Table 113. Supported memory IDs

Memory ID	Description
0	Internal RAM/FLASH (used for the PRINCE configuration).
9h	External FlexSPI device. See Serial NOR Flash through FlexSPI.
110h	External 1-bit SPI NOR FLASH device. See 1-bit Serial NOR flash through SPI.

7.5.14 ReceiveSBFile command

The Receive SB File command (ReceiveSbFile) starts the transfer of an SB file to the target. This command only specifies the size in bytes of the SB file that is sent in the data phase. The SB file is processed as it is received by the bootloader. See SB3.1 firmware update container for more details about the SB file.

Table 114. Parameters for Receive SB File command

Byte #	Command
0–3	Byte count

Data phase: The Receive SB file command has a data phase. The host sends data packets until the target has received the number of bytes specified in the byteCount parameter of the Receive SB File command.

Response: The target returns a GenericResponse packet with a status code set to the kStatus_Success (upon successful execution of the command) or set to an appropriate error code.

7.5.15 TrustProvisioning command

The TrustProvisioning command is a pack of security operations, used for generating or restoring keys and certificates, encrypting or signing data, etc. Table 115 describes the operations supported in the TrustProvisioning command.

Table 115. TrustProvisioning Operations

Operation	Tag	Details
OEM_GEN_MASTER_SHARE	0	This command takes the entropy seed provided by the OEM as input to create shares for initial trust provisioning keys. All remaining ISP trust provisioning commands are dependent on this command.
OEM_SET_MASTER_SHARE	1	Command used by OEM to put the device back in key creation mode with original key shares produced when OEM_GET_MASTER_SHARE command was issued.
OEM_GET_CUST_CERT_DICE_PUK	2	This command is used by OEM design center to harvest public portion of signing device Identity key (NXP_CUST_DICE_CA_PUK) generated using DICE methodology for given RKTH.
HSM_GEN_KEY	3	Used for creating common OEM keys.
HSM_STORE_KEY	4	Used for storing known symmetric keys to be deployed on all devices.
HSM_ENC_BLK	5	Used to encrypt the given block.
HSM_ENC_SIGN	6	Used for signing the data buffer provided.

Command: There are 5-9 parameters required for the TrustProvisioning command. The parameter counts various for different operations. The first parameter is always <Operation Tag> that species which operation to operate. The remaining parameters store the arguments used by the operation. Refer to the next section for the details of the command packet parameters of each operation.

Table 116. Parameters for TrustProvisioning Command

Byte #	Command
0 - 3	Operation tag
4 - 19	Operation arguments
20 - 35	Operation extension arguments

The TrustProvisioning command packet format is shown in the table below.

Table 117. TrustProvisioning command packet format(example)

TrustProvisioning	Parameter	Value
Framing packet	start byte	0x5A
	packet type	0xA4, kFramingPacketType_Command
	length	0x18, 0x00
	crc16	0xFE, 0xE7
Command packet	command tag	0x16

Table continues on the next page...

Table 117. TrustProvisioning command packet format(example) (continued)

TrustProvisioning	Parameter	Value
	flags	0x00 (no data phase)
	reserved	0x00
	parameter count	0x05
	operation tag	0x00000002 (OEM_SET_MASTER_SHARE, refer to Table 121)
	arg0	0x30015000
	arg1	0x20
	arg2	0x30016000
	arg3	0x1000

Data Phase: The TrustProvisioning command has no data phase. The parameters specify the input buffer address, the input data size, the output buffer address, and the output buffer size. The user has to fill the input data into the input buffer using the WriteMemory command before issuing the TrustProvisioning command. ROM will output the data to the output buffer after the success of the TrustProvisioning command execution. Then the user can read the output data from the output buffer using the ReadMemory command. Only the free RAM can be selected as the input or output buffer, otherwise, an error code will be returned in the response.

Response: The target returns a GenericResponse packet once ROM failed to execute the command, and no value is required to be returned except the error status. In other scenarios, ROM will return a TrustProvisioningResponse packet which contains some important values. Refer to the next section for details of the return values of each operation in the response packet. For the GenericResponse, please refer to Table 84. Table 118 describes the TrustProvisioning Response packet.

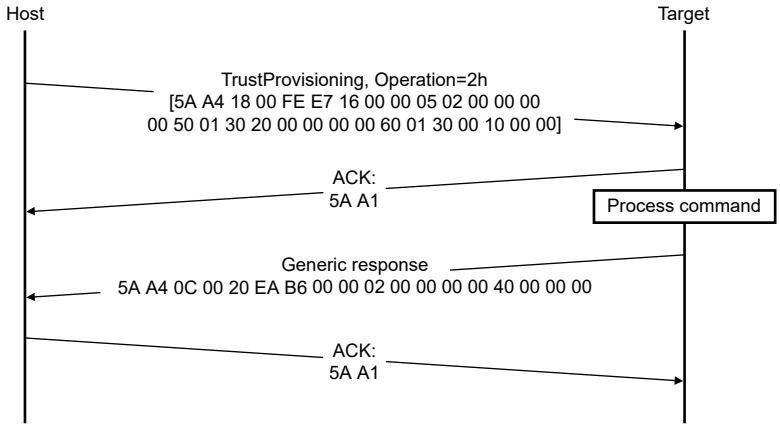


Figure 21. TrustProvisioning Response

Table 118. TrustProvisioning Response packet format (Example)

TrustProvisioningResponse	Parameter	Value
Framing packet	start byte	0x5A
	packet type	0xA4, kFramingPacketType_Command
	length	0x0C, 0x00
	crc16	0x20, 0xEA
Command packet	command tag	0xB6
	flags	0x00 (no data phase)
	reserved	0x00
	parameter count	0x02
	status	0x00000000
	value0	0x40

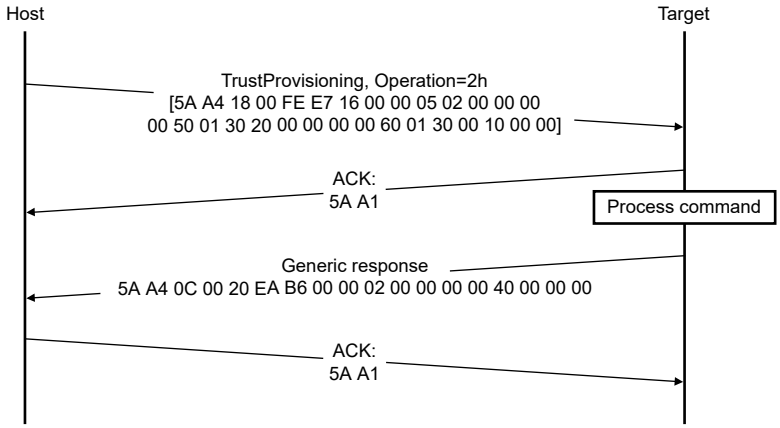


Figure 22. TrustProvisioning Response

7.5.15.1 OEM_GEN_MASTER_SHARE

This command takes the entropy seed provided by the OEM as input, and output two encrypted blobs(the Encrypted OEM Share and the Encrypted OEM Master Share) and a public key(the OEM Customer Certificate Public Key). Table 119 shows the parameters of the OEM_GEN_MASTER_SHARE command. And Table 120 shows the return values of the OEM_GEN_MASTER_SHARE response.

Table 119. OEM_GEN_MASTER_SHARE parameters

Byte	Parameter	Description
0 - 3	Operation tag	0x0

Table continues on the next page...

Table 119. OEM_GEN_MASTER_SHARE parameters (continued)

Byte	Parameter	Description
4 - 7	oemShareInputAddr	The input buffer address where the OEM Share(entropy seed) locates at
8 - 11	oemShareInputSize	The byte count of the OEM Share
12 - 15	oemEncShareOutputAddr	The output buffer address where ROM writes the Encrypted OEM Share to
16 - 19	oemEncShareOutputSize	The output buffer size in byte
20 - 23	oemEncMasterShareOutputAddr	The output buffer address where ROM writes the Encrypted OEM Master Share to
24 - 27	oemEncMasterShareOutputSize	The output buffer size in byte.
28 - 31	oemCustCertPukOutputAddr	The output buffer address where ROM writes the OEM Customer Certificate Public Key to
32 - 35	oemCustCertPukOutputSize	The output buffer size in byte

Table 120. OEM_GEN_MASTER_SHARE returns

Byte	Parameter	Description
0 - 3	status	Operation status
4 - 7	oemEncShareSize	The byte count of the Encrypted OEM Share
8 - 11	oemEncMasterShareSize	The byte count of the Encrypted OEM Master Share
12 - 15	oemCustCertPukSize	The byte count of the OEM Customer Certificate Public Key

7.5.15.2 OEM_SET_MASTER_SHARE

This command takes the entropy seed and the Encrypted OEM Master Share, which is generated by OEM_GET_MASTER_SHARE operation using the same entropy seed, as input to restore the security session. There is no output data for this operation, except a status code in the response packet. Table 121 shows the parameters of the OEM_SET_MASTER_SHARE command and Table 122 shows the return values of the OEM_SET_MASTER_SHARE response.

Table 121. OEM_SET_MASTER_SHARE parameters

Byte	Parameter	Description
0 - 3	Operation tag	0x1
4 - 7	oemShareInputAddr	The input buffer address where the OEM Share(entropy seed) locates at
8 - 11	oemShareInputSize	The byte count of the OEM Share
12 - 15	oemEncMasterShareInputAddr	The input buffer address where the Encrypted OEM Master Share locates at
16 - 19	oemEncMasterShareInputSize	The byte count of the Encrypted OEM Master Share

Table 122. OEM_SET_MASTER_SHARE returns

Byte	Return	Description
0 - 3	status	Operation status

7.5.15.3 OEM_GET_CUST_CERT_DICE_PUK

This command is used to create the initial trust provisioning keys. It takes OEM Root Key Table Hash(RKTH) as input, and output the public portion of the key used for signing device identity. Table 123 shows the parameters of the OEM_GET_CUST_CERT_DICE_PUK command and Table 124 shows the return values of the OEM_GET_CUST_CERT_DICE_PUK response.

Table 123. OEM_GET_CUST_CERT_DICE_PUK parameters

Byte	Parameter	Description
0 - 3	Operation tag	0x2
4 - 7	oemRkthInputAddr	The input buffer address where the OEM RKTH locates at
8 - 11	oemRkthInputSize	The byte count of the OEM RKTH
12 - 15	oemCustCertDicePukOutputAddr	The output buffer address where ROM writes the OEM Customer Certificate Public Key for DICE to
16 - 19	oemCustCertDicePukOutputSize	The output buffer size in byte

Table 124. OEM_GET_CUST_CERT_DICE_PUK returns

Byte	Return	Description
0 - 3	status	Operation status
4 - 7	oemCustCertDicePukOutputSize	The byte count of the OEM Customer Certificate Public Key for DICE

7.5.15.4 HSM_GEN_KEY

This command is used for creating OEM common keys, including encryption keys and signing keys. It outputs the key blob containing private key which is wrapped by NXP_CUST_KEK_INT_SK key and the public portion of the signing key. The key type is specified by the input parameter. Table 125 shows the parameters of the HSM_GEN_KEY command and Table 126 shows the return values of the HSM_GEN_KEY response

Table 125. HSM_GEN_KEY parameters

Byte	Parameter	Description
0 - 3	Operation tag	0x3
4 - 7	keyType	The key to generate 0xC3A5: MFW_ISK, ECDSA Manufacturing Firmware Signing Key 0xA5C3: MFW_ENCK, CKDF Master Key for Manufacturing Firmware Encryption Key

Table continues on the next page...

Table 125. HSM_GEN_KEY parameters (continued)

Byte	Parameter	Description
		0x5A3C: GEN_SIGNK, Generic ECDSA Signing Key 0x3C5A: GET_CUST_MK_SK, CKDF Master Key for Production Firmware Encryption Key
8 - 11	reserved	Reserved, must be zero
12 - 15	keyBlobOutputAddr	The output buffer address where ROM writes the key blob to
16 - 19	keyBlobOutputSize	The output buffer size in byte
20 - 23	ecdsaPukOutputAddr	The output buffer address where ROM writes the public key to
24 - 27	ecdsaPukOutputSize	The output buffer size in byte

Table 126. HSM_GEN_KEY returns

Byte	Return	Description
0 - 3	status	Operation status
4 - 7	keyBlobOutputSize	The byte count of the key blob
8 - 11	ecdsaPukOutputSize	The byte count of the public key

7.5.15.5 HSM_STORE_KEY

This command is used for storing known keys, and generate the corresponding key blob. It wraps the known key, which is given by the customer, using NXP_CUST_KEK_EXT_SK, and output the RFC3394 key blob. The key type and key size are specified by the input parameter. Table 127 shows the parameters of the HSM_STORE_KEY command and Table 128 shows the return values of the HSM_STORE_KEY response.

Table 127. HSM_STORE_KEY parameters

Byte	Parameter	Description
0 - 3	Operation tag	0x4
4 - 7	keyType	The key to generate 1: CKDFK, CKDF Master Key 2: HKDFK, HKDF Master Key 3: HMACK, HMAC Key 4: CMACK, CMAC Key 5: AESK, AES Key 6: KUOK, Key Unwrap Only Key
8 - 11	keyProp	Key size.

Table continues on the next page...

Table 127. HSM_STORE_KEY parameters (continued)

Byte	Parameter	Description
		bit[0]: Key Size, 0 for 128bit, 1 for 256bit bit[31-1]: Reserved, must be 0
12 - 15	keyInputAddr	The input buffer address where the key locates at
16 - 19	keyInputSize	The byte count of the key
20 - 23	keyBlobOutputAddr	The output buffer address where ROM writes the key blob to
24 - 27	keyBlobOutputSize	The output buffer size in byte

Table 128. HSM_STORE_KEY returns

Byte	Return	Description
0 - 3	status	Operation status
4 - 7	keyBlobOutputHeader	The header of the key blob.
8 - 11	keyBlobOutputSize	The byte count of the key blob(header is not included)

7.5.15.6 HSM_ENC_BLK

This command is used to encrypt the given SB3 data block. It uses the encryption key in the given CKDF Master Key Blob to encrypt the SB3 data block. The blob is generated by the HSM_GEN_KEY operation with key type set to MFW_ENCK, or the HSM_STORE_KEY operation with key type set to CKDFK. The parameter, 'kekId', specifics which key is used to wrap the key blob. HSM_GEN_KEY uses NXP_CUST_KEK_IN_SK to wrap the key blob, while HSM_STORE_KEY uses NXP_CUST_KEK_EXT_SK. After the success of the operation, the plaintext data block will be replaced with the encrypted block. Table 129 shows the parameters of the HSM_ENC_BLK command and Table 130 shows the return values of the HSM_ENC_BLK response.

Table 129. HSM_ENC_BLK parameters

Byte	Parameter	Description
0 - 3	Operation tag	0x5
4 - 7	mfgCustMkSk0BlobInputAddr	The input buffer address where the CKDF Master Key Blob locates at
8 - 11	mfgCustMkSk0BlobInputSize	The byte count of the CKDF Master Key Blob
12 - 15	kekId	The CKDF Master Key Encryption Key ID 0x10: NXP_CUST_KEK_INT_SK 0x11: NXP_CUST_KEK_EXT_SK
16 - 19	sb3HeaderInputAddr	The input buffer address where the SB3 Header(block0) locates at
20 - 23	sb3HeaderInputSize	The byte count of the SB3 Header

Table continues on the next page...

Table 129. HSM_ENC_BLK parameters (continued)

Byte	Parameter	Description
24 - 27	blockNum	The index of the block. Due to SB3 Header(block 0) is always un-encrypted, the index starts from block1
28 – 31	blockDataAddr	The buffer address where the SB3 data block locates at
32 - 35	blockDataSize	The byte count of the SB3 data block

Table 130. HSM_ENC_BLK returns

Byte	Return	Description
0 - 3	status	Operation status

7.5.15.7 HSM_ENC_SIGN

This command is used to sign the given data. It uses the private key in the given key blob, which is generated by HSM_GEN_KEY. Table 131 shows the parameters of the HSM_ENC_SIGN command Table 132 shows the return values of the HSM_ENC_SIGN response.

Table 131. HSM_ENC_SIGN parameters

Byte	Parameter	Description
0 - 3	Operation tag	0x6
4 - 7	keyBlobInputAddr	The input buffer address where signing key blob locates at
8 - 11	keyBlobInputSize	The byte count of the signing key blob
12 - 15	blockDataInputAddr	The input buffer address where the data locates at
16 - 19	blockDataInputSize	The byte count of the data
20 - 23	signatureOutputAddr	The output buffer address where ROM writes the signature to
24 - 27	signatureOutputSize	The output buffer size in byte

Table 132. HSM_ENC_SIGN returns

Byte	Return	Description
0 – 3	status	Operation status
4 - 7	signatureOutputSize	The byte count of the signature

7.5.16 GetProperty/SetProperty command properties

This section lists the properties of the GetProperty and SetProperty commands.

Table 133. Properties used by GetProperty and SetProperty commands, sorted by value

Property	Writable	Tag value	Size	Description
CurrentVersion	No	01h	4	Current bootloader version
AvailablePeripherals	No	02h	4	The set of peripherals supported on this chip
FlashStartAddress	No	03h	4	Start address of program flash memory
FlashSizeInBytes	No	04h	4	Size in bytes of program flash memory
AvailableCommands	No	07h	4	The set of commands supported by the bootloader
Check Status	No	08h	4	Return the status based on the specified status identifier 0—CRC status 32-bit return value for CRC check 10401—Application CRC check failed 10402—Application CRC check is inactive 10403—Application CRC check invalid 1—Last Error See details of the last error in GetLastError property.
MaxPacketSize	No	0Bh	4	Maximum supported packet size for currently active peripheral interface.
ReservedRegions	No	0Ch	8 x *n*	List of memory regions reserved by bootloader. Returned as value pairs (<start-address-of-region>,<end-address-of-region>). If HasDataPhase flag is not set, then the response packet parameter count indicates the number of pairs. If HasDataPhase flag is set, then the second parameter is the number of bytes in the data phase. *n* indicates the number of memory region pairs.
SystemDeviceId	No	10h	4	Value from SYSCON's Device ID (DEVICE_ID0) and Chip Revision ID and Number (DIEID) registers.
LifeCycleState	No	11h	4	The life cycle of the device. 5AA5_5AA5h—Device is in development life cycle C33C_C33Ch—Device is in deployment life cycle
UniqueDevice/UUID	No	12h	16	Unique device identification.
Target ROM Version	No	18h	4	Target build version number.
ExternalMemoryAttributes	No	19h	24	List of attributes supported by the specified memory ID (110h = SPINOR FLASH). See description of the return value in ExternalMemoryAttributes property.
IrqNotifierPin	Yes	1ch	4	IRQ Notifier Pin setting:

Table continues on the next page...

Table 133. Properties used by GetProperty and SetProperty commands, sorted by value (continued)

Property	Writable	Tag value	Size	Description
				• bit[7:0]: GPIO pin • bit[15:8]: GPIO port • bit [30:16]: Reserved • bit[31]: Enable flag 0: disable, 1: enable

7.5.16.1 Property definitions

7.5.16.1.1 CurrentVersion property

The value of this property is a 4-byte structure containing the current version of the bootloader.

Table 134. CurrentVersion property fields

Bit	[31:24]	[23:16]	[15:8]	[7:0]
Field	Name = 'K' (4Bh)	Major version	Major version	Bugfix version

7.5.16.1.2 AvailablePeripherals property

The value of this property is the peripherals supported by the bootloader, and the hardware on which it runs.

Table 135. Peripheral bits

Bit	[31:7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Field	Reserved	Reserved	Reserved	USB HID	CAN	SPI slave	I2C slave	LPUART

If the peripheral is available, then the corresponding bit is set in the property value. All reserved bits must = 0.

7.5.16.1.3 AvailableCommands property

This property value is a bit field that indicates the commands enabled in the bootloader. Only commands that can be sent from the host to the target are listed in the bit field. Response commands such as GenericResponse are excluded.

The bit number that identifies whether a command is present is the command's tag value minus 1. The value 1 is subtracted from the command tag because the lowest command tag value is 01h. To determine the bit mask for a given command, use this expression: mask = 1 << (tag – 1).

Table 136. Command bits

Bit	[Others]	[20]	[16]	[15]	[14]	[13]	[12]	[11]	[10]	[9]	[8]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Command																			

7.5.16.1.4 ExternalMemoryAttributes property

The value that this property returns is a 24-byte data structure containing available external memory attributes:

- Start address
- Total size (in KB)

- Page size
- Sector size
- Block size

Below is the breakdown of the 24-byte structure.

Table 137. Fields of ExternalMemoryAttributes property

Field offset	Field Description
0–3	The value returned is a bitmap showing the supported attributes for the external memory, with the corresponding bit field set. 0000_0001h—start address 0000_0002h—total size 0000_0004h—page size 0000_0008h—sector size 0000_0010h—block size
4–7	Start address of external memory
8–11	Total size of external memory in KB
12–15	Page size of external memory in bytes
16–19	Sector size of external memory in bytes
20–23	Block size of external memory in bytes

7.5.16.1.5 GetLastError property

The following table lists the response words and corresponding error conditions.

Table 138. Response word and error description

Response word	Error	Description
0B37_F300h	kLog_Auth_ImageTypeCheck_Fail	When checking the image type pass, the image type did not meet the authentication condition.
0B32_F300h	kLog_Auth_ReadKeyStore_Fail	The read of the keystore part from the RAM image or another keystore region failed.
0B34_F300h	kLog_Auth_VerifyHMAC_Fail	Failed to verify the mac key.
0B35_F300h	kLog_Auth_ImageEntryCheck_Fail	Application entry point is invalid or stack address is invalid.
0B36_F300h	kLog_Auth_Auth_Fail	The authentication of the image failed.
0B36_F301h	kLog_Auth_Auth_Fail_Time	The authentication failed, and the cost time has been loaded into the log data.
0B37_F300h	kLog_Auth_CrcCheck_Fail	CRC checksum is wrong.
0B37_F301h	kLog_Auth_CrcCheck_Fail_Time	The image CRC check failed and the cost time has been loaded into the log data.

Table continues on the next page...

Table 138. Response word and error description (continued)

Response word	Error	Description
0B37_FC00h	kLog_Auth_CrcCheck_Invalid	Either the image type or the image length = 0 did not meet the requirement of the CRC image.
0B38_F300h	kLog_Auth_Dice_Fail	Dice calculation failed.
0C00_F300h	kLog_Jump_Fail	The image jump failed.

Table 139. Response word and error description

Response word	Error	Description
0C00_F500h	kLog_Jump_Fail_Fatal	Failed to jump to application.
0702_F301h	kLog_Recoveryboot_Fail_Reason	Recovery boot failed.
0501_F300h	kLog_Masterboot_Init_Fail	The master boot initialization failed.
0801_F300h	kLog_Ispboot_Init_Fail	The ISP boot initialization failed, meaning there are no available ISP peripherals.
0901_F300h	kLog_BootDevice_Init_Fail	The boot device initialization failed.
0902_F300h	kLog_BootDevice_Read_Fail	The master boot failed to read the initial image.
0A20_F300h	kLog_ImgLoad_InitLoad_Fail	The master boot failed to load the image header part.
0A21_F300h	kLog_ImgLoad_RemainingLoad_Fail	Image header and configuration data loaded successfully; the remaining portion of the image failed to load.
0B30_DC00h	kLog_Auth_SecureBootDisabled	The secure boot has been disabled—no need to do authentication.

7.6 Bootloader status error codes

Table 140 describes the status error codes that the bootloader returns to the host.

Table 140. Bootloader status error codes, sorted by value

Error code	Value	Description
kStatus_Success	0	Operation succeeded without error
kStatus_Fail	1	Operation failed with generic error
kStatus_ReadOnly	2	Requested value is read-only and cannot be changed
kStatus_OutOfRange	3	Requested value is out of range
kStatus_InvalidArgument	4	Requested command's argument is undefined
kStatus_Timeout	5	Timeout occurred
kStatus_NoTransferInProgress	6	No send in progress

Table continues on the next page...

Table 140. Bootloader status error codes, sorted by value (continued)

Error code	Value	Description
kStatus_FLASH_Success	0	API executed successfully
kStatus_FLASH_InvalidArgument	4	Invalid argument was provided
kStatus_FlashSizeError	100	Not used
kStatus_FlashAlignmentError	101	Address or length does not meet required alignment
kStatus_FlashAddressError	102	Address or length is outside addressable memory.
kStatus_FLASH_CommandFailure	105	INT_STATUS[FAIL] = 1
kStatus_FlashUnknownProperty	106	Unknown flash memory property
kStatus_FlashEraseKeyError	107	Provided key does not match programmed flash memory key
kStatus_FlashRegionExecuteOnly	108	Area of flash memory is protected as execute-only
kStatus_FLASH_ExecuteInRamFunctionNotReady	109	Execute-in-RAM function is not available
kStatus_FLASH_CommandNotSupported	111	Flash memory API is not supported
kStatus_FLASH_ReadOnlyProperty	112	Flash memory property is read-only
kStatus_FLASH_InvalidPropertyValue	113	Flash memory property value out of range
kStatus_FLASH_InvalidSpeculationOption	114	Flash memory prefetch speculation option is invalid
kStatus_FLASH_EccError	116	A correctable or uncorrectable error occurred during command execution
kStatus_FLASH_CompareError	117	Destination and source memory contents do not match
kStatus_FLASH_RegulationLoss	118	Loss of regulation during read
kStatus_FLASH_InvalidWaitStateCycles	119	Wait state cycle set to read/write mode is invalid
kStatus_FLASH_OutOfDateCfpaPage	132	CFPA page version is out of date
kStatus_FLASH_BlankIfrPageData	133	Blank page cannot be read
kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce	134	Encrypted flash memory subregions not erased at once

Table continues on the next page...

Table 140. Bootloader status error codes, sorted by value (continued)

Error code	Value	Description
kStatus_FLASH_ProgramVerificationNotAllowed	135	Program verification not allowed when encryption enabled
kStatus_FLASH_HashCheckError	136	Hash check of page data failed
kStatus_FLASH_SealedFfrRegion	137	FFR region sealed
kStatus_FLASH_FfrRegionWriteBroken	138	FFR spec region not allowed to be written discontinuously
kStatus_FLASH_NmpaAccessNotAllowed	139	NMPA region not allowed to be read, written, or erased
kStatus_FLASH_CmpaCfgDirectEraseNotAllowed	140	CMPA configuration region cannot be erased directly
kStatus_FLASH_FfrBankIsLocked	141	FFR bank region locked
kStatus_FLASH_CfpaScratchPageInvalid	148	CFPA scratch page invalid
kStatus_FLASH_CfpaVersionRollbackDisallowed	149	CFPA version rollback not allowed
kStatus_FLASH_ReadHidingAreaDisallowed	150	Flash memory hiding read not allowed
kStatus_FLASH_ModifyProtectedAreaDisallowed	151	Flash firewall page locked Erase and program are not allowed
kStatus_FLASH_CommandOperationInProgress	152	Flash memory state busy Flash memory command is in progress
kStatus_UnknownCommand	10000	Command not recognized
kStatus_SecurityViolation	10001	Security violation happened when receiving disallowed commands
kStatus_AbortDataPhase	10002	Sender requested data phase termination
kStatus_Ping	10003	Ping command received from host
kStatus_NoResponse	10004	No response from host
kStatus_NoResponseExpected	10005	Expected no response from host
kStatus_CommandUnsupported	10006	Unsupported command received
kStatusRomLdrSectionOverrun	10100	Reached the end of SB file processing
kStatusRomLdrSignature	10101	Signature or version is incorrect

Table continues on the next page...

Table 140. Bootloader status error codes, sorted by value (continued)

Error code	Value	Description
kStatusRomLdrSectionLength	10102	The bootOffset or new section count is out of range
kStatusRomLdrUnencryptedOnly	10103	Unencrypted image disabled
kStatusRomLdrEOFReached	10104	End of image file has been reached
kStatusRomLdrChecksum	10105	Checksum of command tag block is invalid
kStatusRomLdrCrc32Error	10106	CRC-32 of the data for a load command is incorrect
kStatusRomLdrUnknownCommand	10107	Unknown command was found in SB file
kStatusRomLdrIdNotFound	10108	No bootable section found in SB file
kStatusRomLdrDataUnderrun	10109	SB state machine waiting for more data
kStatusRomLdrJumpReturned	10110	The function jumped to by the SB file has returned
kStatusRomLdrCallFailed	10111	Call command in SB file failed
kStatusRomLdrKeyNotFound	10112	Matching key to unencrypt the section not found in SB file's key dictionary
kStatusRomLdrSecureOnly	10113	SB file sent is unencrypted Security on the target is enabled
kStatusRomLdrResetReturned	10114	SB file reset operation has unexpectedly returned
kStatusRomLdrRollbackBlocked	10115	Image version rollback event detected
kStatusRomLdrInvalidSectionMacCount	10116	Invalid section MAC count detected in SB file
kStatusRomLdrUnexpectedCommand	10117	Command tag in SB file is unexpected
kStatusRomLdrBadSBKEK	10118	Bad SBKEK detected
kStatusRomLdrPendingJumpCommand	10119	Jump command pending Actual jump implemented in sbloader_finalize()
kStatusMemoryRangeInvalid	10200	Requested address range does not match an entry, or the length extends past the matching entry's end address
kStatusMemoryReadFailed	10201	Memory read failed

Table continues on the next page...

Table 140. Bootloader status error codes, sorted by value (continued)

Error code	Value	Description
kStatusMemoryWriteFailed	10202	Memory write failed
kStatusMemoryCumulativeWrite	10203	Cumulative write occurred due to a write to an unerased flash memory region
kStatusMemoryNotConfigured	10205	Memory was not configured before access
kStatusMemoryAlignmentError	10206	Alignment error during memory access
kStatusMemoryVerifyFailed	10207	Verifying operation failed after erasing or programming flash memory
kStatusMemoryWriteProtected	10208	Memory to be written is protected
kStatusMemoryAddressError	10209	Memory address is invalid or wrong
kStatusMemoryBlankCheckFailed	10210	Check of blank memory failed
kStatusMemoryBlankPageReadDisallowed	10211	Memory is blank Read command is disallowed
kStatusMemoryProtectedPageReadDisallowed	10212	Memory is protected Read command is disallowed
kStatusMemoryFfrSpecRegionWriteBroken	10213	Write operation to FFR region broken
kStatusMemoryUnsupportedCommand	10214	Memory command not supported
kStatus_UnknownProperty	10300	Requested property value undefined
kStatus_ReadOnlyProperty	10301	Requested property value cannot be written
kStatus_InvalidPropertyValue	10302	Specified property value invalid
kStatus_AppCrcCheckPassed	10400	CRC check valid and passed
kStatus_AppCrcCheckFailed	10401	CRC check valid but failed
kStatus_AppCrcCheckInactive	10402	CRC check inactive
kStatus_AppCrcCheckInvalid	10403	CRC check invalid because BCA invalid, or CRC parameters unset (all FFh bytes)
kStatus_AppCrcCheckOutOfRange	10404	CRC check valid, but addresses out of range
kStatus_RomApiExecuteCompleted	0	ROM successfully processed complete SB file or boot image

Table continues on the next page...

Table 140. Bootloader status error codes, sorted by value (continued)

Error code	Value	Description
kStatus_RomApiNeedMoreData	10801	ROM needs more data to continue processing boot image
kStatus_RomApiBufferSizeNotEnough	10802	User buffer not large enough for use by Kboot during execution
kStatus_RomApiInvalidBuffer	10803	User buffer not acceptable for sbloader or authentication
kStatus_FLEXSPI_SequenceExecutionTimeout	6000	FLEXSPI sequence execution timed out
kStatus_FLEXSPI_InvalidSequence	6001	FLEXSPI LUT sequence invalid
kStatus_FLEXSPI_DeviceTimeout	6002	FLEXSPI device timed out
kStatus_FLEXSPINOR_ProgramFail	20100	Page programming failure
kStatus_FLEXSPINOR_EraseSectorFail	20101	Sector erase failure
kStatus_FLEXSPINOR_EraseAllFail	20102	Chip erase failure
kStatus_FLEXSPINOR_WaitTimeout	20103	Execution timeout
kStatus_FlexSPINOR_NotSupported	20104	Page size overflow
kStatus_FlexSPINOR_WriteAlignmentError	20105	Address alignment error
kStatus_FlexSPINOR_CommandFailure	20106	Erase or Program verify error
kStatus_FlexSPINOR_SFDP_NotFound	20107	Timeout occurred during the API call
kStatus_FLEXSPINOR_Unsupported_SFDP_Version	20108	Unrecognized SFDP version
kStatus_FLEXSPINOR_FLASH_NotFound	20109	Flash memory detection failure
kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed	20110	DDR read dummy probe failure
kStatus_IAP_Success	0	IAP API execution succeeded
kStatus_IAP_Fail	1	IAP API execution failed
kStatus_IAP_InvalidArgument	100001	Invalid argument detected during API execution
kStatus_IAP_OutOfMemory	100002	Heap size not large enough during API execution
kStatus_IAP_ReadDisallowed	100003	Read memory operation disallowed during API execution

Table continues on the next page...

Table 140. Bootloader status error codes, sorted by value (continued)

Error code	Value	Description
kStatus_IAP_CumulativeWrite	100004	Flash memory region to be programmed is not empty
kStatus_IAP_EraseFailuire	100005	Erase operation failed
kStatus_IAP_CommandNotSupported	100006	Specific command not supported
kStatus_IAP_MemoryAccessDisabled	100007	Memory access disabled **NOTE** Typically occurs on FLEXSPI NOR if it is not configured properly
kStatus_NBOOT_Success	5A5A_5A5Ahu	Operation successful in NBOOT functions
kStatus_NBOOT_Fail	5A5A_A5A5hu	Operation failed in NBOOT functions
kStatus_NBOOT_InvalidArgument	5A5A_A5F0hu	Invalid argument passed to function in NBOOT functions
kStatus_NBOOT_RequestTimeout	5A5A_A5E1hu	Operation timed out in NBOOT functions.
kStatus_NBOOT_KeyNotLoaded	5A5A_A5E2hu	Requested key not loaded in NBOOT functions
kStatus_NBOOT_AuthFail	5A5A_A5E4hu	Authentication failed in NBOOT functions
kStatus_NBOOT_OperationNotAvaialable	5A5A_A5E5hu	Operation not available on this hardware in NBOOT functions
kStatus_NBOOT_KeyNotAvailable	5A5A_A5E6hu	Key not available in NBOOT functions
kStatus_NBOOT_IvCounterOverflow	5A5A_A5E7hu	Overflow of IV counter (PRINCE/IPED) in NBOOT functions
kStatus_NBOOT_SelftestFail	5A5A_A5E8hu	FIPS self-test failure in NBOOT functions
kStatus_NBOOT_InvalidDataFormat	5A5A_A5E9hu	Invalid data format for example antipole in NBOOT functions
kStatus_NBOOT_IskCertUserDataTooBig	5A5A_A5EAhu	Size of user data in ISK certificate greater than 96 bytes in NBOOT functions
kStatus_NBOOT_IskCertSignatureOffsetTooSmall	5A5A_A5EBhu	Signature offset in ISK certificate smaller than expected in NBOOT functions
kStatus_NBOOT_MemcpyFail	5A5A_845Ahu	Unexpected error detected during nboot_memcpy() in NBOOT functions
kStatus_SKBOOT_Success	5AC3_C35Ahu	Operation successful in SKBOOT functions
kStatus_SKBOOT_Fail	C35A_C35Ahu	Operation failed in SKBOOT functions
kStatus_SKBOOT_InvalidArgument	C35A_5AC3hu	Return invalid argument status in SKBOOT functions

Table continues on the next page...

Table 140. Bootloader status error codes, sorted by value (continued)

Error code	Value	Description
kStatus_MKBOOT_Success	0000_C35Ahu	Operation successful in master KBOOT
kStatus_MKBOOT_Fail	0000_C3C3hu	Operation failed in master KBOOT.
kStatus_MKBOOT_InvalidArgument	0000_5AC3hu	Return Invalid argument status in Master KBOOT

NOTE

In UART, I2C, CAN, and SPI ISP modes, the ROM expects posted responses to be read by the host within (20 ms x number of bytes)—otherwise the ROM reports the terminate command. Because an ACK from the ROM is only two bytes, the host must read those bytes within 40 ms of the ROM posting.

7.7 UART ISP

7.7.1 Introduction

The bootloader integrates an autobaud detection algorithm for the UART peripheral, thereby providing flexible baud rate choices.

Autobaud feature: If you use UART*n* to connect to the bootloader, then you must keep the UART*n*_RX pin high and not left floating during the detection phase. You perform this action to comply with the autobaud detection algorithm. After the bootloader detects the ping packet (5Ah A6h) on UART*n*_RX, the bootloader firmware executes the autobaud sequence.

If the bootloader successfully detects the baud rate, then it sends a ping packet response [(5Ah A7h), protocol version (4 bytes), protocol version options (2 bytes), and crc16 (2 bytes)] at the detected baud rate. The bootloader then enters a loop, waiting for bootloader commands via the UART peripheral.

NOTE

The data bytes of the ping packet must be sent continuously (with no more than 80 ms between bytes) in a fixed UART transmission mode (8-bit data, no parity bit, and 1 stop bit). If the bytes of the ping packet are sent one by one with more than 80 ms delay between them, then the autobaud detection algorithm may calculate an incorrect baud rate. In this instance, the autobaud detection state machine should be reset.

The baud rate is closely related to the chip core and system clock frequencies. The typical supported baud rates are:

- 9600
- 19200
- 38400
- 57600
- 115200
- 230400
- 460800
- 1000000

Packet transfer: After autobaud detection succeeds, bootloader communications can take place over the UART peripheral:

- Figure 23 shows how the host detects an ACK from the target.
- Figure 24 shows how the host detects a ping response from the target
- Figure 25 shows how the host detects a command response from the target

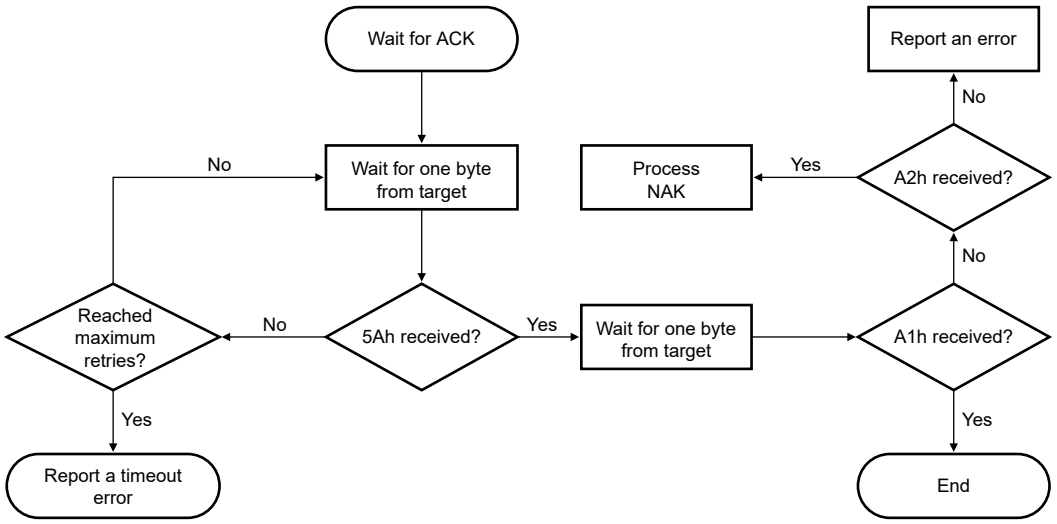


Figure 23. Host reads an ACK from target via UART

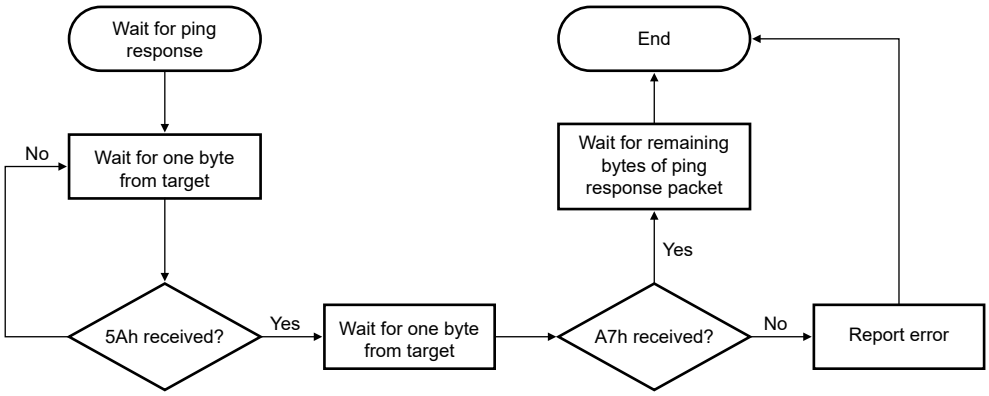


Figure 24. Host reads a ping response from target via UART

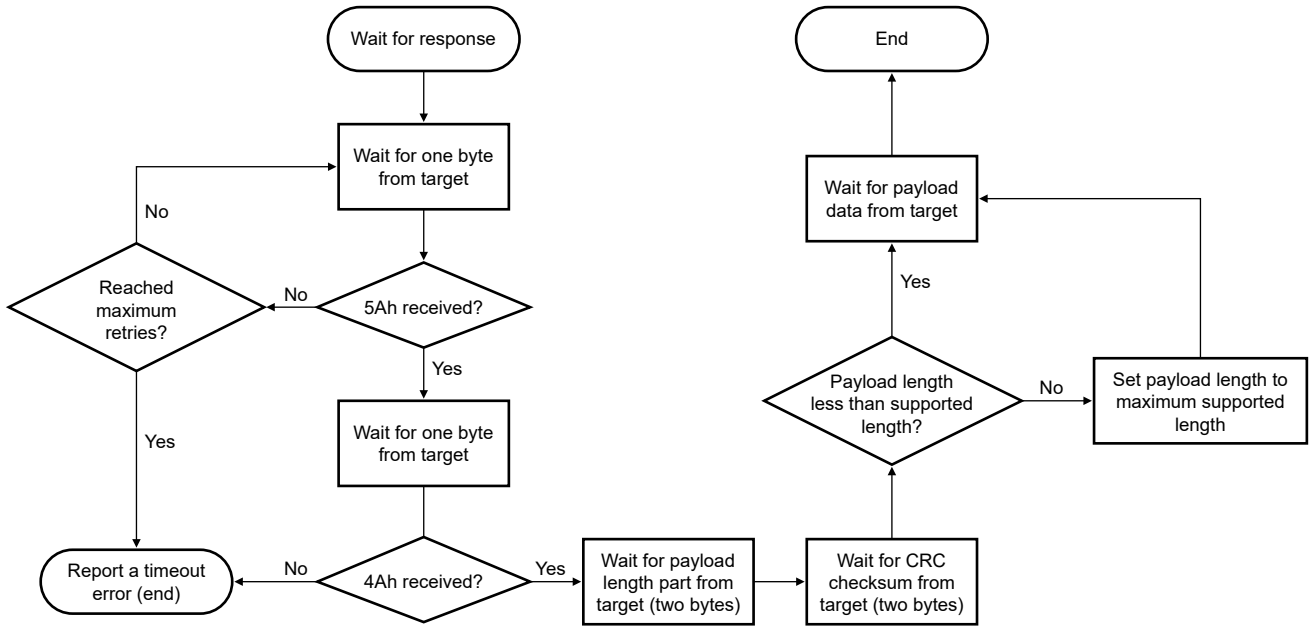


Figure 25. Host reads a command response from target via UART

For more information on UART ISP command, response, and data formats, see Bootloader packet types. For information on UART ISP commands, see Command packet.

7.8 I²C ISP

7.8.1 Introduction

The bootloader supports loading data into flash memory via the I²C peripheral, where the I²C peripheral serves as the I²C target. The transfer uses a 7-bit target address. The bootloader uses 10h as the I²C target address and supports up to 400 kbit/s as the I²C baud rate. You can also reconfigure the I2C slave address with the value from CMPA[I2C_SLAVE_ADDR](0x0100403C[7:0]).

The maximum supported I²C baud rate depends on the core clock frequency when the bootloader is running. The typical baud rate is 400 kbit/s with factory settings.

Because the I²C peripheral serves as an I²C target device, the host must start each transfer and fetch each outgoing packet:

- The host sends an incoming packet with a selected I²C target address, and the direction bit is set as write.
- The host reads an outgoing packet with a selected I²C target address, and the direction bit is set as read.
- If the target is busy with processing or preparing data, then the target sends 00h as the response to the host.

The following flowcharts show the communication flow of the host reading the ping and ACK packets, and the corresponding responses from the target.

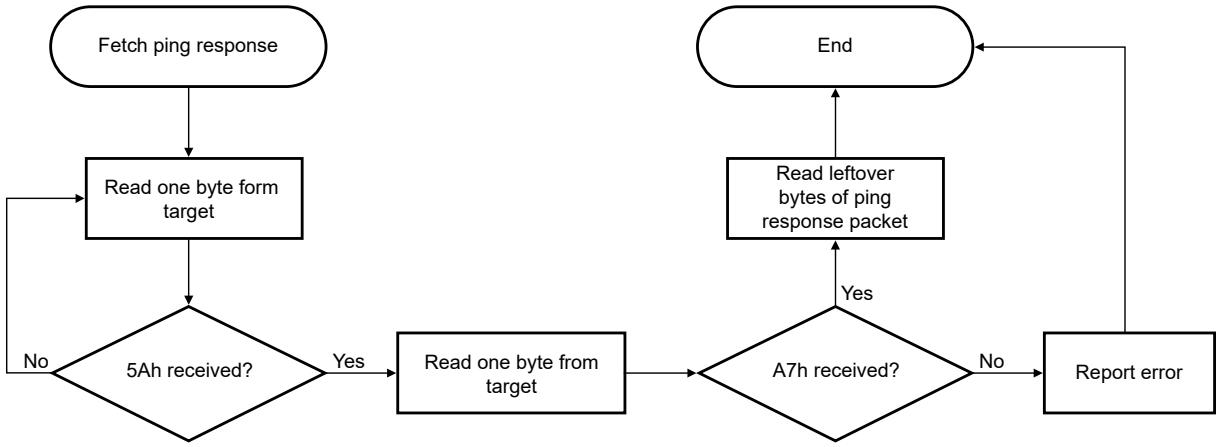


Figure 26. Host reads ping response from target via I²C

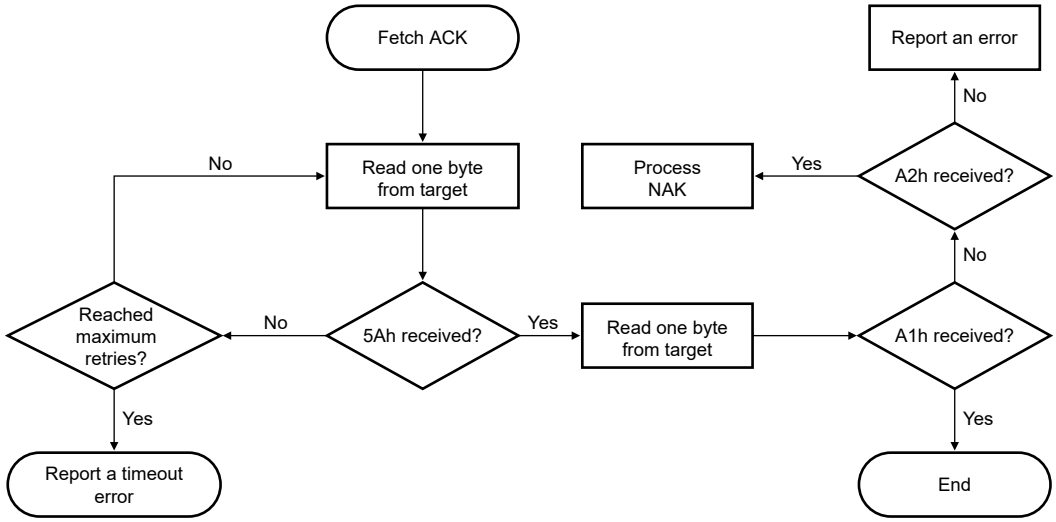


Figure 27. Host reads ACK packet from target via I²C

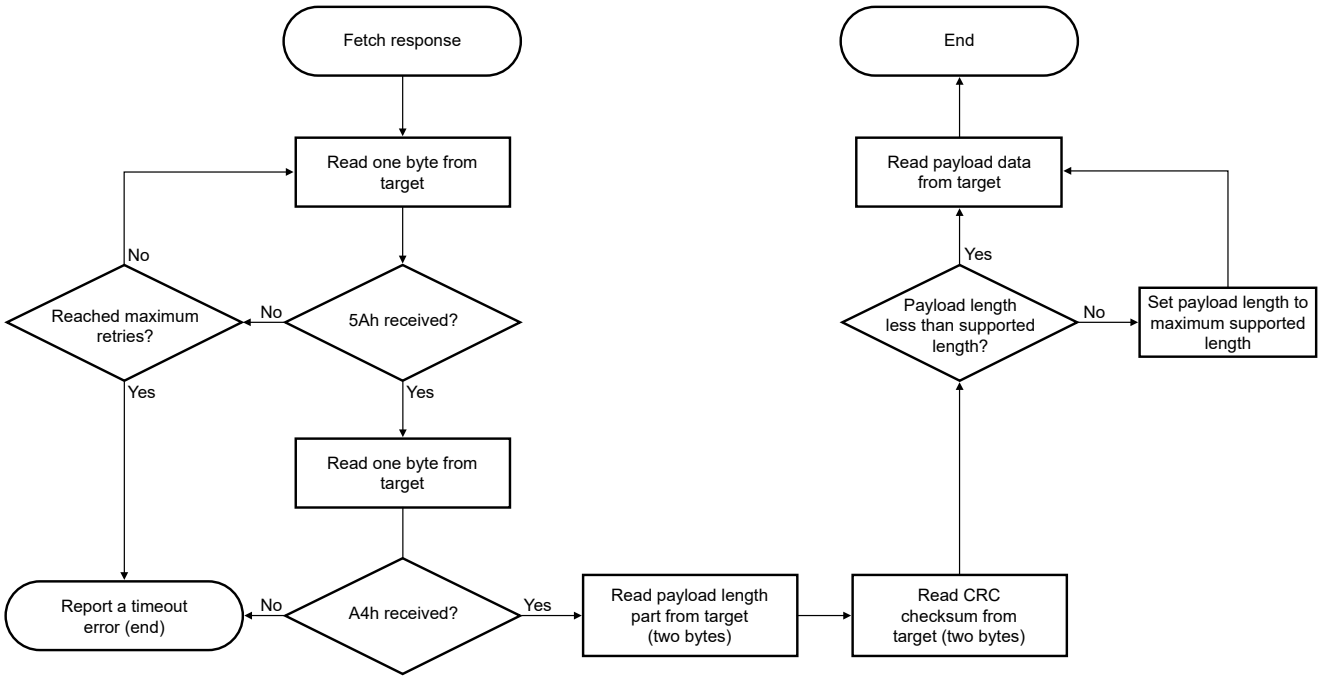


Figure 28. Host reads response from target via I²C

For more information on I²C ISP command, response, and data formats, see Bootloader packet types.

For more information on I²C ISP commands, see Bootloader command set.

7.9 SPI ISP

7.9.1 Introduction

The bootloader supports loading data into flash memory via the SPI peripheral, where the SPI peripheral serves as an SPI target. The SPI transfer should be in SPI mode 3 with 8 data bits.

The maximum supported baud rate of the SPI depends on the core clock frequency when the bootloader is running. The typical baud rate is 2000 kbit/s with the factory settings. The actual baud rate is lower or higher than 2000 kbit/s, depending on the actual value of the core clock.

Because the SPI peripheral serves as an SPI target device, the host must start each transfer and must fetch each outgoing packet.

The transfer on SPI is slightly different from I²C:

- The host receives 1 byte after it sends out any byte.
- Received bytes must be ignored when the host is sending out bytes to the target.
- The host starts reading bytes by sending 00h to the target.

The target sends the byte 00h as a response to the host if the target is under the following conditions:

- Processing incoming packet
- Preparing outgoing data
- Received invalid data

The bootloader also supports the active notification pin (nIRQ pin) to notify the host processor that it is busy or ready for new commands or data.

To accelerate the SPI transfer between the host and the bootloader, the bootloader provides the nIRQ pin, which you enable with the SetProperty command. After you enable it, the host must wait until it sees a negative edge on the nIRQ pin before reading any data from the bootloader. It must also wait until the nIRQ pin is high before sending any data to the bootloader.

See the figure below for the typical physical connection between the host and the bootloader device.

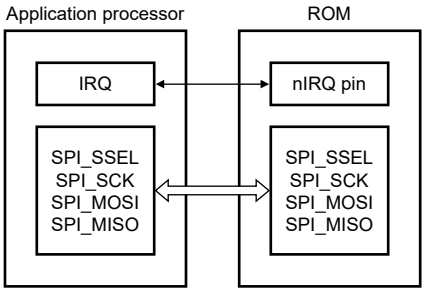


Figure 29. Physical interface for SPI ISP

The following flowcharts show how the host reads a ping response, an ACK, and a command response from the target via SPI, without the nIRQ pin enabled.

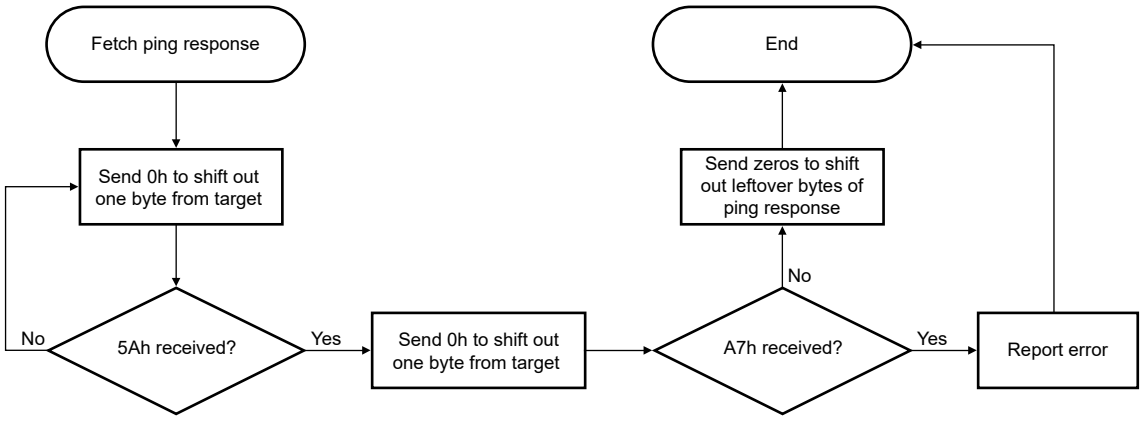


Figure 30. Host reads ping packet from target via SPI

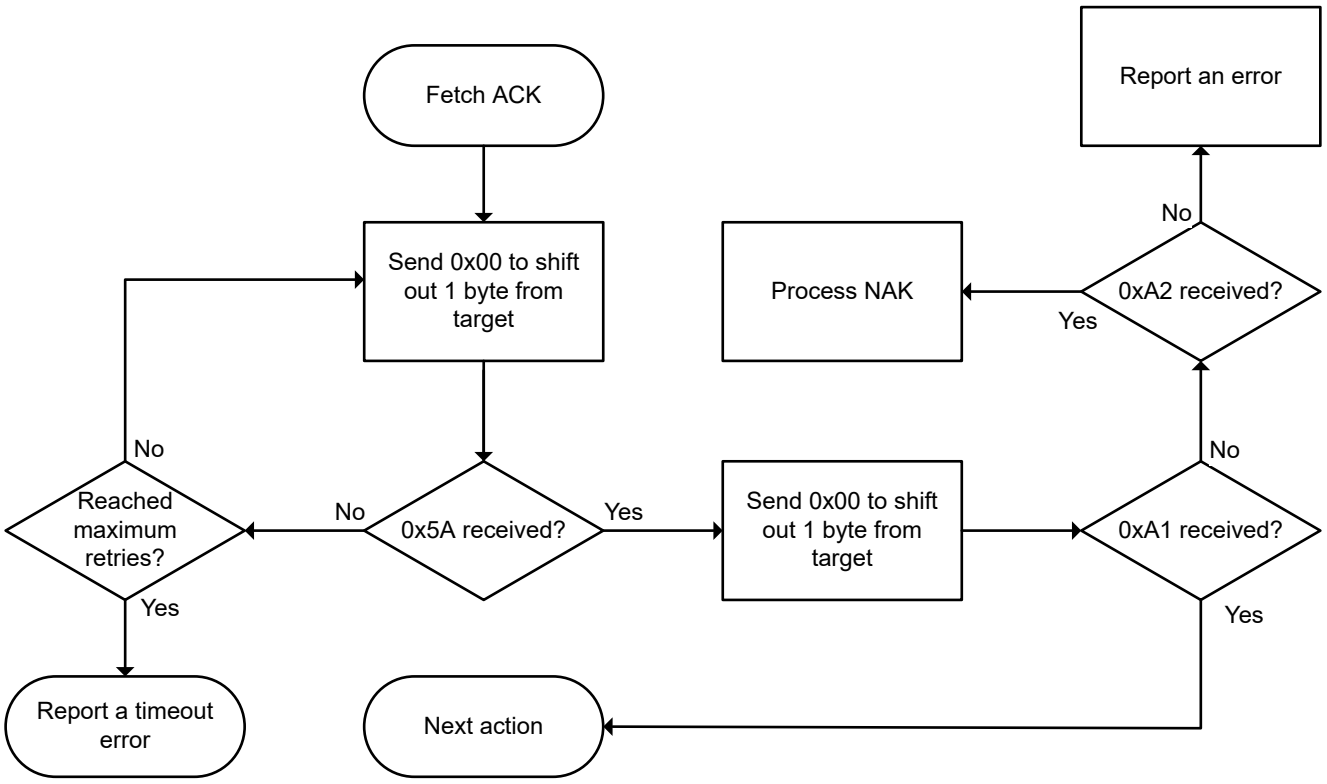


Figure 31. Host reads ACK from target via SPI

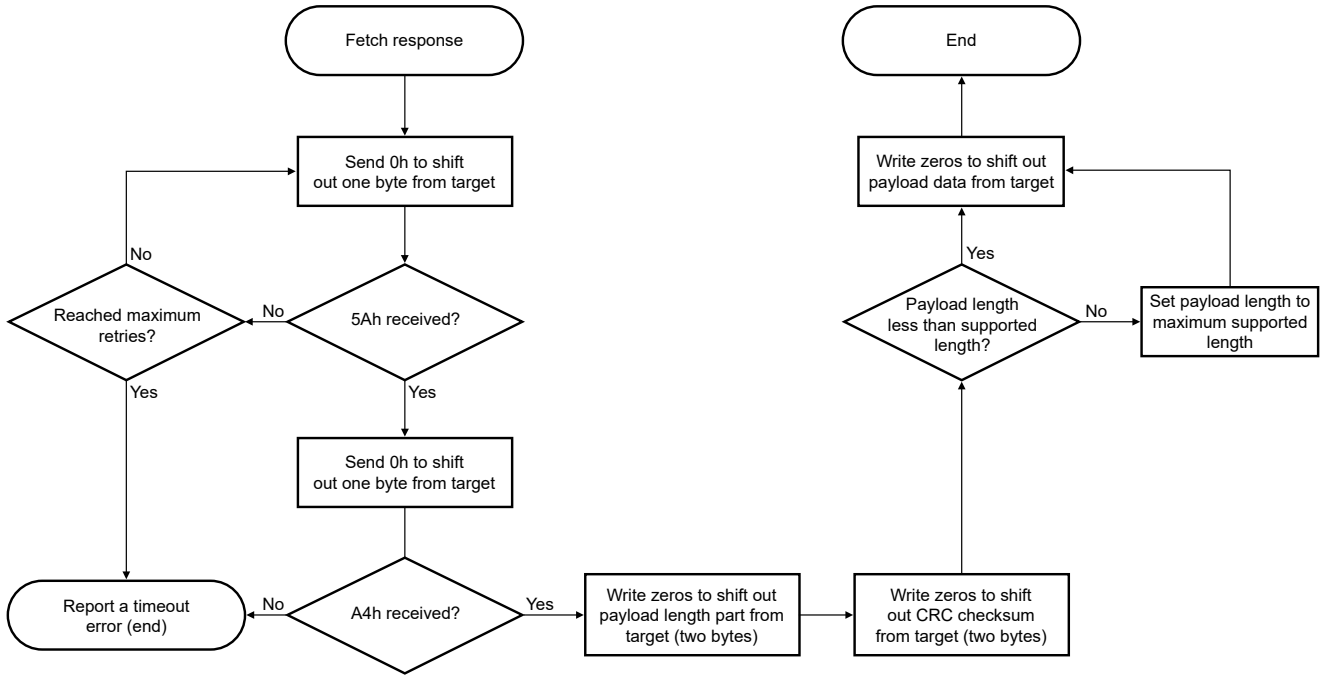


Figure 32. Host reads response from target via SPI

For more information on the command, response, and data formats of SPI ISP, see Bootloader packet types.

For more information on SPI ISP commands, see Bootloader command set.

7.9.2 Host read ACK(5Ah, A1h) packet timing restriction

If the ISP protocol does not use the IRQ pin, then the host must use the data-fetching timing for ACK from the chip:

- After sending out the command, the host must wait for at least 100 µs before polling the start byte 5Ah of the ACK packet sent by the chip. If the fetched data is not 5Ah, delay 100 µs and poll to reread 5Ah.
- For the ACK packet (5Ah, A1h): After the host receives 5Ah, you must add a 50 µs delay before fetching the A1h from the chip.

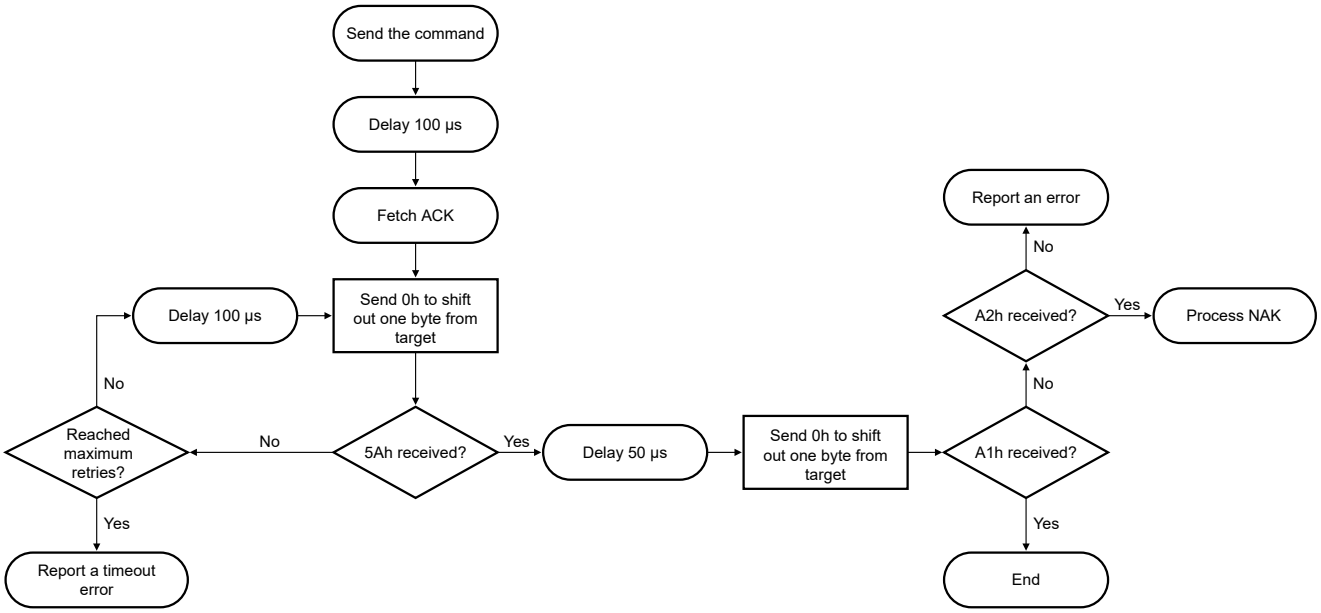


Figure 33. Host read ACK(5Ah, A1h) packet timing restriction flow

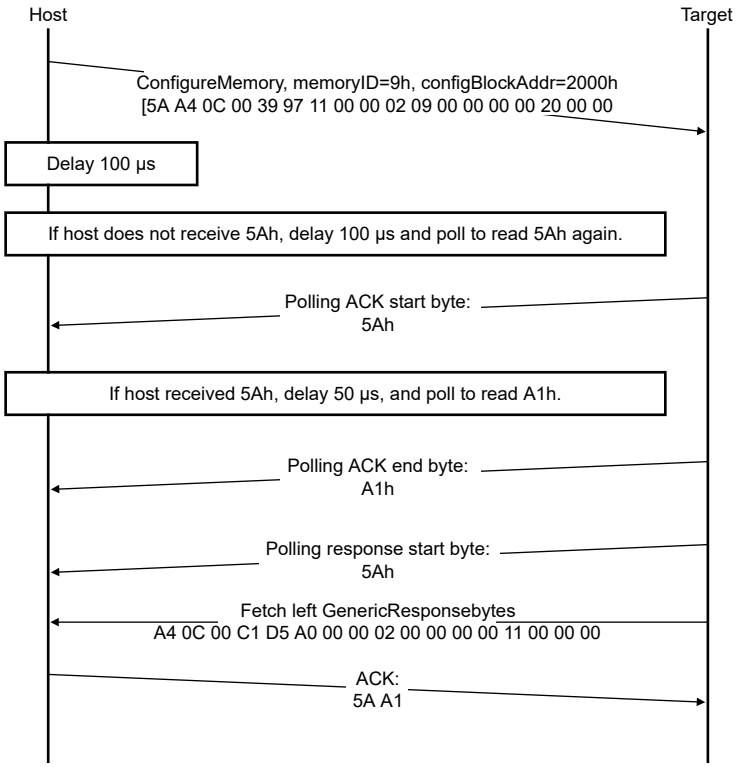


Figure 34. Host read ACK(5Ah, A1h) packet timing restriction

7.10 USB ISP

7.10.1 Introduction

The bootloader supports ISP using the USB peripheral. The target is implemented as a USB human interface device (HID) class.

USB HID does not use framing packets when transferring data through the USB HID class. Instead, USB HID uses the packetization inherent in the USB protocol itself. The chip's ability to NAK out transfers (until they can be received) provides the required flow control. The built-in CRC of each USB packet provides the required error detection.

7.10.1.1 Device descriptor

The bootloader configures the default USB VID, PID, and strings as follows:

Default VID and PID:

- VID = 1FC9h
- PID = 014fh

Default strings:

- Manufacturer [1] = "NXP SEMICONDUCTOR INC"
- Product [2] = "USB COMPOSITE DEVICE"

You can customize the USB VID, PID, and strings using the CMPA of the flash memory. For example, you can customize the USB VID and PID by writing:

- The new VID to the CMPA location 0100_4034h in flash memory
- The new PID to the usbPid field of the CMPA in flash memory

7.10.1.2 Endpoints

The HID peripheral uses three endpoints:

- Control (0)
- Interrupt IN (1)
- Interrupt OUT (2)

The interrupt OUT endpoint is optional for HID class devices. The chip bootloader uses the interrupt OUT endpoint as a pipe, where the firmware can NAK send requests from the USB host.

7.10.1.3 HID Reports

The bootloader USB HID peripheral defines and uses four HID reports. The report ID determines the direction and type of packet sent in the report—otherwise, the contents of all reports are the same.

Table 141. HID reports assigned for the bootloader

Report ID	Packet type	Direction
1	Command	OUT
2	Data	OUT
3	Command	IN
4	Data	IN

Each report has a maximum size of 60 bytes. The maximum payload size is 56 bytes. In addition, there is a 4-byte report header that indicates the length (in bytes) of the payload and a report ID sent to the packet.

The actual data sent in all reports uses the following format:

Table 142. Data format sent in USB HID packet

Report ID	Data
0	Report ID
1	Padding
2	Packet length LSB
3	Packet length MSB
4	Packet[0]
5	Packet[1]
6	Packet[2]
...	...
N+4-1	Packet[N-1]

If the HID report descriptor defines more than one report, the report ID must be included in the data. The actual data sent and received has a maximum length of 35 bytes. The packet length header is in little-endian format and it is set to the size (in bytes)

of the packet sent in the report. This size does not include the report ID or the packet length header itself. During a data phase, a packet size of 0 indicates a data phase abort request from the receiver.

For more information on USB ISP command and response formats, see Command packet.

For more information on USB ISP data format, see Response packet.

For more information on USB ISP commands, see Bootloader command set.

7.11 CAN ISP

The bootloader supports loading data into flash through the CAN peripheral. It supports four predefined auto-detect speeds on CAN transferring:

- 125 KHz
- 250 KHz
- 500 KHz
- 750 KHz
- 1 MHz

The current CAN IP can support up to 1 MHz using the ROM ISP feature, so the default baud rate is set to 1 MHz.

In host applications, you can specify the baud rate for CAN by writing one of the values in the following list to (0100_4028[31:28]) in CMPA:

- 0000: Auto baud detection (125 KHz, 250 KHz, 500 KHz, 750 KHz, 1 MHz)
- 0001: 125 kbit/s
- 0010: 250 kbit/s
- 0011: 500 kbit/s
- 0100: 750 kbit/s
- 0101 and above: 1000 kbit/s and auto baud detection

Initially, the bootloader enters the listen mode with the default speed of 1 MHz. After the host sends a ping to a specific node, it generates traffic on the CAN bus. Because the bootloader is in a listen mode, it can check whether the local node speed is correct by detecting errors. If there is an error, some traffic is visible, but it may not be at the right speed to read the correct data. If this happens, the speed setting changes and checks for errors again. No errors indicates that the speed is correct.

The settings change back to the normal receiving mode to check if there is a package for this node. It then stays in this speed until another host using another speed tries to communicate with any node. It repeats the process to detect a correct speed before sending host time-out and aborting the request.

The host side should have a reasonable time tolerance during the auto-speed detection period. If it sends a time-out signal, that indicates either there is no response from the specific node, or there is a real error and it must report the error to the application.

This flowchart demonstrates the communication flow for how the host reads the ping packet, ACK, and response from the target.

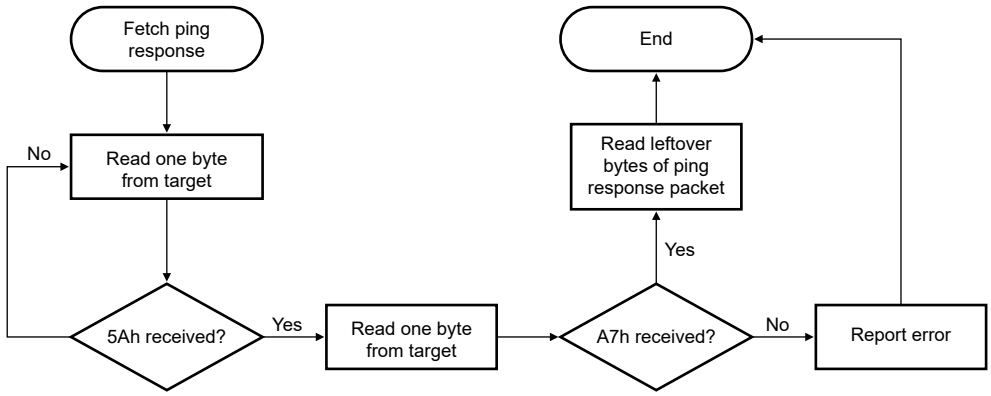


Figure 35. Host reads ping response from target via CAN

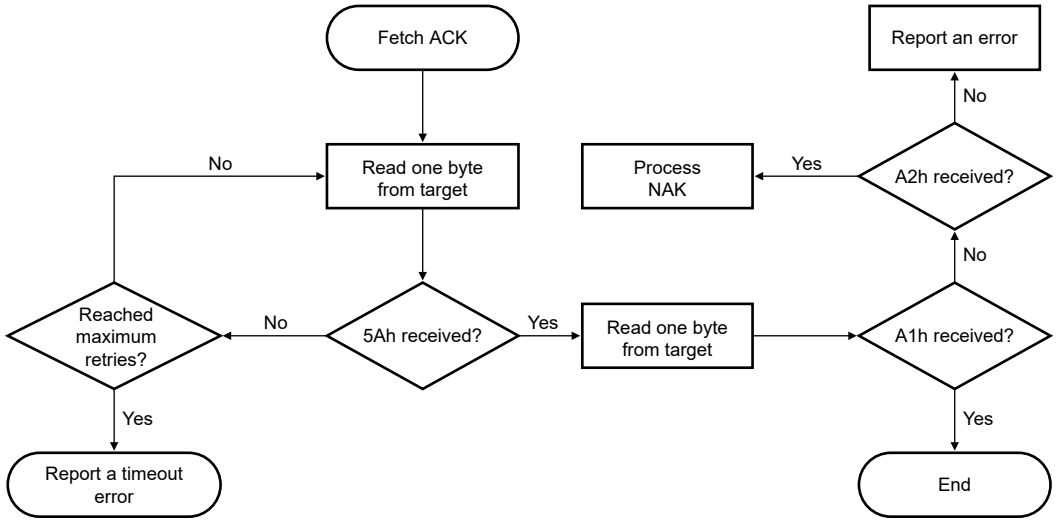


Figure 36. Host reads ACK packet from target via CAN

Chapter 8
Non-secure Boot ROM

8.1 Overview

This chip supports both on-chip flash memory image boot and external NOR flash memory image boot via the FlexSPI interface.

You can use the boot ROM to erase, program, and read the on-chip or external flash memory, which means you can use the boot ROM to download the boot image into the on-chip or external flash memory via the ISP interfaces.

Following are the boot modes available for the boot ROM:

- Master boot mode
- ISP boot mode
- Recovery boot mode

The boot ROM uses the ISP pins or CMPA configuration to select Master boot mode for on-chip flash memory boot and FlexSPI boot (see Table 144 for more information).

Also, the boot ROM takes responsibility of the boot flow. It selects whether to boot from on-chip flash memory, external flash memory, or ISP mode. It even supports secure boot such as image authentication by ECDSA (P256, P365) or CMAC (128 bits).

There are many boot-related parameters in customer manufacturing/factory programmable area (CMPA) and customer in-field programmable area (CFPA). You can use the boot ROM to update these settings in ISP mode or by using ROM API.

For all details related to CMPA and CFPA, including their fields and address settings, see the IFR map file MCXNx4x_IFR_Fusemap.xlsx, attached to this document.

NOTE

This chapter is also available in the *MCX Nx4x* Reference Manual, with differences related to security functionality. The chapter in the Reference Manual does not show the security relevant information in it.

NOTE

IFR and PFR refer to "Protected Flash Regions" in this document.

8.1.1 Features

The boot ROM (256 KB on-chip with bootloader) allows various boot options and APIs. It supports:

- Automatic booting from internal flash memory, based on ISP pins or the CMPA setting in the IFR.
- In-application programming (IAP) calls (see IAP APIs for more information).
- Flash memory API for programming internal and external FlexSPI NOR flash memory. See the following for more information:
  - FLASH APIs
  - FLEXSPI FLASH Driver APIs
- SPI flash memory recovery boot from 1-bit SPI flash memory devices (see SPI flash memory recovery for more information).

8.2 Functional description

8.2.1 Boot mode and ISP download modes based on the ISP pin

The internal boot ROM memory stores the boot code. After a reset, the Arm processor starts its code execution from this memory. You must execute the bootloader code every time the part is powered on, reset, or wakes up from a deep power-down mode.

Images must be stored in the flash memory or in the serial NOR flash memory connected to the FlexSPI controller. The code is then validated, and the boot ROM vectors to on-chip or off-chip flash memory.

Depending on the values of the CMPA fields, ISP pins, and image header type definition, the bootloader decides whether to boot from flash memory, off-chip flash memory, or run into ISP mode. See IFR region definitions. The boot ROM reads the status of the ISP pins to determine the boot source. See Table 143 for details.

Table 143. Boot mode and ISP download modes based on the ISP pin

Boot mode	ISPMODE_N (P0_6 pin)	Description
Flash memory boot	High	The boot ROM looks for a valid image in the flash memory. If it does not find a valid image, it looks for one in the recovery boot device, if the recovery boot is enabled (9:8, word 0 in CMPA). If it still fails to find a valid image, the boot ROM enters ISP boot mode based on the ISP_BOOT_IF bits defined in Table 144.
ISP boot	Low	One of the serial interfaces (UART, I^2C, SPI, USBFS0, or USBHS1) is used to download the image from the host into the flash memory. The first valid probe message on UART, I^2C, SPI, CAN, or USB becomes inactive on that interface.

NOTE

For the default USB HS ISP mode, you require crystal 24 MHz that you can configure in CMPA. For USB FS ISP mode, the boot ROM uses FRO.

8.2.2 ISP download mode based on the ISP_BOOT_IF CMPA bits

Table 144. ISP download mode based on the ISP_BOOT_IF CMPA bits (word0[6:4])

ISP boot mode	ISP_MODE_2	ISP_MODE_1	ISP_MODE_0	Functionality
Auto ISP	0	0	0	The boot ROM probes the active peripheral from one of the following serial interfaces and downloads the image from the probed peripherals: • UART • I^2C • SPI • CAN • USBFS0 • USBHS1
UART ISP	0	0	1	UART is used to download the image.
SPI ISP	0	1	0	The SPI slave is used to download the image.
I^2C slave ISP	0	1	1	The I^2C slave is used to download the image.
USB0 slave ISP	1	0	0	The USB HID class is used to download the image of the USB0 port.

Table continues on the next page...

Table 144. ISP download mode based on the ISP_BOOT_IF CMPA bits (word0[6:4]) (continued)

ISP boot mode	ISP_MODE_2	ISP_MODE_1	ISP_MODE_0	Functionality
USB1 slave ISP	1	0	1	The USB HID class is used to download the image of the USB1 port.
FlexCAN slave ISP	1	1	0	The FlexCAN slave is used to download the image.
Disable ISP	1	1	1	ISP mode is disabled.

8.2.3 Boot ROM pin assignments

Table 145 shows ISP pin assignments (the default ones) that the boot ROM code uses. You can change these assignments by using the CMPA settings (see the IFR map file attached to this document for more information).

Table 145. Boot ROM pin assignments

ISP pin	Port pin assignment
ISPMODE_N	P0_6
UART ISP mode	
FC4_UART_TXD	P1_9
FC4_UART_RXD	P1_8
I^2C ISP mode	
FC0_I2C_SDA	P0_16
FC0_I2C_SCL	P0_17
SPI Flash Memory Recovery mode	
FC7_SPI_SDO	P3_9
FC7_SPI_SDI	P3_8
FC7_SPI_PCS0	P3_0
FC7_SPI_SCK	P3_7
SPI ISP mode	
FC3_SPI_SCK	P1_1
FC3_SPI_PCS	P1_3
FC3_SPI_SDO	P1_0
FC3_SPI_SDI	P1_2

Table continues on the next page...

Table 145. Boot ROM pin assignments (continued)

ISP pin	Port pin assignment
USBFS0 and USBHS1 ISP mode	
USB0_VBUS	P4_12
USB0_DP	—
USB0_DM	—
USB1_VBUS	—
USB1_DP	—
USB1_DM	—
FlexCAN ISP mode	
CAN_TXD	P1_10
CAN_RXD	P1_11
FlexSPI pins	
FLEXSPI0_A_SS0_b	P3_0
FLEXSPI0_A_SCLK	P3_7
FLEXSPI0_A_DQS	P3_6
FLEXSPI0_A_DATA0	P3_8
FLEXSPI0_A_DATA1	P3_9
FLEXSPI0_A_DATA2	P3_10
FLEXSPI0_A_DATA3	P3_11
FLEXSPI0_A_DATA4	P3_12
FLEXSPI0_A_DATA5	P3_13
FLEXSPI0_A_DATA6	P3_14
FLEXSPI0_A_DATA7	P3_15
FLEXSPI0_B_SS0_b	P2_2
FLEXSPI0_B_SCLK	P2_3
FLEXSPI0_B_DQS	P2_1

Table continues on the next page...

Table 145. Boot ROM pin assignments (continued)

ISP pin	Port pin assignment
FLEXSPI0_B_DATA0	P2_4
FLEXSPI0_B_DATA1	P2_5
FLEXSPI0_B_DATA2	P2_6
FLEXSPI0_B_DATA3	P2_7
FLEXSPI0_B_DATA4	P2_8
FLEXSPI0_B_DATA5	P2_9
FLEXSPI0_B_DATA6	P2_10
FLEXSPI0_B_DATA7	P2_11
FLEXSPI_RESET	See the IFR map file attached to this document.

8.2.4 Image header

Top-level boot flow shows the top-level boot process, which starts after the reset is released.

The CPU clock has a default frequency of 48 MHz, based on 144 MHz FRO by default. You can set this frequency to 72 MHz, 144 MHz, or 150 MHz by using the BOOT_SPEED setting in CMPA. When the Cortex-M33 core starts the bootloader, the SWD access is disabled; and therefore, the debugger cannot connect to the CPU during this period. The boot ROM determines the boot mode based on the reset state of the ISP pins.

NOTE

If VDD_CORE is supplied from an external source, then the 48 MHz BOOT_SPEED option must be used and VDD_CORE must be in the mid voltage (1.0 V nominal) range.

After completing boot mode detection, the bootloader starts validating the vector table and image header if the image is present in the boot media (internal or external flash memory).

The boot ROM does the following for image validity check:

- Validates the image using header and CMPA settings when secure boot is enabled. See the "Secure Boot ROM" chapter in the *Security Reference Manual* for details.
- Validates the image using CRC32 when secure boot is not enabled, if the CRC check is present in the image header.
- Validates whether SP and PC are valid RAM and NVM addresses of the chip.
- Validates the TZ-M image type if the aforementioned image checks pass.

The beginning of the image follows the format described in Table 146. The bootloader begins scanning for user images by examining the image type marker located at offset 24h. If the value matches any supported image type markers, the image header's validation begins, and after its completion, the qualification continues by examining the TZM "imageType" field:

- If it is a CRC image, you use the "image length" field value as the length to perform a CRC calculation based on the image in the flash memory (see Table 146 for more). The CRC calculation begins at offset 0h from the beginning of the image sector and continues up to the number of bytes specified by the image length, which does not include the "offsetToExtendedHeader" field that makes up the CRC value field. This means that the calculated CRC skips the CRC value field. The result is then compared to the "offsetToExtendedHeader" entry in the structure and the image is considered valid if a match exists; otherwise, the image is considered invalid. The CRC calculation is not performed if the image is not a CRC image.

- If it is a signed image, the "image length" field value is used as the length to perform authentication (ECDSA and CMAC), which is performed on the image in the internal or external flash memory. The authentication begins at offset 0h from the beginning of the image sector, and continues up to the number of bytes specified by the image length. The "offsetToExtendedHeader" field value points to the offset that holds the certificates.

Table 146. Image header

Offset (hex)	Size (bytes)	Symbol	Description
00	4	Initial SP	Stack pointer
04	4	Initial PC	Application's first execution instruction
08	24	Vector table	Cortex-M33 core's vector table entries
20	4	image length	Length of the current image (total length, including the signature), which is set to the actual image length if the image type is another value
24	4	imageType	Image type, bit[7:0]: • 0h: Plain image • 2h: Plain image with CRC • 4h: Signed image • 5h: Plain image with CRC • 6h: SB3 manifest Load_to_RAM image: The image has a nonzero image_load_address and image_length in the image header. Other values are reserved. Bit[10]: Indicates whether the image version is included in ImageType[31:16]: • 0: Image version is not included in "imageType." • 1: Image version is included in "Image Type," applicable to the internal flash memory XIP use case only. Bit[13]: Indicates whether the image includes TZM preset data: • 0: Image does not contain TZM preset data. • 1: Image contains TZM preset data. Bits[31:16]: Image version (for on-chip flash memory) when bit[10] = 1
28	4	offsetToExtendedHeader	Offset to extended header: • For a signed image (imageType = 4h), this is the offset to the certificate header block • For a CRC image (imageType = 2h or 5h), this is the crcChecksum value
2C	8	Vector table	Cortex-M33 core's vector table entries

Table continues on the next page...

Table 146. Image header (continued)

Offset (hex)	Size (bytes)	Symbol	Description
34	4	imageExecutionAddress	The execution address of the image is: • Set to 0 if image type is XIP • Set to actual image execution address if the image type is loaded to RAM
38	—	Vector table	Cortex-M33 core's vector table entries

8.2.5 Top-level boot flow

The following figure shows the top-level boot process, which begins after the reset is released.

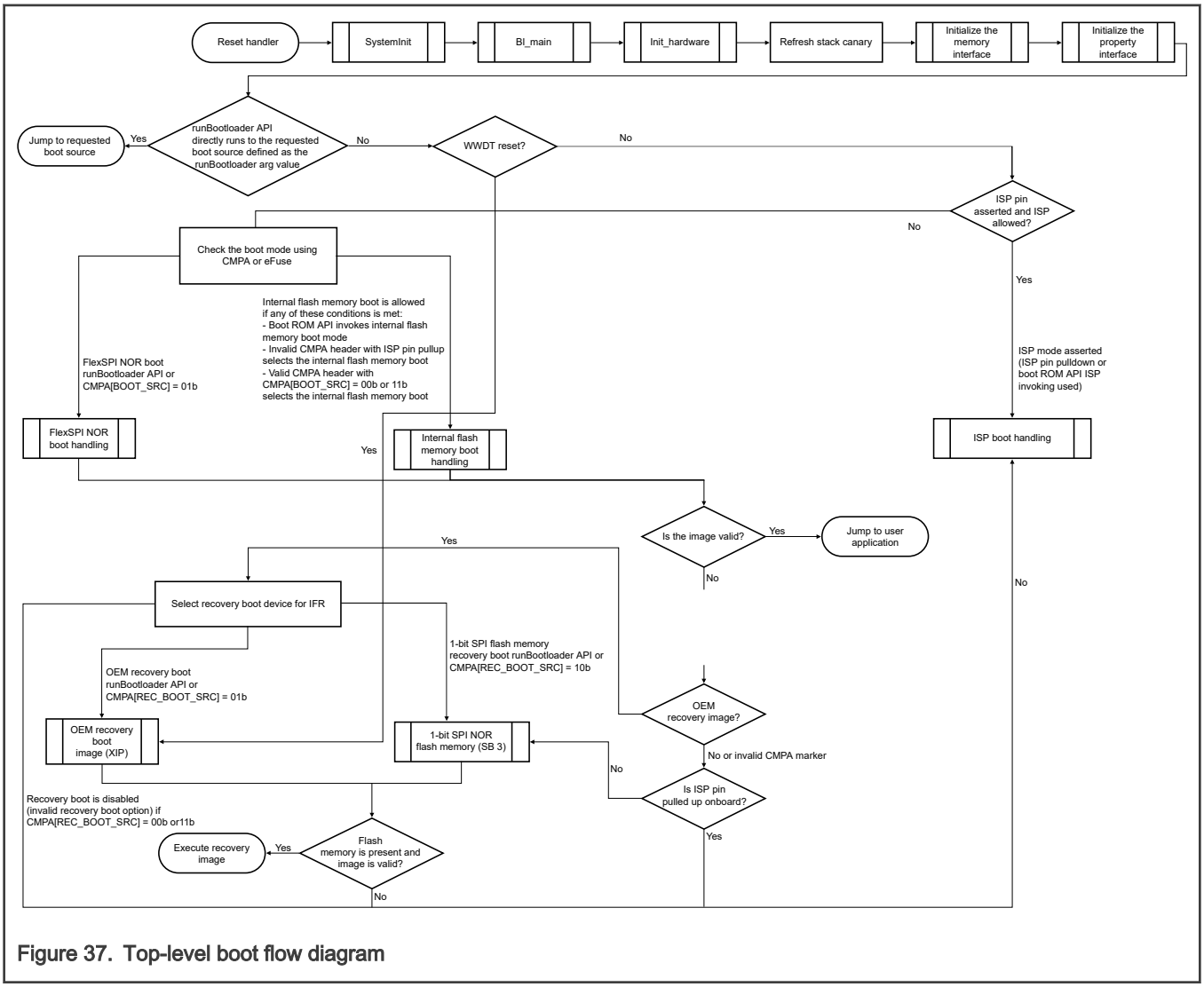


Figure 37. Top-level boot flow diagram

8.2.6 SPI flash memory recovery

Support is provided for a recovery boot from an external 1-bit SPI flash memory device in which the SB3.1 image is stored. The SB3.1 file is an encrypted and signed command script file that supports programming flash memory, IFR, and other configuration

commands. You can implement this feature during OTA using a recovery media model in which an external SPI flash memory is used to store a factory image in the SB3.1 format. If the image in the main flash memory is corrupted, the boot ROM attempts to recover the chip by booting or executing the SB3.1 file that is present on the external flash memory device.

See the "ROM firmware update using SB file" section in the "Secure Boot ROM" chapter in the MCX Nx4x Security Reference Manual for details related to the SB3.1 file format.

When booting from an internal or external flash memory using FlexSPI, if the flash memory image is deemed invalid, the chip checks REC_BOOT_SRC (bits 9:8) in the CMPA BOOT_CFG field (0100_4000h) to determine whether SPI flash memory recovery is selected.

The SB file can program the image into internal flash memory. The following commands are available for SB file recovery mode:

```
#define SBLOADER_V3_CMD_SET_IN_REC_MODE \\
    ((1u << kSB3_CmdErase) | (1u << kSB3_CmdLoad) | (1u << kSB3_CmdExecute) |
 (1u << kSB3_CmdProgramFuse) |      \\
    (1u << kSB3_CmdProgramIFR) | (1u << kSB3_CmdCopy) | (1u << kSB3_CmdLoadKeyBlob) |
 (1u << kSB3_CmdConfigMem) | \\
    (1u << kSB3_CmdFillMem))| (1u << kSB3_CmdFwVerCheck))
```

Code Listing 1. Commands for SB file recovery mode

See 1-bit serial NOR flash memory through SPI for more information.

8.3 Boot modes

8.3.1 Master boot mode

Master boot mode supports the following boot devices:

- Internal flash memory boot
- FlexSPI NOR flash memory boot
- SPI 1-bit NOR recovery boot
- Secondary bootloader boot

Table 147. Image offsets for different boot media

Boot media	Image offset
Internal flash memory boot	0h
FlexSPI NOR flash memory boot	1000h
SPI 1-bit NOR recovery boot	0h
Secondary bootloader boot	0h

8.3.1.1 Internal flash memory boot

The CPU clock is set to the boot speed (that the BOOT_SPEED field specifies) in the CMPA region and boots directly from the internal flash memory or external FlexSPI NOR flash memory based on the BOOT_SRC settings in CMPA. After you select the internal flash memory boot as the boot media, the boot ROM tries to find a valid boot image in the internal flash memory. For more information, see Boot flow.

8.3.1.1.1 Boot flow

Figure 38 shows internal flash memory boot flow:

- The CMPA[FLASH_REMAP_SIZE] field specifies the second image in terms of size. If the field is 0, it indicates that the second image is not present.
- The Secure_FW_Version field [31:0], word 2 in CFPA specifies the minimum allowed image version.
- If the image version flag at imageType[10] is set, the image version can be found at imageType[31:16]; otherwise, the image version is treated as 0.
- In the image list, the newer one is image 0, and the older one is image 1. If the image version information is missing or equal, the image starting from address 0 is treated as image 0, and the image that FLASH_REMAP_SIZE specifies is treated as image 1, which starts from the second bank of the flash memory when the flash memory swap is enabled.

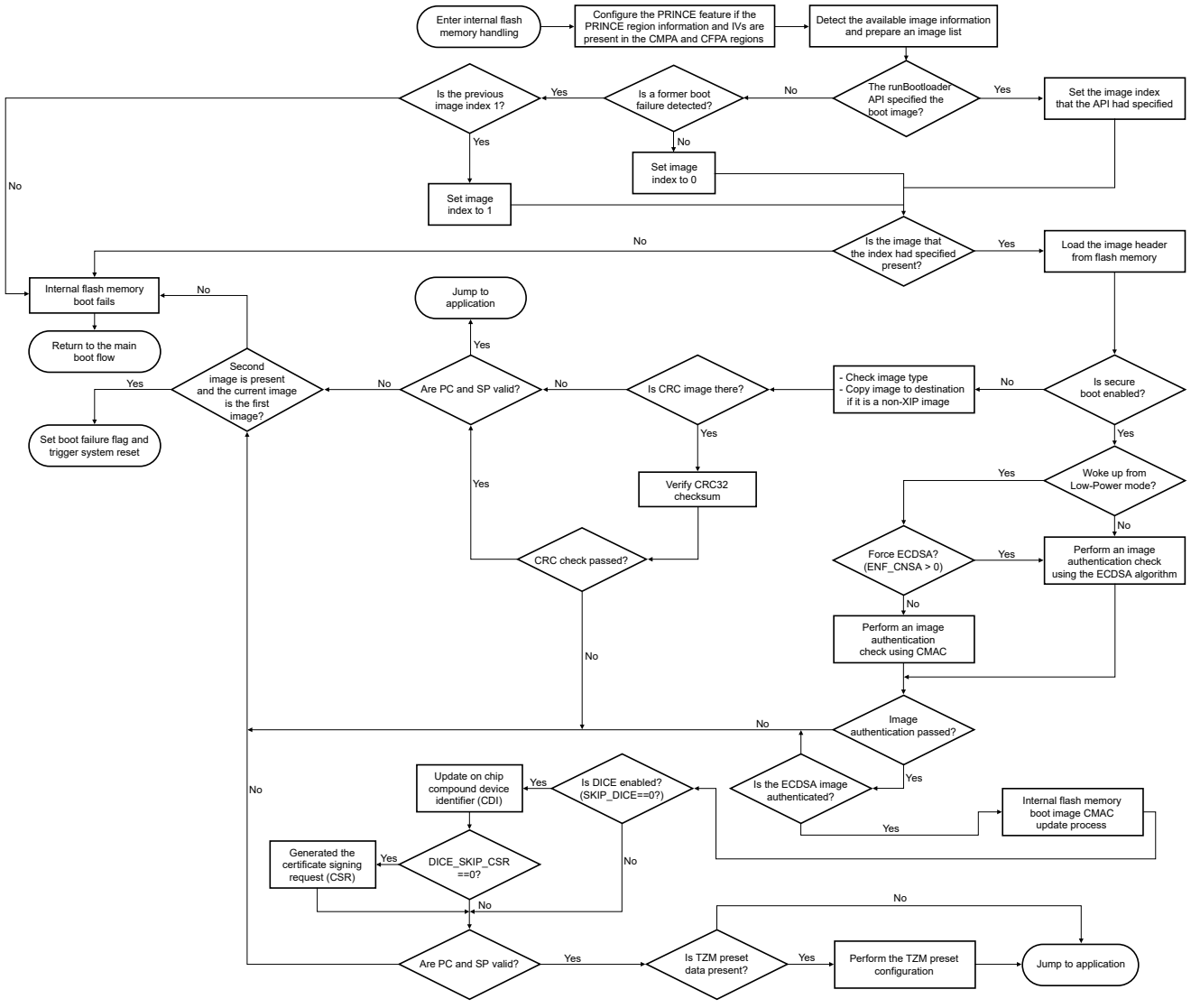


Figure 38. Internal flash memory boot flow

8.3.1.1.2 Encryption and decryption using PRINCE

See ISP commands to enable PRINCE for information on how to use the PRINCE feature to boot and execute from encrypted internal flash memory, and encrypt the data programmed to internal flash memory.

8.3.1.1.3 Internal flash memory dual image boot

The boot ROM supports dual image boot for internal flash memory. This means that two boot images can be placed in the flash memory region. The boot ROM decides which image to boot first based on the image version. It boots the one with the newer image version first, and in case of a failure, it boots the older one. The image version is the image header offset, 24h (see Table 146). Bit 10 shows whether the image contains the image version; if bit 10 is 0, it means that the image has no image version. In this case, the boot ROM considers the image version as 0.

Table 148. Internal flash memory boot image version (image header offset: 24h)

image_type	Field name	Field description
Bits 31–16	Image version	Active when the remap feature is enabled (remap offset and remap size are not zero in CMPA)
Bit 10	IMG_VER_INCLUDED_IN_IMG_TYPE	If this field is 0, the image has no version. If this field is 1, the image has a version in the dual image boot.

8.3.1.1.3.1 Remap feature

The following figure shows the internal flash memory controller remap function. When FLASH_REMAP_SIZE is a nonzero value in CMPA, the internal flash memory AHB access changes the access address based on the current active flash memory bank number. For example, when FLASH_REMAP_SIZE = 2h, and the remap size is 96 KB ((2h + 1) × 32 KB), access to 0h is remapped to flash memory bank1 start offset 0 when the bank1 image is active. The boot ROM can implement a dual image boot with two images via this feature. You set the remap size of the image in the CMPA and eFuse regions.

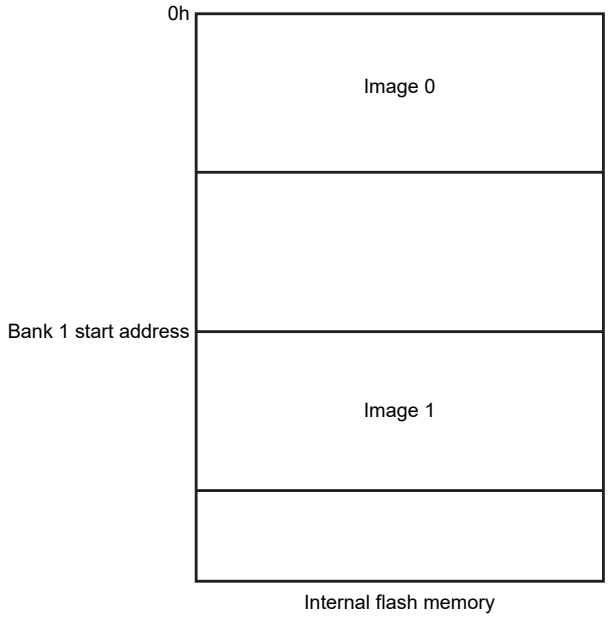


Figure 39. Internal flash memory dual image layout

8.3.1.1.3.2 Remap settings

The following table shows the internal flash memory dual image boot remap setting in CMPA. You must perform the settings defined in this table in CMPA to let the boot ROM know where the second image is placed. You can, then, enable dual image boot by setting the dual image boot image version in the image header, according to the information provided in the aforementioned sections. For more information about the internal flash memory boot flow, see Boot flow.

Table 149. Boot image 1 remap size

Region	Address	Word name	Description
CMPA	0100_4004h	FLASH_REMAP_SIZE	Indicates the flash memory remap size from the flash memory bank1 address, 0h, to the boot ROM.
eFuse	Index: 25	FLASH_REMAP_SIZE[4:0]	Indicates the flash memory remap size from the flash memory bank1 address, 0h, to the boot ROM. Also, nonzero FLASH_REMAP_SIZE[4:0] in eFuse replaces the FLASH_REMAP_SIZE[4:0] data in the CMPA field.

8.3.1.1.3.3 XOM access control settings

In some scenarios, you may need to set some internal flash memory regions as execute-only-memory (XOM) or read-only memory (ROM), or hide flash memory regions to protect the code.

For example, if ACL_SEC_0 = 2, it means that the first sector (32 KB) of the flash memory global access control (GLBAC*n*) index is 2. As shown in Table 150, this sector is set as ROM.

Table 150. Flash memory MBC setting index

GLBAC*n* index	Read	Write	Execute	Lock	Description
0	1	1	1	0	Default flash memory behavior (R/W/X unlocked)
1	1	1	0	1	Data flash memory with this setting: R/W + locked
2	1	0	1	1	ROM with this setting: RX + locked
3	1	0	0	1	Data read-only memory (DROM) with this setting: R + locked
4	1	0	1	0	ROM with this setting: RX unlocked
5	0	0	1	0	XOM with this setting: XOM unlocked
6	0	0	1	1	XOM with this setting: XOM + locked
7	0	0	0	1	Hidden (no access + locked)

Follow below rules to change the ACL_SEC_*n* fields.

- After you select a locked access level, the subsequent updates of this field can be done with higher lock level only.
  - If current sector ACL value (GLBAC*n* index) is greater than the new value, then it is permitted except if the current value is 4 or 5.
    - 7 (___L) > 6 (__XL) > 3 (R__L) > 2 (R_XL) > 1 (RW_L) > 0 (RWX_)
  - If current sector ACL value is 0, 4, or 5 then any new value is permitted.

The boot ROM supports setting flash memory attributes in the CFPA region. Before jumping into the user code, the boot ROM loads the settings of access control for the flash memory sector that you specify in CFPA. Table 151 shows access control settings for flash memory in CFPA.

Table 151. Access control settings for the flash memory sector

CFPA word	Flash memory address (hex)	Word name	Description
Word48	0100_00C0	FLASH_ACL_0_7	Selects the flash memory MBC setting index. See Table 150 for more information.
Word49	0100_00C4	FLASH_ACL_8_15	Same as above
Word50	0100_00C8	FLASH_ACL_16_23	Same as above
Word51	0100_00CC	FLASH_ACL_24_31	Same as above
Word52	0100_00E0	FLASH_ACL_32_39	Same as above
Word53	0100_00E4	FLASH_ACL_40_47	Same as above
Word54	0100_00E8	FLASH_ACL_48_55	Same as above
Word55	0100_00EC	FLASH_ACL_55_63	Same as above

8.3.1.1.3.4 ISP commands to enable PRINCE

PRINCE mode is used for real-time encryption of data that is written to on-chip flash memory and decryption of encrypted flash memory data, during read, to allow asset protection. You can enable PRINCE during internal flash memory update when performing ISP programming for internal flash memory.

See Configuration parameters for the PRINCE region for PRINCE enable parameters, and perform the procedure in Table 152 to enable PRINCE.

Table 152. ISP commands to enable PRINCE

Step number	Step	ISP command to perform the step
1	Select PRINCE region0 and set the encryption from 0h to 8000h. (50h is a fixed tag for PRINCE; last two bits of "0x50000000" indicate 0:b00 (prince region number)).	`blhost -p comxx -- fill-memory 0x2002_0000 4 0x50000000`
2	Set the PRINCE encryption region start address.	`blhost -p comxx -- fill-memory 0x2002_0004 4 0x0`
3	Set the PRINCE encryption region length.	`blhost -p comxx -- fill-memory 0x2002_0008 4 0x8000`
4	Enable PRINCE region0.	`blhost -p comxx -- configure-memory 0x0 0x2002_0000`

8.3.1.1.3.5 Configuration parameters for the PRINCE region

```
typedef struct
{
    uint32_t target_prince_region : 2; // 0/1/2/3
    uint32_t reserved : 22;
    uint32_t tag : 8; // Fixed to 0x50 ('P')
} prince_prot_region_option_t;

typedef struct
{
    prince_prot_region_option_t option;
    uint32_t start;
    uint32_t length;
} prince_prot_region_arg_t;
```

Code Listing 2. Configuration parameters for the PRINCE region

8.3.1.1.3.6 Dual image boot with PRINCE regions 0 and 1 enabled

The dual image boot feature supports dual image boot for internal flash memory. The boot ROM includes the PRINCE feature, which means you can use the dual image boot with the PRINCE feature enabled. IV and KEY for each region are different, and even for the same region, IV and KEY vary by chip.

Perform the procedure and use the corresponding commands listed in Table 153 to enable PRINCE region0 for image0 and PRINCE region1 for image1. To erase the flash memory region and program the two boot images, see Table 154.

Table 153. Dual image boot with PRINCE regions 0 and 1 enabled

Step number	Step	Command to perform the step
Sequence of steps and corresponding commands to enable PRINCE region0 for image0		
1	Select the internal flash memory PRINCE region 0 to be enabled.	`blhost -p comxx -- fill-memory 0x2002_0000 4 0x50000000`
2	Set the start address for PRINCE region0.	`blhost -p comxx -- fill-memory 0x2002_0004 4 0x0`
3	Set the size for PRINCE region0.	`blhost -p comxx -- fill-memory 0x2002_0008 4 0x10000`
4	Trigger PRINCE region0 to be enabled.	`blhost -p comxx -- configure-memory 0x0 0x2002_0000`
Sequence of steps and corresponding commands to enable PRINCE region1 for image1		
1	Select the internal flash PRINCE region 1 to be enabled.	`blhost -p comxx -- fill-memory 0x2002_0000 4 0x50000001`
2	Enable PRINCE region1 for image1 (loaded at the bank1 start address).	`blhost -p comxx -- fill-memory 0x2002_0004 4 0x100000`
3	Set the size for PRINCE region1.	`blhost -p comxx -- fill-memory 0x2002_0008 4 0x10000`
4	Trigger PRINCE region1 to be enabled.	`blhost -p comxx -- configure-memory 0x0 0x2002_0000`

Table 154. Erasing the flash memory region and programming the two boot images

Step number	Step	Sequence of commands to perform the step
1	Erase the flash memory region.	`blhost -p comxx -- flash-erase-region 0x0 0x10000` `blhost -p comxx -- flash-erase-region 0x100000 0x10000`
2	Program the two boot images.	`blhost -p comxx -- write-memory 0x0 image0.bin` `blhost -p comxx -- write-memory 0x100000 image1.bin`

NOTE

After you enable the PRINCE feature, the boot ROM does not disable it for nonencrypted boot, which means, if PRINCE is used via the ROM feature and you try to boot without PRINCE, it may cause failure because the boot ROM does not accept lower security level for PRINCE boot.

8.3.1.1.3.7 CMPA settings with PRINCE enabled

The following figure shows CMPA settings for internal flash dual image boot with PRINCE enabled.

Table 155. Remapping settings for dual image

CMPA word	Flash memory address	Word name	Value	Details
WORD1	0100_4004h	FLASH_REMAP_SIZE	1h	Remap size: 10000h

After all settings are updated, the boot ROM follows dual image boot as the boot flow progresses.

8.3.1.2 FlexSPI NOR flash memory boot

The boot ROM supports access to different Quad and Octal SPI NOR flash memory chips from various vendors via the FlexSPI interface using 1-bit, 4-bit (Quad), or 8-bit (Octal) mode. It uses the flash memory configuration block (FCB) located at offset 400h on the flash memory chip or flash memory auto-probe feature that CMPA specifies:

- If FLEXSPI_AUTO_PROBE_EN is blown, the bootloader performs the flash memory auto-probe sequence using parameters blown in:
  - The FLEXSPI_PROBE_TYPE field that defines the flash memory auto-probe type.
  - The FLEXSPI_PROBE_TYPE and FLEXSPI_FLASH_TYPE fields that define the flash memory type.
  - The FLEXSPI_FREQUENCY field that defines the flash memory access speed.

  The auto-probe feature is used for detecting the external flash FCB information automatically. When FLEXSPI_AUTO_PROBE_EN is blown, and the preceding necessary fields are programmed, the boot ROM tries to identify the external flash information. If the identification is successful, it obtains a self-identifying FCB configuration block. If the recognition fails, it continues to attempt to read the user FCB configuration block to obtain external flash information.

  When FLEXSPI_AUTO_PROBE_EN=1, which is only supported on chips that are JESD216-compliant, the boot ROM is able to detect the flash memory information via the read SFDP (5Ah) command. The ROM uses the SFDP information to construct an FCB instead of using one from the application image.

- If FLEXSPI_AUTO_PROBE_EN is not blown, the bootloader looks at offset 400h on the flash memory chip; if data at offset 400h is equal to 4246_4346h, the bootloader reads the whole 512-byte FCB into the on-chip SRAM and configures the FlexSPI controller using this FCB accordingly.

  After performing the aforementioned operation, the bootloader starts performing the normal boot flow. See Figure 40 for more information.

Table 156. FlexSPI FCB settings

Field	Offset (hex)	Size (bytes)	Description
tag	000	4	Configuration block tag Setting: 4246_4346h
version	004	4	Configuration block that the boot ROM uses internally Setting: 5601_0400h
Reserved	008	4	Reserved for future use
readSampleClkSrc	00C	1	Read sampling clock source options: 0 – Internal sampling 1 – Loopback from DQS pad 2 – Loopback from SCK pad 3 – External DQS signal
csHoldTime	00D	1	CS hold time in terms of flash memory clock cycles Recommended value is 3.
csSetupTIme	00E	1	CS set up time in terms of flash memory clock cycles The recommended value is 3.
columAddressWidth	00F	1	Column address width: • Is set to 3 for HyperFlash NOR flash memories. • Is set to 0 for other flash memory chips.
deviceModeCfgEnable	010	1	Enables the chip mode configuration sequence: 1 – Enable 0 – Disable
deviceModeType	011	1	Argument for Device mode configuration For example, Quad enable setting in Status Register2 for some Quad SPI flash memory devices: 0 – No mode change 1 – Quad enable (switch from SPI to Quad mode) 2 – Spi2Xpi (switch from SPI to DPI, QPI, or OPI mode) 3 – Xpi2Spi (switch from DPI, QPI, or OPI to SPI mode)

Table continues on the next page...

Table 156. FlexSPI FCB settings (continued)

Field	Offset (hex)	Size (bytes)	Description
waitTimeCfgCommands	012	2	Wait time (in terms of 100 µs) for Device mode configuration command
deviceModeSeq	014	4	Device mode configuration sequence: Byte 0 – Number of required sequences Byte 1 – Sequence index
deviceModeArg	018	4	Argument for Device mode configuration For example, Quad enable setting in Status Register2 for some Quad SPI flash memory devices
configCmdEnable	01C	1	Configuration command enable: 0 – Ignores configuration command 1 – Enables configuration command
configModeType	01D	3	Configuration command type that supports up to three types; each type is combined with a configuration command sequence
configCmdSeqs	020	12	Configuration command that supports up to three sequences
Reserved	02C	4	Reserved for future use
configCmdArgs	030	12	Configuration command that supports up to three arguments
Reserved	03C	4	Reserved for future use
controllerMiscOption	040	4	Miscellaneous controller configuration options: Bit 0 – Differential clock enable: set to 1 for HyperFlash NOR flash memory 1V8 device and set to 0 for other devices Bit 3 – WordAddressableEnable: set to 1 for HyperFlash NOR flash memory and set to 0 for other devices Bit 4 – SafeConfigFreqEnable: set to 1 if expecting to configure the chip with a safe frequency Bit 6 – DDR mode enable: set to 1 if DDR read is expected Other bits – Reserved; set to 0
deviceType	044	1	Device type 1 – Serial NOR flash memory
sflashPadType	045	1	Data pad used in Read command: 1 – Single pad 2 – Dual pads 4 – Quad pads

Table continues on the next page...

Table 156. FlexSPI FCB settings (continued)

Field	Offset (hex)	Size (bytes)	Description
			8 – Octal pads
serialClkFreq	046	1	Serial clock frequency SDR: 1 – 30MHz 2 – 50MHz 3 – 60MHz 4 – 75MHz 5 – 100MHz DDR: 1 – 30MHz 2 – 50MHz
lutCustomSeqEnable	047	1	LUT customization enable—required when you cannot perform the program and erase operations using one LUT sequence (currently, only one is applicable to HyperFlash NOR flash memory)
Reserved	048	8	Reserved for future use
sflashA1Size	050	4	Size of flash memory connected to A1
sflashA2Size	054	4	Size of flash memory connected to A2
sflashB1Size	058	4	Size of flash memory connected to B1
sflashB2Size	05C	4	Size of flash memory connected to B2
csPadSettingOverride	060	4	Pad override value for CS pin Use this value (if nonzero) to configure the CS pin; otherwise, use the default boot ROM setting
sclkPadSettingOverride	064	4	Serial clock pad setting override value
dataPadSettingOverride	068	4	Data pad setting override value
dqsPadSettingOverride	06C	4	DQS pad setting override value
timeoutInMs	070	4	Timeout value (in terms of ms) to terminate the busy check
commandInterval	074	4	CS deselect interval between two commands
dataValidTime	078	4	Clock edge to data valid time
busyOffset	07C	2	Busy offset, valid value: 0-31

Table continues on the next page...

Table 156. FlexSPI FCB settings (continued)

Field	Offset (hex)	Size (bytes)	Description
busyBitPolarity	07E	2	Busy flag polarity: 0 – Busy flag is 1 when the flash memory device is busy 1 – Busy flag is 0 when the flash memory device is busy
lookupTable	080	256	16 LUT sequences—each sequence consists of four words (see the FlexSPI chapter for more information)
lutCustomSeq	180	48	Customizable LUT sequences
Reserved	1B0	16	Reserved for future use
pageSize	1C0	4	Page size of flash memory
sectorSize	1C4	4	Sector size of flash memory
ipcmdSerialClkFreq	1C8	1	Serial clock frequency for IP commands: 0 – The same with read command Others – The same definition as that of the serial clock frequency
isUniformBlockSize	1C9	1	Indicates whether the sector size is the same as the block size: 0 – No 1 – Yes
isDataOrderSwapped	1CA	1	Data order (D0, D1, D2, D3) is swapped with (D1, D0, D3, D2)
Reserved	1CB	1	Reserved for future use
serialNorType	1CC	1	Serial NOR flash memory type: 0, 1, 2, and 3
needExitNoCmdMode	1CD	1	Exits NoCmd mode before other IP commands
halfClkForNonReadCmd	1CE	1	Half serial clock for nonread command: true or false
needRestoreNoCmdMode	1CF	1	Restores NoCmd mode after IP command execution
blockSize	1D0	4	Flash memory block size
flashStateCtx	1D4	4	Flash memory state context
Reserved	1D8	40	Reserved for future use

8.3.1.2.1 Reset

During FlexSPI boot, the boot process requires the FlexSPI flash memory device to be in a certain mode, for example, 1-bit SPI compatible mode. The flash memory device is, by default, in this mode after a POR reset. That is because the power-up sequence resets it along with the boot ROM device. However, the flash memory device does not remain in 1-bit SPI compatible mode if you configure it in DPI, QPI, or Octal mode when any non-POR resets happen (for example, watchdog timer and external pin reset).

In such a case, the boot process requires special processing to restore the flash memory device to 1-bit SPI compatible mode before accessing it. You can achieve this by using a GPIO to assert a reset pin on the flash memory device. The bootloader can perform the reset process and reset the flash memory device to 1-bit SPI compatible mode if the FLEXSPI_RESET_ENABLE field is a nonzero value in CMPA, using the GPIO that the FLEXSPI_RESET_GPIO_PORT and FLEXSPI_RESET_GPIO_PIN fields specify in combination.

NOTE

For Octal flash memory, after the boot ROM boots up, the flash memory is configured to Octal mode. The boot ROM cannot reset the NOR flash memory from Octal mode to standard SPI mode if it is a pin or soft reset causing reboot fail. The only way to solve this is to use a GPIO to reset the Octal NOR flash memory during reboot. There is no such problem in case of Quad SPI flash memory.

Table 157. FlexSPI boot configurations in CMPA WORD32(0100_4010h)(FLEXSPI_BOOT_CFG[0])

Field name	Enum name	Description	Offset	Width	Value (binary)
FLEXSPI_AUTO_PROBE_EN	—	Quad SPI or Octal SPI flash memory auto-probe feature enable	0	1	—
FLEXSPI_PROBE_TYPE	—	Quad SPI or Octal SPI flash memory probe type	1	3	—
FLEXSPI_PROBE_TYPE	QUADSPI_NOR	Quad SPI NOR flash memory	—	—	000b
FLEXSPI_PROBE_TYPE	MICRON_OCTAL	Micron Octal flash memory	—	—	001b
FLEXSPI_PROBE_TYPE	MACRONIX_OCTAL	Macronix Octal flash memory	—	—	010b
FLEXSPI_PROBE_TYPE	ADESTO_OCTAL	Adesto Octal flash memory	—	—	011b
FLEXSPI_PROBE_TYPE	—	Reserved	—	—	100b
FLEXSPI_PROBE_TYPE	—	Reserved	—	—	101b
FLEXSPI_PROBE_TYPE	—	Reserved	—	—	110b
FLEXSPI_PROBE_TYPE	—	Reserved	—	—	111b
FLEXSPI_FLASH_TYPE	—	Typical serial NOR flash memory types	4	3	—
FLEXSPI_FLASH_TYPE	QSPI_ADDR_3B	Quad SPI flash memory address (by default, the chip supports 3B read)	—	—	000b
FLEXSPI_FLASH_TYPE	—	Reserved	—	—	001b
FLEXSPI_FLASH_TYPE	—	Reserved	—	—	
FLEXSPI_FLASH_TYPE	HYPER_3V3	HyperFlash NOR flash memory 3V3	—	—	011b

Table continues on the next page...

Table 157. FlexSPI boot configurations in CMPA WORD32(0100_4010h)(FLEXSPI_BOOT_CFG[0]) (continued)

Field name	Enum name	Description	Offset	Width	Value (binary)
FLEXSPI_FLASH_TYPE	OSPI_DDR_MXIC	MXIC Octal DDR	—	—	100b
FLEXSPI_FLASH_TYPE	OSPI_DDR_MICRON	Micron Octal DDR	—	—	101b
FLEXSPI_FLASH_TYPE	—	Reserved	—	—	110b
FLEXSPI_FLASH_TYPE	—	Reserved	—	—	111b
FLEXSPI_DUMMY_CYCLES	—	Quad SPI or Octal SPI dummy cycles for read command	7	4	—
FLEXSPI_DUMMY_CYCLES	PROBE_DUMMY_CYCLE	Dummy cycles probed automatically	—	—	0000b
FLEXSPI_DUMMY_CYCLES	—	Number of dummy cycles	—	—	Other value
FLEXSPI_FREQUENCY	—	Quad SPI or Octal SPI flash interface frequency	11	3	—
FLEXSPI_FREQUENCY	FLEXSPI_75MHZ	75 MHz	—	—	000b
FLEXSPI_FREQUENCY	FLEXSPI_60MHZ	60 MHz	—	—	001b
FLEXSPI_FREQUENCY	FLEXSPI_50MHZ	50 MHz	—	—	010b
FLEXSPI_FREQUENCY	FLEXSPI_100MHZ	100 MHz	—	—	011b
FLEXSPI_FREQUENCY	—	Reserved	—	—	—
FLEXSPI_FREQUENCY	—	Reserved	—	—	—
FLEXSPI_FREQUENCY	—	Reserved	—	—	—
FLEXSPI_FREQUENCY	—	Reserved	—	—	—
FLEXSPI_RESET_PIN_ENABLE	—	FLEXSPI_RESET_PIN to use to reset the flash memory device	14	1	—
FLEXSPI_RESET_PIN_ENABLE	NO_RESET	Quad SPI or Octal SPI flash memory device reset pin is not connected or is unavailable	—	—	—
FLEXSPI_RESET_PIN_ENABLE	EN_RESET	Quad SPI or Octal SPI flash memory device reset pin is connected to a GPIO (FLEXSPI_RESET_PIN)	—	—	—

Table continues on the next page...

Table 157. FlexSPI boot configurations in CMPA WORD32(0100_4010h)(FLEXSPI_BOOT_CFG[0]) (continued)

Field name	Enum name	Description	Offset	Width	Value (binary)
FLEXSPI_RESET_PIN	—	GPIO port and pin number to use for the Quad SPI or Octal SPI flash memory reset function	15	8	—
FLEXSPI_RESET_PIN	GPIO_PORT	GPIO port	0	3	—
FLEXSPI_RESET_PIN	GPIO_PIN_NUM	GPIO pin number	3	5	—
FLEXSPI_HOLD TIME	—	Wait time before accessing serial flash memory	23	2	—
FLEXSPI_HOLD TIME	WAIT_500_US	500 µs	—	—	00b
FLEXSPI_HOLD TIME	WAIT_1_MS	1 ms	—	—	01b
FLEXSPI_HOLD TIME	WAIT_3_MS	3 ms	—	—	10b
FLEXSPI_HOLD TIME	WAIT_10_MS	10 ms	—	—	11b
FLEXSPI_PWR_HOLD_TIME	—	Delay (wait time) after POR before accessing Quad SPI or Octal SPI flash memory devices, in addition to the delay that the QSPI_HOLD TIME field specifies	25	4	—
FLEXSPI_PWR_HOLD_TIME	NO_DELAY	No delay	—	—	0000b
FLEXSPI_PWR_HOLD_TIME	100US_DELAY	Additional 100 µs	—	—	0001b
FLEXSPI_PWR_HOLD_TIME	500US_DELAY	Additional 500 µs	—	—	0010b
FLEXSPI_PWR_HOLD_TIME	1MS_DELAY	Additional 1 ms	—	—	0011b
FLEXSPI_PWR_HOLD_TIME	10MS_DELAY	Additional 10 ms	—	—	0100b
FLEXSPI_PWR_HOLD_TIME	20MS_DELAY	Additional 20 ms	—	—	0101b
FLEXSPI_PWR_HOLD_TIME	40MS_DELAY	Additional 40 ms	—	—	0110b
FLEXSPI_PWR_HOLD_TIME	60MS_DELAY	Additional 60 ms	—	—	0111b
FLEXSPI_PWR_HOLD_TIME	80MS_DELAY	Additional 80 ms	—	—	1000b
FLEXSPI_PWR_HOLD_TIME	100MS_DELAY	Additional 100 ms	—	—	1001b
FLEXSPI_PWR_HOLD_TIME	120MS_DELAY	Additional 120 ms	—	—	1010b
FLEXSPI_PWR_HOLD_TIME	140MS_DELAY	Additional 140 ms	—	—	1011b

Table continues on the next page...

Table 157. FlexSPI boot configurations in CMPA WORD32(0100_4010h)(FLEXSPI_BOOT_CFG[0]) (continued)

Field name	Enum name	Description	Offset	Width	Value (binary)
FLEXSPI_PWR_HOLD_TIME	160MS_DELAY	Additional 160 ms	—	—	1100b
FLEXSPI_PWR_HOLD_TIME	180MS_DELAY	Additional 180 ms	—	—	1101b
FLEXSPI_PWR_HOLD_TIME	200MS_DELAY	Additional 200 ms	—	—	1110b
FLEXSPI_PWR_HOLD_TIME	220MS_DELAY	Additional 220 ms	—	—	1111b
PORT_SEL	—	—	31	1	—

8.3.1.2.2 Boot flow

Figure 40 illustrates the FlexSPI NOR flash memory boot flow. It shows details related to normal image load and authentication flow:

- The FLEXSPI_BOOT_CFG[0] and FLEXSPI_BOOT_CFG[1] fields in CMPA specify the second image in terms of offset and size. If both are 0, it means the second image is not present.
- If the image version at each flash memory offset 600h is valid, use the image version; otherwise, the image version is treated as 0.
- In the image list, the newer one is image 0, and the older one is image 1. If the image version information is missing or equal, the image starting from address 0 is treated as image 0, and the image that the FLEXSPI_BOOT_CFG[0] and FLEXSPI_BOOT_CFG[1] fields specify is treated as image 1.

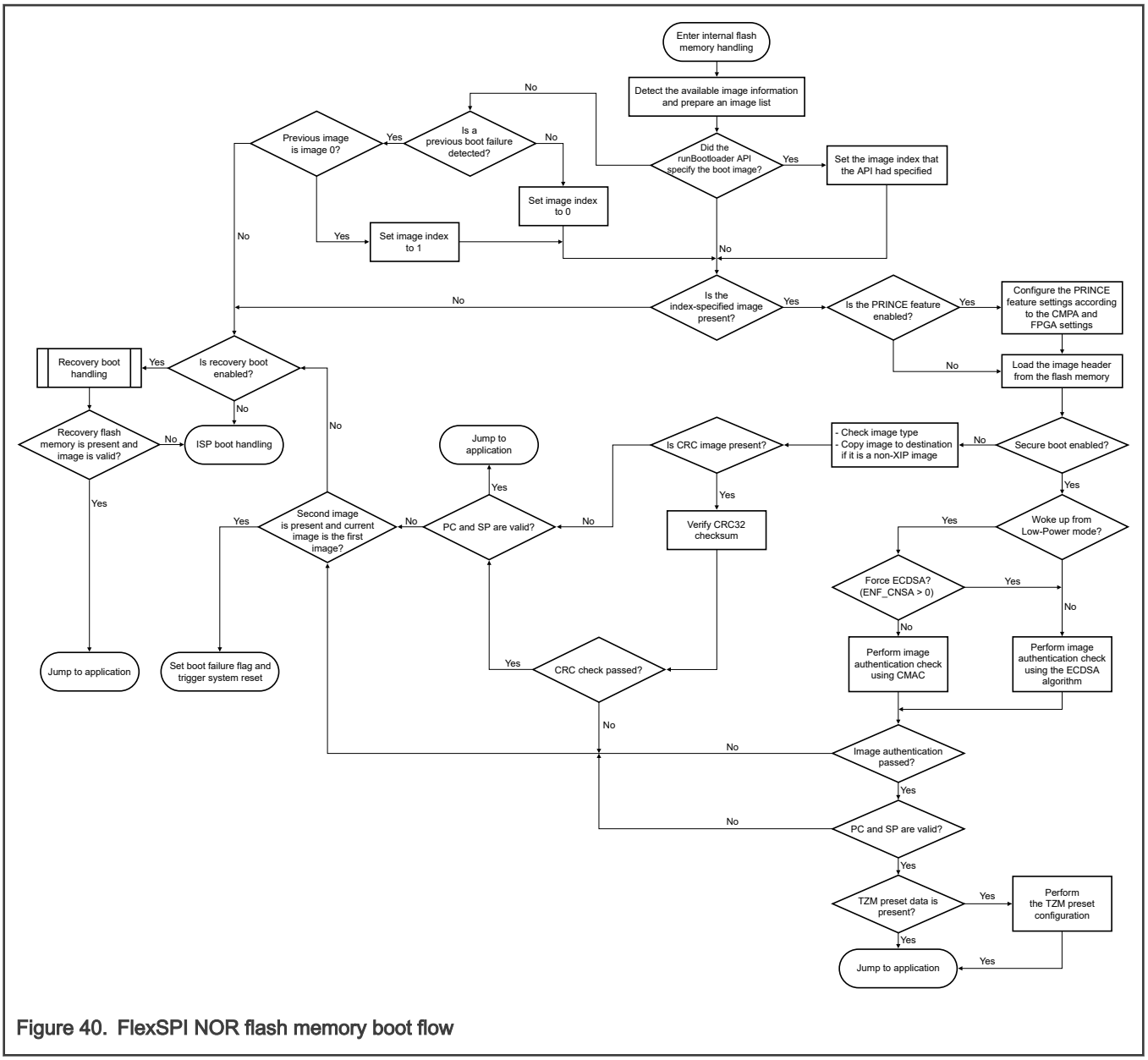


Figure 40. FlexSPI NOR flash memory boot flow

8.3.1.2.3 Dual image ping-pong boot

The FlexSPI controller supports the remap function, which can remap the AHB accessing the address of the serial NOR flash memory to another address with the same offset. In this way, FlexSPI boot supports the second image boot when the first image boots fails.

For dual image ping-pong boot, each of the boot images has its image version, which the boot ROM uses to first select the latest boot image. If the latest image boot fails, the boot ROM uses the old image to boot again.

8.3.1.2.4 Basic remapping function

The following figure shows the basic FlexSPI remap function. When the remap offset is set, FlexSPI memory access changes the access address, adding the offset, as shown in this figure. For example, when the offset is set to 2 MB (200000h), the access to 8000_0000h via FlexSPI is remapped to 8020_0000h. Using this remap feature, the boot ROM can implement a dual image boot with two images. You must set the offset and remap size of the image in the CMPA region.

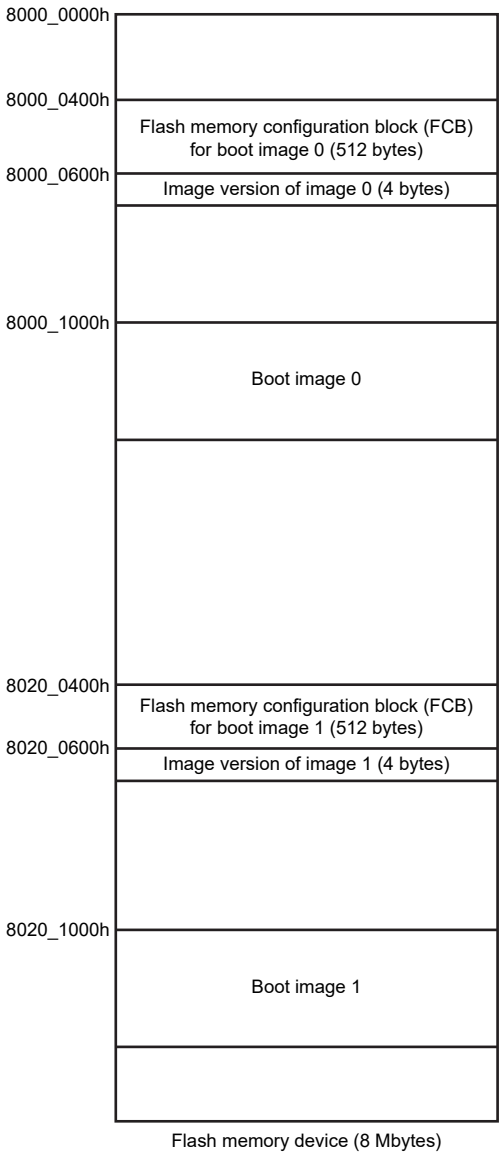


Figure 41. FlexSPI image remap (offset = 200000h)

8.3.1.2.5 Image remap offset and size configuration

The configurations related to FlexSPI dual image boot are allocated in CMPA word5 (0100_4014h) "FLEXSPI_BOOT_CFG [1]" word. See the following table and CMPA in the IFR table for details.

Table 158. Boot image 1 remap offset (CMPA: 0100_4014h)

Name that is filled	Bit offset	Width	Value	Description
FLEXSPI_IMAGE_OFFSET	7	10	*x*	*x* multiplied by 256 KB

Table 159. Boot Image size (FLEXSPI_BOOT_CFG [1])

Name that is filled	Bit offset	Width	Value	Description
FLEXSPI_REMAP_IMAGE_SIZE	17	4	0	Same size as that of the second image offset
			13	256 KB
			14	512 KB
			15	768 KB
			x	*x* multiplied by 1 MB

NOTE

FlexSPI remap size cannot exceed the start address of boot image 1.

8.3.1.2.6 Boot image version

Each boot image version is placed at offset 600h of the FlexSPI boot device (see Basic remapping function).

The image version uses 4 bytes—the lower 2 bytes are the real image version number, and the upper 2 bytes are the invert value of the lower 2 bytes of the image version. All other values that do not follow this rule are regarded as invalid values—if the boot image version is not valid, the boot ROM does not boot that image at all.

However, if the image version is all FFFF_FFFFh (the condition in which you do not program the image version), it is also considered as a valid value. When one of the image versions is FFFF_FFFFh and the other one is a valid image version, the boot ROM always first boots the image with the valid value, and not the one with image version as FFFF_FFFFh. If both image versions are FFFF_FFFFh, the boot ROM always boots the first image residing at the lower address in the FlexSPI NOR flash memory device.

A valid image version must always start from 1, which is FFFE_0001h.

The following table shows some examples of the boot image version.

Table 160. Boot image settings

Image version	Validity	Version number
FFFE_0001h	Valid	1
FFFD_0002h	Valid	2
0000_0003h	Invalid	Does not follow the image version rules
FFFF_FFFFh	Valid	Lower boot sequence
0000_FFFFh	Valid	Largest version, FFFFh

8.3.1.2.7 Ping-Pong boot flow

For FlexSPI boot, the boot ROM always tries to find the latest boot image to boot. If boot fails, the boot ROM tries to find an old boot image. If the boot still fails, it enters ISP boot mode. The following flowchart shows Ping-Pong boot flow.

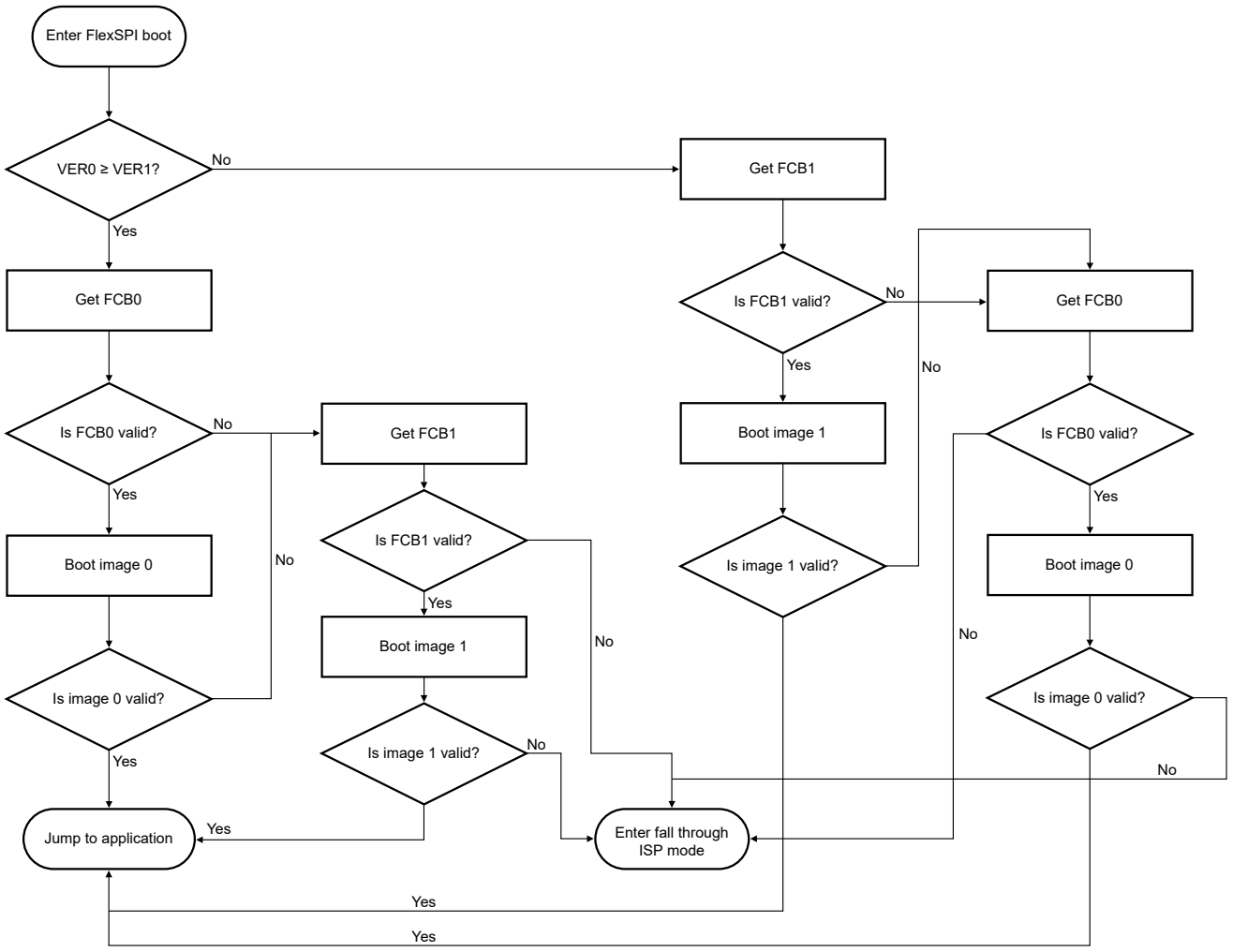


Figure 42. FlexSPI Ping-Pong boot flow

8.3.1.2.8 Configuring the FlexSPI NOR flash memory device

The boot ROM supports the booting of the image from an external FlexSPI NOR flash memory device. This section describes how to use the "blhost" tool to program the image into an external nonvolatile memory for booting. The "blhost" tool uses UART, SPI, I^2C, USB HID, and FlexCAN to communicate with the ROM code via ROM ISP mode.

See Table 145 for FlexSPI pin assignments of NOR flash memory connections.

8.3.1.2.8.1 Boot image programming

The boot ROM supports booting from different types of NOR flash memories such as Quad SPI, Octal SPI, and HyperFlash, which connect to FlexSPI pins that the boot ROM uses. For FlexSPI NOR boot, you must program FCB at offset 8000_0400h, and the boot image at offset 8000_1000h. The following sections provide information on how to configure and program the boot image into NOR flash memory for common NOR flash memory types.

8.3.1.2.8.2 Configuration options, settings, and parameters

8.3.1.2.8.2.1 Options

Table 161. FlexSPI serial NOR flash memory configuration options

Offset	Field	Description
0	Option0	See Table 162 for details.
4	Option1	Optional; effective only if the "option_size" field in option0 configuration is a nonzero value. See Table 163 for more information.

8.3.1.2.8.2.2 Option 0 field descriptions

Table 162. Option 0 field descriptions

Field	Bits	Description
tag	31:28	Tag of the configuration option, fixed to 0Ch
option_size	27:24	Size in bytes = (option size + 1) × 4 It is 0 only if option0 is required.
device_type	23:20	Device detection type: 0 – Read SFDP for SDR commands 1 – Read SFDP for DDR read commands 2 – Reserved 3 – HyperFlash 3V 4 – Macronix Octal DDR 6 – Micron Octal DDR 8 – Adesto EcoXiP DDR
query_pad	19:16	Data pads during Query command: (read SFDP or read MID) 0 – 1 2 – 4 3 – 8
cmd_pad	15:12	Data pads during flash memory access command: 0 – 1 2 – 4 3 – 8
quad_mode_setting	11:8	Quad SPI mode enable setting: 0 – Not configured 1 – Set bit 6 in Status register 1

Table continues on the next page...

Table 162. Option 0 field descriptions (continued)

Field	Bits	Description
		2 – Set bit 1 in Status register 2 3 – Set bit 7 in Status register 2 4 – Set bit 1 in Status register 2 by using the 0x31 command You use this setting to configure flash memory when it enters Quad SPI mode. You must reset the flash memory into stand SPI mode after the chip is reset because the boot ROM does not do that. **NOTE** This field is effective only if the chip is compliant with JESD216 (9 longword SFDP table).
misc_mode	7:4	Miscellaneous mode setting: 0 – Disabled 1 – Enable 0-4-4 mode for high random read performance 3 – Data Order Swapped mode (for MXIC OctaFlash memory only) 5 - Select the FlexSPI data sample source as internal loop back; see FlexSPI usage for more details 6 - Configure the FlexSPI NOR flash memory running at stand SPI mode **NOTE** Experimental feature only. Do not use in production: keep it as 0 for normal usage.
max_freq	3:0	Maximum flash memory operation speed: For SDR: 0 – 30 MHz (default) 1 – 50 MHz 2 – 60 MHz 3 – 75 MHz 4 – 100 MHz For DDR: 0 – 30 MHz 1 – 50 MHz

8.3.1.2.8.2.3 Option 1 field descriptions

Table 163. Option 1 field descriptions

Field	Bits	Description
Reserved	31:8	Reserved for future use

Table continues on the next page...

Table 163. Option 1 field descriptions (continued)

Field	Bits	Description
dummy_cycles	7:0	Dummy cycles for the Read command: 0 – Use detected dummy cycle Others – Use dummy cycles provided in the flash memory data sheet

8.3.1.2.8.2.4 Option settings

The following table shows typical FlexSPI serial NOR flash memory configuration option settings.

Table 164. FlexSPI serial NOR flash memory configuration option settings

FlexSPI mode name	Access type	Setting
Quad SPI NOR	Quad SPI SDR read	Option0 = C000_0002h (60 MHz)
Quad SPI NOR	Quad SPI DDR read	Option0 = C010_0002h (60 MHz)
HyperFlash 3V0	—	Option0 = C033_3002h (60 MHz)
MXIC OPI DDR (OPI DDR enabled by default)	—	C043_3001h (30 MHz)
Micron Octal DDR	—	Option0 = C060_0002h (60 MHz)
Micron OPI DDR	—	Option0 = C060_3002h (60 MHz)
Micron OPI DDR (DDR read enabled by default)	—	Option0 = C063_3002h (60 MHz)
Adesto OPI DDR	—	Option0 = C080_3002h (60 MHz)

8.3.1.2.8.2.5 Configuration parameters

```
              typedef struct _serial_nor_config_option
              {
          union
          {
            struct
            {
              uint32_t max_freq : 4;           // Maximum supported Frequency
              uint32_t misc_mode : 4;          // miscellaneous mode
              uint32_t quad_mode_setting : 4;  // Quad mode setting
              uint32_t cmd_pads : 4;           // Command pads
              uint32_t query_pads : 4;         // SFDP read pads
              uint32_t device_type : 4;        // Device type
              uint32_t option_size : 4;        // Option size, in terms of uint32_t, size =
(option_size + 1)
              uint32_t tag : 4;                    // Tag, must be 0x0C
            } B;
            uint32_t U;
          } option0;

          union
          {
            struct
```

```
            {
              uint32_t dummy_cycles : 8;     // Dummy cycles before read
              uint32_t status_override : 8;   // Override status register value during device mode
configuration
              uint32_t pinmux_group : 4;      // The pinmux group selection
              uint32_t dqs_pinmux_group : 4;  // The DQS Pinmux Group Selection
              uint32_t drive_strength : 4;    // The Drive Strength of FlexSPI Pads
              uint32_t flash_connection : 4;   // Flash connection option: 0 - Single Flash
connected to port A,  2 - Single Flash connected to Port B

            } B;
            uint32_t U;
          } option1;

               } serial_nor_config_option_t;
```

Code Listing 3. FlexSPI NOR flash memory device configuration parameters

8.3.1.2.8.3 Configuring, erasing, and programming via blhost

8.3.1.2.8.3.1 Configure via blhost

To configure a FlexSPI NOR flash memory device, the configuration parameter must be stored in RAM first. Consider Macronix MX25L25645G, which connects to the FlexSPI interface, as an example. The following figure shows the configuration parameter for FlexSPI, which is C000_0001h. You select a configuration parameter according to the flash memory type.

```
C:\blhost>blhost.exe -p com4 fill-memory 0x2000f000 4 0xc0000001
Ping responded in 1 attempt(s)
Inject command 'fill-memory'
Successful generic response to command 'fill-memory'
Response status = 0 (0x0) Success.
```

Code Listing 4. Setting the FlexSPI configuration parameter in RAM

8.3.1.2.8.3.2 Read, erase, and program via blhost

Use the configuration parameter stored in RAM (see Configure via blhost) to configure FlexSPI, as shown in the following figure. After that, you can read, erase, and program the flash memory. For details related to blhost tool commands, use the help section of the blhost tool.

```
C:\blhost>blhost.exe -p com4 configure-memory 0x9 0x2000f000
Ping responded in 1 attempt(s)
Inject command 'configure-memory'
Successful generic response to command 'configure-memory'
Response status = 0 (0x0) Success.
```

Code Listing 5. Configuring FlexSPI using the parameter stored in RAM

```
C:\blhost\blhost>blhost.exe -p com3 read-memory 0x08000400 0x100
Ping responded in 1 attempt(s)
Inject command 'read-memory'
Successful response to command 'read-memory'
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

```
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
(1/1) 100% Completed!
Successful generic response to command 'read-memory'
Response status = 0 (0x0) Success.
Response word 1 = 256 (0x100)
Read 256 of 256 bytes.
```

Code Listing 6. Reading the NOR flash memory via the blhost tool

```
C:\blhost\blhost>blhost.exe -p com3 flash-erase-region 0x08000400 0x200
Ping responded in 1 attempt(s)
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.
```

Code Listing 7. Erasing the NOR flash memory via the blhost tool

```
C:\blhost\blhost>blhost.exe -p com3 write-memory 0x08001000 boot_image.bin
Ping responded in 1 attempt(s)
Inject command 'write-memory'
Preparing to send 5796 (0x16a4) bytes to the target.
Successful generic response to command 'write-memory'
(1/1) 100% Completed!
Successful generic response to command 'write-memory'
Response status = 0 (0x0) Success.
Wrote 5796 of 5796 bytes.
```

Code Listing 8. Programming the boot image into NOR flash memory via the blhost tool

```
C:\blhost\blhost>blhost.exe -p com3 read-memory 0x08001000 0x100
Ping responded in 1 attempt(s)
Inject command 'read-memory'
Successful response to command 'read-memory'
00 00 20 00 f9 d1 10 00 eb c8 10 00 5b cb 10 00
6f cc 10 00 bb cd 10 00 b7 d1 10 00 17 d3 10 00
a4 16 00 00 02 40 00 00 1d 26 cd c5 23 d3 10 00
9d d6 10 00 00 c0 10 00 9f d6 10 00 a1 d6 10 00
99 d4 10 00 a1 d4 10 00 a9 d4 10 00 b1 d4 10 00
b9 d4 10 00 c1 d4 10 00 c9 d4 10 00 d1 d4 10 00
d9 d4 10 00 e1 d4 10 00 e9 d4 10 00 f1 d4 10 00
f9 d4 10 00 01 d5 10 00 09 d5 10 00 11 d5 10 00
19 d5 10 00 21 d5 10 00 29 d5 10 00 31 d5 10 00
39 d5 10 00 41 d5 10 00 49 d5 10 00 51 d5 10 00
```

```
59 d5 10 00 61 d5 10 00 69 d5 10 00 71 d5 10 00
79 d5 10 00 81 d5 10 00 89 d5 10 00 91 d5 10 00
99 d5 10 00 a1 d5 10 00 a9 d5 10 00 b1 d5 10 00
b9 d5 10 00 c1 d5 10 00 c9 d5 10 00 d1 d5 10 00
d9 d5 10 00 e1 d5 10 00 e9 d5 10 00 f1 d5 10 00
f9 d5 10 00 01 d6 10 00 09 d6 10 00 11 d6 10 00
(1/1) 100% Completed!
Successful generic response to command 'read-memory'
Response status = 0 (0x0) Success.
Response word 1 = 256 (0x100)
Read 256 of 256 bytes.
```

Code Listing 9. Reading the boot image back via the blhost tool

8.3.1.2.8.3.3 Generate and program FCB via blhost

Generate and program the FlexSPI NOR FCB into the flash memory for FlexSPI boot, with an FCB at offset 8000_0400h. You use this FCB to configure the FlexSPI interface when booting the image from an external NOR flash memory via the FlexSPI interface. You can generate the FCB using the previous FlexSPI configuration parameter (C000_0002h). Store the configured FCB and program parameter into RAM (see Code Listing 10). The boot ROM uses these in the next step to automatically generate and program the FCB into flash memory at offset 8000_0400h (see Code Listing 11).

```
C:\blhost>blhost.exe -p com4 fill-memory 0x2000f000 4 0xf000000f
Ping responded in 1 attempt(s)
Inject command 'fill-memory'
Successful generic response to command 'fill-memory'
Response status = 0 (0x0) Success.
```

Code Listing 10. Storing the configured FCB parameter into RAM

```
C:\blhost>blhost.exe -p com4 configure-memory 0x9 0x2000f000
Ping responded in 1 attempt(s)
Inject command 'configure-memory'
Successful generic response to command 'configure-memory'
Response status = 0 (0x0) Success.
```

Code Listing 11. FCB generation and programming into the flash memory at offset 8000_0400h

```
C:\blhost\blhost>blhost.exe -p com3 read-memory 0x08000400 0x100
Ping responded in 1 attempt(s)
Inject command 'read-memory'
Successful response to command 'read-memory'
46 43 46 42 00 04 01 56 00 00 00 00 01 03 03 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 01 04 02 00 00 00 00 00 00 00 00 00
00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ec 04 20 0a 00 1e 04 32 04 26 00 00 00 00 00 00
05 04 04 24 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
21 04 20 08 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
(1/1) 100% Completed!
Successful generic response to command 'read-memory'
Response status = 0 (0x0) Success.
Response word 1 = 256 (0x100)
Read 256 of 256 bytes.
```

Code Listing 12. Reading the FCB back via the blhost tool

8.3.1.2.8.4 Typical configuration parameters

You require different configuration parameters for different FlexSPI NOR flash memory types. The following table shows some typical flash memory configuration parameters.

Table 165. Typical NOR flash memory configuration parameters

Device vendor	Device type	Flash memory type	Configuration parameter	Flash memory mode	Comment
Adesto	Octal SPI flash memory	ATXP032	C080_3001h	DDR	• ATXP032 has three versions, REV A, REV B, and REV C. For REV A, the boot ROM only supports auto probe boot, the OTP fuse word. • You must configure FLEXSPI_BOOT_CFG[0] as 7h, with no need to configure FCB.
Cypress	HyperFlash memory	S26KS512S	C023_3001h	DDR	For HyperFlash boot, you must configure FLEXSPI_BOOT_CFG[0] as 20h before boot.
Macronix	Quad SPI flash memory	MX25L25645G	C000_0001h	SDR	—
Macronix	Quad SPI flash memory	MX25L25645G	C010_0001h	DDR	—
Macronix	Octal SPI flash memory	MX25UM51345G	C040_3001h	DDR	—
Micron	Octal SPI flash memory	MT35XL512ABA	C060_3001h	DDR	—

8.3.1.2.9 Programming the boot image into the flash memory

The following table lists the procedure and corresponding commands to program the boot image into the external flash memory with all data encrypted (see Configuring the FlexSPI NOR flash memory device for more information).

Table 166. FlexSPI NOR flash memory configuration

Step number	Step	Command to perform the step
1	Fill in the flash configuration parameter.	`blhost.exe -p com4 fill-memory 0x20020000 4 0xc0000001`
2	Enable the flash memory.	`blhost.exe -p com4 configure-memory 0x9 0x20020000`
3	Fill in the FCB generation parameters.	`blhost.exe -p com4 fill-memory 0x20020004 4 0xf000000f`
4	Program FCB at flash memory offset 8000_0400h.	`blhost.exe -p com4 configure-memory 0x9 0x20020004`
5	Erase the flash memory region from 8000_0000h to 8001_0000h.	`blhost.exe -p com4 flash-erase-region 0x80001000 0x10000`
6	Program the boot image into the flash memory from offset 8000_1000h.	`blhost.exe -p com4 write-memory 0x80001000 image.bin`

8.3.1.2.10 FlexSPI NOR flash memory real-time encryption and decryption with IPED

The boot ROM devices offer support for real-time encryption and decryption for external flash memory using the IPED encryption algorithm. Compared to AES, the IPED engine is faster because it can decrypt and encrypt without adding extra latency. IPED operates as data is read or written, without the need to first store data in RAM and then encrypt or decrypt to another space. It operates on a block size of 64 bits with a 128-bit key. This functionality is useful for asset protection, such as securing application code that resides in external NOR flash memory.

The boot ROM supports seven regions for encryption and decryption. Each crypto region resides at a memory address boundary of the external flash memory from 8000_0000h to 8FFF_FFFFh. All seven regions have their start address at 8000_0000h. These seven regions must not overlap; otherwise, it leads to undefined hardware behavior. For each region, there is a combination of KEY and IV that is used as an input for data encryption and decryption. The IV and KEY of each region are different. Even for the same region, IV and KEY vary for different chips. The boot ROM takes charge of the IV and KEY management, which means you do not need to consider IV and KEY when using the boot ROM's IPED feature. The hardware bus feeds IV and KEY into the IPED engine, and you remain unaware of what IV or KEY is.

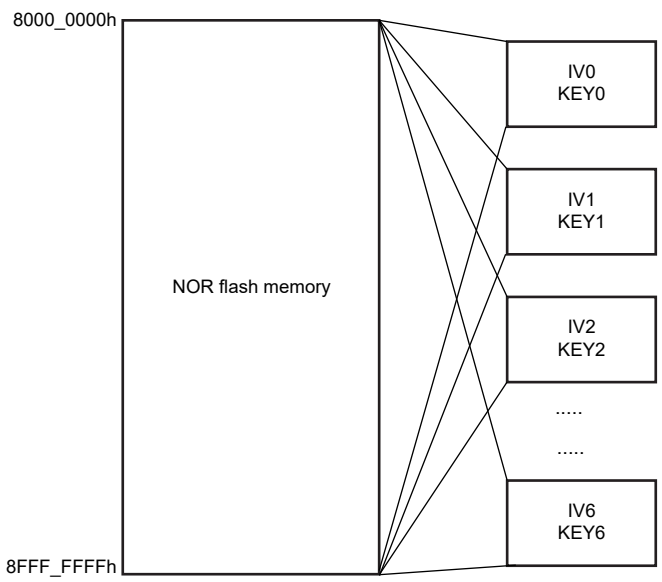


Figure 43. IPED regions

8.3.1.2.11 Functioning of the IPED region

The IPED engine has seven regions (the following figure considers region0 as an example). For each region, there is a start and an end address that you can set to decide the encryption area. As shown in the following figure, region0 ranges from 8000_1000h to 8001_0000h. After the region is enabled, the programming or reading of data from 8000_1000h to 8001_0000h is encrypted or decrypted using IV and KEY. This is hardware logic, and you do not need to consider it. The area that region0 does not cover is considered as normal area.

NOTE

Ensure that the encryption start and end address settings of the IPED engine are 256 bytes aligned. This means that the IPED engine ignores the last 8 bits if they have nonzero values. For example, if the address is 8000_00FFh, the address is considered as 8000_0000h.

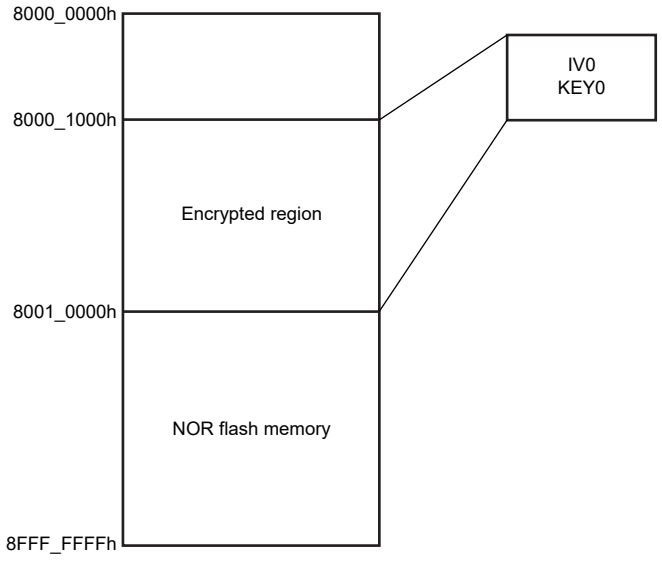


Figure 44. IPED region0

8.3.1.2.12 Image programming through IPED

The boot ROM supports real-time encryption and booting of the boot image when programming the boot image into the external flash memory. Using this feature, you can protect your boot image for security. The following figure shows real-time programming of the external flash memory through IPED region0. Before performing flash programming, you must set the chip in ISP mode (see Table 143 and Table 144 for more information). When the chip is in ISP mode, you can use the ISP interface (UART, SPI, I^2C, FlexCAN, or USB-HID) to communicate with the chip for ISP operations. For more information about the ISP protocol, see ISP protocol.

8.3.1.2.12.1 ISP commands for real-time encryption programming

Table 167. ISP commands for real-time encryption programming of external flash memory

Function	Sequence of ISP commands

Enabling IPED for CTR mode	`blhost.exe -p com4 fill-memory 0x20020000 4 0x49000000`
	`blhost.exe -p com4 fill-memory 0x20020004 4 0x80001000`
	`blhost.exe -p com4 fill-memory 0x20020008 4 0x80010000`
	`blhost.exe -p com4 configure-memory 0x9 0x20020000`
	`blhost.exe -p com4 fill-memory 0x20020000 4 0xc0000001`
	`blhost.exe -p com4 configure-memory 0x9 0x20020000`
	`blhost.exe -p com4 fill-memory 0x20020004 4 0xf000000f`
	`blhost.exe -p com4 configure-memory 0x9 0x20020004`
	`blhost.exe -p com4 flash-erase-region 0x80001000 0x10000`
	`blhost.exe -p com4 write-memory 0x80000000 image.bin`

Enabling IPED for GCM mode	`blhost.exe -p com4 fill-memory 0x20020000 4 0x49800000`
	`blhost.exe -p com4 fill-memory 0x20020004 4 0x80001000`
	`blhost.exe -p com4 fill-memory 0x20020008 4 0x80010000`
	`blhost.exe -p com4 configure-memory 0x9 0x20020000`
	`blhost.exe -p com4 fill-memory 0x20020000 4 0xc0000001`
	`blhost.exe -p com4 configure-memory 0x9 0x20020000`
	`blhost.exe -p com4 fill-memory 0x20020004 4 0xf000000f`
	`blhost.exe -p com4 configure-memory 0x9 0x20020004`
	`blhost.exe -p com4 flash-erase-region 0x80001000 0x10000`
	`blhost.exe -p com4 write-memory 0x80001000 image.bin`

8.3.1.2.12.2 IPED region selection and enablement

Perform the procedure and corresponding commands listed in the following table to select and enable IPED region0.

Table 168. IPED region selection and enablement

Step number	Step	Command to perform the step
1	Select IPED region0.	`blhost.exe -p com4 fill-memory 0x20020000 4 0x49000000`

Table continues on the next page...

Table 168. IPED region selection and enablement (continued)

Step number	Step	Command to perform the step
2	Set the IPED region0 encryption start address as 8000_1000h.	`blhost.exe -p com4 fill-memory 0x20020004 4 0x80001000`
3	Set the IPED region0 encryption end address as 8001_0000h.	`blhost.exe -p com4 fill-memory 0x20020008 4 0x80010000`
4	Enable the IPED region0.	`blhost.exe -p com4 configure-memory 0x9 0x20020000`

Following is additional information about the values used in these commands:

- 2002_0000h is a RAM address that you can specify.
- 4900_0000h is a fixed tag value used to select the IPED region. The last "0" means select region0; if the last digit is "1," it means select region1 (for example, 4900_0001h).
- 8000_1000h is the IPED region start address.
- 8001_0000h is the IPED region end address. You can change this depending on the boot image size.

See IPED region configuration parameters for more information.

8.3.1.2.12.2.1 IPED region configuration parameters

```
typedef struct _flexspi_iped_region_option
{
    uint32_t iped_region : 3; // 0~7
    uint32_t reserved : 14;
    uint32_t iped_ahbwr_en : 1; // AHB write decryption enable
    uint32_t iped_ahbrd_en : 1; // AHB read decryption enable
    uint32_t iped_ip_en : 1; // IP encryption enable
    uint32_t iped_ipwr_en : 1; // IP write encryption enable
    uint32_t iped_config_lock : 1; // CTX cnrtol lock
    uint32_t enhanced_encrypt : 1; // Enable double encryption
    uint32_t iped_gcm : 1; // IPED GCM mode enable
    uint32_t tag : 8; // Fixed to 0x49 ('I')
} flexspi_iped_prot_region_option_t;

typedef struct _flexspi_iped_region_arg
{
    flexspi_iped_prot_region_option_t option;
    uint32_t start;
    uint32_t end;
} flexspi_iped_region_arg_t;
```

Code Listing 13. IPED region configuration parameters

8.3.1.2.12.3 Boot the IPED encrypted image

After the boot image is encrypted through the IPED region and programmed into the external flash memory, the boot ROM can boot the encrypted image. During booting, the IPED engine decrypts the image in real time. Before booting, you must perform a few IPED-related settings in the CMPA region according to the defined image encryption region user. These settings must be aligned with the image encryption region; otherwise, the boot process may fail. See Table 170, considering region0 as an example. The encryption start address is 80001000h. Therefore, the CMPA setting for IPED0_START[31:8]: IPED_REG0_START_ADDR must be 800010h, with iped_region set to 0h, 1h, 2h, 3h, 4h, 5h, or 6h. See Programming the boot image into the flash memory for more information.

Table 169. Boot the IPED encrypted image

Step number	Step	Command to perform the step
1	Set the IPED region0 encryption start address as 8000_1000h. (IPED0_START = 8000_1000h)	`blhost.exe -p com4 fill-memory 0x2000f004 4 0x80001000`
2	Set the IPED region0 encryption end address as 8001_0000h. According to the IPED region encryption end address setting, IPED0_END = 8001_0000h.	`blhost.exe -p com4 fill-memory 0x20020008 4 0x80010000`

After you update the CMPA setting and the boot image is also programmed into the flash memory with IPED enabled, change the boot mode to FlexSPI NOR boot from IPED encryption boot.

Table 170. IPED region parameters in CMPA

CMPA word	Flash memory address (hex)	Word name	Setting	Details
Word44	0104_00B0	IPED0_START	IPED0_START[31:8]: IPED_REG0_START_ADDR IPED0_START[7:1]: Reserved IPED0_START[0]: ENABLE_GCM_MODE	IPED_REG0_START_ADDR: IPED region0 decryption start address for FlexSPI boot ENABLE_GCM_MODE: Enables IPED region as GCM mode during boot 0 – GCM is disabled (CTR mode) 1 – GCM is enabled
Word45	0104_00B4	IPED0_END	IPED0_END[31:8]: IPED_REG0_END_ADDR IPED0_END[7:2]: Reserved IPED0_END[1:0]: LOCK_EN	LOCK_EN: Locks the external IPED context settings 00 – Region is unlocked 01, 10, 11 – Region is locked **NOTE** When locked, you cannot update the IPED region configuration register until the next reset.

NOTE

Settings for regions 0–6 are the same and the boot ROM supports only these regions.

8.3.1.2.12.3.1 Dual image boot with IPED real-time encryption

Because there are seven IPED regions and the boot ROM also supports dual image boot (see Dual image ping-pong boot), you can use the boot ROM's dual image boot feature along with the IPED encryption boot. Before programming the two boot images,

you must enable the corresponding IPED regions for each boot image. Programming the dual image with IPED regions enabled shows typical examples for dual image boot along with IPED enabled for these two regions.

8.3.1.2.12.3.1.1 Programming the dual image with IPED regions enabled

The following table lists steps to program the two images described in Dual image boot with IPED real-time encryption into the NOR flash memory with encryption by the IPED engine.

Table 171. Programming the dual image with IPED regions enabled

Step number	Step	Sequence of commands to perform the step

1	Enable IPED region 0 (the encryption address ranges from 8000_1000h to 8000_4000h).	`blhost.exe -p com4 fill-memory 0x20020000 4 0x49000000` (GCM mode: 4980_0000h)
		`blhost.exe -p com4 fill-memory 0x20020004 4 0x80001000`
		`blhost.exe -p com4 fill-memory 0x20020008 4 0x80004000`
		`blhost.exe -p com4 configure-memory 0x9 0x20020000`

2	Enable IPED region 1 (the encryption address ranges from 8004_1000h to 8004_4000h, with the second image offset at 40000h).	`blhost.exe -p com4 fill-memory 0x20020000 4 0x49000001` (GCM mode: 4980_0001h)
		`blhost.exe -p com4 fill-memory 0x20020004 4 0x80041000`
		`blhost.exe -p com4 fill-memory 0x20020008 4 0x80044000`
		`blhost.exe -p com4 configure-memory 0x9 0x20020000`

3	Enable the FlexSPI interface.	`blhost.exe -p com4 fill-memory 0x20020000 4 0xc0000001` `blhost.exe -p com4 configure-memory 0x9 0x20020000`
4	Program the FCB at address 0800_0400h (the boot ROM automatically performs this step after the previous three steps are complete.)	`blhost.exe -p com4 fill-memory 0x20020004 4 0xf000000f` `blhost.exe -p com4 configure-memory 0x9 0x20020004`
5	Read the FCB at the address 8000_0400h and save it as FCB.bin.	`blhost.exe -p com4 read-memory 0x80000400 0x200 FCB.bin`
6	Erase the flash memory content before further programming.	`blhost.exe -p com4 flash-erase-region 0x80001000 0x4000` `blhost.exe -p com4 flash-erase-region 0x80040000 0x4000`

Table continues on the next page...

Table 171. Programming the dual image with IPED regions enabled (continued)

Step number	Step	Sequence of commands to perform the step
7	Configure the FCB.bin program at address 8004_0400h, which is used for the second image boot.	`blhost.exe -p com4 write-memory 0x8004_0400 FCB.bin`
8	Program the two boot images.	`blhost.exe -p com4 write-memory 0x80001000 image0.bin` `blhost.exe -p com4 write-memory 0x80041000 image1.bin`

8.3.1.2.12.3.1.2 CMPA settings for dual image boot with IPED

The following table shows the dual image IPED encryption boot CMPA settings. You must define all these settings in CMPA. The dual image encryption boot with IPED then follows the boot flow of the dual image ping-pong boot.

For the remap offset, see Dual image ping-pong boot. For IPED region settings, see Boot the IPED encrypted image.

Table 172. Boot image 1 remap offset and remap size setting (CMPA: 3E284h)

CMPA word	Flash memory address (hex)	Word name	Value (hex)	Details
WORD4	0104_0014	FLEXSPI_BOOT_CFG[1]	0000_0080	0000_0080h = 0'b00000000_00000000_00000000_10000000 Remap offset: 1 × 256 K (0004_0000h) Remap size: same as remap offset

Table 173. IPED region settings in CMPA for dual image boot

CMPA word	Flash memory address (hex)	Word name	Value (hex)
Word44	0104_00B0	IPED0_START	8000_1000 (GCM mode: 80001001h)
Word45	0104_00B4	IPED0_END	8000_4000
Word46	0104_00B8	IPED1_START	8004_1000 (GCM mode: 80041001h)
Word47	0104_00BC	IPED1_END	8004_4000

8.3.2 Secondary bootloader mode

Secondary boot mode can be enabled by setting CMPA[BOOT_SRC] as 2. The image loaded in the Bank1_IFR0 region (0x0100_8000 to 0x0100_FFFF) will be set as the primary boot mode, and the secondary boot image will boot first after the device is reset. The secondary boot image can be plain, crc or signed image, but cannot be set as the SB file.

Also the Bank1_IFR0 region (0x0100_8000 to 0x0100_FFFF) can be used as the Bank1_IFR0 recovery Boot area by setting the CMPA[REC_BOOT_SRC] as 1, then the Bank1_IFR0 region image will be set as the recovery boot image to boot when the primary image boot fail.

When the Bank1_IFR0 image is larger than the 32KB, user can also use the specific memory of the internal flash for the Bank1_IFR0 image usage based on the CMPA[REC_IMG_EXTn] corresponding bits, and each bit represent 32KB internal flash size. After the Bank1_IFR0 image boot, the specific memory of the internal flash and Bank1_IFR0 region will also be configured with the specific MBC as the bellow table. Suggest to use the last memory blocks of the internal flash for the Bank1_IFR0 image usage.

Based on the CMPA[OEM_BANK1_IFR0_PROT] setting, after the secondary bootloader image(SBL) boot or the Bank1_IFR0 recovery boot boot image boot, the Bank1_IFR0 region and internal flash region will be configured as the following table:

Lifecycle	CMPA[OEM_BANK1_IFR0_PROT]	Secondary boot mode & internal flash (CMPA[REC_IMG_EXTn] corresponding block) MBC setting	Bank1_IFR0 recovery boot & Internal flash(CMPA[REC_IMG_EXTn] corresponding block) MBC setting
Develop (0x3)	NA	GLBAC0	GLBAC0
Develop2 (0x7), In-field (0xF), In-field Locked (0xCF), Field Return OEM (0x1F)	0	GLBAC4	GLBAC4
	1	GLBAC4	
	2	GLBAC2	
	3	GLBAC6	
	4	GLBAC4	
	5		
	6		
	7		

Example 1: Under the LC = 7, when the Bank1_IFR0 region is used for the Secondary boot mode(CMPA[BOOT_SRC] as 2), the REC_IMG_EXT1[31] is set as 1 means the internal flash corresponding last block 32K (0x1F_8000 – 0x1F_FFFF) will be used for Secondary boot mode image usage. And set the CMPA[OEM_BANK1_IFR0_PROT] as 2, then the Bank1_IFR0 region and internal flash last block 32K (0x1F_8000 – 0x1F_FFFF) will be configured with GLBAC2 after the SBL boot.

Example 2: Under the LC = 7, when the Bank1_IFR0 region is used for the Bank1_IFR0 recovery boot mode(CMPA[REC_BOOT_SRC] as 1), and under the LC = 7 the REC_IMG_EXT1[31] is set as 1 means the internal flash corresponding last block 32K (0x1F_8000 – 0x1F_FFFF) will be used for Bank1_IFR0 recovery boot image usage, then the Bank1_IFR0 region and internal flash last block 32K (0x1F_8000 – 0x1F_FFFF) will be configured with GLBAC4 after the Bank1_IFR0 recovery boot.

8.3.3 ISP boot mode

The boot ROM supports an ISP which is mainly used for:

- Downloading the image (initial or updated) into the internal or external flash memory from the host.
- Provisioning the chip during production (configuring Secure mode or programming key data, ISP fall-through mode, and lock settings).

There are a number of conditions that can cause the boot ROM to enter ISP mode. By default, all entry methods are allowed, but CMPA fields can be used to restrict which ISP entry methods are allowed.

Table 174. ISP entry methods

ISP entry method	Description	CMPA field to allow/disallow
Boot fail	If the boot ROM does not locate a valid image, then the ROM can fall-through to ISP mode.	ISP_FT_ENTRY
API	The runBootloader API can be called from an application to request ISP mode entry.	ISP_API_ENTRY
Debug mailbox	The debug mailbox includes an Enter ISP mode command which can be used to request ISP entry from the debug port.	ISP_DM_ENTRY and ISP_CMD_EN[1]
ISP pin	If the ISP pin is asserted at reset, then the boot ROM can go straight to ISP mode without attempting to find a boot image.	ISP_PIN_ENTRY

1. ISP_CMD_EN is a field within the SOCU configuration. The combination of the fused DCFG_CC_SOCU plus the CFPA's DCFG_CC_SOCU and CMPA's CC_SOCU determine the access rights for debug commands including the Debug Mailbox Enter ISP mode command. The Debug Mailbox chapter is present in MCX Nx4x Security Reference Manual.

NOTE

The ISP_DISABLE fuse field can also be used to restrict ISP entry methods. For an ISP entry to work, it must be enabled by both the CMPA and fuse settings.

In ISP mode, the boot ROM can communicate over a variety of serial interfaces. By default, the device UART, I2C, SPI, CAN, and USB HID are supported. The CMPA's ISP_BOOT_IF can be used to select a specific serial interface to be used, or it can be left in the default Auto ISP mode which will scan all the available interfaces looking for activity. There are also fuses that can be used to disable ISP mode on specific interfaces:

- ISP_CAN_Dis
- ISP_USB1_Dis
- ISP_USB0_Dis
- ISP_I2C_Dis
- ISP_SPI_Dis
- ISP_UART_Dis

For ISP mode to work, the interface selected by ISP_BOOT_IF must be allowed by the ISP_xxx_Dis fuses. See the In-System Programming (ISP) chapter for more details on ISP protocol, packet types, commands, and operation for each serial communication type.

8.3.4 Recovery boot mode

The boot ROM supports a recovery boot from an external 1-bit SPI flash memory device or an image loaded into IFR0 (0x0100_8000 to 0x0100_FFFF). The table below summarizes the recovery boot options.

Table 175. Recovery boot options

Recovery boot type	Selected by	Image format
External 1-bit SPI flash	• CMPA REC_BOOT_SRC = 0b10 • Floating ISP pin	• SB3.1 image
IFR0 (0x0100_8000 to 0x0100_FFFF)	CMPA REC_BOOT_SRC = 0b01	XIP image

See SPI flash memory recovery for more information.

NOTE

The default pins used for 1-bit SPI flash recovery boot overlap with the FlexSPI pins. This was done intentionally to allow a memory to be used as a recover device and also to be used for FlexSPI operation later.

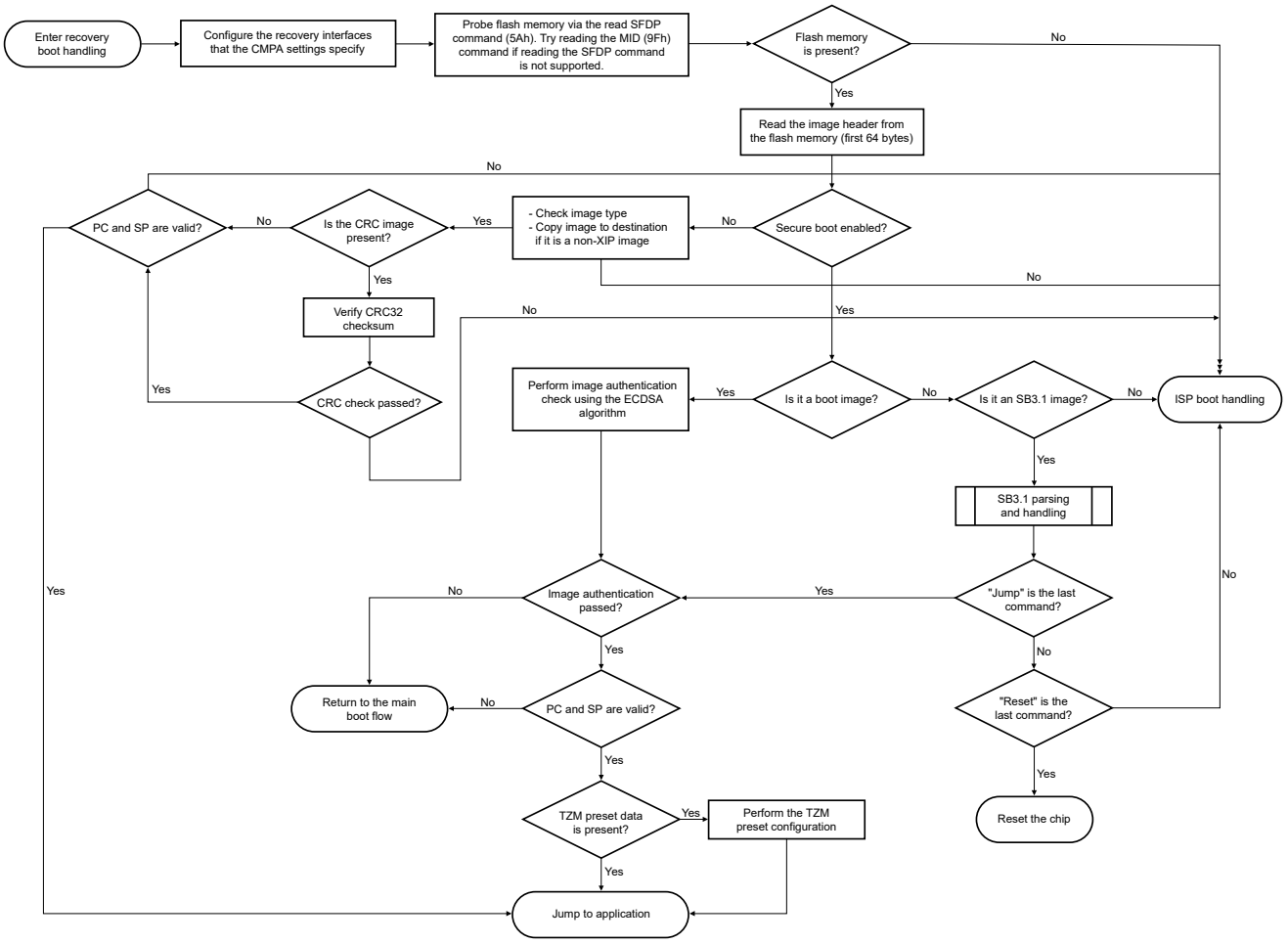


Figure 45. Recovery boot flow

8.4 External memory support

The following table describes the external memory devices, with their memory identifiers, that the boot ROM ISP command supports.

To use an external memory device correctly, the boot ROM enables specific external memory devices, with their corresponding configuration profiles, using a preassigned memory identifier. Without enabling these external memory devices, you cannot access them using the boot ROM ISP command.

Table 176. Memory IDs for external memory devices

Memory identifier	External memory device
09h	Serial NOR over the FlexSPI module
110h	Serial NOR and EEPROM over the SPI module

8.4.1 Serial NOR flash memory through FlexSPI

8.4.1.1 FlexSPI NOR FCB

The boot ROM supports the read, write, and erase functions for external serial NOR flash memory devices via FlexSPI. Before accessing these devices, you must configure the FlexSPI module properly, using a simplified FlexSPI NOR configuration block or a complete 512-byte FlexSPI NOR configuration block. See Table 177 for settings.

The boot ROM can generate the 512-byte FlexSPI NOR configuration block based on the simplified flash memory configuration option block for most serial NOR flash memory devices in the market.

To customize the lookup table (LUT) sequence for a specific chip, you must enable the “lutCustomSeqEnable” field and fill in the corresponding “lutCustomSeq” field. See Table 178 for the predefined LUT index for serial (SPI) NOR.

Table 177. FlexSPI NOR FCB

Field	Offset (hex)	Size (bytes)	Description
tag	000	4	Configuration block tag Must be set to 4246_4346h
version	004	4	Configuration block that the boot ROM uses internally; its value is fixed to 5601_0400h
Reserved	008	4	Reserved for future use
readSampleClkSrc	00C	1	Read sampling clock source option: 0 – Internal sampling 1 – Loopback from DQS pad 2 – Loopback from SCK pad 3 – External DQS signal
csHoldTime	00D	1	CS hold time in terms of flash memory clock cycles Recommended value is 3.
csSetupTIme	00E	1	CS set up time in terms of flash memory clock cycles Recommended value is 3.
columAddressWidth	00F	1	Column address width: • 3 for HyperFlash • 0 for other flash memory devices
deviceModeCfgEnable	010	1	Enables the Device mode configuration sequence
deviceModeType	011	1	Argument for Device mode configuration For example, Quad enable setting in Status register 2 for some Quad SPI flash memory devices
waitTimeCfgCommands	012	2	Wait time in terms of 100 µs for the Device mode configuration command

Table continues on the next page...

Table 177. FlexSPI NOR FCB (continued)

Field	Offset (hex)	Size (bytes)	Description
deviceModeSeq	014	4	Device mode configuration sequence: Byte 0 – Number of required sequences Byte 1 – Sequence index
deviceModeArg	018	4	Argument for Device mode configuration For example, Quad enable setting in Status register 2 for some Quad SPI flash memory devices
configCmdEnable	01C	1	Configuration command enable: 0 – Ignores configuration command 1 – Enables configuration Command
configModeType	01D	3	Configuration command types; supports up to three types Each type is combined with a configuration command sequence.
configCmdSeqs	020	12	Configuration command types; supports up to three sequences
Reserved	02C	4	Reserved for future use
configCmdArgs	030	12	Configure command argument; supports up to three arguments
Reserved	03C	4	Reserved for future use
controllerMiscOption	040	4	Miscellaneous controller configuration options: Bit 0 – Differential clock enable; set to 1 for the HyperFlash 1V8 device, and set to 0 for other devices Bit 3 – WordAddressableEnable; set to 1 for HyperFlash and set to 0 for other devices Bit 4 – SafeConfigFreqEnable; set to 1 for configuring the device with a safe frequency Bit 6 – DDR mode enable; set to 1 if DDR read is expected Other bits – Reserved; set to 0
deviceType	044	1	Device type: 1 - Serial NOR
sflashPadType	045	1	Data pad used in Read command: 1 – Single pad 2 – Dual pads

Table continues on the next page...

Table 177. FlexSPI NOR FCB (continued)

Field	Offset (hex)	Size (bytes)	Description
			4 – Quad pads 8 – Octal pads
serialClkFreq	046	1	Serial clock frequency SDR: 1 – 30MHz 2 – 50MHz 3 – 60MHz 4 – 75MHz 5 – 100MHz DDR: 1 – 30MHz 2 – 50MHz
lutCustomSeqEnable	047	1	LUT customization enable that is required if you cannot perform the program or erase operations using one LUT sequence (currently only applicable to HyperFlash)
Reserved	048	8	Reserved for future use
sflashA1Size	050	4	Size of flash memory connected to A1
sflashA2Size	054	4	Size of flash memory connected to A2
sflashB1Size	058	4	Size of flash memory connected to B1
sflashB2Size	05C	4	Size of flash memory connected to B2
csPadSettingOverride	060	4	Pad override value for CS pin Use this value (if nonzero) to configure the CS pin; otherwise use the default ROM setting
sclkPadSettingOverride	064	4	Serial clock pad setting override value
dataPadSettingOverride	068	4	Data pad setting override value
dqsPadSettingOverride	06C	4	DQS pad setting override value
timeoutInMs	070	4	Timeout value in terms of ms to terminate the busy check
commandInterval	074	4	CS deselect interval between two commands
dataValidTime	078	4	Clock edge to data valid time

Table continues on the next page...

Table 177. FlexSPI NOR FCB (continued)

Field	Offset (hex)	Size (bytes)	Description
busyOffset	07C	2	Busy offset; valid value: 0–31
busyBitPolarity	07E	2	Busy flag polarity: 0 – Indicates that the busy flag is set to 1 when the flash memory device is busy 1 – Indicates that the busy flag is set to 0 when the flash memory device is busy
lookupTable	080	256	16 LUT sequences Each sequence consists of four words (see the "FlexSPI" chapter for more information).
lutCustomSeq	180	48	Customizable LUT sequences
Reserved	1b0	16	Reserved for future use
pageSize	1C0	4	Page size of flash memory
sectorSize	1C4	4	Sector size of flash memory
ipcmdSerialClkFreq	1C8	1	Serial clock frequency for IP command: 0 – The same with Read command Others – The same definition as serialClkFreq
isUniformBlockSize	1C9	1	Indicates whether the sector size is the same as block size: 0 – No 1 – Yes
isDataOrderSwapped	1CA	1	Indicates whether the data order (D0, D1, D2, D3) is swapped (D1, D0, D3, D2)
Reserved	1CB	1	Reserved for future use
serialNorType	1CC	1	Serial NOR flash memory type: 0, 1, 2, or 3
needExitNoCmdMode	1CD	1	Exit NoCmd mode before other IP commands as needed
halfClkForNonReadCmd	1CE	1	Half the serial clock for nonread command: true or false
needRestoreNoCmdMode	1CF	1	Restore NoCmd mode after IP command execution as needed
blockSize	1D0	4	Flash memory block size

Table continues on the next page...

Table 177. FlexSPI NOR FCB (continued)

Field	Offset (hex)	Size (bytes)	Description
flashStateCtx	1D4	4	Flash memory state context
Reserved	1D8	40	Reserved for future use

8.4.1.2 LUT index pre-assignment for FlexSPI NOR

Table 178. LUT index pre-assignment for FlexSPI NOR

Name	Index in LUT	Description
Read	0	Read command sequence
ReadStatus	1	Read status command
ReadStatusXpi	2	Read status command under OPI mode
WriteEnable	3	Write enable command sequence
WriteEnableXpi	4	Write enable command under OPI mode
EraseSector	5	Erase sector command
EraseBlock	8	Erase block command
PageProgram	9	Page program command
ChipErase	11	Full chip erase
ExitNoCmd	15	Exit no command mode as needed
Reserved	6, 7, 10, 12, 13, 14	All reserved indexes can be freely used for other purposes

8.4.2 FlexSPI NOR configuration option block settings

The FlexSPI NOR configuration option block is organized by a 4-bit unit and is expandable (see Table 179 and Table 180 for the current definitions of the block).

The boot ROM detects FCB via reading an SFDP command, which most JESD216(A/B)-compliant flash memory devices support. However, JESD216A/B only defines the dummy cycles for Quad SPI SDR read. To get the dummy cycles for DDR or DTR read mode, the boot ROM supports auto-probing by writing test patterns to offset 200h on the external flash memory devices. For optimal timing, readSampleClkSrc is set to 1 for flash memory devices that do not support externally provided DQS pad inputs. It is set to 3 for flash memory devices that support externally provided DQS pad inputs such as Octal SPI flash memory or HyperFlash NOR flash memory. The FlexSPI_DQS pad is not used for other purposes.

Table 179. FlexSPI NOR configuration option 0 block

Field: Option0 | Offset: 0 Details							
Tag[31:28]	Option size[27:24]	Device detection type[23:20]	Query CMD pad(s) [19:16]	CMD pads(s) [15:12]	Quad SPI enable type[11:8]	Misc mode[7: 4]	Max freq[3:0]
Ch (fixed)	Size in bytes = (option size + 1) × 4 (provides scalability for future use)	Software-defined device types used for configuration block auto-detection 0: Quad SPI SDR 1: Quad SPI DDR 2: HyperFlash 1V8 3: HyperFlash 3V 4: MXIC OPI DDR 6: Micron OPI DDR 7: Micron Octal SDR 8: Adesto OPI DDR 9: Adesto EcoXiP SDR	Command pads (1, 4, and 8) for the SFDP command 0: 1 2: 4 3: 8	Command pads (1, 4, and 8) for the flash memory device 0: 1 2: 4 3: 8	Quad enable sequence 1: QE bit is bit 6 in StatusReg1 2: QE bit is bit 1 in StatusReg2 3: QE bit is in bit 7 in StatusReg2 4: QE bit is bit 1 in StatusReg2; enable command is 0x31	Miscellaneous mode for the selected flash memory types **NOTE** Experimental feature only. Do not use in production: keep it as 0 for normal usage. 0: Not enabled 1: Enable 0-4-4 mode for high random read performance 3: Data orders wrapped mode (for MXIC OctaFlash only) 5: Select the FlexSPI data sample source as internal loop back; see FlexSPI usage for more 6: Configure the FlexSPI NOR flash memory running at stand SPI mode	Maximum flash memory operation speed 0: Don't change the FlexSPI clock setting

Table 180. FlexSPI NOR configuration option 1 block (optional)

Field: Option1 (optional) | Offset: 4 Details			
flash_connection[31:28]	Reserved[27:16]	status_override[15:8]	Dummy cycle[7:0]
Flash connection option (FlexSPI port A or B) 0: Single flash memory connected to port A 2: Single flash memory connected to port B		Override status register value used for Device mode configuration	Dummy cycles for SDR and DDR read commands 0: Use detected dummy cycles Other values: Use dummy cycles provided in the flash memory data sheet

NOTE

CMD pad(s) value is:

- Always 0h for devices that work under 1-1-4, 1-4-4, 1-1-8, or 1-8-8 mode.
- 2 for devices that only support 4-4-4 mode for high performance.
- 3 for devices that only support 8-8-8 mode for high performance.

8.4.3 Typical use cases for the FlexSPI NOR configuration block

Table 181. Use cases for the FlexSPI NOR configuration block

FlexSPI mode name	Access type	Setting
Quad SPI NOR	Quad SDR read	Option0 = C000_0002h (60 MHz)
Quad SPI NOR	Quad DDR read	Option0 = C010_0002h (60 MHz)
HyperFlash 3V0	—	Option0 = C033_3002h (60 MHz)
MXIC OPI DDR	OPI DDR enabled by default	Option0 = C043_3002h (60 MHz)
Micron Octal DDR	DDR read	Option0 = C060_0002h (60 MHz)
Micron OPI DDR	DDR read	Option0 = C060_3002h (60 MHz)
Micron OPI DDR	DDR read enabled by default	Option0 = C063_3002h (60 MHz)
Adesto OPI DDR	DDR read	Option0 = C080_3002h (60 MHz)

8.4.4 Programming a serial NOR flash memory device using the FlexSPI NOR configuration option

The boot ROM supports generating the complete FCB using the configure-memory command. It also supports programming the generated FCB at the start of the flash memory by using a specific option, "0xF000000F." The following table shows an example of how to configure and access HyperFlash (assuming it is a blank HyperFlash device). See Configuring the FlexSPI NOR flash memory device for more information.

Table 182. Programming a serial NOR flash memory device using the FlexSPI NOR configuration option

Step number	Step	Command to perform the step
1	Write the option block to SRAM address 2002_000h.	`blhost -u <PID,VID> - fill-memory 0x20020000 0x04 0xc0333007`
2	Configure HyperFlash using the option block.	`blhost -u <PID,VID> - configure-memory 0x09 0x20020000`
3	Write specific option to SRAM address 2002_000h.	`blhost -u <PID,VID> - fill-memory 0x20020000 0x04 0xf000000f`
4	Program FCB to the offset 400h of the HyperFlash device.	`blhost -u <PID,VID> - configure-memory 0x09 0x20020000`
5	Erase the HyperFlash device from address of specified size.	`blhost -u <PID,VID> – flash-erase-region <addr> <size>`
6	Write the image to the address.	`blhost -u <PID,VID> - write-memory <addr> image.bin`

8.4.5 1-bit serial NOR flash memory through SPI

8.4.5.1 Manual and Auto configuration methods

The boot ROM supports programming 1-bit SPI NOR flash memory devices that support a 3-byte address read, 03h, or 4-byte address read, 13h, via the flash memory configuration option block. It defines the SPI clock frequency as 48 MHz.

The boot ROM supports either a manually configured option or an auto-detected option. When using the manually configured option, you must specify all the flash memory information (flash size, sector size, page size, and so on) in the option block. When using the auto-detected option, which is only supported on devices that are JESD216 compliant, the boot ROM is able to detect the flash memory information via the read SFDP (5Ah) command. Using this command, you can set all the flash memory information to zeros. Table 183 shows details required for configuring the SPI NOR flash memory by using these methods. The read page (03h) is used for the default read command. If the flash memory size is greater than 16 MB (detected by the read SFDP command), the 13h command is used for the page read. If the SFDP command is not supported, the boot ROM uses the read MID (9Fh) to detect whether a device is connected to it.

8.4.5.2 SPI NOR configuration option block settings

Table 183. SPI NOR configuration option block settings (Option0 field settings)

Tag[31:28]	Reserved[27:16]	Flash memory info set[15:12]	Flash memory size[11:8]	Sector size[7:4]	Page size[3:0]
Fixed at Ch		Memory type used to configure and access the NOR flash memory: 0 – Manual (selects flash memory parameter via the flash memory data sheet) 2 – Auto (detects flash memory parameter via the SFDP table)	Memory capacity of the NOR flash memory device: 1 – 1 MB 2 – 2 MB 3 – 4 MB 4 – 8 MB 5 – 16 MB 6 – 32 MB 7 – 64 MB 8 – 64 MB 9 – 128 MB 10 – 256 MB	Sector size of the NOR flash memory device: 0 – 4 KB 1 – 8 KB 2 – 32 KB 3 – 64 KB 4 – 128 KB 5 – 256 KB	Page size of the NOR flash memory device: 0 – 256 bytes 1 – 512 bytes 2 – 1 KB 3 – 128 KB

See Table 184 for information about how to program a 1-bit serial NOR flash memory device via the SPI NOR configuration option block.

8.4.5.3 Programming via the SPI NOR configuration option block

The boot ROM supports programming the serial NOR flash device via the SPI interface using the SPI NOR configuration option block. The following table shows an example of how to configure or program the option block. Before writing the configuration option block, you select the SPI peripheral index by writing the value C070_2000h. (SPI index is the SPI interface used to access serial NOR flash memory.)

Table 184. Programming a 1-bit serial NOR flash memory device via the SPI NOR configuration option block

Step number	Step	Command to perform the step
1	Write the option block to SRAM address 2002_0000h; the SPI instance is 7.	`blhost -p comxx - fill-memory 0x20020000 0x04 0xc0702000`
2	Configure serial NOR flash memory using the option block.	`blhost -p comxx - configure-memory 0x110 0x20020000`
3	Erase the serial NOR flash memory device from address of specified size.	`blhost -p comxx - flash-erase-region <addr> <size> 0x110`
4	Write the recovery boot image or SB3 file to the address in the external 1-bit flash memory.	`blhost -p comxx - write-memory <addr> image.bin 0x110`

8.5 IFR region definitions

The IFR region is used as the persistent storage for the secure boot and the SoC specific parameters. It starts at fixed address 0x01000000. Refer Protected Flash Region for details.

Table 185. IFR layout for the ROM devices

Region	Field	Description
0x01000000-0x01003FFF	Customer In-field Programmable Area (CFPA)	See the IFR map for more details.
0x01004000-0x01005FFF	Customer Manufacturing/Factory Programmable Area (CMPA)	See the IFR map for more details.
0x01006000-0x01007FFF	Reserved area	Reserved
0x01008000-0x0100FFFF	OEM SBL/OEM recovery boot area	See the IFR map for more details.
0x01010000-0x01014000	NXP Manufacturing Programmed Area	See the IFR map for more details.

See the IFR sheet in the attached "MCXNx4x_IFR_Fusemap.xlsx" document for detailed layout.

Customer Manufacturing/Factory Programmable Area(CMPA)

Customer can use the ISP program and ROM API (ffr_cust_factory_page_write) to program the CMPA region in the Develop (0x3) lifecycle. And the CMPA region will be frozen in the Develop2 (0x7), In-field (0xF), In-field Locked (0xCF), Field Return OEM (0x1F) lifecycles.

Under the Develop(0x3) lifecycle, customer can use the ISP program and ROM API (ffr_cust_factory_page_write with seal_part “false” parameter and CMAC value set as all 0x0) to program the CMPA CRC on the secure part. CMPA CMAC can be calculated by ISP program or by IAP (if CFPA[CMPA_UPD] >= 0x2).”

The CMPA CMAC authenticity and integrity is only executed when the lifecycle is equal or larger than the Develop2 (0x7) lifecycle. Customer can use the “customer defined” CMPA area (0x0100_4200 – 0x0100_5FFF) to store their own manufacturing data such as certificate store, keystore, any other configuration data etc. This region can be programmed with the following steps under the Develop (0x3) lifecycle:

- Configure the MBC MBC0_DOM0_MEM1_BLK_CFG_W0[10:8] with GLBAC0 to add the write attribute of CMPA or set GAC0 to this filed.
- Call the API flash_init and ffr_init.
- Use the ROM API flash_erase to erase the whole CMPA page.

- Use the ROM API flash_program to program the “customer defined” data into the CMPA area (0x0100_4200 – 0x0100_5FFF).
- If needed, Customer can use the ROM API flash_read to read the region to check that or use the ROM API flash_verify_program to verify whether the data is programmed right.

When the CMPA has the “customer defined” data, the later CMPA update like the ROM API(ffr_cust_factory_page_write) or ISP CMPA program will always read the original “customer defined” data back and program with that data in the new CMPA page update.

Customer In-field Programmable Area (CFPA)

Customer can use the ISP program and ROM API (ffr_infield_page_write) to program the CFPA with CRC value. After the reset (excluding warm reset caused by wake-up from deep power-down state), the CMAC of CFPA page will be calculated by boot ROM and appended to CFPA page. Hence user application should set the CMAC field in CFPA page with 0xFF or 0x00 during API call. On subsequent reset, the boot ROM authenticates and integrity-checks CFPA using CMAC operation when the lifecycle is Develop2 (0x7), In-field (0xF) and In-field Locked (0xCF).

CFPA pages have the ping-pong page feature for the customer to update CFPA more reliably. This has the roll-back check and use the previously valid CFPA page when the updated CFPA is invalid.

During the ROM boot flow, one CFPA page is active (which will be write-protected), and other CFPA page is left in-active(write enabled) for ISP or ROM API (ffr_infield_page_write) to program the CFPA under the Develop(0x3), Develop2 (0x7), In-field (0xF), In-field Locked (0xCF) lifecycles.

Customer can use the “customer defined” CFPA ping area (0x0100_0200 – 0x0100_1FFF) and CFPA pong area (0x0100_2200 – 0x0100_3FFF) to store their own manufacturing data such as certificate store, keystore, any other configuration data etc. This region can be programmed with the following steps under the Develop(0x3) lifecycle:

- First, call the API flash_init and ffr_init.
- Use the ROM API flash_erase to erase the whole in-active CFPA page.
- Use the ROM API flash_program to program the “customer defined” data into the in-active CFPA customer defined area (offset from 0x200 to 0x1fff). If user program the data into the current active CFPA area, the device will reset because accessing the locked protect region.
- If needed, you can use the ROM API flash_read to read the region to check that or use the ROM API flash_verify_program to verify whether the data is programmed right.

  When the CFPA has the “customer defined” data, user will meet two situations, the first is user want to update the CFPA with the original customer defined data. The user can call the ROM API(ffr_infield_page_write) or ISP CFPA program to update the CFPA, and the new programmed CFPA will read the original “customer defined” data back and program with that data in the new CFPA page update automatically. The second situation is user want to update the “customer defined” data, user need to follow the above flash_erase and flash_program steps to program the customer defined data with larger value in the first word, and then user can call the ROM API(ffr_infield_page_write) or ISP CFPA program to update the CFPA and the new “customer defined” data will be used.

The monotonic counter and flags feature fields are listed in the table below.

NOTE

For other fields, see the section Secure boot related configuration fields in IFR.

CFPA word	Flash memory address (hex)	Word name	Description
Word32	0100_0080	MCTR_CUST_CTR0	Customer counter for word rollback check: the updated MCTR_CUST_CTR0 should be equal or larger than origin MCTR_CUST_CTR0.

Table continues on the next page...

Table continued from the previous page...

CFPA word	Flash memory address (hex)	Word name	Description
Word33	0100_0084	MCTR_CUST_CTR1	Same as above
Word34	0100_0088	MCTR_CUST_CTR2	Same as above
Word35	0100_008c	MCTR_CUST_CTR3	Same as above
Word36	0100_0090	MCTR_CUST_CTR4	Same as above
Word37	0100_0094	MCTR_CUST_CTR5	Same as above
Word38	0100_0098	MCTR_CUST_CTR6	Same as above
Word39	0100_009c	MCTR_CUST_CTR7	Same as above
Word40	0100_00a0	MFLAG_CUST_0	Customer flag for word burn-bits rollback check: the updated MFLAG_CUST_0 should have the origin MFLAG_CUST_0 set bits.
Word41	0100_00a4	MFLAG_CUST_1	Same as above
Word42	0100_00a8	MFLAG_CUST_2	Same as above
Word43	0100_00ac	MFLAG_CUST_3	Same as above
Word44	0100_00b0	MFLAG_CUST_4	Same as above
Word45	0100_00b4	MFLAG_CUST_5	Same as above
Word46	0100_00b8	MFLAG_CUST_6	Same as above
Word47	0100_00bc	MFLAG_CUST_7	Same as above

The monotonic counter

- The MCTR_CUST_CTR0 - MCTR_CUST_CTR7 value in the CFPA (0x0100_0080 – 0x0100_009F) are loaded by ISP program and ROM API (ffr_infield_page_write), and each word will do the roll-back check: the newer MCTR_CUST_CTRn value of the updated CFPA should not be less than the origin MCTR_CUST_CTRn of the current active CFPA.
- If the MCTR_CUST_CTRn word value of the updated CFPA is set as 0xFFFFFFFFu, then the CFPA will continue use the original MCTR_CUST_CTRn of the current CFPA.

The flags feature

- The MFLAG_CUST_0 - MFLAG_CUST_7 value in the CFPA (0x0100_00a0 – 0x0100_00BF) are loaded by ISP program and ROM API (ffr_infield_page_write), and each word will do the burn-back check. When the origin MFLAG_CUST_n has set some bits, the newer MFLAG_CUST_n of the updated CFPA should also set the same bits besides other new bits set or not. For example, the current active CFPA set the MFLAG_CUST_0 with 0x3, the updated CFPA should set the MFLAG_CUST_0 with 0x3 or 0x7, etc., but cannot set as 0x2.

8.6 eFuse definitions

The boot ROM supports the fuse block and defines the logical fuse layout provided in the following table, with exported APIs for fuse read and program.

See the fuse map file attached to this document for more information.

Table 186. eFuse definitions in the boot ROM

Fuse word index	Bit width	Field name	Description
0	32	FUSE WORD0	Boot configuration lock
1	32	FUSE WORD1	Life cycle state, debug setting, and ROTK setting
2	32	FUSE WORD2	Image key revoke and PRINCE, PUF, and NXP provisioning firmware setting
3	32	FUSE WORD3	ISP pin selection setting and boot configuration setting
4	32	FUSE WORD4	FLEXSPI_BOOT_CFG0
5	32	FUSE WORD5	FLEXSPI_BOOT_CFG1
6	32	FUSE WORD6	Secure related setting0
7	32	FUSE WORD7	Secure related setting1
8	32	FUSE WORD8	Blank fuse
9	32	FUSE WORD9	Blank fuse
10	32	FUSE WORD10	Blank fuse
11	32	FUSE WORD11	ROTKH[31:0]
12	32	FUSE WORD12	ROTKH[63:32]
13	32	FUSE WORD13	ROTKH[95:64]
14	32	FUSE WORD14	ROTKH[127:96]
15	32	FUSE WORD15	ROTKH[159:128]
16	32	FUSE WORD15	ROTKH[191:160]
17	32	FUSE WORD17	ROTKH[223:192]
18	32	FUSE WORD18	ROTKH[255:224]
19	32	FUSE WORD19	ROTKH[287:256]
20	32	FUSE WORD20	ROTKH[319:288]
21	32	FUSE WORD21	ROTKH[351:320]
22	32	FUSE WORD22	ROTKH[383:352]
23	32	FUSE WORD23	NPX_CTX0_WD0[31:0]
24	32	FUSE WORD24	NPX_CTX0_WD1[31:0]
25	32	FUSE WORD25	NPX_CTX1_WD0[31:0]
26	32	FUSE WORD26	NPX_CTX1_WD1[31:0]
27	32	FUSE WORD27	NPX_CTX2_WD0[31:0]
28	32	FUSE WORD28	NPX_CTX2_WD1[31:0]
29	32	FUSE WORD29	NPX_CTX3_WD0[31:0]
30	32	FUSE WORD30	NPX_CTX3_WD1[31:0]

Table continues on the next page...

Table 186. eFuse definitions in the boot ROM (continued)

Fuse word index	Bit width	Field name	Description
31	32	FUSE WORD31	IPED_CTX0_CRC32[31:0]; CRC value of the IPED start and end address setting
32–38	32	FUSE WORD32-38	Reserved
39	32	FUSE WORD39	Reserved
40	32	FUSE WORD40	Reserved
41	32	FUSE WORD41	Reserved
42	32	FUSE WORD42	Reserved
43–81	32	FUSE WORD43-81	CUST FUSE WORD0-26

Chapter 9
Secure Boot ROM

9.1 Overview

The Secure part of ROM boot loader provides the following basic operations:

- Secure boot
- Secure firmware update
- Security related miscellaneous functions

The ROM bootloader provides the NBOOT APIs to allow integration of loader operations into customer applications.

NOTE

IFR and PFR refer to "Protected Flash Regions" in this document.

9.2 Functional description

9.2.1 Secure Boot

Secure boot prevents unauthorized code from being executed on a given product. It achieves this level of security by always leaving the device's ROM in an executing mode when coming out of a reset. This allows the ROM to examine the first user executable image resident in internal flash memory to determine the authenticity of that code. If the code is authentic, then control is transferred to it. This establishes a chain of trusted code from the ROM to the user boot code. This chain can be further extended, through the verification of digital signatures associated with additional code layers.

Cipher-based Message Authentication Code (CMAC) and Elliptic Curve Digital Signature Algorithm (ECDSA) are used in this architecture to verify authenticity of the boot code. The boot code is always signed with ECDSA private keys. The corresponding ECDSA public keys used for signature verification (Root of Trust Keys) are contained in certificate block that is contained in the signed image. Support is provided for up to four Root of Trust keys.

During first boot of the application image, the ECDSA algorithm is always used to verify authenticity of the image. After successful authentication, the bootloader calculates CMAC over the whole image in case that IMG_UPD field in CFPA is configured to do so. Device specific CMAC key is used. Calculated CMAC is then stored in CFPA, so that it can be used for authentication during next boot in case that SEC_BOOT_EN field in Customer Manufacturing/Factory Programmable Area (CMPA) is configured to do so.

The device can be configured to boot plain images during development. In this case, the ROM does not check the image to be booted, or performs only CRC32 checking, depending on the configuration.

9.2.2 Secure firmware update

If firmware updates are to be performed in the field when secure boot is enabled, then a secure firmware update mechanism is preferred. Otherwise, inauthentic firmware may be written to the device, preventing it from booting. In the most basic sense, secure firmware updates simply perform an authentication of the new firmware prior to committing it to memory. In this case, the chain of trust is extended from the old, currently executing code, to the new code.

Another use case for secure firmware update is to hide the application binary code during transit over public media such as the web. This is accomplished by encrypting the firmware update image. As the new firmware is written into device memory, it is decrypted. If On-the-fly PRINCE or IPED encryption is already configured and enabled, the firmware will be again encrypted during the write operation.

In this architecture, both cases of secure firmware update are supported. The SB file format is encrypted and digitally signed. The SB file can be loaded via interfaces such as USB, UART, etc., or can be provided to the ROM API as a complete binary file.

9.2.2.1 Extending the chain of trust

After secure boot has transferred CPU control to user code, that code may need to load additional pieces of code. This establishes another link in the chain of trust. The process can continue when many nested sub-modules are required, with each parent code module authenticating the next link in the chain.

The NBOOT API is used from customer code to verify signatures on the additional code images. Using the API to verify signatures gives complete control to the customer code over what additional code must be signed and how that code is organized in memory.

9.2.3 Miscellaneous functions

ROM provides support for various security related functions:

- DICE (Device Identifier Composition Engine)
- Load of TrustZone-M configuration during ROM secure boot
- Booting from encrypted internal Flash regions using NPX peripheral module
- Booting from encrypted external FlexSPI NOR flash using IPED peripheral module
- Debug authentication
- CMPA and CFPA authentication using CMAC
- Device provisioning (ROM embedded support for the initial secure provisioning of the boot keys)

9.2.4 Image authentication flow diagram

During CMAC based image authentication, bootloader computes CMAC value over the boot image and compares it to expected image CMAC stored in CFPA page. In case CMAC based authentication fails, ECDSA authentication is used instead. ECDSA image authentication flow is described in the figure below:

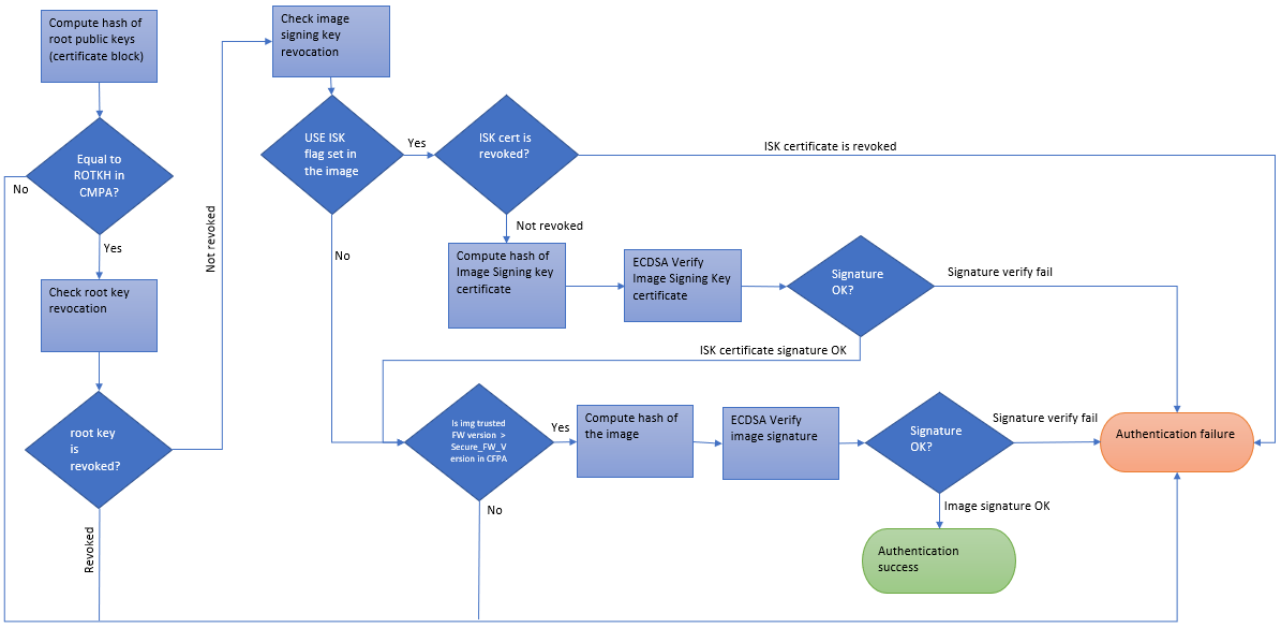


Figure 46. ECDSA Image Authentication Flow chart

9.2.4.1 Internal Flash CMAC authentication boot

For on-chip flash boot, ROM supports image CMAC authentication boot as internal flash boot flow shows. This is a new feature introduced as a part of secure boot and provides faster alternative to the more secure ECDSA based image authentication method. The ROM bootloader will use CMAC based image authentication:

1. During normal boot when SECURE_BOOT_CFG[1:0] in CMPA is configured to allow booting of CMAC signed images (SEC_BOOT_EN == b'10)
2. During boot from power-down & deep-power down state when SECURE_BOOT_CFG[4:3] in CMPA is configured to perform CMAC check of the active image

   The expected CMAC values for image 0 and 1 are stored in CFPA page. These values are entirely managed by ROM bootloader and cannot be changed by the user. When new boot image is loaded or the image is updated, user is responsible to set IMG_UPD flag in CFPA page to inform boot ROM that the expected CMAC values needs to be recalculated. The ROM will then perform complete ECDSA authentication of the new image during first boot. As soon as the image passes the full ECDSA authentication, new image CMAC values are calculated and stored into CFPA. IMG_UPD flag will be cleared automatically when this process is done. Once valid image CMAC in present in the CFPA page, CMAC authentication can be used.

NOTE

Only internal flash boot supports the CMAC image authentication

Table 187. CMAC update flags in CFPA page

CFPA Word Name	FLASH Address	Word bit field	Bit field name	Definition
CFPA_PAGE_VERSION	0100_0004	CFPA_PAGE_VERSION[25:24]	IMG_UPD	Image updated. 00 - No action 01 - Update image 0 CMAC 10 - Update image 1 CMAC 11 - Update SBL image CMAC. Value is written in SBL_IMG0_CMAC field.
		CFPA_PAGE_VERSION[29:27]	CMPA_UPD	CMPA page updated through ROM API. Thus compute CMAC on sub-sequent boot. This field is checked only in Provision/ Develop (0x3) LC state. * 000 - No action * 010 - Update CMAC field in CMPA page. * 011 - Update CMAC field in CMPA page and OTP. Advance OTP_LC_STATE to Develop 2 (0x7). * 101 - Update CMAC field in CMPA page and OTP. Advance OTP_LC_STATE to In-field (0xF). * 110 - Update CMAC field in CMPA page and OTP. Advance OTP_LC_STATE to In-field Locked (0xCF). * All other combinations are ignored.

Table 188. Internal flash Image CMAC value

CFPA Word	FLASH Address	Word Name	Definition
Word56	0100_00E0	SBL_IMG0_CMAC	Internal flash boot image 0/SBL CMAC value word0
Word57	0100_00E4	SBL_IMG0_CMAC	Internal flash boot image 0/SBL CMAC value word1
Word58	0100_00E8	SBL_IMG0_CMAC	Internal flash boot image 0/SBL CMAC value word2
Word59	0100_00EC	SBL_IMG0_CMAC	Internal flash boot image 0/SBL CMAC value word3
Word60	0100_00F0	IMG1_CMAC	Internal flash boot image 1 CMAC value word0
Word61	0100_00F4	IMG1_CMAC	Internal flash boot image 1 CMAC value word1
Word62	0100_00F8	IMG1_CMAC	Internal flash boot image 1 CMAC value word2
Word63	0100_00FC	IMG1_CMAC	Internal flash boot image 1 CMAC value word3

Table 189. Image authentication type selection

CMPA Word Name	FLASH Address	Word Name	Bit Field Name	Definition
SECURE_BOOT_CFG	0104_0050	SECURE_BOOT_CFG[1:0]	SEC_BOOT_EN	Secure boot enforcement. This field defines the minimum image verification procedure (CRC32, CMAC, ECDSA sign). The Image type field in header indicates the type of verification data (checksum or signature) included in it. 00 - All Image types are allowed. 01 - Only CRC32 or signed (CMAC or ECDSA) images are allowed. 10 - Only signed (CMAC or ECDSA) images are allowed. 11 - Only ECDSA signed images are allowed. Plain < CRC32 < CMAC < ECDSA
		SECURE_BOOT_CFG[4:3]	LP_SEC_BOOT	Secure boot option for low-power wake from power-down & deep-powerdown.

Table continues on the next page...

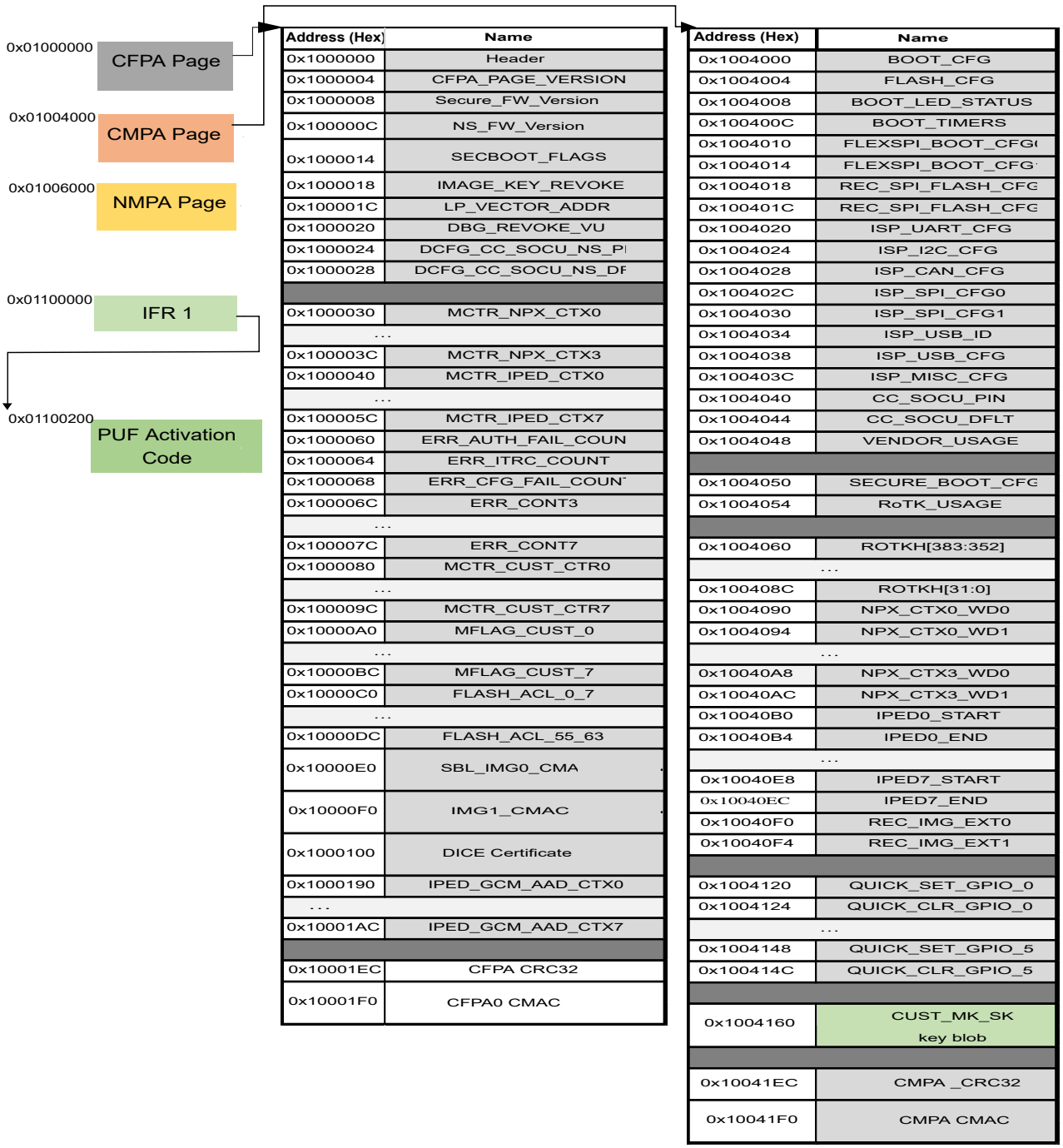
Table 189. Image authentication type selection

CMPA Word Name	FLASH Address	Word Name	Bit Field Name	Definition
				00 - Same as cold boot (use option from SEC_BOOT_EN) 01 - CRC32 check of active image 10 - Jump to vector address specified in CFPA (LP_VECTOR_ADDR). 11 - CMAC check of active image

9.2.4.2 Data structures

9.2.4.2.1 Overview

This device stores secure boot configuration, PUF activation and customer key store code in IFR. It resides at the end of the flash region and can be programmed through ROM in ISP mode.



Address (Hex)	Name
0x1000000	Header
0x1000004	CFPA_PAGE_VERSION
0x1000008	Secure_FW_Version
0x100000C	NS_FW_Version
0x1000014	SECBOOT_FLAGS
0x1000018	IMAGE_KEY_REVOKE
0x100001C	LP_VECTOR_ADDR
0x1000020	DBG_REVOKE_VU
0x1000024	DCFG_CC_SOCU_NS_PI
0x1000028	DCFG_CC_SOCU_NS_DF
0x1000030	MCTR_NPX_CTX0
...	
0x100003C	MCTR_NPX_CTX3
0x1000040	MCTR_IPED_CTX0
...	
0x100005C	MCTR_IPED_CTX7
0x1000060	ERR_AUTH_FAIL_COUN
0x1000064	ERR_ITRC_COUNT
0x1000068	ERR_CFG_FAIL_COUN
0x100006C	ERR_CONT3
...	
0x100007C	ERR_CONT7
0x1000080	MCTR_CUST_CTR0
...	
0x100009C	MCTR_CUST_CTR7
0x10000A0	MFLAG_CUST_0
...	
0x10000BC	MFLAG_CUST_7
0x10000C0	FLASH_ACL_0_7
...	
0x10000DC	FLASH_ACL_55_63
0x10000E0	SBL_IMG0_CMA
0x10000F0	IMG1_CMAC
0x1000100	DICE Certificate
0x1000190	IPED_GCM_AAD_CTX0
...	
0x10001AC	IPED_GCM_AAD_CTX7
0x10001EC	CFPA CRC32
0x10001F0	CFPA0 CMAC

Address (Hex)	Name
0x1004000	BOOT_CFG
0x1004004	FLASH_CFG
0x1004008	BOOT_LED_STATUS
0x100400C	BOOT_TIMERS
0x1004010	FLEXSPI_BOOT_CFG(
0x1004014	FLEXSPI_BOOT_CFG
0x1004018	REC_SPI_FLASH_CFG
0x100401C	REC_SPI_FLASH_CFG
0x1004020	ISP_UART_CFG
0x1004024	ISP_I2C_CFG
0x1004028	ISP_CAN_CFG
0x100402C	ISP_SPI_CFG0
0x1004030	ISP_SPI_CFG1
0x1004034	ISP_USB_ID
0x1004038	ISP_USB_CFG
0x100403C	ISP_MISC_CFG
0x1004040	CC_SOCU_PIN
0x1004044	CC_SOCU_DFLT
0x1004048	VENDOR_USAGE
0x1004050	SECURE_BOOT_CFG
0x1004054	RoTK_USAGE
0x1004060	ROTKH[383:352]
...	
0x100408C	ROTKH[31:0]
0x1004090	NPX_CTX0_WD0
0x1004094	NPX_CTX0_WD1
...	
0x10040A8	NPX_CTX3_WD0
0x10040AC	NPX_CTX3_WD1
0x10040B0	IPED0_START
0x10040B4	IPED0_END
...	
0x10040E8	IPED7_START
0x10040EC	IPED7_END
0x10040F0	REC_IMG_EXT0
0x10040F4	REC_IMG_EXT1
0x1004120	QUICK_SET_GPIO_0
0x1004124	QUICK_CLR_GPIO_0
...	
0x1004148	QUICK_SET_GPIO_5
0x100414C	QUICK_CLR_GPIO_5
0x1004160	CUST_MK_SK key blob
0x10041EC	CMPA _CRC32
0x10041F0	CMPA CMAC

Figure 47. IFR

NOTE

For detailed description of CFPA and CMPA bit fields, please refer to IFR map spreadsheet attached to this manual.

9.2.4.2.1 Key storage in Protected Flash Region

ROM provides area for storing CUST_MK_SK in CMPA at address 0x01004160. All keys are stored in RFC3394 blobs, which are wrapped using die specific key (NXP_DIE_KEK_SK). CUST_MK_SK Blob is the only one used by the bootloader the rest is available for storing of user keys. Key blobs can be created using SB_STORE_KEY command from SB3.1 file.

Device unique PUF activation code stored at 0x01100200 is generated during device provisioning at NXP factory. It cannot be changed.

Table 190. PUF Activation code structure

Address	Size (bytes)	Name	Description
0x01100200	4	Key Store Header	Marker. A value of 0x95959595 means that Activation code is valid.
0x01100204	4	PUF Discharge time	Time in milliseconds to wait until PUF SRAM fully discharges. Only effective when PUF Start fails. Set to zero to use default discharge time.
0x01100208	1000	Activation Code	Device specific PUF activation code generated by enroll command during manufacturing process.

9.3 Keys

CUST_MK_SK is stored in form of RFC3394 blob starting from 0x01004160 and it is used by bootloader to decrypt SB3.1 encryption key during processing of SB file by bootloader. CUST_MK_SK is generated during device provisioning process by HSM_GEN_KEY (random key) or by HSM_STORE_KEY (user defined key) commands. To store this key into IFR, SB_STORE _KEY command should be used. SB_STORE_KEY command takes the key blob, loads it to ELS. Then writes the key in RFC3394 blob into CMPA at given offset.

9.4 Secure boot related configuration fields in IFR

9.4.1 CMPA page

The CMPA (Customer Manufacturing/Factory Programmable Area) page contains settings for a signed image in secure boot configuration, NPX configuration (if encrypted internal flash is needed), IPED configuration (if encrypted external flash is needed), 48 bytes of Root Key Table Hash (RKTH) and CUST_MK_SK key blob used by bootloader to decrypt SB file. Only secure boot related fields are described in this chapter.

Table 191. Configuration overview

Address	Bytes	Has OTP variant	Name	Description
0x01004050	4	Yes	SECURE_BOOT _CFG	Secure boot configuration flags.
0x01004054	4	Yes	RoTK_USAGE	RoT keys usage properties, DICE config, NPX context lock
0x01004060	48	Yes	RKTH	Root Key Table Hash.
0x01004090	4	Yes	NPX_CTX0_WD0	Bitmap mask for NPX Prince context 0 (sectors 0 -31)
0x01004094	4	Yes	NPX_CTX0_WD1	Bitmap mask for NPX Prince context 0 (sectors 32-63)

Table continues on the next page...

Table 191. Configuration overview (continued)

Address	Bytes	Has OTP variant	Name	Description
0x01004098	4	Yes	NPX_CTX1_WD0	Bitmap mask for NPX Prince context 1 (sectors 0 -31)
0x0100409C	4	Yes	NPX_CTX1_WD1	Bitmap mask for NPX Prince context 1 (sectors 32-63)
0x010040A0	4	Yes	NPX_CTX2_WD0	Bitmap mask for NPX Prince context 2 (sectors 0 -31)
0x010040A4	4	Yes	NPX_CTX2_WD1	Bitmap mask for NPX Prince context 2 (sectors 32-63)
0x010040A8	4	Yes	NPX_CTX3_WD0	Bitmap mask for NPX Prince context 3 (sectors 0 -31)
0x010040AC	4	Yes	NPX_CTX3_WD1	Bitmap mask for NPX Prince context 3 (sectors 32-63)
0x010040B0	4	No	IPED0_START	IPED_CTX0_START_ADDR
0x010040B4	4	No	IPED0_END	IPED_CTX0_END_ADDR
0x010040B8	4	No	IPED1_START	IPED_CTX1_START_ADDR
0x010040BC	4	No	IPED1_END	IPED_CTX1_END_ADDR
0x010040C0	4	No	IPED2_START	IPED_CTX2_START_ADDR
0x010040C4	4	No	IPED2_END	IPED_CTX2_END_ADDR
0x010040C8	4	No	IPED3_START	IPED_CTX3_START_ADDR
0x010040CC	4	No	IPED3_END	IPED_CTX3_END_ADDR
0x010040D0	4	No	IPED4_START	IPED_CTX4_START_ADDR
0x010040D4	4	No	IPED4_END	IPED_CTX4_END_ADDR
0x010040D8	4	No	IPED5_START	IPED_CTX5_START_ADDR
0x010040DC	4	No	IPED5_END	IPED_CTX5_END_ADDR
0x010040E0	4	No	IPED6_START	IPED_CTX6_START_ADDR
0x010040E4	4	No	IPED6_END	IPED_CTX6_END_ADDR
0x010040E8	4	No	IPED7_START	IPED_CTX7_START_ADDR
0x010040EC	4	No	IPED7_END	IPED_CTX7_END_ADDR
0x01004160	48	No	CUST_MK_SK key blob	CUST_MK_SK wrapped in RFC3394 key blob

NOTE

OTP variant of the configuration has always higher priority than the value stored in CMPA. Please refer to OTP fuse map for address and bit field definitions.

Table 192. SECURE_BOOT_CFG word bit field definitions

Address	Bit(s)	Name	Description
0x01004050	1:0	SEC_BOOT_EN	Secure boot enforcement. This field defines the minimum image verification procedure (CRC32, CMAC, ECDSA sign). The Image type field in header indicates the type of verification data (checksum or signature) included in it. 2'b00 - All Image types are allowed. 2'b01 - Only CRC32 or signed (CMAC or ECDSA) images are allowed. 2'b10 - Only Signed (CMAC or ECDSA) images are allowed. 2'b11 - Only ECDSA signed images are allowed.
	2	RESERVED	Reserved, filled with zeros
	4:3	LP_SEC_BOOT	Secure boot option for low-power wake from power-down & deep-powerdown. 2'b00 - Same as cold boot 2'b01 - CRC check for IFR & CRC32 check of active image 2'b10 - CRC check for IFR & jump to vector address specified in CFPA. 2'b11 - CRC check for IFR & CMAC check of active image
	7:5	RESERVED	Reserved, filled with zeros
	9:8	ENF_CNSA	Enforce CNSA suite approved algorithms for secure boot, secure update and debug authentication. 00 - All algorithms allowed. 01, 10, 11 - Only ECC P-384 keys, SHA384 and AES256 algorithms are used **NOTE** CMAC image authentication is only allowed in 2b'00 setting.
	11:10	ENF_TZM_PRESET	TrustZone-M mode 2'b00: Ignored 2'b01: Enforce preset TZM data in image manifest. 2'b10: Enforce preset TZM data in image manifest. 2'b11: Enforce preset TZM data in image manifest.
	13:12	ITRC_ZEROIZE	RAM zeroize on ITRC event. Zeroize long-term storage RAM (AO_RAM) on subsequent boot when Intrusion & Tamper control signals an intrusion event. 2b'00 - No action.

Table continues on the next page...

Table 192. SECURE_BOOT_CFG word bit field definitions (continued)

Address	Bit(s)	Name	Description
			2b'01 - Always Zeroize PKC RAM on ITRC event only. 2b'10 - Zeroize PKC & AO RAM on ITRC event only. 2b'11 - Always Zeroize PKC & AO RAM on all reset boot.
	15:14	ACTIVE_IMG_PROT	Protection of active image. This field defines protection of flash area occupied by active image. MBC module is used for implementing image protection functionality. The protection is enforced on AHB access. Only applicable to internal flash. 2b'00 - Protection is defined using the CFPA FLASH_ACL settings. 2b'01 - Write protect active image area with sticky lock. GLBAC2 is used. FLASH_ACL settings are ignored. 2b'10 - Write protect active image area without sticky lock. GLBAC4 is used. FLASH_ACL settings are ignored. 2b'11 - XOM protect active image area with sticky lock. GLBAC6 is used. FLASH_ACL settings are ignored.
	17:16	FIPS_SHA_STEN	Enable self-test for SHA2 block on power-up. Needed for FIPS certification. If this field is non-zero, run self-test and log result in the ELS_AS_BOOT_LOG1 register (see SYSCON chapter for details of this register). 2'b00 not included 2'b01: On failure continue to boot. 2'b10: On failure enter ISP mode for recovery. 2'b11: On failure lock the device to enforce power-cycle.
	19:18	FIPS_AES_STEN	Enable self-test for AES block on power-up. Needed for FIPS certification. If this field is non-zero, run self-test and log result in the ELS_AS_BOOT_LOG1 register (see SYSCON chapter for details of this register). 2'b00 not included 2'b01: On failure continue to boot. 2'b10: On failure enter ISP mode for recovery. 2'b11: On failure lock the device to enforce power-cycle.
	21:20	FIPS_ECDSA_STEN	Enable self-test for ECDSA block on power-up. Needed for FIPS certification. If this field is non-zero, run self-test and log result in the ELS_AS_BOOT_LOG1 register (see SYSCON chapter for details of this register) 2'b00 not included 2'b01: On failure continue to boot.

Table continues on the next page...

Table 192. SECURE_BOOT_CFG word bit field definitions (continued)

Address	Bit(s)	Name	Description
			2'b10: On failure enter ISP mode for recovery. 2'b11: On failure lock the device to enforce power-cycle.
	23:22	FIPS_DRBG_STEN	Enable self-test for DRBG block on power-up. Needed for FIPS certification. If this field is non-zero, run self-test and log result in the ELS_AS_BOOT_LOG1 register (see SYSCON chapter for details of this register). 2'b00 not included 2'b01: On failure continue to boot. 2'b10: On failure enter ISP mode for recovery. 2'b11: On failure lock the device to enforce power-cycle.
	25:24	FIPS_CMAC_STEN	Enable self-test for CMAC block on power-up. Needed for FIPS certification. If this field is non-zero run self-test and log result in BOOT_STATE register. 2'b00 not included 2'b01: On failure continue to boot 2'b10: On failure enter ISP mode for recovery. 2'b11: On failure lock the device to enforce power-cycle.
	27:26	FIPS_KDF_STEN	Enable self-test for KDF block on power-up. Needed for FIPS certification. If this field is non-zero run self-test and log result in BOOT_STATE register. 2'b00 not included 2'b01: On failure continue to boot 2'b10: On failure enter ISP mode for recovery. 2'b11: On failure lock the device to enforce power-cycle.
	29:28	FIPS_PUF_STEN	Enable self-test for PUF block on power-up. Needed for FIPS certification. If this field is non-zero, run self-test and log result in the ELS_AS_BOOT_LOG1 register (see SYSCON chapter for details of this register). 2'b00 - not included 2'b01 - On failure continue to boot. 2'b10 - On failure enter ISP mode for recovery. 2'b11 - On failure lock the device to enforce power-cycle.
	31:30	RESERVED	Reserved, filled with zeros

Table 193. RoTK_USAGE

Address	Bit(s)	Name	Description
0x01004054	2:0	RoTK0_Usage	RoT key 0 usage properties. 000 - Usable as debug CA, image CA, FW CA, image and FW key. 001 - Usable as debug CA only. 010 - Usable as image (boot & FW) CA only. 011 - Usable as debug, boot & FW image CA. 100 - Usable as image key & FW update key only. 101 - Usable as boot image key only. 110 - Usable as FW update image key only. 111 - Key slot is not used.
	5:3	RoTK1_Usage	RoT key 1 usage properties. 000 - Usable as debug CA, image CA, FW CA, image and FW key. 001 - Usable as debug CA only. 010 - Usable as image (boot & FW) CA only. 011 - Usable as debug, boot & FW image CA. 100 - Usable as image key & FW update key only. 101 - Usable as boot image key only. 110 - Usable as FW update image key only. 111 - Key slot is not used.
	8:6	RoTK2_Usage	RoT key 2 usage properties. 000 - Usable as debug CA, image CA, FW CA, image and FW key. 001 - Usable as debug CA only. 010 - Usable as image (boot & FW) CA only. 011 - Usable as debug, boot & FW image CA. 100 - Usable as image key & FW update key only. 101 - Usable as boot image key only. 110 - Usable as FW update image key only. 111 - Key slot is not used.
	11:9	RoTK3_Usage	RoT key 3 usage properties. 000 - Usable as debug CA, image CA, FW CA, image and FW key. 001 - Usable as debug CA only. 010 - Usable as image (boot & FW) CA only. 011 - Usable as debug, boot & FW image CA. 100 - Usable as image key & FW update key only.

Table continues on the next page...

Table 193. RoTK_USAGE (continued)

Address	Bit(s)	Name	Description
			101 - Usable as boot image key only. 110 - Usable as FW update image key only. 111 - Key slot is not used.
	12	SKIP_DICE	Skip DICE computation. 0 - Enable DICE 1 - Disable DICE
	13	DICE_INC_NXP_CFG	Include NXP area (IFR1) containing specific part configuration data defined during chip manufacturing process. 0 - not included 1 - included
	14	DICE_INC_CUST_CFG	Include data from CMPA page (512 bytes) in DICE computation. 0 - not included 1 - included
	23:15	RESERVED	Reserved, filled with zeros
	25:24	NPX_LOCK_CTX0	Lock on-chip flash NPX PRINCE context 0 settings. 00 - Region is not locked. 01 - Region is locked. But no erase check. 10, 11 - Region is locked. Erase check enabled. Flash programming ROM API enforces the erase check. When erase check is enabled all sectors selected in the context should be erased together and only contiguous sectors can be selected in context bitmap.
	27:26	NPX_LOCK_CTX1	Lock on-chip flash NPX PRINCE context 1 settings. 00 - Region is not locked. 01 - Region is locked. But no erase check. 10, 11 - Region is locked. Erase check enabled. Flash programming ROM API enforces the erase check. When erase check is enabled all sectors selected in the context should be erased together and only contiguous sectors can be selected in context bitmap.
	29:28	NPX_LOCK_CTX2	Lock on-chip flash NPX PRINCE context 2 settings. 00 - Region is not locked. 01 - Region is locked. But no erase check. 10, 11 - Region is locked. Erase check enabled. Flash programming ROM API enforces the erase check. When erase check is enabled all sectors selected in the context should be erased together and only contiguous sectors can be selected in context bitmap.

Table continues on the next page...

Table 193. RoTK_USAGE (continued)

Address	Bit(s)	Name	Description
	31:30	NPX_LOCK_CTX3	Lock on-chip flash NPX PRINCE context 3 settings. 00 - Region is not locked. 01 - Region is locked. But no erase check. 10, 11 - Region is locked. Erase check enabled. Flash programming ROM API selected the erase check. When erase check is enabled all sectors selected in the context should be erased together and only contiguous sectors can be selected in context bitmap.

Table 194. RKTH layout in CMPA

Address	Description	Address	Description
0x01004060	RKTH[383:352]	0x01004078	RKTH [191:160]
0x01004064	RKTH[351:320]	0x0100407C	RKTH [159:128]
0x01004068	RKTH[319:288]	0x01004080	RKTH [127:96]
0x0100406C	RKTH [287:256]	0x01004084	RKTH [95:64]
0x01004070	RKTH[255:224]	0x01004088	RKTH [63:32]
0x01004074	RKTH [223:192]	0x0100408C	RKTH [31:0]

RKTH:

For ECC P-256 keys, RKTH is a 32-byte SHA-256 digest of four SHA-256 digests computed over four OEM public keys (OEM has four private-public key pairs in case one of its private keys becomes compromised). If ECC P-384 keys are used, RKTH is 48-byte SHA-384 digest. Actual size of the RKTH in CMPA is then determined by the size of the used hash. For P-256 keys, the RKTH size is only 32 bytes. In this case RKTH should be also written starting from 0x01004060, but only RKTH [383:352] up to RKTH [159:128] will be used. Remaining 16 bytes are unused and should stay filled with zeros. The structure of this table is shown in Figure 48.

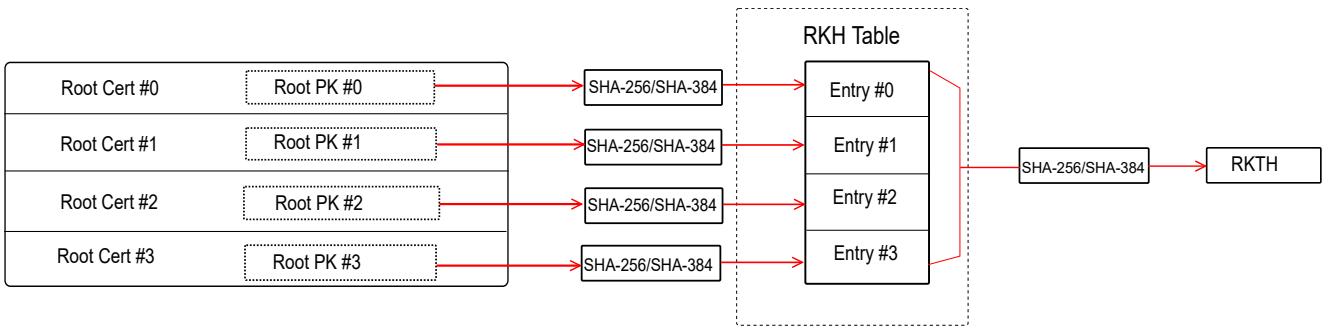


Figure 48. RKTH generation process

If we denote OEM public keys as OEM_PBK1, OEM_PBK2, OEM_PBK3, OEM_PBK4, then RKTH is computed as:

```
RKTH = SHA-256(SHA-256(OEM_PBK1), SHA-256(OEM_PBK2), SHA-256(OEM_PBK3), SHA-256(OEM_PBK4))
```

The number of hashes of keys in the RKH table must range from at least 1 through a maximum of 4. Unused table entries must be set to all 0 bytes. When searching the RKH table for a key's hash, the loader will stop at the first entry that is all zeroes.

The extra root public keys and root certificates must be created in advance and are held in reserve in case a public key has to be revoked. The customer is responsible for implementing the mechanism to determine whether a key needs to be revoked, and

to then set the appropriate RoTKx_EN configuration in CFPA. This is usually accomplished through an authenticated connection with a server during a firmware update.

NOTE

Only one of the root certificates whose keys are listed in the RKH table may be included in the certificate table at a time.

NPX_CTXx_WD0/1:

When on-the-fly encryption/decryption of internal flash using NPX Prince is enabled, ROM configures the sector encryption enable bits for a given memory region according to a value stored in this word as shown in Table 195.

Table 195. NPX_CTXx_WD0 sector enable bits

Address	Bit(s)	Name	Description
0x01004090 + 8*x	31:0	NPX_CTXx_WD0	NPX Prince Context x bitmap (sectors 31 - 0) Each bit in this field enables encryption/decryption of flash sector at offset 32kB*n, where n is the bit number. A 0 in bit n means encryption and decryption of data associated with sector n is disabled. A 1 in bit n means that data written to sub-region n during flash programming will be encrypted, and flash AHB reads from sector n will be decrypted using the NPX PRINCE.

Table 196. NPX_CTXx_WD1 sector enable bits

Address	Bit(s)	Name	Description
0x01004094 + 8*x	31:0	NPX_CTXx_WD1	NPX Prince Context x bitmap (sectors 63 - 32) Enable bits for flash sectors 63 - 32.

IPED_CTXx_START/END_ADDR:

When on-the-fly encryption/decryption of external flash using IPED is enabled, ROM configures the region encryption based on configuration stored in Table 197 and Table 198.

Table 197. IPED_CTXx_START_ADDR - IPED region start address

Address	Bit(s)	Name	Description
0x010040B0 + 8*x	0	GCM_MODE	GCM mode enable. 0 - Region is enabled in CTR mode. 1 - Region is enabled in GCM mode.
	1	AHBERR_DIS	Disable AHB bus error. If GCM authentication fails generates bus error or not. 0 - Bus error enabled. 1 - Bus error disabled.
	31:8	IPED_CTXx_START_ADDR	Upper 24-bits of IPED region start address.

Table 198. IPED_CTXx_END_ADDR - IPED region END address

Address	Bit(s)	Name	Description
0x010040B4 + 8*x	1:0	LOCK_EN	Lock the external IPED context settings. 00 - Region is not locked. 01, 10, 11 - Region is locked.
	31:8	IPED_CTXx_END_ADDR	Upper 24-bits of IPED region end address.

CUST_MK_SK_BLOB:

CUST_MK_SK is stored in form of RFC3394 blob and it is used by bootloader to decrypt SB3.1 encryption key during processing of SB file by bootloader. CUST_MK_SK is generated during device provisioning process by HSM_KEY_GEN (random key) or by HSM_STORE_KEY (user defined key) commands. To store this key into CMPA, SB_STORE_KEY command should be used. SB_STORE_KEY command takes the key blob, loads it to ELS. Then writes the key in RFC3394 blob into CMPA at given offset.

Table 199. CUST_MK_SK key blob

Address	Bytes	Name	Description
0x01004160	48	CUST_MK_SK_BLOB	RFC3394 wrapped key blob of CUST_MK_SK

9.4.2 CFPA page

The CFPA (Customer Field Programmable Area) page contains additional editable settings for signed image, secure boot configuration and monotonic erase counters for NPX/IPED engines. Only secure boot related fields are described in this chapter.

Table 200. CFPA page layout

Address	Byte(s)	Name	Description
0x01000004	4	CFPA_PAGE_VERSION	CFPA page version monotonic counter and update flags
0x01000008	4	Secure_FW_ Version	Secure firmware version (Monotonic counter)
0x0100000C	4	NS_FW_Version	Non Secure firmware version (Monotonic counter)
0x01000014	4	SECBOOT_FLAGS	RoT keys enable and OEM Return Life cycle erase done flag
0x01000018	4	IMAGE_KEY _REVOKE	Image key revocation ID
0x01000030	4	MCTR_NPX_CTX0	Erase count for internal NPX region 0 (Monotonic counter)
0x01000034	4	MCTR_NPX_CTX1	Erase count for internal NPX region 1 (Monotonic counter)
0x01000038	4	MCTR_NPX_CTX2	Erase count for internal NPX region 2 (Monotonic counter)
0x0100003C	4	MCTR_NPX_CTX3	Erase count for internal NPX region 3 (Monotonic counter)
0x01000040	4	MCTR_IPED_CTX0	Erase count for external IPED region 0 (Monotonic counter)
0x01000044	4	MCTR_IPED_CTX1	Erase count for external IPED region 1 (Monotonic counter)
0x01000048	4	MCTR_IPED_CTX2	Erase count for external IPED region 2 (Monotonic counter)
0x0100004C	4	MCTR_IPED_CTX3	Erase count for external IPED region 3 (Monotonic counter)
0x01000050	4	MCTR_IPED_CTX4	Erase count for external IPED region 4 (Monotonic counter)
0x01000054	4	MCTR_IPED_CTX5	Erase count for external IPED region 5 (Monotonic counter)

Table continues on the next page...

Table 200. CFPA page layout (continued)

Address	Byte(s)	Name	Description
0x01000058	4	MCTR_IPED_CTX6	Erase count for external IPED region 6 (Monotonic counter)
0x0100005C	4	MCTR_IPED_CTX7	Erase count for external IPED region 7 (Monotonic counter)
0x01000060	4	ERR_AUTH_FAIL_COUNT	Authentication failure counter during boot
0x01000064	4	ERR_ITRC_COUNT	Tamper event counter
0x010000E0	16	SBL_IMG0_CMAC	Expected CMAC value calculated over boot image 0
0x010000F0	16	IMG1_CMAC	Expected CMAC value calculated over boot image 1
0x01000100	144	DICE Certificate	DICE Certificate generated by boot ROM
0x01000190	4	IPED_GCM_AAD_CTX0	Additional Authentication Data for IPED region 0
0x01000194	4	IPED_GCM_AAD_CTX1	Additional Authentication Data for IPED region 1
0x01000198	4	IPED_GCM_AAD_CTX2	Additional Authentication Data for IPED region 2
0x0100019C	4	IPED_GCM_AAD_CTX3	Additional Authentication Data for IPED region 3
0x010001A0	4	IPED_GCM_AAD_CTX4	Additional Authentication Data for IPED region 4
0x010001A4	4	IPED_GCM_AAD_CTX5	Additional Authentication Data for IPED region 5
0x010001A8	4	IPED_GCM_AAD_CTX6	Additional Authentication Data for IPED region 6
0x010001AC	4	IPED_GCM_AAD_CTX7	Additional Authentication Data for IPED region 7

CFPA_PAGE_VERSION: CFPA page monotonic counter. Upper bits are used as flags to trigger update of image or CMPA CMAC and DICE certificate during next boot. Please refer to Table 201.

Table 201. CFPA_PAGE_VERSION bit field description

Address	Bit(s)	Name	Description
0x01000004	23:0	CFPA page version	CFPA page version (Monotonic counter, should start from 1)
	25:24	IMG_UPD	Image updated. 00 - No action 01 - Update image 0 CMAC 10 - Update image 1 CMAC 11 - Update SBL image CMAC. Value written in SBL_IMG0_CMAC field.
	26	RESERVED	Should be set to zero
	29:27	CMPA_UPD	CMPA page was updated, thus compute CMPA CMAC signature on sub-sequent boot. User is responsible to always set CMPA_UPD bits correspondingly when updating CMPA. This field is checked only in Develop (0x3) LC state. 000 - No action 010 - Update CMAC field in CMPA page.

Table continues on the next page...

Table 201. CFPA_PAGE_VERSION bit field description (continued)

Address	Bit(s)	Name	Description
			011 - Update CMAC field in CMPA page and OTP. Advance OTP_LC_STATE to Develop2 (0x7). 101 - Update CMAC field in CMPA page and OTP. Advance OTP_LC_STATE to In-field (0xF). 110 - Update CMAC field in CMPA page and OTP. Advance OTP_LC_STATE to In-field Locked (0xCF). * All other combinations updates CMAC field in CMPA page.
	30	RESERVED	Should be set to zero
	31	DICE_UPD	Update DICE certificate during next boot. 0 - No action 1 - Generate certificate.

Secure_FW_version: Optionally used during SB3.1 file loading. The secure FW version value specified in the elftosb.json file used to create the SB3.1 file must always be greater or equal to value written in this configuration word. Otherwise, if this version check command is included in the SB3.1 file, the file load will be rejected.

NS_FW_version: Optionally used during SB3.1 file loading. The non-secure FW version value specified in the elftosb.json file used to create the SB3.1 file must always be greater or equal to value written in this configuration word. Otherwise, if this version check command is included in the SB3.1 file, the file load is rejected.

SECBOOT_FLAGS: Each of four RoT Keys can be revoked. When trying to boot Images that are signed using a revoked RoT key, they will be rejected during the authentication process and fail to boot if SEC_BOOT_EN is set to boot only signed images. The OEM_RETURN_ERASE_DONE flag is managed automatically by boot ROM.

Table 202. SECBOOT_FLAGS bit field description

Address	Bit(s)	Name	Description
0x01000014	1:0	RoTK0_EN	RoT Key 0 enable 2'b00: RoT Key 0 is enabled 2'b01: RoT Key 0 is enabled 2'b10: RoT Key 0 is revoked 2'b11: RoT Key 0 is revoked
	3:2	RoTK1_EN	RoT Key 1 enable 2'b00: RoT Key 1 is enabled 2'b01: RoT Key 1 is enabled 2'b10: RoT Key 1 is revoked 2'b11: RoT Key 1 is revoked
	5:4	RoTK2_EN	RoT Key 2 enable 2'b00: RoT Key 2 is enabled 2'b01: RoT Key 2 is enabled

Table continues on the next page...

Table 202. SECBOOT_FLAGS bit field description (continued)

Address	Bit(s)	Name	Description
			2'b10: RoT Key 2 is revoked 2'b11: RoT Key 2 is revoked
	7:6	RoTK3_EN	RoT Key 3 enable 2'b00: RoT Key 3 is enabled 2'b01: RoT Key3 is enabled 2'b10: RoT Key3 is revoked 2'b11: RoT Key 3 is revoked
	27:8	RESERVED	Should be filled with zeros
	31:28	OEM_RETURN_ERASE_DONE	ROM sets this field to 0xA after erasing the user flash and OEM assets. This operation is done on first boot after OTP_LC_STATE is advanced to Field return OEM and beyond.

IMAGE_KEY _REVOKE: This value is checked during the image authentication process. Lower 24 bits of IMAGE_KEY_REVOKE field in CFPA and 8 bits of OTP_IMAGE_KEY_REVOKE field are combined to form a 32 bit unary encoded monotonic counter which is checked against “constraints” field of ISK certificate inside image certificate block. Constraints field of image certificate must be always higher then this combined field to be able to boot this image.

MCTR_NPX_CTXn: Monotonic erase counter for NPX region n. This value is used by bootloader to dynamically compute region IV. This counter will increment by one, during each erase cycle of the corresponding flash region. User should not write anything in this field. This field is entirely handled by ROM. Final IV value for given region used by NPX for encryption/decryption is computed by ROM bootloader and incorporates device UUID, NPX region number and MCTR_NPX_CTXn.

MCTR_IPED_CTXn: Monotonic erase counter for IPED region n. This value is used by bootloader to dynamically compute region IV. This counter will increment by one, during each erase cycle of the corresponding flash region. User should not write anything in this field. This field is entirely handled by ROM. Final IV value for given region used by IPED for encryption/decryption is computed by ROM bootloader and incorporates device UUID, IPED region number and MCTR_IPED_CTXn.

ERR_AUTH_FAIL_COUNT: Authentication failure counter during boot, SB3, debug authentication (Monotonic counter).

ERR_ITRC_COUNT: Tamper event counter (ITRC reset, WDT reset, WDT reset, GDET reset, Tamper pin) (Monotonic counter).

SBL_IMG0_CMAC: Expected CMAC value of index 0 boot image. This value is generated during first boot of the application image after successful ECDSA based authentication when IMG_UPD is set to 0b01. If CMAC based authentication is allowed in CMPA, CMAC will be then used for image authentication during boot or low power wakeup.

IMG1_CMAC: Expected CMAC value of index 1 boot image. This value is generated during first boot of the application image after successful ECDSA based authentication when IMG_UPD is set to 0b10. If CMAC based authentication is allowed in CMPA, CMAC will be then used for image authentication during boot or low power wakeup.

DICE Certificate: DICE Certificate Signing Request (CSR) is generated after DICE_UPD (bit 31 in CFPA_PAGE_VERSION) flag is set. The DICE CSR can be generated only in case that secure boot is enabled and SKIP_DICE is set to 0 in CMPA. Signature of this certificate can be verified by NXP_CUST_DICE_CA_PUK generated by ISP GET_OEM_CUST_CERT_DICE_PUK provisioning command.

IPED_GCM_AAD_CTXx: Additional Authentication Data for IPED region n. This value is optional and only used when IPED region operates in GCM mode. ROM will copy this value to corresponding IPEDCTXnAAD0 register during boot. User who wants FCB page data and version page data integrity protected can for example calculate CRC32 of those pages and write it the corresponding contexts IPED_GCM_AAD_CTXn field in the CFPA page.

9.5 Plain image structure

Unsigned Plain CRC images on ROM are supported only when SEC_BOOT_EN configuration in CMPA is set to allow booting of all or CRC32 images. This image type is mainly used during development life cycle state.

The structure of unsigned CRC images is shown in Figure 49.

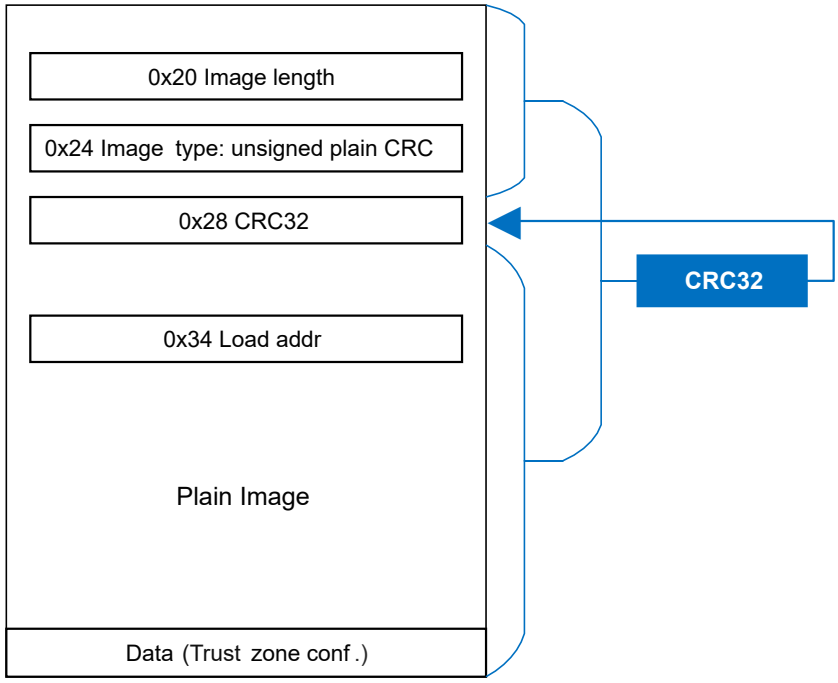


Figure 49. Structure of Unsigned CRC images

NOTE

When Image Type is 0x0, CRC32 checking is bypassed. Such an image can be used as a generic image during development.

9.6 Signed image structure

Images are signed using the ECDSA P-256 or P-384 algorithm. The digest is computed using SHA-256 for ECDSA P-256 signed images and SHA-384 for ECDSA P-384 signed images.

The structure of signed images is shown in Figure 50.

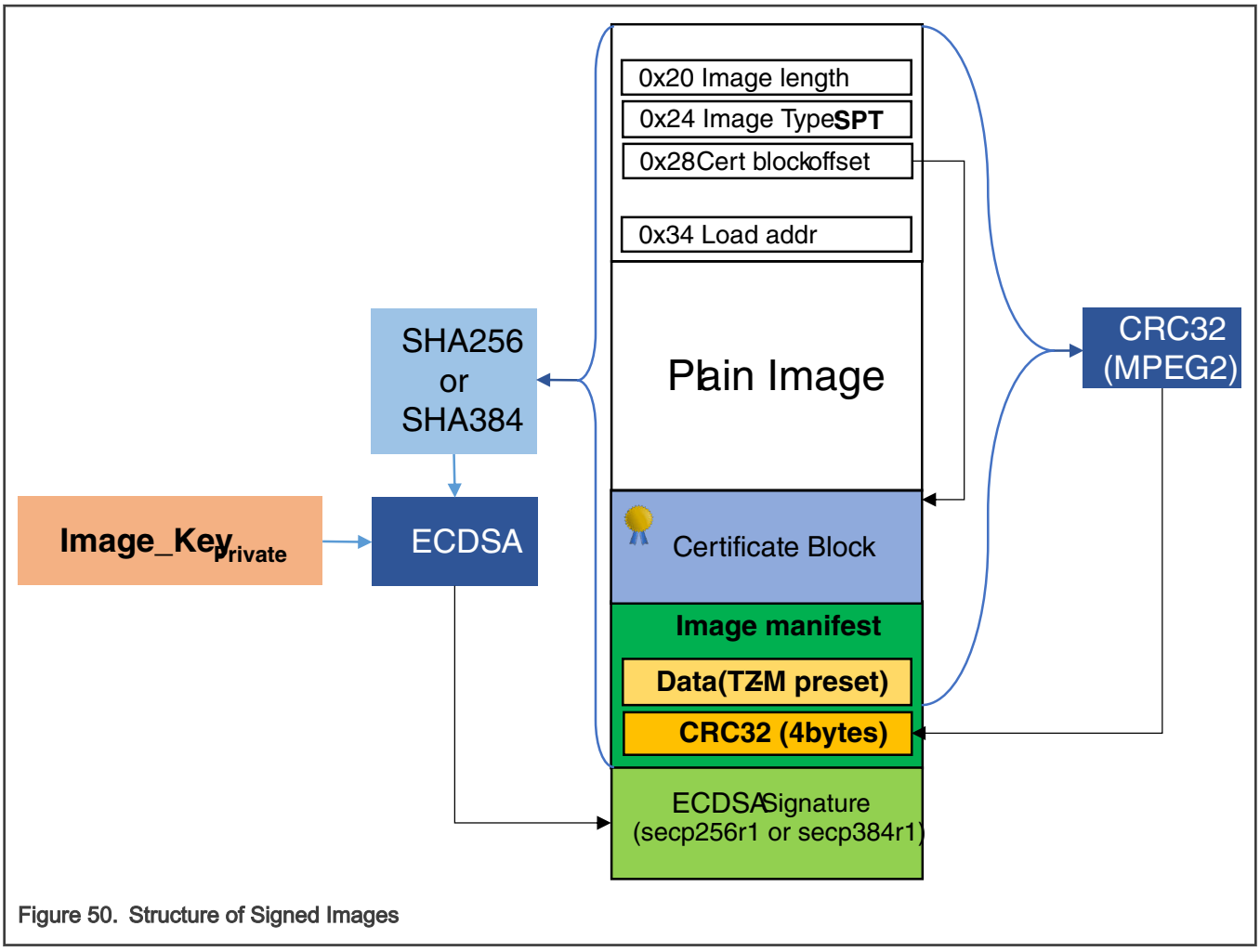


Figure 50. Structure of Signed Images

Image length - total length of the image in bytes including signature

Table 203. ROW Image Type (word at offset 0x24)

31:14	Reserved	Set to 0
13	TZ-M Preset	0: No TZ-M peripherals preset. 1: TZ-M peripherals preset. The TZ-M related peripherals are configured by bootloader based on data stored in image manifest
12:8	Reserved	Set to 0
5:0	Image Type	0x0: plain image 0x2: plain image with CRC 0x4: Xip plain signed 0x5: Xip plain with CRC 0x6: SB3 manifest 0x7: NXP reserved for provisioning

Table continues on the next page...

Table 203. ROW Image Type (word at offset 0x24) (continued)

		0x8: NXP reserved for provisioning Other values are reserved.
7:6	Image Subtype	0x0: default 0x1: recovery

Header Offset -A 32-bit offset from the beginning of the signed image to the certificate block header, called offsetToCertificateBlockInBytes, must reside at offset 0x28 from the start of the signed image. An executable code image in internal flash is expected to start with an NVIC vector table. The word at offset 0x28 is a reserved slot in the vector table.

As an example, if an image resides in flash at a non-zero address (say 0x8000), and its certificate block header is at address 0x24000, then the word at 0x8028 will contain the value 0x1C000.

Here is a standard Cortex-M33 NVIC vector table with the offset to the certificate block header highlighted.

Table 204. Cortex-M33 NVIC vector table

Offset (hex)	Usage
0	Initial SP
4	Reset
8	**NMI**
C	HardFault
10	MemManage
14	BusFault
18	UsageFault
1C	*Reserved*
20	Image Length
24	Image Type
28	**offsetToCertificateBlockInBytes**
30	SVC
34	DebugMon
38	*Reserved*
3C	SysTick

9.6.1 Certificate block

See the Certificate block v2.1 section.

9.6.2 Image Manifest

Image Manifest contains additional image data such as firmware version, optional Trust-Zone preset data and CRC32 value of image used for fast low power wake-up image authentication.

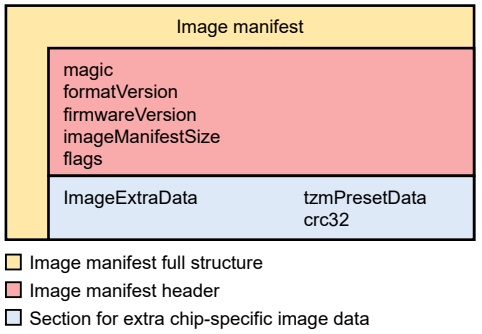


Figure 51. Image manifest

Table 205. Image Manifest

Image Manifest	Offset	Data Type (little endian)	Description
magic	0x0	uint32_t	Fixed 4-byte string "imgm" without the trailing NULL, 0x6D676D69u.
formatVersion	0x4	uint32_t	Fixed value "1.0", 0x00010000u, major = 1, minor = 0.
firmwareVersion	0x8	uint32_t	Value compared with Secure_FW_Version monotonic counter value stored in CMPA. If value is lower than value in IFR, then is image rejected (rollback protection).
imageManifestSize	0xC	uint32_t	Total size of image manifest in bytes including image extra data.
flags	0x10	uint32_t	Reserved
tzmPresetData	0x14	uint8_t[tzm_size]	Trust-Zone preset data, optional. For size and content please refer to chapter 31.10.1 Trust-Zone Preset Data, data must be aligned to 4 bytes.
crc32	0x14 + tzm_size	uint32_t	CRC32 checksum of boot image (MPEG2 CRC32 specification), for content of CRC32 checksum calculation please refer to Figure 50. Used for fast low power wake-up image authentication.

9.7 ROM firmware update using SB file

The Secure Binary (SB) image format is a command-based firmware update image. The SB file can be considered as a type of script (commands and data), for which the ROM is the interpreter. The ROM silicon supports version 3.1 of the SB image format. The ReceiveSBFile command verifies the digital signature.

Details about SB3.1 firmware update container are in SB3.1 firmware update container.

9.8 Key management

The main purpose of key management is to protect confidentiality and unauthorized access to the keys that are stored inside the device. Edge Lock Secure Subsystem (ELS) is designed in such way, that the value of stored keys remains unknown to any

software or firmware running outside of ELS. The PUF and NPX/IPED peripherals are connected to the ELS using dedicated private key buses, that are not exposed to system memory. When key needs to be stored outside ELS, it is unloaded in form of wrapped blob, which can be unwrapped just inside ELS and only on device for which it was created.

- Key provisioning - A process in which keys are being created inside the device. Keys and their properties are brought into the device for the first time.
- Key loading - After the keys and their properties have been provisioned or created using key generation or key derivation function, they will reside (wrapped) in non-volatile memory. To use those keys, they need to be first imported into ELS Key loading operation is used during provisioning as well as the normal device operation (run-time). Some OEM application keys may need to be made available to application software, in which case either ELS or PUF will be delivering them.
- Key unloading - Keys and their properties that are available in the ELS key store, needs to be wrapped by RFC3394 blob before shipping out to the external world, e.g. non-volatile memory. Key unloading is the mechanism that will be used for that. Similar to key loading, key unloading is used either during provisioning as well as the normal operation (run-time).
- Key derivation - During this process, existing key is used to derive a new key using a key derivation function (CKDF, HKDF…).
- Key generation - Key generation, in contrast to key derivation, is responsible for generating (creating) keys from "scratch", e.g. by using a DRBG output.
- Key access control - The key access is explicitly controlled by the rules defined in corresponding key properties. Due to a special input used during key derivation function, implicit access control is also achieved for all the derived keys.

9.8.1 PUF role in key management

The initial device identity is created at NXP during trust provisioning process that takes place at the NXP manufacturing floor. At that time PUF is enrolled to generate device specific PUF Activation Code (AC) which is stored starting from address 0x01100200 in IFR1. From that point, ROM on each start-up of the device, starts the PUF using this AC and generates Device Unique Key called NXP_DIE_MK_SK. This key is then loaded into ELS using KEYIN command. After that, PUF is entirely available to be used by user, since rest of the key operations takes place inside ELS.

9.8.2 ELS key management

Once the NXP_DIE_MK_SK is received by ELS, the ROM is responsible for generating of various keys used during bootloader execution. NXP_DIE_MK_SK key is used as a master key when deriving remaining die unique keys in the key hierarchy. Key hierarchy is shown at Figure 52. For further details about key generation methods, please refer to Edge Lock Secure Subsystem (ELS) chapter.

Table 206. Key summary

Key name	Generation method	Parent key	ELS key store properties
NXP_DIE_MK_SK	KEYIN form PUF	-	0x60010161
NXP_DIE_MEM_ENC_SK	CKDF	NXP_DIE_MK_SK	0x88000320
NXP_DIE_EXT_MEM_ENC_SK	CKDF	NXP_DIE_MK_SK	0x88000320
NXP_DIE_PFR_AUTH_SK	CKDF	NXP_DIE_MK_SK	0x400020E1
NXP_DIE_FW_AUTH_SK	CKDF	NXP_DIE_MK_SK	0x400020A1
NXP_DIE_KEK_SK	CKDF	NXP_DIE_MK_SK	0x404000E1
NXP_DIE_MEM_IV_ENC_SK	CKDF	NXP_DIE_MK_SK	0x001000A0
NXP_DIE_DICE_UDS_MK_SK	CKDF	NXP_DIE_MK_SK	0x400200E1

Table continues on the next page...

Table 206. Key summary (continued)

Key name	Generation method	Parent key	ELS key store properties
CUST_DIE_DICE_CDI_MK_SK	HKDF	NXP_DIE_DICE_UDS_MK_SK	0x440200E1
CUST_DIE_DICE_CA_PRK_SEED	HKDF	CUST_DIE_DICE_CDI_MK_SK	0x040000E1
CUST_DIE_DICE_CA_PRK	KEYGEN	CUST_DIE_DICE_CA_PRK_SEED	0x100440E1
CUST_MK_SK	User defined	-	0x100100E1
CUST_FW_ENC_SK	CKDF	CUST_MK_SK	0x00010020
CUST_FW_SB3KBLK_ENC_SK(N)	CKDF	CUST_FW_ENC_SK	0x00100020

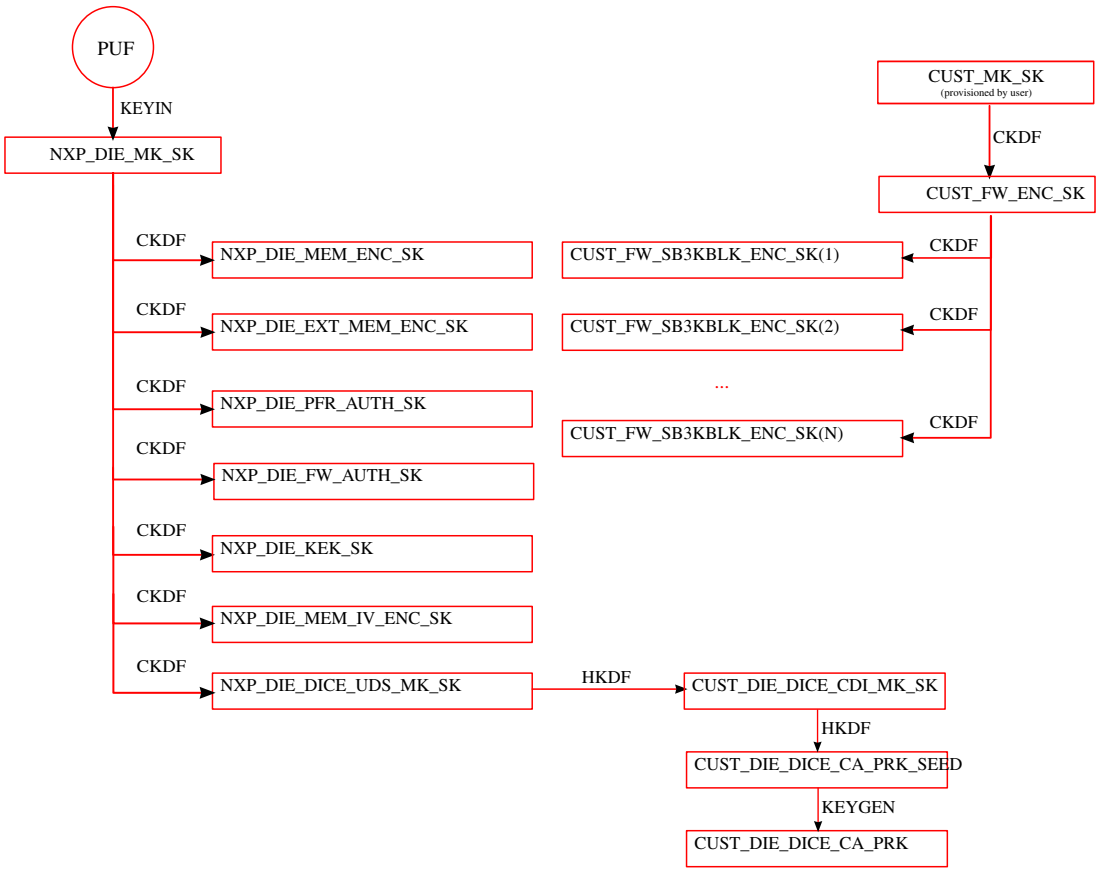


Figure 52. Key hierarchy

9.8.1 Die unique keys

Die unique keys are derived from NXP_DIE_MK_SK key supplied to ELS from PUF. ELS is designed such that the value of NXP_DIE_MK_SK as well as keys derived from this key remains unknown to any software or firmware running outside of ELS. This fact makes it possible to preserve the confidentiality of all other keys on the device either by keeping them inside of ELS.

9.8.1.1 NXP_DIE_MK_SK

256-bit key derived from NXP provisioned PUF Activation code using GET_KEY operation. Key is loaded via dedicated secret key bus from PUF into ELS using KEYIN command, where it is used as a main key for further derivation of all remaining keys used by ROM. This key cannot be unloaded from ELS

9.8.1.2 NXP_DIE_MEM_ENC_SK

128-bit key used as encryption key for internal flash bus encryption engine module (NPX). This key is key is loaded into NPX IP block from ELS via dedicated key bus.

9.8.1.3 NXP_DIE_EXT_MEM_ENC_SK

128-bit key used as encryption key for external flash bus encryption engine module (IPED). This key is key is loaded into IPED IP block from ELS via dedicated key bus.

9.8.1.4 NXP_DIE_PFR_AUTH_SK

256-bit CMAC key used during boot to verify authenticity of CFPA and CMPA IFR pages.

9.8.1.5 NXP_DIE_FW_AUTH_SK

256-bit CMAC key used to calculate and verify authenticity of boot image when CMAC based image authentication option is enabled.

9.8.1.6 NXP_DIE_KEK_SK

256-bit key used for wrapping of RFC3394 blobs stored in IFR Customer Key store Area.

9.8.1.7 NXP_DIE_MEM_IV_ENC_SK

128-bit key for AES ECB encryption of IVs for both NPX and IPED bus encryption engines. Final IV value for given region used by NPX/IPED for encryption is computed by ROM bootloader and incorporates device UUID, encryption region number and MCTR_NPX_CTXn or MCTR_IPED_CTXn as follows:

IV = AES_ECB_ENC(NXP_DIE_MEM_IV_ENC_SK, ivSeed[127:0])

where ivSeed is defined as:

For NPX:

ivSeed[127:0] = {UUID[95:0] ^ regionNumber[1:0], MCTR_NPX_CTXn[31:0]}

For IPED:

ivSeed[127:0] = {UUID[95:0] ^ regionNumber[1:0], MCTR_IPED_CTXn[31:0]}

9.8.1.8 CUST_DIE_DICE_CDI_MK_SK

256-bit CDI key created from UDS key and ELS runtime fingerprint.

9.8.1.9 CUST_DIE_DICE_CA_PRK_SEED

256-bit DICE CSR signing key.

9.8.1.10 CUST_DIE_DICE_CA_PRK

256-bit Key pair CUST_DIE_DICE_CA. Public key CUST_DIE_DICE_CA_PUK signed with NXP_CUST_DICE_CA_PRK.

9.8.1.11 CUST_FW_ENC_SK

256-bit SB3 key derivation key.

9.8.1.12 CUST_FW_SB3KBLK_ENC_SK(N)

256-bit SB3 block-n encryption key.

9.8.2 Pre-shared keys

All pre-shared keys that needs to be supplied externally into ELS are stored in Customer Key store Area located in CMPA starting from 0x01004160. These keys are stored in RFC3394 key blob format, which is wrapped using device specific NXP_DIE_KEK_SK key. For bootloader purpose, only CUST_MK_SK needs to be present in Customer Key store Area. The remaining space can be used to store one additional key blob, used by user application. ROM provides special SB3 command for wrapping and loading of external keys into CMPA key store area. Please refer to Secure trust provisioning for more details.

9.8.2.1 CUST_MK_SK key blob

CUST_MK_SK key is used by the bootloader to decrypt SB3.1 file. To be able to process SB3.1 files, this RFC3394 key blob key needs to be present in Customer Key store Area at address 0x01004160. Generation and loading process of this key takes place during device provisioning. Refer to Secure trust provisioning for more details.

9.9 ELS key store state after boot

Several keys that were generated during startup by the ROM bootloader are kept loaded in the ELS key store when the user application starts executing. This allows user and bus crypto engines to use these keys when needed. In case that any of the key is deleted from ELS key slot upon user request, the only way to restore it is to reset the device, since only ROM code is capable to generate these keys at startup again. Table 207 shows summary of the keys present in ELS key store after boot:

Table 207. Keys present in ELS key store after boot

Key name	ELS key slot number	Key definition
NXP_DIE_MK_SK	0 & 1	Die unique key
NXP_DIE_MEM_ENC_SK	2	Encryption key for internal flash bus encryption engine (NPX)
NXP_DIE_EXT_MEM_ENC_SK	3	Encryption key for external flash bus encryption engine (IPED)
NXP_DIE_MEM_IV_ENC_SK	4	Key for AES ECB encryption of IVs for both NPX and IPED bus encryption engines
CUST_DIE_DICE_CA_PRK	9 & 10	Private key from the CUST_DIE_DICE_CA key pair (Available only when DICE_UPD==1 in CFPA)

NOTE

256-bit keys always occupies two ELS key slots.

9.10 ELS key store usage from ROM API

ELS key store is also used by ROM bootloader during processing of SB file through ROM API. All keys generated during this process are automatically deleted when SB file loading is finalized. ELS key slots used by these keys must be always empty before

using ROM API to process the SB file. Otherwise, the SB loader ROM API will return an error. The following table shows summary of the keys generated during SB file processing.

Table 208. Keys used by ROM API during SB file processing

Key name	ELS key slot number	Key definition
NXP_DIE_KEK_SK	5 & 6	Key used for unwrapping of RFC3394 wrapped key blobs
CUST_MK_SK	18 & 19	Main key used for SB3 processing
CUST_FW_ENC_SK	16 & 17	SB3 key derivation key
CUST_FW_SB3KBLK_ENC_SK(n)	14 & 15	SB3 block-n encryption key

9.11 Secure trust provisioning

The purpose of trust provisioning process is to establish NXP genuine device proof in the chip during manufacturing process in NXP factory and to allow OEMs to securely provision/inject CUST_MK_SK, initial CFPA/CMPA configuration together with RKTH and corresponding application image to prepare device for its first use.

9.11.1 Trust provisioning keys

When working with keys during trust provisioning, all of the exported keys are stored in RFC3394 blobs so that the key confidentiality is retained. For this purpose ROM uses NXP_CUST_KEK_INT_SK and NXP_CUST_KEK_EXT_SK as a wrapping keys. These keys are internally derived from OEM_INPUT entropy seed value provided by OEM when OEM_GEN_MASTER_SHARE command is executed.

9.11.1.1 NXP_CUST_KEK_INT_SK

256-bit key derived from OEM_INPUT. Used to wrap on-chip generated keys using HSM_GEN_KEY ISP command into RFC3394 key blob when unloaded from ELS.

9.11.1.2 NXP_CUST_KEK_EXT_SK

256-bit key derived from OEM_INPUT. Used to wrap known symmetric keys supplied by user in plain format. HSM_STORE_KEY ISP command wraps the keys in RFC3394 key blob using this key when unloaded from ELS.

9.11.2 ISP trust provisioning commands

This section describes functionality and purpose of individual ISP provisioning commands. Trust provisioning ISP commands do not have the data phase and operate exclusively with memory pointers. ISP write-memory command needs to be used to preload input data into memory before executing of the command. Output data can be then read by read-memory ISP command. For detailed info related to command parameter syntax please refer to TrustProvisioning command

9.11.2.1 OEM_GEN_MASTER_SHARE

This command takes the entropy seed provided by the OEM as input to create shares for initial trust provisioning keys. All remaining ISP trust provisioning commands are dependent on this command.

- Input:
  - OEM_INPUT: 128-bit entropy seed value provided by the OEM.
- Output:
  - ENC_OEM_SHARE: This GCM encrypted blob is embedded in manufacturing firmware SB3 file in ISK field for the receiving device to reconstruct the OEM diversified root keys.
  - ENC_OEM_MASTER_SHARE: This GCM encrypted blob can be used in future to re-create FW files and should be kept secret. It is used as input to OEM_SET_MASTER_SHARE command.

— NXP_CUST_CA_PUK: Public portion of OEM TP public key to validate the device ID keys for "over production protection flow".

```
 Example:
blhost.exe -p COM4 -- write-memory 0x20001000 OEM_SHARE_INPUT.bin
blhost.exe -p COM4 -- trust-provisioning oem_gen_master_share 0x20001000 0x10 0x20002000 0x30
0x20003000 0x40 0x20004000 0x40
blhost.exe -p COM4 -- read-memory 0x20002000 0x30 encOemShare.bin
blhost.exe -p COM4 -- read-memory 0x20003000 0x40 encOemMasterShare.bin
blhost.exe -p COM4 -- read-memory 0x20004000 0x40 oemCertificate.bin
```

9.11.2.2 OEM_SET_MASTER_SHARE

Command used by OEM to put the device back in key creation mode with original keys produced when OEM_GET_MASTER_SHARE command was issued. Needs to be used only in case if user needs to re-establish previously interrupted trust provisioning session generated using OEM_GEN_MASTER_SHARE i.e. after reset or power loss.

- Inputs:
  - OEM_INPUT: 128-bit entropy seed value provided by the OEM.
  - ENC_OEM_MASTER_SHARE: Blob obtained from OEM_GEN_MASTER_SHARE command.
- Output:
  - None

```
Example:
blhost.exe -p COM4 -- write-memory 0x20001000 OEM_SHARE_INPUT.bin
blhost.exe -p COM4 -- write-memory 0x20002000 encOemMasterShare.bin
blhost.exe -p COM4 -- trust-provisioning oem_set_master_share 0x20001000 16 0x20002000 64
```

9.11.2.3 OEM_GET_CUST_CERT_DICE_PUK

Command used by OEM to provide it share to create the initial trust provisioning keys.

- Input:
  - OEM_RKTH: 256-bit SHA2 hash digest of the root keys table used by OEM for secure boot, FW update and debug authentication.
- Output:
  - NXP_CUST_DICE_CA_PUK: Public portion of OEM diversified ELS key used for signing device Identity keys generated using DICE methodology during CSR harvesting phase.

9.11.2.4 HSM_GEN_KEY

Command used for creating common OEM keys. Can create ECDSA P-256 key pairs usable as root keys, image signing key or debug authentication key. The key blobs generated by this command are used by OEM for future FW update file generation or debug certificate generations. Generated key blobs can also be used to deploy on all device through OEM provisioning/ manufacturing firmware by using SB_STORE_KEY SB command. This command is usable only after ISP_SET_OEM_SHARE or ISP_GEN_OEM_MASTER_SHARE commands are issued.

- Inputs:
  - Key_type:
    - MFW_ISK_FLAG: ECDSA manufacturing firmware (MFW) signing key (NXP_CUST_FW_AUTH_PRK).

- MFW_ENCK_FLAG: CKDF master key for manufacturing firmware (MFW) encryption key (NXP_CUST_FW_MK_SK). Will generate MFG_CUST_MK_SK0_BLOB usable with ISP_HSM_ENC_BLK command.
- GEN_SIGNK_FLAG: Generic ECDSA signing key
  - Used for signing production FW image keys, RoT keys or debug keys.
  - Use KEYGEN with DRBG seed input parameter to create these keys.
- GEN_CUST_MK_SK_FLAG: CKDF master key for production firmware encryption key (CUST_MK_SK). Random derivation data is used to derive this key. Will generate MFG_CUST_MK_SK0_BLOB usable with ISP_HSM_ENC_BLK command.

- Outputs:
  - KEY_BLOB: RFC3394 key blob generated by ELS KEYOUT command.
    - Uses NXP_CUST_KEK_INT_SK for this purpose to allow usage of key blobs on another EVK board. OEM share is also used for generation of this key.
  - ECDSA_PUK: Public portion of the key.

```
Example:
blhost.exe -p com4 -- trust-provisioning hsm_gen_key MFWISK 0 0x20001000 48 0x20003000 64
blhost.exe -p com4 -- read-memory 0x20001000 48 CUST_FW_AUTH_PRK.bin
blhost.exe -p com4 -- read-memory 0x20003000 64 CUST_FW_AUTH_PRK_PUK.bin
```

9.11.2.5 HSM_STORE_KEY

Command used for storing known symmetric keys to be deployed on all devices. For example CKDF master key used as root for FW encryption (CUST_MK_SK).

- Inputs:
  - Key_type:
    - CUST_CKDFK_FLAG: Customer master key. Example OEM production FW encryption key (CUST_MK_SK). Will generate MFG_CUST_MK_SK0_BLOB usable with ISP_HSM_ENC_BLK command.
    - CUST_HKDFK_FLAG: HKDF master key.
    - CUST_HMACK_FLAG: HMAC key.
    - CUST_CMACK_FLAG: CMAC key.
    - CUST_AESK_FLAG: AES key.
    - CUST_KUOK_FLAG: Key unwrap only key.
  - Key_prop - Key size and PPROT restrictions.
    - bit[0]: Key Size
      - 0 - 128 bit key
      - 1 - 256 bit key
    - bit[29-1]: Reserved, must be 0
    - bit[30]: PPROT0
      - 0 - ELS command must be sent in privileged/not privileged mode to make use of the key
      - 1 - ELS command must be sent in privileged mode to make use of the key
    - bit[31]: PPROT1
      - 0 - ELS command must be sent in secure/not secure mode to make use of the key

        - 1 - ELS command must be sent in secure mode to make use of the key
    - Key_value - Input key in plain text format
- Output:
    - KEY_BLOB: RFC3394 key blob generated by KEYOUT command (ELS). Uses NXP_CUST_KEK_EXT_SK for this purpose to allow usage of key blobs on another device. OEM share is also used in producing this key.

```
Example:
blhost.exe -p com4 -- write-memory 0x20004000 USER_KEY.bin
blhost.exe -p com4 -- trust-provisioning hsm_store_key CKDFK 0x1 0x20004000 32 0x20005000 48
blhost.exe -p com4 -- read-memory 0x20005000 0x30 CUST_MK_SK.bin
```

9.11.2.6 HSM_ENC_BLK

Command used to encrypt the given SB3 data block sliced by the PC tool. This command is only supported after issuance of ISP_GEN_OEM_SHARE or SET_OEM_MASTER_SHARE. The encryption is done in-place.

- Inputs:
    - MFG_CUST_MK_SK0_BLOB: Blob generated using HSM_KEY_GEN command with key type set to MFW_ENCK_FLAG or GEN_CUST_MK_SK_FLAG. This type of blob is wrapped using NXP_CUST_KEK_INT_SK. Blob generated using ISP_HSM_STORE_KEY command with key type set to CUST_CKDFK_FLAG. This type of blob is wrapped using NXP_CUST_KEK_EXT_SK.
    - Wrapping key ID: Flag indicating whether NXP_CUST_KEK_INT_SK (Key ID = 16) or NXP_CUST_KEK_EXT_SK (Key ID = 17) key is used for wrapping of MFG_CUST_MK_SK0_BLOB.
    - SB3_HEADER_DATA: SB3 header containing file size, Firmware version and timestamp data. Except for hash digest of block 0, all other fields should be valid.
    - Block_num: SB3 Block number. This is used to derive the encryption key for the given SB3 data block.
    - Input data: Input buffer to be encrypted. The encryption is done in-place.
    - Size: Size of the data buffer to be encrypted.
- Output:
    - Output data: Encrypted data. The encryption is done in-place, so plaintext data in input buffer will be replaced with encrypted data.

```
Example:
blhost.exe -p com4 -- write-memory 0x2000C000 SB3_HEADER_DATA.bin
blhost.exe -p com4 -- write-memory 0x2000A000 CUST_FW_ENC_SK.bin
blhost.exe -p com4 -- write-memory 0x2000D000 plain_block_0.bin
blhost.exe -p com4 -- trust-provisioning hsm_enc_blk 0x2000A000 48 16 0x2000C000 60 0 0x2000D000 256
blhost.exe -p com4 -- read-memory 0x2000D000 256 encrypted_block_0.bin
```

9.11.2.7 HSM_ECK_SIGN

Command used for signing the data buffer provided.

- Inputs:
    - Key_blob: ECDSA private key to be used for signing. The key blob is generated using HSM_KEY_GEN command.
    - Data buffer: Buffer to be signed.
- Output:

— Signature data: ECDSA signature of the data buffer.

```
Example:
blhost.exe -p com4 -- write-memory 0x20008000 CUST_FW_AUTH_PRK.bin
blhost.exe -p com4 -- write-memory 0x2000F000 SB3_HEADER.bin
blhost.exe -p com4 -- trust-provisioning hsm_enc_sign 0x20008000 48 0x2000F000 220 0x20010000 64
blhost.exe -p com4 -- read-memory 0x20010000 64 SIGNATURE.bin
```

9.11.3 Storing generated keys

Keys generated using trust provisioning commands can be stored in CMPA Area in IFR (0x01004160). First 48 bytes of key store area is reserved for storing of CUST_MK_SK key blob, which is used during decryption of the regular SB3.1 file when receive-sb-file command is executed. Remaining area is available to the user for storing of his own application defined keys. Storing of the keys into key store especially the CUST_MK_SK is the main purpose of the OEM provisioning SB file.

9.11.3.1 LOAD_KEY_BLOB

This SB command is used to store keys which are generated either by HSM_GEN_KEY or HSM_STORE_KEY ISP commands into the device key store area in CMPA. At first, the input key blob is loaded into ELS and unwrapped by NXP_CUST_KEK_INT_SK or NXP_CUST_KEK_EXT_SK key. Then the key is wrapped again using device specific NXP_DIE_KEK_SK and unloaded from ELS into CMPA key store area at given offset. This command is usable inside OEM provisioning/manufacturing firmware.

- Inputs:
  - Key blob: RFC3394 key blob generated either by HSM_GEN_KEY or HSM_STORE_KEY ISP commands.
  - Wrapping key ID: Flag indicating whether input blob is wrapped by NXP_CUST_KEK_INT_SK (HSM_GEN_KEY) or NXP_CUST_KEK_EXT_SK (HSM_STORE_KEY) key.
  - Offset: Absolute offset of CMPA key store area where the output key blob should be loaded.
- Output:
  - Key blob: Output blob is wrapped using die specific key (NXP_DIE_KEK_SK) and exported out from ELS using KEY_OUT command. This blob is automatically stored in CMPA key store area at user specified offset.

Please refer to elftosb Tool User's manual for details about the LOAD_KEY_BLOB command usage and syntax.

9.12 SB3.1 firmware update container

The Secure Binary (SB) container brings secure and easy way to upload or update firmware in embedded device during either the manufacturing process or end-customer's device lifecycle.

The SB container in version 3.1 (SB3.1) uses latest cryptographical algorithms to provide authenticity and confidentiality of carried firmware. The security level of SB3.1 is configurable and with the best available configuration you can full fill the Commercial National Security Algorithm Suite (CNSA) recommendation. By several available security configurations can be controlled the boot time and security level, which fits the best for the required use case. Authenticity of SB3.1 container is ensured by digital signature based on Elliptic Curve Cryptography (ECC) and confidentiality of SB3.1 container is ensured by use of Advance Encryption System (AES) in Cipher Block Chain (CBC) mode.

9.12.1 SB3.1 structure

SB3.1 is characterized as chain of blocks (figure SB3.1 structure), and each Block i contains hash digest of block i+1.

Besides the hash digest of Block 1, the block 0 also contains digital signature, which guarantee authenticity of hash digest of block 1 and thus the whole chain. By digital signature verification of Block 0 followed by gradual verification of all hashes in chain for all following blocks, authenticity of whole SB3.1 chain is verified. The last block in chain (Block N) contains only zeroes instead of hash digest value.

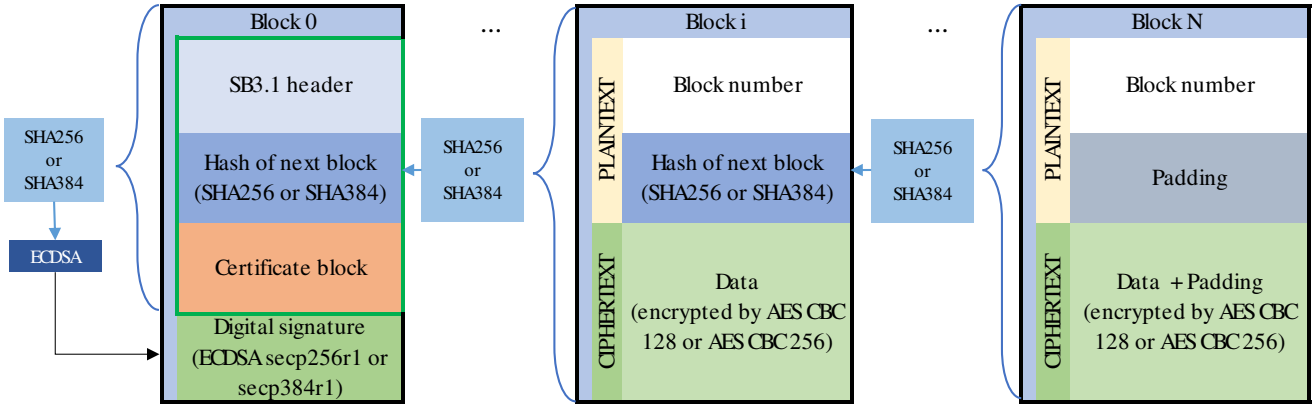


Figure 53. SB3.1 structure

9.12.2 SB3.1 Block 0

Block 0, also known as SB3.1 manifest, is different from the rest of SB3.1 blocks. Block 0 (see full content of Block 0 in Figure 55) does not contain any firmware data, instead, it holds the configuration and other data needed for authentication process. It is compatible with signed boot image format, where SB3.1 header and Hash of the next block acts as image data, so the same function can be used for authentication of signed boot image and SB3.1 Block 0. The size of Block 0 is various and depends on the selected level of security.

Hash algorithm used for hash of next block calculation depends on the type of elliptic curve used for digital signature of Block 0. If secp256r1 is selected, then SHA256 is used, if secp384r1 is selected, then SHA384 is used.

The digital signature of Block 0 is calculated over whole Block 0 (green framed data in Figure 54 and Figure 55) by use of SHA256 if secp256r1 is selected for ECDSA digital signature, or SHA384 if secp384r1 is selected for ECDSA digital signature.

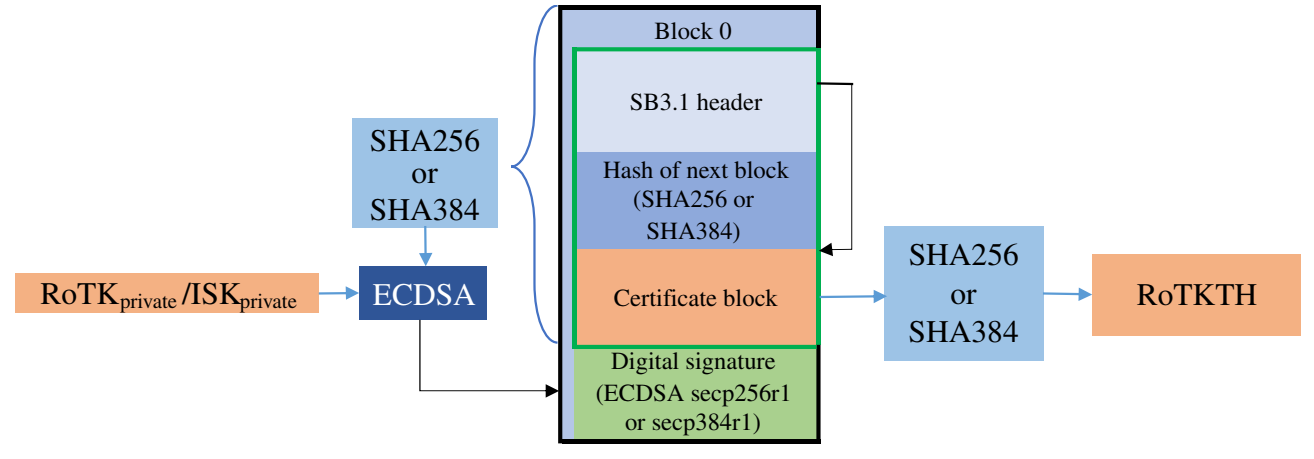


Figure 54. SB3.1 Block 0

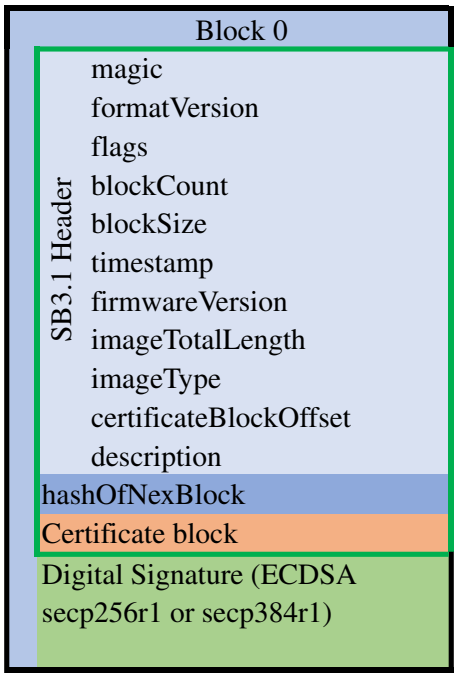


Figure 55. SB3.1 Block 0 full structure

- **SB3.1 Header** – object
  - **magic** – uint32_t, little endian

    Fixed 4-byte string "sbv3" without the trailing NULL, 0x33766273u.
  - **formatVersion** – uint32_t, little endian

    Fixed to "3.1", 0x00030001u, major = 3, minor = 1.
  - **flags** – uint32_t, little endian

    Not used, reserved for future use.
  - **blockCount** – uint32_t, little endian

    Total number of data blocks (not including the block 0).
  - **blockSize** – uint32_t, little endian

    Size in bytes of one data block. All data blocks have the same size.
  - **timestamp** – uint64_t, little endian

    Cryptographic nonce, e.g. number of seconds from 01/01/2000. Value used as one of the inputs into SB3.1 key derivation process.
  - **firmwareVersion** – uint32_t, little endian

    Version number of the included firmware.
  - **imageTotalLength** – uint32_t, little endian

    Total block 0 length in bytes, including block 0 signature.
  - **imageType** – uint32_t, little endian

    Image type, 0x06u for regular SB3.1, other values reserved.
  - **certificateBlockOffset** – uint32_t, little endian

Offset from start of block 0 to the certificate block. This allows the signed image verification code to verify the signature over the block 0.

— **description** – uint8_t[16]

Free text field for file description.

- **hashOfNexBlock** – uint8_t[32] or uint8_t[48]

  Hash is computed over whole Block 1 after encryption of data section. Size is 32 bytes (SHA256) or 48 bytes (SHA384). Hash algorithm is selected based on EC type used for signing of Block 0. Secp256r1 -> SHA256 or secp384r1 -> SHA384.

- **Certificate block** – uint8_t[variable size]

  Refer to Certificate block v2.1 for details.

- **Digital Signature** – uint8_t[64] or uint8_t[96]

  The ECDSA signature of hash (SHA256 or SHA384) over the header + hash of next block + certificate block data. Hash algorithm is based on EC type used for signature (secp256r1 or secp384r1). Secp256r1 -> SHA256 or secp384r1 -> SHA384. Signature coordinates (r,s) stored in big-endian.

9.12.3 SB3.1 Block i (Data block)

Block i, also known as data block, contains data payload separated into fixed size data chunks. The data chunk size is currently set to 256 bytes. Data blocks are numbered from 1 to N. The total number of data blocks (N) in SB3.1 is corresponding to the size of payload divided by data chunk size (see SB3.1 data chunk). When payload is not aligned to block size, in last block (Block N) padding is added in form of zeros to have data chunk aligned to data chunk size. Hash of next block for last block is filled with zeroes. See full structure details in Table 2 - Block i (data block).

Each data chunk is encrypted by AES CBC 128 or AES CBC 256 algorithm using unique key per block. Keys are derived from CUST_MK_SK key, which is external input to SB3.1 key derivation process.

SB3.1 key derivation process is following NIST SP 800-108 (Recommendation for Key Derivation Using Pseudorandom Functions) specification. Refer to SB3.1 key derivation process for more details.

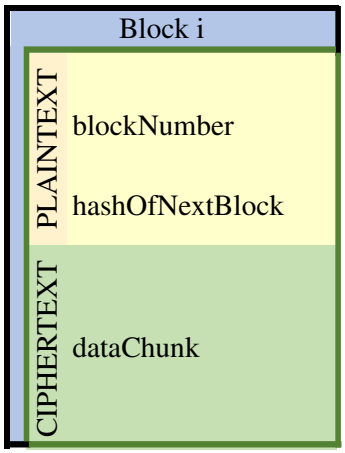


Figure 56. Block i

- **blockNumber** – uint32_t, little endian

  Number of current block, starting from 1 to N.

- **hashOfNextBlock** – uint8_t[32] or uint8_t[48]

  SHA256 or SHA384 digest of whole block i+1 (blockNumber || Next Block Hash[32]/[48] || dataChunk[blockSize]). Hash algorithm is selected based on EC type used for signing of Block 0. Secp256r1 -> SHA256 or secp384r1 -> SHA384.

- **dataChunk** – uint8_t[blockSize]

Payload data encrypted by aes_cbc128(CUST_FW_SB3KBLK_ENC_SK_128[i]) or aes_cbc256(CUST_FW_SB3KBLK_ENC_SK_256[i]). AES algorithm is selected based on EC type used for signing of Block 0. Secp256r1 -> aes_cbc128 or secp384r1 -> aes_cbc256.

9.12.4 SB3.1 key derivation process

SB3.1 key derivation process accords with NIST SP 800-108 (Recommendation for Key Derivation Using Pseudorandom Functions) specification. As pseudo random function (PRF) uses CMAC algorithm and key derivation function is running in counter mode, SB3.1 derivation function is named CMAC Key Derivation Function (CKDF).

SB3.1 Key derivation inputs 256-bit long symmetric key named Customer Master Key Symmetric Key (CUST_MK_SK). After that, derive Customer Firmware Encryption Symmetric Key (CUST_FW_ENC_SK) with CKDF function by use of CUST_MK_SK and timestamp as part of derivation data. CUST_FW_ENC_SK is then used as CKDF input for derivation of key for each data block named Customer Firmware Secure Binary 3 Key Block Encryption Symmetric Key CUST_FW_SB3KBLK_ENC_SK(N), where block number is used as part derivation data. Detailed structure of derivation data is described later in the chapter.

The size of derived CUST_FW_ENC_SK and CUST_FW_SB3KBLK_ENC_SK(N) is driven by EC type used for signing of Block 0. If secp256r1 is used, then CUST_FW_ENC_SK and CUST_FW_SB3KBLK_ENC_SK(N) have 128 bits. If secp384r1 is used, then CUST_FW_ENC_SK and CUST_FW_SB3KBLK_ENC_SK(N) have 256 bits.

Table 209. SB3.1 key derivation process

Key		Description
Key derivation key:	CUST_FW_ENC_SK = CKDF(CUST_MK_SK, timestamp)	Initial key derivation key is derived from CUST_MK_SK
Key encryption/ decryption keys:	CUST_FW_SB3KBLK_ENC_SK(0) = N/A	Block 0 is not encrypted.
	CUST_FW_SB3KBLK_ENC_SK(1) = CKDF(CUST_FW_ENC_SK, 0x1)	Encryption/decryption key for block 1.
	CUST_FW_SB3KBLK_ENC_SK(2) = CKDF(CUST_FW_ENC_SK, 0x2)	Encryption/decryption key for block 2.
	…	…
	CUST_FW_SB3KBLK_ENC_SK(N) = CKDF(CUST_FW_ENC_SK, N)	Encryption/decryption key for block N.

Pseudo code of used key derivation algorithm (CKDF):

For i = 1 to n, do

1. K(i) := PRF (KI, Label || Context || [L]2 || [i]2)
2. result(i) := result(i-1) || K(i).

Table 210. Key derivation data

Input	Data type (length)	CUST_FW_ENC_SK	CUST_FW_SB3KBLK_ENC_SK(N)
K_I	uint8_t[32]	Symmetric 256bit key -> CUST_MK_SK.	CUST_FW_ENC_SK
Label	uint8_t[12] (96bits)	Timestamp (uint64_t) value from SB3.1 Block 0 header in little endian with added zero padding to 96 bits. Example: "A170AD270000000000000000".	BlockNumber (uint32_t) value from SB3.1 Block i header in little endian with added zero padding to 96 bits. Example for block 14: "0E000000000000000000000".

Table continues on the next page...

Table 210. Key derivation data (continued)

Input	Data type (length)	CUST_FW_ENC_SK	CUST_FW_SB3KBLK_ENC_SK(N)
Context	uint8_t[12] (96bits)	If CUST_FW_ENC_SK is 128 bits, value is fixed to "0000000000000000c0010020".	If CUST_FW_SB3KBLK_ENC_SK(N) is 128 bits, it is fixed to "0000000000000000C0100020".
		If CUST_FW_ENC_SK is 256 bits, value is fixed to "0000000000000000C0010021".	If CUST_FW_SB3KBLK_ENC_SK(N) is 256 bits, it is fixed to "0000000000000000C0100021".
L	uint32_t - Big endian	Value is corresponding to derived key size. If CUST_FW_ENC_SK is 128 bits, it is fixed to "00000080".	Value is corresponding with derived key size. If CUST_FW_SB3KBLK_ENC_SK(N) is 128 bits, it is fixed to "00000080".
		If CUST_FW_ENC_SK is 256 bits, it is fixed to "00000100".	If CUST_FW_SB3KBLK_ENC_SK(N) is 256 bits, it is fixed to "00000100".
i	uint32_t - Big endian	Counter. If CUST_FW_ENC_SK is 128 bits, only one iteration is executed with value "00000001".	Counter. If CUST_FW_SB3KBLK_ENC_SK(N) is 128 bits, only one iteration is executed with value "00000001".
		If CUST_FW_ENC_SK is 256 bits, two iterations are done with value "00000001" followed by value "00000002".	If CUST_FW_SB3KBLK_ENC_SK(N) is 256 bits, two iterations are done with value "00000001" followed by value "00000002".

Key derivation example:

1. CUST_FW_ENC_SK 256 bit derivation
   - CUST_MK_SK: "24e517d4ac417737235b6efc9afced8224e517d4ac417737235b6efc9afced82"
   - Data1: "8b71ad270000000000000000000000000000000c00100210000010000000001"
   - Data2: "8b71ad270000000000000000000000000000000c00100210000010000000002"
   - Res1 = PRF(CUST_MK_SK, Data1) = "68fd9ef140290488eca5736aa9f4b4a5"
   - Res2 = PRF(CUST_MK_SK, Data2) = "cf437c8618809047ec1d46f70523481a"
2. CUST_FW_SB3KBLK_ENC_SK(3) 256 bit derivation
   - CUST_FW_ENC_SK: "68fd9ef140290488eca5736aa9f4b4a5cf437c8618809047ec1d46f70523481a"
   - Data1: "0300000000000000000000000000000000000000c01000210000010000000001"
   - Data2: "0300000000000000000000000000000000000000c01000210000010000000002"
   - Res1 = PRF(CUST_FW_ENC_SK, Data1) = "4b2afc98b4ca03fc0de090be76d3beb2"
   - Res2 = PRF(CUST_FW_ENC_SK, Data2) = "729fb4b3149b3ea05f414a2dd0a193ce"
   - CUST_FW_SB3KBLK_ENC_SK(3): "4b2afc98b4ca03fc0de090be76d3beb2729fb4b3149b3ea05f414a2dd0a193ce"

9.12.5 Certificate block v2.1

Certificate block is the most important part of SB3.1 and signed boot image. The Certificate block used in SB3.1 and corresponding signed boot image has version 2.1 which is only for elliptic curve cryptography (ECC). Certificate block requires to generate from 1 to 4 key pairs, based on secp256r1 or secp384r1, EC type can't be mixed (only secp256r1 or secp384r1). Hash digest of public key(s) is stored in Root of Trust Key Table (RoTKT), size of Root of Trust Key Hash (RoTKH) record depends on EC type. If secp256r1 is provided, then SHA256 is used, if secp384r1 is provided, then SHA384 is used.

The hash algorithm rule, which can be applied on all ECDSA signatures in current certificate block design, is that if secp256r1 curve is used for signing, then SHA256 digest of signed data should be used as input; if secp384r1 curve is used for signing, then SHA384 digest of signed data should be used as input.

Hash digest (SHA256 or SHA384 depending on root certificate EC type) of RoTKH records is stored in non-volatile memory of the device (Fuse, IFR, which locks the possibility of later change of Root certificate and creates Root of Trust in the device. The final hash of hashes is named Root of Trust Key Table Hash (RoTKTH). If only one root certificate is used, then RoTKTH is calculated as hash (SHA256 or SHA384 depending on root certificate EC type) directly from public key.

Next important input is Image Signing Key (ISK). If ISK is not used, then ISK certificate section is removed from Certificate block and one of the root certificates is used for signature of whole Block 0 or boot image. If ISK certificate is provided, then the ROTRecord and iskCertificate are signed by one of the root keys, and then the ISK private key is used for signing block 0 (for an SB file) or the boot image. ISK EC type can have only same or lower size in compare to root certificate. This means if RoTKs are based on secp256r1, then ISK can be only secp256r1, if root certificates are based on secp384r1, then secp256r1 or sec384r1 can be used for ISK. The private part of ISK key pair needs to be provided as external input for ECDSA signature of boot image or Block 0.

The signature of ISK certificate is done by hash calculation over RoTRecord and iskCertificate (green framed data in figure Certificate block). The hash algorithm is selected based on EC type of RoTK, if secp256r1 is provided, then SHA256 is used, if secp384r1 is provided, then SHA384 is used. The private part of selected root key pair needs to be provided as external input for ECDSA signature.

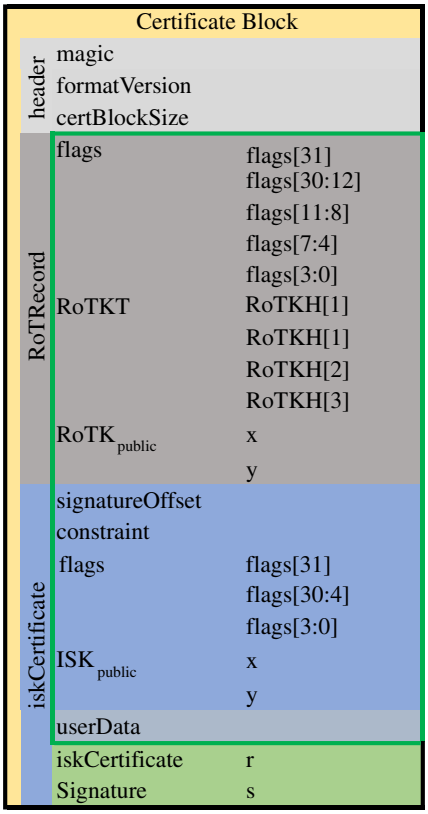


Figure 57. Certificate Block

- **SB3.1 Header** – object
  - **magic** – uint32_t, little endian

    Fixed 4-byte string "chdr" without trailing NULL, 0x72646863.
  - **formatVersion** – uint32_t, little endian

    Fixed to "2.1", 0x00020001, major = 2, minor = 1.

- **certBlockSize** – uint32_t, little endian

  Total size of Certificate block in bytes.

- **RoTRecord** – object
  - **flags** – uint32_t, little endian
    - **flags[bit 31]**: NoCA flag, if set to 0, used RoTK acts as Certificate Authority and is used to sign ISK certificate, does not sign the full image. If set to 1, used RoTK does not act as Certificate Authority and signs directly the full image or SB3 Block0. If the NoCa flag is set to 1, then the iskCertificate section is not present in the certificate block.
    - **flags[bits 30:12]**: Reserved for future use.
    - **flags[bits 7:4]**: Used root cert number [0-3] (specify root cert used to ISK/image signature).
    - **flags[bits 3:0]**: Type of root certificate, secp256r1 = 0x1u or secp384r1 = 0x2u, other values are reserved.
  - **RoTKT** – object

    Root of Trust keys Table, optional object (in case when only one RoTK is specified for device, RoTKT is not present, 1 to 4 RoTK can be specified for device), so if RoTKT is present in Certificate Block, then RoTKT table contains 2 to 4 RoTKH records.
    - **RoTKH[0-3]** – uint8_t[32] or uint8_t[48]

      Root of Trust Key Hash is SHA256 or SHA384 of RoTKpublic. Hash algorithm is selected based on RoTK EC type (secp256r1 -> SHA256 or secp384r1 -> SHA384). Same RoTKs and RoTKTH values are shared between debug authentication, SB3.1 firmware updates container and signed boot image.
  - **$RoTK_{public}$** – uint8_t[64] or uint8_t[96]

    X and Y coordinates of public key of selected RoTK, which will be used for signing and verification of ISK certificate or SB3.1 Block 0, field size depends on EC type (secp256r1 or secp384r1).

- **iskCertificate** – object

  Optional object, If ISK is not used (driven by NoCA flag), the whole iskCertificate section is missing.
  - **signatureOffset** – uint32_t, little endian

    Offset in bytes to ISK certificate signature from the beginning of iskCertificate object.
  - **constraint** – uint32_t, little endian

    Constraint = certificate version, compared with monotonic counter in fuses.
  - **flags** – uint32_t, little endian
    - **flags[bit 31]**: User data flag, if set to 1, user data are included in ISK certificate.
    - **flags[bits 30:4]**: Reserved for future use.
    - **flags[3:0]**: Type of ISK certificate, secp256r1 = 0x1u or secp384r1 = 0x2u, other values are reserved.
  - **ISK_{public}** – uint8_t[64] or uint8_t[96]

    ISK public key. If root certificate is secp256r1, ISK can be also only secp256r1, if root is secp384r1, then ISK can be secp256r1 or secp384r1. Public key stored in big-endian
  - **userData** – uint8_t[0-96]

    Optional, variable size. Can contain user data, e.g. EC public key, which is signed by ISK signature. Maximal size limited to 96 bytes (EC384 key pair). Data should be aligned to 4 bytes.
  - **iskCertificate Signature** – uint8_t[64] or uint8_t[96]

    ECDSA signature of SHA256 or SHA384 of green framed data on certificate block figure based on EC type of RoTK (secp256r1 or secp384r1). Secp256r1 -> SHA256 or secp384r1 -> SHA384. Signature stored in big-endian.

9.12.6 SB3.1 data chunk

The data area of all blocks following the header block is a contiguous sequence of bytes, called the payload, which is divided into equal-sized chunks. Each chunk is placed into its own block.

For a given block size, the data chunk size is:

chunkSize = blockSize – blockOverhead

blockSize = 256 bytes.

blockOverhead = 4bytes for block number (uint32_t) + SHA256 or SHA384 digest of next block (32 or 48 bytes)

Data stream can be divided into multiple sections. Each section is started by section header:

```
struct section_header {
   uint32_t sectionUid; //0x1u
   uint32_t sectionType; //0x1u
   uint32_t length;
   uint32_t _pad; //zeores
};
```

The length field of the section header can be used to skip over the section when searching for a given section. In current implementation is supported only one section header with section UID fixed to 1 and only data range section supported as section type.

Data range section (section type = 1) contains one or more ranges of data to be loaded and the target addresses. Only one data range section is supported on this device.

Data range section consists of one or more data range commands, where each command starting with a header:

```
struct range_header {
        uint32_t tag;
     uint32_t startAddress;
     uint32_t length;
     uint32_t cmd;
};
```

The startAddress and length specify the target memory address and data range to be written to the memory address. Tag carries a magic number 0x55aaaa55 as an identifier of data range header. Cmd field is an enum of various actions (commands) to be performed with the data range. Some commands contain also command specific extended header of 16 bytes added right after range header. More details are available in description of each command. Each data range command starts by range header, if it is not specified differently in detailed command description below.

List of supported data range commands:

```
enum CommandType {
       kSB3_COMMAND_erase = 0x1u,
    kSB3_COMMAND_load = 0x2u,
    kSB3_COMMAND_execute = 0x3u,
    kSB3_COMMAND_programFuses = 0x5u,
    kSB3_COMMAND_programIfr = 0x6u,
    kSB3_COMMAND_copy = 0x8u,
    kSB3_COMMAND_loadKeyBlob = 0xAu,
    kSB3_COMMAND_configureMemory = 0xBu,
    kSB3_COMMAND_fillMemory = 0xCu,
    kSB3_COMMAND_fwVersionCheck = 0xDu,
};
```

- **1–Erase**

Performs a flash erase of the given address range. The erase will be rounded up to the sector size.

```
struct range_header {
      uint32_t tag; // 0x55aaaa55
      uint32_t startAddress;
      uint32_t length;
      uint32_t cmd; //0x1u
};
Followed by:
struct range_header_memory_data {
      uint32_t memoryId;
      uint32_t _ pad0; //zeros
      uint32_t _ pad1; //zeros
      uint32_t _pad2; //zeros
};
```

- 2–Load

  If set, then the data to write immediately follows the range header memory data. Padding must be appended after the data such that the start of the next range or section header is 16-byte aligned. The length field contains the actual data length, not the aligned length.

```
struct range_header {
     uint32_t tag; // 0x55aaaa55
     uint32_t startAddress;
     uint32_t length;
     uint32_t cmd;//0x2u
};
Followed by:
struct range_header_memory_data {
     uint32_t memoryId;
     uint32_t _ pad0; //zeros
     uint32_t _ pad1; //zeros
     uint32_t _pad2; //zeros
};
```

- 3–Execute

  Perform authentication and jump to the code immediately after receive-sb operation is complete. So, test code can be loaded and executed out of RAM or flash. Command requires startAddress where signed/plain image is loaded.

- 5–programFuses

  The startAddress will be the address of fuse register, length will be number of fuse words to program. The data to write to the fuse registers will immediately follow the header.

- 6–programIFR

  The startAddress will be the address into the IFR region, length will be in number of bytes to write to IFR region. The data to write to IFR region at the given address will immediately follow the header.

- 8–copy

  Used for copying data from one place to another. Command size is fixed to 32 bytes. Structure of command is described below.

```
struct range_header {
    uint32_t tag; // 0x55aaaa55
    uint32_t startAddress; // address to copy from
    uint32_t length; // number of bytes to copy
    uint32_t cmd; // 0x8
    };
```

```
Followed by:
struct copy_command_data {
    uint32_t destinationAddress; //target address of copy
    uint32_t memoryIdFrom;
    uint32_t memoryIdTo;
    uint32_t _pad0; //zeros
};
```

- **10–loadKeyBlob**

  Wrapped key blob immediately follows the load key blob range header. The length field contains the actual data length of keyblob. Expected keyblob size is 32 or 48 bytes. More details about keyblob are available in LOAD_KEY_BLOB. Structure of command is described below.

```
struct load_keyblob_range_header {
    const uint32_t tag; // 0x55aaaa55
    uint16_t offset; // offset in IFR keystore
    uint16_t keyWrapId; //16 = NXP_CUST_KEK_INT_SK, 17 = NXP_CUST_KEK_EXT_SK
    uint32_t length;
    uint32_t cmd; // 0xA
};
```

- **11–configureMemory**

  Command used to automatic configure of specified memory. Command size is fixed to 16 bytes. Structure of command is described below.

```
struct config_memory_header {
    const uint32_t tag; // 0x55aaaa55
    uint32_t memoryId;
    uint32_t address;
    uint32_t cmd; // 0xB
};
```

- **12–fillMemory**

  The startAddress specifies the address of memory region to be filled with pattern and size as parameters of a command.

```
struct range_header {
  uint32_t tag; // 0x55aaaa55
  uint32_t startAddress;
  uint32_t length; // number of repeats of pattern
  uint32_t cmd; //0xCu
};
Followed by:
struct fill_command_data {
  uint32_t pattern;
  uint32_t memoryId;
  uint32_t _pad0; //zeros
  uint32_t _pad1; //zeros
};
```

- **13–fwVersionCheck**

  Check whether the FW version value specified in command for specific counterId is acceptable. FW version value in command must be greater than that programmed in IFR to be acceptable else rollback will be detected.

```
struct fw_version_check_range_header {
const uint32_t tag = NBOOT_RANGE_SECTION_TAG;
      uint32_t value; //value to compare with counter
```

```
      uint32_t counterId;
      uint32_t cmd; //0xDu
};
enum counterId {
             kNBOOT_CNT_nonsecure = 0x1u,
             kNBOOT_CNT_secure = 0x2u,
  };
```

It is an error to have a cmd field with a value of 0.

Example of payload

- section_header
  - sectionUid = 0x1
  - sectionType = 0x1 (Data range section)
  - length = n
- range_header
  - tag = 0x55aaaa55
  - startAddress = 0x0
  - length = 8192
  - cmd = 0x1 (erase)
- range_header_memory_data
  - memory_id = 0x0
  - _pad0 = 0x0
  - _pad1 = 0x0
  - _pad2 = 0x0
- range_header
  - tag = 0x55aaaa55
  - startAddress = 0x0
  - length = 8192
  - cmd = 0x2 (load
- range_header_memory_data
  - memory_id = 0x0
  - _pad0 = 0x0
  - _pad1 = 0x0
  - _pad2 = 0x0
- (8192 bytes of data)
- range_header
  - tag = 0x55aaaa55
  - startAddress = 0x0000_2000
  - length = 16
  - cmd = 0x2 (load
- range_header_memory_data

    - memory_id = 0x0
    - _pad0 = 0x0
    - _pad1 = 0x0
    - _pad2 = 0x0
- (16 bytes of data)
- range_header
    - tag = 0x55aaaa55
    - startAddress = 0x0
    - length = 0
    - cmd = 0x3 (execute)
- range_header
    - tag = 0x55aaaa55
    - startAddress = 0x10
    - length = 16 (16 fuse registers of 32-bit each)
    - Cmd = 0x5 (programFuse)
- (16 * 4 bytes of data)
- range_header
    - Tag = 0x55aaaa55
    - startAddress = 0x3E400
    - length = 512 bytes
    - Cmd = 0x6 (programIFR)
- (512 bytes of data)
- range_header
    - Tag = 0x55aaaa55
    - startAddress = 0x25
    - length = 0xFF words
    - Cmd = 0xC (fillMemory)
- fill_command_data
    - pattern = 0x89ABCDEF
    - memoryId = 0x0
    - _pad0 = 0x0
    - _pad1 = 0x0
- fw_version_check_range_header
    - tag = 0x55aaaa55
    - value = 0x3 (FW version value to be checked)
    - counterId = 0x1 (nonsecure), 0x2 (secure)
    - cmd = 0xD (checkFwVersion)

Above example shows different kinds of range headers within a section.

9.12.7 SB3.1 processing

The following figure shows SB3.1 processing flow.

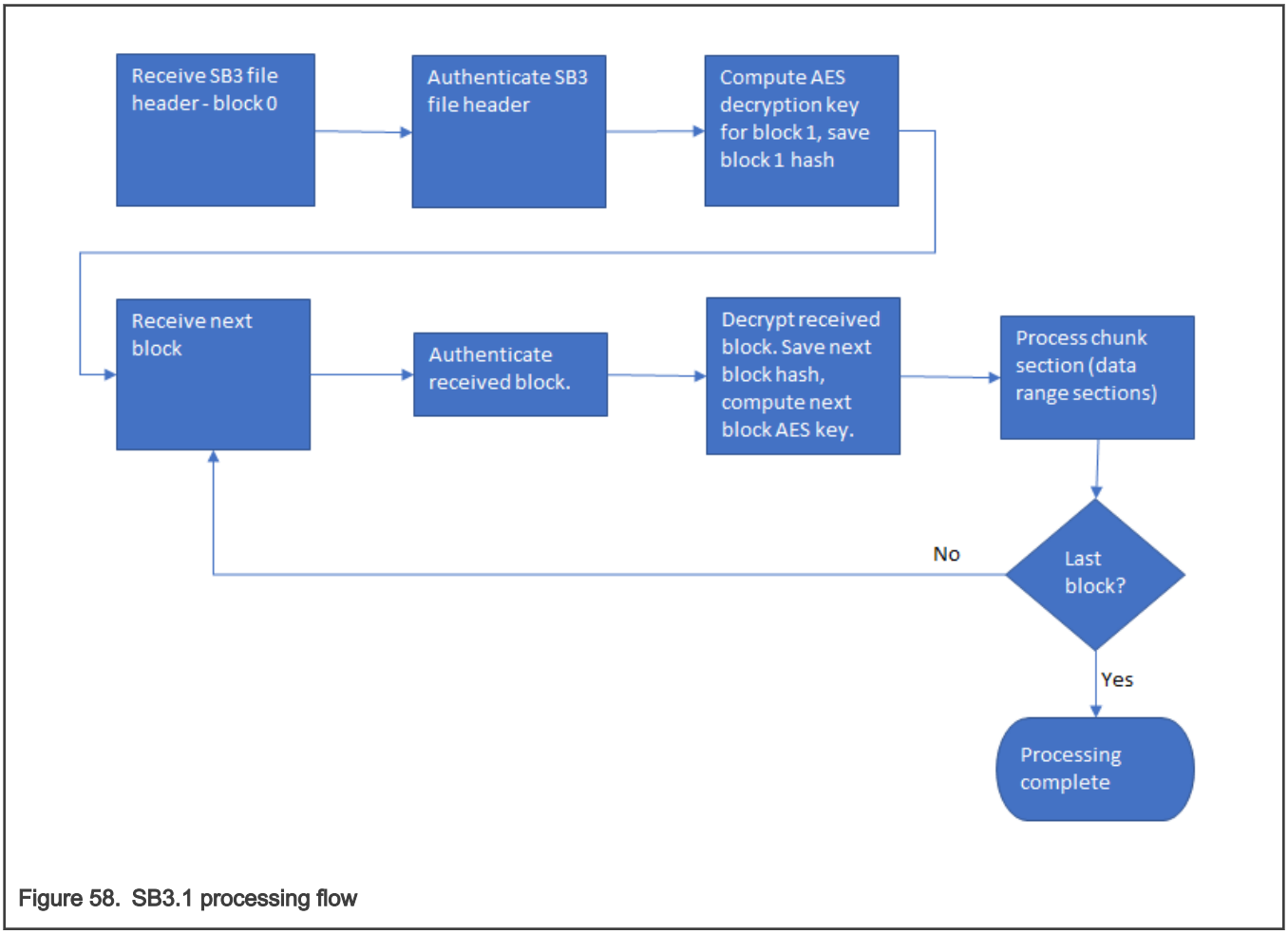


Figure 58. SB3.1 processing flow

Refer to SB3.1 Block 0 for more details about SB3.1 Block0 (header) authentication step.

9.13 On-chip Security Sensors

The ROM uses various on-chip sensors to protect secure boot flow against misuse of the device or disclosure of critical assets like cryptographic keys or personal data. All sensors are connected into Intrusion and Tamper Response Controller (ITRC). The ROM configures ITRC during device initialization and enables following ITRC outputs:

- OUT3 - RAM ZEROIZE
- OUT4 - chip RESET

The RAM_ZEROIZE signal is checked by ROM during the device RESET. If the signal is set, ROM zeroizes PKC and RAMA based on user settings ITRC_ZEROIZE in CMPA. The ITRC configuration is not locked and can changed later by user application code.

The on-chip sensors, which are utilized during boot process can be seen in Table 211 .

Table 211. Enabled on-chip sensors during boot process

ITRC input	Sensor Description
0	Glitch detector interrupts (GDET0 and GDET1)

Table continues on the next page...

Table 211. Enabled on-chip sensors during boot process (continued)

ITRC input	Sensor Description
2	CDOG0
4	SPC VDD_CORE low voltage detect (SPC_VD_STAT[COREVDD_LVDF])
7	Secure violation interrupt (Memory Block Checker interrupt or secure AHB matrix violation)
9	SPC VDD_CORE glitch detect (SPC_VDD_CORE_GLITCH_DETECT_SC[GLITCH_DETECT_FLAG])
10	Error flags from PKC module (PKC_ACCESS_ERR register)
11	CDOG1
16	SPC VDD_SYS low voltage detect (SPC_VD_STAT[SYSVDD_LVDF])
17	SPC VDD_IO low voltage detect (SPC_VD_STAT[IOVDD_LVDF])
33	GDET0 and GDET1 SFR error that occurs when invalid address has been accessed in GDET0 and GDET1 block.
34	SPC VDD_CORE high voltage detect (SPC_VD_STAT[COREVDD_HVDF])
35	SPC VDD_SYS_HVD high voltage detect (SPC_VD_STAT[SYSVDD_HVDF])
36	SPC VDD_IO high voltage detect (SPC_VD_STAT[IOVDD_HVDF])

9.14 ROM TrustZone Support

9.14.1 TrustZone Preset Data

ROM provides support for TrustZone data configuration during boot process. The TrustZone preset data includes:

- VTOR, VTOR_NS, NVIC_ITNS0, NVIC_ITNS1 (CPU0) registers
- Secure MPU
- Non-secure MPU
- SAU
- Secure AHB Controller
- MBC
- ITRC

If the TrustZone preset is enabled, the ROM, after image validation, configures all TrustZone related registers by data provided at the end of the image. If corresponding lock bits are set, the registers are also locked, so any further registers modification is not possible until next reset.

This feature increases robustness of the user application since the user application jumps into pre-configured TrustZone environment and it doesn't need to contain any TrustZone configuration code.

9.14.1.1 Master Boot Image with TrustZone Preset Data

The information whether image file contains TrustZone configuration data or not is defined in the vector section of the image header at offset 0x24, bit 13 (TZM_PRESET). The position of preset data block in image can be seen in Table 203.

Table 212. TrustZone Preset Data

Bitfield	Name	Bitfield value
Bit 13	TZ-M Preset	0: Trustzone preset data not present. 1: Trustzone preset data present.

9.14.1.2 TrustZone Preset Data Structure

The TrustZone preset data structure is defined by following C structure:

```
typedef struct _tzm_secure_config
{
    uint32_t tzm_magic;               /*!< It contains four letters "TZ-M" to identify start of block
*/

    uint32_t cm33_vtor_addr;          /*!< CM33 Secure vector table address */
    uint32_t cm33_vtor_ns_addr;       /*!< CM33 Non-secure vector table address */
    uint32_t cm33_nvic_itns0;         /*!< CM33 Interrupt target non-secure register 0 */
    uint32_t cm33_nvic_itns1;         /*!< CM33 Interrupt target non-secure register 1 */
    uint32_t cm33_nvic_itns2;         /*!< CM33 Interrupt target non-secure register 2 */
    uint32_t cm33_nvic_itns3;         /*!< CM33 Interrupt target non-secure register 3 */
    uint32_t cm33_nvic_itns4;         /*!< CM33 Interrupt target non-secure register 4 */
    uint32_t cm33_misc_ctrl;          /*!< Miscellaneous CM33 settings:
                                        AIRCR.SYSRESETREQS
                                        AIRCR.BFHFNMINS
                                        AIRCR.PRIS
                                        SCR.SLEEPDEEPS
                                        SHCSR.SECUREFAULTENA */
    uint32_t cm33_nsacr;              /*!< CM33 Non-secure Access Control Register */
    uint32_t cm33_cppwr;              /*!< CM33 Coprocessor Power Control Register */
    uint32_t cm33_cpacr;              /*!< CM33 Coprocessor Access Control Register */

    uint32_t cm33_mpu_ctrl;           /*!< MPU Control Register.*/
    uint32_t cm33_mpu_mair0;          /*!< MPU Memory Attribute Indirection Register 0 */
    uint32_t cm33_mpu_mair1;          /*!< MPU Memory Attribute Indirection Register 1 */
    uint32_t cm33_mpu_rbar0;          /*!< MPU Region 0 Base Address Register */
    uint32_t cm33_mpu_rlar0;          /*!< MPU Region 0 Limit Address Register */
    uint32_t cm33_mpu_rbar1;          /*!< MPU Region 1 Base Address Register */
    uint32_t cm33_mpu_rlar1;          /*!< MPU Region 1 Limit Address Register */
    uint32_t cm33_mpu_rbar2;          /*!< MPU Region 2 Base Address Register */
    uint32_t cm33_mpu_rlar2;          /*!< MPU Region 2 Limit Address Register */
    uint32_t cm33_mpu_rbar3;          /*!< MPU Region 3 Base Address Register */
    uint32_t cm33_mpu_rlar3;          /*!< MPU Region 3 Limit Address Register */
    uint32_t cm33_mpu_rbar4;          /*!< MPU Region 4 Base Address Register */
    uint32_t cm33_mpu_rlar4;          /*!< MPU Region 4 Limit Address Register */
    uint32_t cm33_mpu_rbar5;          /*!< MPU Region 5 Base Address Register */
    uint32_t cm33_mpu_rlar5;          /*!< MPU Region 5 Limit Address Register */
    uint32_t cm33_mpu_rbar6;          /*!< MPU Region 6 Base Address Register */
    uint32_t cm33_mpu_rlar6;          /*!< MPU Region 6 Limit Address Register */
    uint32_t cm33_mpu_rbar7;          /*!< MPU Region 7 Base Address Register */
    uint32_t cm33_mpu_rlar7;          /*!< MPU Region 7 Limit Address Register */

    uint32_t cm33_mpu_ctrl_ns;        /*!< Non-secure MPU Control Register.*/
    uint32_t cm33_mpu_mair0_ns;       /*!< Non-secure MPU Memory Attribute Indirection Register 0 */
    uint32_t cm33_mpu_mair1_ns;       /*!< Non-secure MPU Memory Attribute Indirection Register 1 */
    uint32_t cm33_mpu_rbar0_ns;       /*!< Non-secure MPU Region 0 Base Address Register */
```

```
    uint32_t cm33_mpu_rlar0_ns;       /*!< Non-secure MPU Region 0 Limit Address Register */
    uint32_t cm33_mpu_rbar1_ns;       /*!< Non-secure MPU Region 1 Base Address Register */
    uint32_t cm33_mpu_rlar1_ns;       /*!< Non-secure MPU Region 1 Limit Address Register */
    uint32_t cm33_mpu_rbar2_ns;       /*!< Non-secure MPU Region 2 Base Address Register */
    uint32_t cm33_mpu_rlar2_ns;       /*!< Non-secure MPU Region 2 Limit Address Register */
    uint32_t cm33_mpu_rbar3_ns;       /*!< Non-secure MPU Region 3 Base Address Register */
    uint32_t cm33_mpu_rlar3_ns;       /*!< Non-secure MPU Region 3 Limit Address Register */
    uint32_t cm33_mpu_rbar4_ns;       /*!< Non-secure MPU Region 4 Base Address Register */
    uint32_t cm33_mpu_rlar4_ns;       /*!< Non-secure MPU Region 4 Limit Address Register */
    uint32_t cm33_mpu_rbar5_ns;       /*!< Non-secure MPU Region 5 Base Address Register */
    uint32_t cm33_mpu_rlar5_ns;       /*!< Non-secure MPU Region 5 Limit Address Register */
    uint32_t cm33_mpu_rbar6_ns;       /*!< Non-secure MPU Region 6 Base Address Register */
    uint32_t cm33_mpu_rlar6_ns;       /*!< Non-secure MPU Region 6 Limit Address Register */
    uint32_t cm33_mpu_rbar7_ns;       /*!< Non-secure MPU Region 7 Base Address Register */
    uint32_t cm33_mpu_rlar7_ns;       /*!< Non-secure MPU Region 7 Limit Address Register */

    uint32_t cm33_sau_ctrl;           /*!< SAU Control Register.*/
    uint32_t cm33_sau_rbar0;          /*!< SAU Region 0 Base Address Register */
    uint32_t cm33_sau_rlar0;          /*!< SAU Region 0 Limit Address Register */
    uint32_t cm33_sau_rbar1;          /*!< SAU Region 1 Base Address Register */
    uint32_t cm33_sau_rlar1;          /*!< SAU Region 1 Limit Address Register */
    uint32_t cm33_sau_rbar2;          /*!< SAU Region 2 Base Address Register */
    uint32_t cm33_sau_rlar2;          /*!< SAU Region 2 Limit Address Register */
    uint32_t cm33_sau_rbar3;          /*!< SAU Region 3 Base Address Register */
    uint32_t cm33_sau_rlar3;          /*!< SAU Region 3 Limit Address Register */
    uint32_t cm33_sau_rbar4;          /*!< SAU Region 4 Base Address Register */
    uint32_t cm33_sau_rlar4;          /*!< SAU Region 4 Limit Address Register */
    uint32_t cm33_sau_rbar5;          /*!< SAU Region 5 Base Address Register */
    uint32_t cm33_sau_rlar5;          /*!< SAU Region 5 Limit Address Register */
    uint32_t cm33_sau_rbar6;          /*!< SAU Region 6 Base Address Register */
    uint32_t cm33_sau_rlar6;          /*!< SAU Region 6 Limit Address Register */
    uint32_t cm33_sau_rbar7;          /*!< SAU Region 7 Base Address Register */
    uint32_t cm33_sau_rlar7;          /*!< SAU Region 7 Limit Address Register */

    uint32_t flash00_mem_rule0;         /*!< FLASH 00 Memory Rule Register 0 */
    uint32_t flash00_mem_rule1;         /*!< FLASH 00 Memory Rule Register 1 */
    uint32_t flash00_mem_rule2;         /*!< FLASH 00 Memory Rule Register 2 */
    uint32_t flash00_mem_rule3;         /*!< FLASH 00 Memory Rule Register 3 */

    uint32_t flash01_mem_rule0;         /*!< FLASH 01 Memory Rule Register 0 */
    uint32_t flash01_mem_rule1;         /*!< FLASH 01 Memory Rule Register 1 */
    uint32_t flash01_mem_rule2;         /*!< FLASH 01 Memory Rule Register 2 */
    uint32_t flash01_mem_rule3;         /*!< FLASH 01 Memory Rule Register 3 */

    uint32_t flash02_mem_rule;          /*!< FLASH 02 Memory Rule Register */

    uint32_t flash03_mem_rule;          /*!< FLASH 03 Memory Rule Register */

    uint32_t rom_mem_rule0;           /*!< ROM Memory Rule Register 0 */
    uint32_t rom_mem_rule1;           /*!< ROM Memory Rule Register 1 */
    uint32_t rom_mem_rule2;           /*!< ROM Memory Rule Register 2 */
    uint32_t rom_mem_rule3;           /*!< ROM Memory Rule Register 3 */

    uint32_t ramx_mem_rule0;           /*!< RAMX Memory Rule Register 0 */
    uint32_t ramx_mem_rule1;           /*!< RAMX Memory Rule Register 1 */
    uint32_t ramx_mem_rule2;           /*!< RAMX Memory Rule Register 2 */
    uint32_t ramx_mem_rule3;           /*!< RAMX Memory Rule Register 3 */

    uint32_t rama_mem_rule;           /*!< RAMA Memory Rule Register */
```

```
    uint32_t ramb_mem_rule;           /*!< RAMB Memory Rule Register */

    uint32_t ramc_mem_rule0;           /*!< RAMC Memory Rule Register 0 */
    uint32_t ramc_mem_rule1;           /*!< RAMC Memory Rule Register 1 */

    uint32_t ramd_mem_rule0;           /*!< RAMD Memory Rule Register 0 */
    uint32_t ramd_mem_rule1;           /*!< RAMD Memory Rule Register 1 */

    uint32_t rame_mem_rule0;           /*!< RAME Memory Rule Register 0 */
    uint32_t rame_mem_rule1;           /*!< RAME Memory Rule Register 1 */

    uint32_t ramf_mem_rule0;           /*!< RAMF Memory Rule Register 0 */
    uint32_t ramf_mem_rule1;           /*!< RAMF Memory Rule Register 1 */

    uint32_t ramg_mem_rule0;           /*!< RAMG Memory Rule Register 0 */
    uint32_t ramg_mem_rule1;           /*!< RAMG Memory Rule Register 1 */

    uint32_t ramh_mem_rule0;           /*!< RAMH Memory Rule Register 0 */

    uint32_t apb_bridge0_mem_rule0;   /*!< APB Bridge 0 Memory Rule Register 0 */
    uint32_t apb_bridge0_mem_rule1;   /*!< APB Bridge 0 Memory Rule Register 1 */
    uint32_t apb_bridge0_mem_rule2;   /*!< APB Bridge 0 Memory Rule Register 2 */
    uint32_t apb_bridge0_mem_rule3;   /*!< APB Bridge 0 Memory Rule Register 3 */

    uint32_t apb_bridge1_mem_rule0;   /*!< APB Bridge 1 Memory Rule Register 0 */
    uint32_t apb_bridge1_mem_rule1;   /*!< APB Bridge 1 Memory Rule Register 1 */
    uint32_t apb_bridge1_mem_rule2;   /*!< APB Bridge 1 Memory Rule Register 2 */

    uint32_t aips_bridge0_mem_rule0;  /*!< AIPS BRIDGE 0 Peripherals  Memory Rule Register 0 */
    uint32_t aips_bridge0_mem_rule1;  /*!< AIPS BRIDGE 0 Peripherals  Memory Rule Register 1 */
    uint32_t aips_bridge0_mem_rule2;  /*!< AIPS BRIDGE 0 Peripherals  Memory Rule Register 2 */
    uint32_t aips_bridge0_mem_rule3;  /*!< AIPS BRIDGE 0 Peripherals  Memory Rule Register 3 */

    uint32_t ahb_periph0_mem_rule0;       /*!< AHB Peripherals 0 Memory Rule Register 0 */
    uint32_t ahb_periph0_mem_rule1;       /*!< AHB Peripherals 0 Memory Rule Register 1 */
    uint32_t ahb_periph0_mem_rule2;       /*!< AHB Peripherals 0 Memory Rule Register 2 */

    uint32_t aips_bridge1_mem_rule0;  /*!< AIPS BRIDGE 1 Peripherals  Memory Rule Register 0 */
    uint32_t aips_bridge1_mem_rule1;  /*!< AIPS BRIDGE 1 Peripherals  Memory Rule Register 1 */

    uint32_t ahb_periph1_mem_rule0;       /*!< AHB Peripherals 1 Memory Rule Register 0 */
    uint32_t ahb_periph1_mem_rule1;       /*!< AHB Peripherals 1 Memory Rule Register 1 */
    uint32_t ahb_periph1_mem_rule2;       /*!< AHB Peripherals 1 Memory Rule Register 2 */

    uint32_t aips_bridge2_mem_rule0;  /*!< AIPS BRIDGE 2 Peripherals  Memory Rule Register 0 */
    uint32_t aips_bridge2_mem_rule1;  /*!< AIPS BRIDGE 2 Peripherals  Memory Rule Register 1 */

    uint32_t aips_bridge3_mem_rule0;  /*!< AIPS BRIDGE 3 Peripherals  Memory Rule Register 0 */
    uint32_t aips_bridge3_mem_rule1;  /*!< AIPS BRIDGE 3 Peripherals  Memory Rule Register 1 */
    uint32_t aips_bridge3_mem_rule2;  /*!< AIPS BRIDGE 3 Peripherals  Memory Rule Register 2 */
    uint32_t aips_bridge3_mem_rule3;  /*!< AIPS BRIDGE 3 Peripherals  Memory Rule Register 3 */

    uint32_t aips_bridge4_mem_rule0;  /*!< AIPS BRIDGE 4 Peripherals  Memory Rule Register 0 */
    uint32_t aips_bridge4_mem_rule1;  /*!< AIPS BRIDGE 4 Peripherals  Memory Rule Register 1 */
    uint32_t aips_bridge4_mem_rule2;  /*!< AIPS BRIDGE 4 Peripherals  Memory Rule Register 2 */
    uint32_t aips_bridge4_mem_rule3;  /*!< AIPS BRIDGE 4 Peripherals  Memory Rule Register 3 */

    uint32_t ahb_sec_ctrl_periph_rule;   /*!< AHB Secure Controller Peripheral Rule Register */

    uint32_t fspi0_reg0_mem_rule0;    /*!< FLEXSPI0 Region 0 Memory Rule Register 0 */
```

```
    uint32_t fspi0_reg0_mem_rule1;    /*!< FLEXSPI0 Region 0 Memory Rule Register 1 */
    uint32_t fspi0_reg0_mem_rule2;    /*!< FLEXSPI0 Region 0 Memory Rule Register 2 */
    uint32_t fspi0_reg0_mem_rule3;    /*!< FLEXSPI0 Region 0 Memory Rule Register 3 */
    uint32_t fspi0_reg1_mem_rule;     /*!< FLEXSPI0 Region 1 Memory Rule Register */
    uint32_t fspi0_reg2_mem_rule;     /*!< FLEXSPI0 Region 2 Memory Rule Register */
    uint32_t fspi0_reg3_mem_rule;     /*!< FLEXSPI0 Region 3 Memory Rule Register */
    uint32_t fspi0_reg4_mem_rule;     /*!< FLEXSPI0 Region 4 Memory Rule Register */
    uint32_t fspi0_reg5_mem_rule;     /*!< FLEXSPI0 Region 5 Memory Rule Register */
    uint32_t fspi0_reg6_mem_rule;     /*!< FLEXSPI0 Region 6 Memory Rule Register */
    uint32_t fspi0_reg7_mem_rule0;    /*!< FLEXSPI0 Region 7 Memory Rule Register 0 */
    uint32_t fspi0_reg7_mem_rule1;    /*!< FLEXSPI0 Region 7 Memory Rule Register 1 */
    uint32_t fspi0_reg7_mem_rule2;    /*!< FLEXSPI0 Region 7 Memory Rule Register 2 */
    uint32_t fspi0_reg7_mem_rule3;    /*!< FLEXSPI0 Region 7 Memory Rule Register 3 */
    uint32_t fspi0_reg8_mem_rule;     /*!< FLEXSPI0 Region 8 Memory Rule Register */
    uint32_t fspi0_reg9_mem_rule;     /*!< FLEXSPI0 Region 9 Memory Rule Register */
    uint32_t fspi0_reg10_mem_rule;    /*!< FLEXSPI0 Region 10 Memory Rule Register */
    uint32_t fspi0_reg11_mem_rule;    /*!< FLEXSPI0 Region 11 Memory Rule Register */
    uint32_t fspi0_reg12_mem_rule;    /*!< FLEXSPI0 Region 12 Memory Rule Register */
    uint32_t fspi0_reg13_mem_rule;    /*!< FLEXSPI0 Region 13 Memory Rule Register */

    uint32_t sec_gpio_pint_mask0;     /*!< Secure GPIO PINT Mask Register 0 */
    uint32_t sec_gpio_pint_mask1;     /*!< Secure GPIO PINT Mask Register 1 */
    uint32_t sec_int_reg0;            /*!< Secure Interrupt Mask for CPU1 Register 0 */
    uint32_t sec_int_reg1;            /*!< Secure Interrupt Mask for CPU1 Register 1 */
    uint32_t sec_int_reg2;            /*!< Secure Interrupt Mask for CPU1 Register 2 */
    uint32_t sec_int_reg3;            /*!< Secure Interrupt Mask for CPU1 Register 3 */
    uint32_t sec_int_reg4;            /*!< Secure Interrupt Mask for CPU1 Register 4 */

    uint32_t sec_mask_lock;           /*!< Secure GPIO Mask Lock Register 2 */

    uint32_t master_sec_reg;          /*!< Master Secure Level Register */
    uint32_t master_sec_anti_pol_reg; /*!< Master Secure Level Anti-pole Register */

    uint32_t cm33_lock_reg0;           /*!< CM33 Lock Control Register 0 */
    uint32_t cm33_lock_reg1;           /*!< CM33 Lock Control Register 1 */

    uint32_t misc_ctrl_dp_reg;        /*!< Secure Control Duplicate Register */
    uint32_t misc_ctrl_reg;           /*!< Secure Control Register */

    /* GLBAC 1,2,3,6,7 are not included because they are locked after reset */
    uint32_t mbc0_memn_glbac0;         /*!< MBC Global Access Control Register 0 */
    uint32_t mbc0_memn_glbac4;         /*!< MBC Global Access Control Register 4 */
    uint32_t mbc0_memn_glbac5;         /*!< MBC Global Access Control Register 5 */

    uint32_t mbc0_dom0_mem0_blk_cfg_w0;    /*!< MBC Memory Block Configuration Word 0 */
    uint32_t mbc0_dom0_mem0_blk_cfg_w1;    /*!< MBC Memory Block Configuration Word 1 */
    uint32_t mbc0_dom0_mem0_blk_cfg_w2;    /*!< MBC Memory Block Configuration Word 2 */
    uint32_t mbc0_dom0_mem0_blk_cfg_w3;    /*!< MBC Memory Block Configuration Word 3 */
    uint32_t mbc0_dom0_mem0_blk_cfg_w4;    /*!< MBC Memory Block Configuration Word 4 */
    uint32_t mbc0_dom0_mem0_blk_cfg_w5;    /*!< MBC Memory Block Configuration Word 5 */
    uint32_t mbc0_dom0_mem0_blk_cfg_w6;    /*!< MBC Memory Block Configuration Word 6 */
    uint32_t mbc0_dom0_mem0_blk_cfg_w7;    /*!< MBC Memory Block Configuration Word 7 */

    /* mbc0_dom_0_mem1_blk_cfg_w0, mbc0_dom_0_mem2_blk_cfg_w0, IFR pages cannot be configured by user
*/

    /* ITRC inputs 0-15 */
    uint32_t itrc_irq_out0_sel0;           /*!< ITRC IRQ Trigger source selector register 0 */
    uint32_t itrc_irq_out0_sel1;           /*!< ITRC IRQ Trigger source selector register 1 */
```

```
    uint32_t itrc_css_reset_out1_sel0;      /*!< ITRC CSS_RESET Trigger source selector register 0 */
    uint32_t itrc_css_reset_out1_sel1;      /*!< ITRC CSS_RESET Trigger source selector register 1 */

    uint32_t itrc_puf_zeroize_out2_sel0;      /*!< ITRC PUF_ZEROIZE Trigger source selector register
0 */
    uint32_t itrc_puf_zeroize_out2_sel1;      /*!< ITRC PUF_ZEROIZE Trigger source selector register
1 */

    uint32_t itrc_ram_zeroize_out3_sel0;      /*!< ITRC RAM_ZEROIZE Trigger source selector register
0 */
    uint32_t itrc_ram_zeroize_out3_sel1;      /*!< ITRC RAM_ZEROIZE Trigger source selector register
1 */

    uint32_t itrc_chip_reset_out4_sel0;      /*!< ITRC CHIP_RESET Trigger source selector register 0
*/
    uint32_t itrc_chip_reset_out4_sel1;      /*!< ITRC CHIP_RESET Trigger source selector register 1
*/

    uint32_t itrc_itr_out_out5_sel0;      /*!< ITRC ITR_OUT0 Trigger source selector register 0 */
    uint32_t itrc_itr_out_out5_sel1;      /*!< ITRC ITR_OUT0 Trigger source selector register 1 */

    uint32_t itrc_itr_out_out6_sel0;      /*!< ITRC ITR_OUT1 Trigger source selector register 0 */
    uint32_t itrc_itr_out_out6_sel1;      /*!< ITRC ITR_OUT1 Trigger source selector register 1 */

    /* ITRC inputs 16-31 */
    uint32_t itrc_irq_out0_sel0_1;            /*!< ITRC IRQ Trigger source selector register 0 */
    uint32_t itrc_irq_out0_sel1_1;            /*!< ITRC IRQ Trigger source selector register 1 */

    uint32_t itrc_css_reset_out1_sel0_1;      /*!< ITRC CSS_RESET Trigger source selector register 0
*/
    uint32_t itrc_css_reset_out1_sel1_1;      /*!< ITRC CSS_RESET Trigger source selector register 1
*/

    uint32_t itrc_puf_zeroize_out2_sel0_1;      /*!< ITRC PUF_ZEROIZE Trigger source selector
register 0 */
    uint32_t itrc_puf_zeroize_out2_sel1_1;      /*!< ITRC PUF_ZEROIZE Trigger source selector
register 1 */

    uint32_t itrc_ram_zeroize_out3_sel0_1;      /*!< ITRC RAM_ZEROIZE Trigger source selector
register 0 */
    uint32_t itrc_ram_zeroize_out3_sel1_1;      /*!< ITRC RAM_ZEROIZE Trigger source selector
register 1 */

    uint32_t itrc_chip_reset_out4_sel0_1;      /*!< ITRC CHIP_RESET Trigger source selector register
0 */
    uint32_t itrc_chip_reset_out4_sel1_;      /*!< ITRC CHIP_RESET Trigger source selector register 1
*/

    uint32_t itrc_itr_out_out5_sel0_1;      /*!< ITRC ITR_OUT0 Trigger source selector register 0 */
    uint32_t itrc_itr_out_out5_sel1_1;      /*!< ITRC ITR_OUT0 Trigger source selector register 1 */

    uint32_t itrc_itr_out_out6_sel0_1;      /*!< ITRC ITR_OUT1 Trigger source selector register 0 */
    uint32_t itrc_itr_out_out6_sel1_1;      /*!< ITRC ITR_OUT1 Trigger source selector register 1 */

    /* ITRC inputs 32-47 */
    uint32_t itrc_irq_out0_sel0_2;            /*!< ITRC IRQ Trigger source selector register 0 */
    uint32_t itrc_irq_out0_sel1_2;            /*!< ITRC IRQ Trigger source selector register 1 */

    uint32_t itrc_css_reset_out1_sel0_2;      /*!< ITRC CSS_RESET Trigger source selector register 0
*/
```

```
    uint32_t itrc_css_reset_out1_sel1_2;      /*!< ITRC CSS_RESET Trigger source selector register 1
*/

    uint32_t itrc_puf_zeroize_out2_sel0_2;      /*!< ITRC PUF_ZEROIZE Trigger source selector
register 0 */
    uint32_t itrc_puf_zeroize_out2_sel1_2;      /*!< ITRC PUF_ZEROIZE Trigger source selector
register 1 */

    uint32_t itrc_ram_zeroize_out3_sel0_2;      /*!< ITRC RAM_ZEROIZE Trigger source selector
register 0 */
    uint32_t itrc_ram_zeroize_out3_sel1_2;      /*!< ITRC RAM_ZEROIZE Trigger source selector
register 1 */

    uint32_t itrc_chip_reset_out4_sel0_2;      /*!< ITRC CHIP_RESET Trigger source selector register
0 */
    uint32_t itrc_chip_reset_out4_sel1_2;      /*!< ITRC CHIP_RESET Trigger source selector register
1 */

    uint32_t itrc_itr_out_out5_sel0_2;      /*!< ITRC ITR_OUT0 Trigger source selector register 0 */
    uint32_t itrc_itr_out_out5_sel1_2;      /*!< ITRC ITR_OUT0 Trigger source selector register 1 */

    uint32_t itrc_itr_out_out6_sel0_2;      /*!< ITRC ITR_OUT1 Trigger source selector register 0 */
    uint32_t itrc_itr_out_out6_sel1_2;      /*!< ITRC ITR_OUT1 Trigger source selector register 1 */

    uint32_t gpio0_pcns;                    /**< GPIO0 Pin Control Non-Secure */
    uint32_t gpio0_pcnp;                    /**< GPIO0 Pin Control Non-Privilege */
    uint32_t gpio0_icns;                    /**< GPIO0 Interrupt Control Non-Secure */
    uint32_t gpio0_icnp;                    /**< GPIO0 Interrupt Control Non-Privilege */
    uint32_t gpio0_lock;                    /**< GPIO0 Lock Register */

    uint32_t gpio1_pcns;                    /**< GPIO1 Pin Control Non-Secure */
    uint32_t gpio1_pcnp;                    /**< GPIO1 Pin Control Non-Privilege */
    uint32_t gpio1_icns;                    /**< GPIO1 Interrupt Control Non-Secure */
    uint32_t gpio1_icnp;                    /**< GPIO1 Interrupt Control Non-Privilege */
    uint32_t gpio1_lock;                    /**< GPIO1 Lock Register */

    uint32_t gpio2_pcns;                    /**< GPIO2 Pin Control Non-Secure */
    uint32_t gpio2_pcnp;                    /**< GPIO2 Pin Control Non-Privilege */
    uint32_t gpio2_icns;                    /**< GPIO2 Interrupt Control Non-Secure */
    uint32_t gpio2_icnp;                    /**< GPIO2 Interrupt Control Non-Privilege */
    uint32_t gpio2_lock;                    /**< GPIO2 Lock Register */

    uint32_t gpio3_pcns;                    /**< GPIO3 Pin Control Non-Secure */
    uint32_t gpio3_pcnp;                    /**< GPIO3 Pin Control Non-Privilege */
    uint32_t gpio3_icns;                    /**< GPIO3 Interrupt Control Non-Secure */
    uint32_t gpio3_icnp;                   /**< GPIO3 Interrupt Control Non-Privilege */
    uint32_t gpio3_lock;                   /**< GPIO3 Lock Register */

    uint32_t gpio4_pcns;                    /**< GPIO4 Pin Control Non-Secure */
    uint32_t gpio4_pcnp;                   /**< GPIO4 Pin Control Non-Privilege */
    uint32_t gpio4_icns;                    /**< GPIO4 Interrupt Control Non-Secure */
    uint32_t gpio4_icnp;                    /**< GPIO4 Interrupt Control Non-Privilege */
    uint32_t gpio4_lock;                    /**< GPIO4 Lock Register */

    uint32_t gpio5_pcns;                   /**< GPIO5 Pin Control Non-Secure */
    uint32_t gpio5_pcnp;                    /**< GPIO5 Pin Control Non-Privilege */
    uint32_t gpio5_icns;                   /**< GPIO5 Interrupt Control Non-Secure */
    uint32_t gpio5_icnp;                    /**< GPIO5 Interrupt Control Non-Privilege */
    uint32_t gpio5_lock;                    /**< GPIO5 Lock Register */
```

```
} tzm_secure_config_t;
```

Almost all data in TZ-M preset data structure is one to one copy to the corresponding register. The data with specific function is described in next section.

9.14.2 TZ-M Data with Specific Function

1. uint32_t tzm_magic

   tzm_magic is used to identify correct start of TZ-M preset data block. This value must be set to four letters “TZ-M”. In little endian format it corresponds to value 0x4d2d5a54. Every TZ-M preset data block must start with this value otherwise boot process fails.

2. uint32_t cm33_misc_ctrl

   The cm33_misc_ctrl variable controls configuration of several core TrustZone related features spread among different SCB core registers. For better data size efficiency, the control of these features was merged into single 32-bit variable. The list of core features configured by this variable is listed below.

Table 213. cm33_misc_ctrl variable definition

Bit	Description
31-5	Reserved
4	Defines value of SYSRESETREQ bit in AIRCR register
3	Defines value of BFHFNMINS bit in AIRCR register
2	Defines value of PRIS bit in AIRCR register
1	Defines value of SLEEPDEEPS bit in SCR register
0	Defines value of SECUREFAULTENA bit in SHCSR register

9.14.3 TrustZone Preset Data Restrictions

The TrustZone configuration takes effect already in ROM code execution before a jump to the user application. Therefore, the user’s TrustZone configuration data must allow ROM code to successfully jump to user application. This means that user’s TrustZone settings must configure:

1. Whole ROM space (0x13000000-0x1303FFFF) as secure privilege,
2. The RAM space 0x30000000-0x30003FFF as secure privilege
3. In the case, that secure MPU is used,
   - The whole ROM space (0x13000000-0x1303FFFF) must be configured for code execution.
   - The RAM space 0x30012000-0x30017FFF must be available for R/W privilege access.
4. The ROM code validates TZ-M preset data in configured registers. Therefore if register contains reserved or read only bits, the values of such bits in TZ-M preset data block has to match default value read after reset. For example if some register contains reserved bits, which are read as logical “1”, then also TrustZone preset data must set the same reserved bits as “1".

If any condition above is not met, boot process fails during pre-set data configuration.

Chapter 10
Debug Mailbox (DBGMB)

10.1 Chip-specific DBGMB information

Table 214. Reference links to related information[1]

Topic	Related module	Reference
Full description	Debug Mailbox	Debug Mailbox
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

10.1.1 Module instances

This device has one instance of the Debug Mailbox module.

10.2 Overview

This chapter outlines the debug capabilities implemented in this chip through DBGMB. Debugging uses the Arm Serial Wire Debug (SWD) interface. This chapter is for debug tool developers and assumes that you know the Arm SWD interface and CoreSight (TM) debug and trace technology.

DBGMB is accessible through two paths:

- By software running on the chip using the DBGMB register interface as the access method.
- By tools software running on a PC connected to a device over SWD pins. These tools access DBGMB using the Arm Debug Interface (ADIv5) specification where the individual nodes or ports are called Access Ports (AP). In this context, we are using the DBGMB AP (DM-AP).

10.2.1 Block diagram

This figure shows the top-level debug ports and connections.

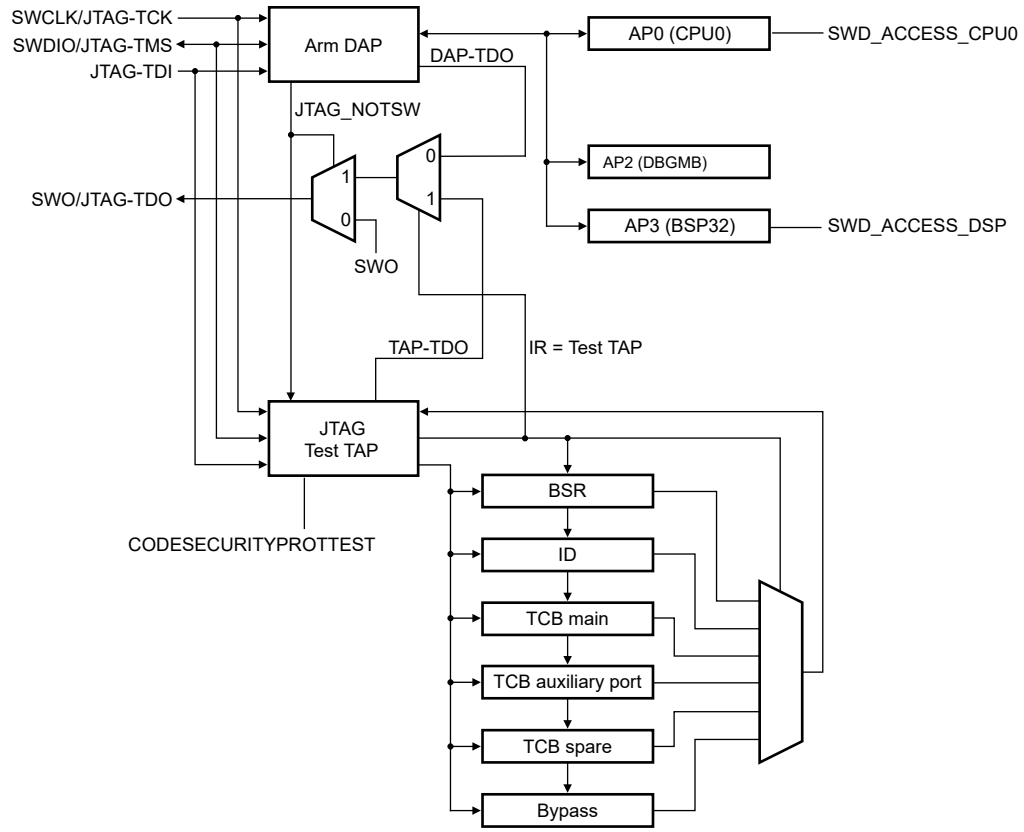


Figure 59. Serial Wire Debug (SWD) internal connections

10.2.2 Ports

Table 215. Ports

Port	Description
Arm DAP	Debug access port with a serial wire port (SWJ-DP) which interprets incoming data and routes it to the appropriate AP
AP0 (CPU0)	Debug access port for the Cortex-M33 core instantiated as CPU0
AP2 (DBGMB)	Debug access port for the debug mailbox. DBGMB: • Sends and receives messages from the code executing from the ROM. • Is always enabled. ROM can send and receive data externally. • Implements the NXP debug authentication protocol.
AP3 (BSP32)	Debug access port for CoolFlux BSP32

For JTAG identifier details, see the "System Controller (SYSCON)" chapter.

10.2.3 Features

- Support for Arm SWD mode
- Cortex-M33 CPU instruction trace capability via trace port, with output via a serial wire viewer
- Direct debug access to all memories, registers, and peripherals

- No target resource requirements for a debugging session
- Support for setting instruction breakpoints
- Support for setting data watchpoints, which you can use as triggers
- Optional additional software-controlled trace for the CPU via the instrumentation trace macrocell (ITM)

10.2.4 Glossary

Table 216. Glossary

Abbreviation	Term	Description
RoT	Root of Trust	Vendor-owned key pair that authorizes data assets via cryptographic signatures. The public part of the key is typically pre-configured in products so that data from untrusted sources can be cryptographically verified. The vendor public key used by the device to verify the signature of this DC. (The corresponding private key was used to sign the DC).
RoTpub	RoT Public Key	The vendor public key used by the device to verify the signature of this DC. (The corresponding private key was used to sign the DC.)
RoTID	RoT Identifier	RoTID allows the debugger to infer which RoT public keys are acceptable to the device. If the debugger does not provide such a credential, the authentication process fails.
RoTMETA	RoT metadata	The RoT metadata required by the device to corroborate: the ROTID sent in the DAC, the field in this DC, and any additional RoT state not stored within the device. This metadata allows different RoT identification, management, and revocation solutions to be handled.
SoCC	SoC Class	A unique identifier for a set of SoCs that require no SoC-specific differentiation in their debug authentication. The main usage is to allow a different set of debug domains and options to be negotiated between the device configuration and credentials. If the granularity of debug control warrants it, a class can contain a single revision of a single SoC model.
DCK	Debug Credential Key	A user-owned key pair. The public part of the key is associated with a DC, the private part is held by the user and used to produce signatures during authentication.
DC	Debug Credential	A user public key and associated attributes, bound together and signed by a RoT, serves as an *identity*.
CC	Credential Constraint	In product configuration, CCs are limitations on the DCs that the device accepts for authentication. In DCs, CCs are vendor/RoT-authorized usages of the DC, as well as inputs to the desired debug behavior.
VU	Vendor Usage	A CC (constraint) value that is opaque to the debug authentication protocol itself but which can be leveraged by vendors in product-specific ways.
SoCU	SoC Usage	A CC (constraint) value that is a bit mask, and whose bits are used in an SoC-specific manner. These bits are typically used for controlling which debug domains are accessed via the authentication protocol. Device-specific debug options can also be managed in this way.

Table continues on the next page...

Table 216. Glossary (continued)

Abbreviation	Term	Description
CB AB	Credential Beacon Authentication beacon	A value that is passed through the authentication protocol, which is not interpreted by the protocol but is instead made accessible to the application being debugged. A credential beacon is associated with a DC and is therefore vendor/RoT-signed. An authentication beacon is provided and signed by the debugger during the authentication process.
DCFG_*	Debug Config	Refers to device configuration settings stored in OTP
CMPA		Customer manufacturing programmable area
CFPA		Customer field-programmable area

10.3 Functional description

10.3.1 Basic configuration

This chip supports Arm's serial wire debug (SWD) interface. SWD is the default function for pins PIO0_0 (SWCLK) and PIO0_9 (SWDIO) after a reset. If you use serial wire output (SWO), you must enable it in the application code by selecting the SWO function on PIO0_8 or PIO_18 and enabling the trace clock (see External signals).

The ROM controls debug access via a remote host, and it is enabled only when permitted through the chip configuration and when you follow the correct protocol to initiate a debug session. If you have configured the chip for debug authentication, you must initiate a debug session following the correct authentication sequence. When a chip is in the development life-cycle state, use the debug session protocol described in Debug session protocol. When the chip is in deployed life-cycle state, use the debug authentication protocol described in Debug authentication.

10.3.2 Debug Access Port (DAP)

The DAP with a serial wire port (SWJ-DP) interprets incoming data and routes it to the appropriate AP. External signals describes the external I/O pins that interface with DAP. The DAP block is always enabled, but you can use the I/O pins that provide access to the SWD signals for other functions controlled by software.

10.3.3 Arm Cortex-M33 access port

The debug access port (DAP) for the Arm Cortex-M33 processor is disabled during power-on reset (POR) or during the assertion of the reset pin. The ROM enables the DAP when you follow the correct debug initiation procedures. If you are not using the DAP, you can use the debug enablement protocol to initiate a debug session. If debug authentication is required, see Debug authentication.

The debug authentication process allows control of the DBGEN, NIDEN, SPIDEN, and SPNIDEN authentication signals connected to Cortex-M33 as described below.

Table 217. Authentication signals

Signal	Description
DBGEN	Invasive debugging of TrustZone for Armv8-M architecture-defined nonsecure domain • Breakpoints and watch points halt the processor on a specific activity. • A debug connection examines and modifies registers and memory, and it provides single-step execution.

Table continues on the next page...

Table 217. Authentication signals (continued)

Signal	Description
NIDEN	Noninvasive debugging of TrustZone for a defined nonsecure domain • Collects information on instruction execution and data transfers. • Delivers trace in real time to off-chip tools to merge data with source code on a development workstation for future analysis.
SPIDEN	Invasive debugging of TrustZone for Armv8-M architecture-defined secure domain
SPNIDEN	Noninvasive debugging of TrustZone for Armv8-M architecture-defined secure domain

10.3.4 Debugger mailbox access port (DM-AP)

The DM-AP provides a register-based mailbox accessible by CPU0 and the device debug port (DP) of the MCU. This port is always enabled. An external host communicating through the SWD interface can exchange messages and data with the boot code executing from the ROM on CPU0. This port implements the NXP debug authentication protocol. See Debug authentication for a description of the protocol used to initiate a debug session from a host debug system.

The boot ROM implements a debug mailbox protocol to interact with tools via the SWD interface.

The debug mailbox protocol has the following features:

- Request-and-response based, where the requests and responses use the same basic structure
- Support for relatively large command and response data
- 32-bit word alignment for all commands and responses
- Support for data above 32 bits via an ACK_TOKEN that moderates the transfer in 32-bit value chunks

10.3.5 Debug session protocol

When DBGMB fetches instructions from the ROM address range during boot, the DAP of CPU0 is disabled, regardless of device life-cycle state or DCFG_SOCU settings. This document refers to this mechanism as Boot-ROM protection. The method to initiate a debug session varies depending on the device state and intended debugging scenario. The scenarios described in the rest of these sections are:

Table 218. Debug session scenarios

Scenario	Description
Debug session with uninitialized or invalid image or ISP mode	If DBGMB detects no valid image, the ROM proceeds into In-System Programming (ISP) mode and waits to be booted via one of its serial interfaces. In ISP mode, the debug interface is disabled.
Debug session with valid application	Program control is in ISP mode, initiated because the ISP pin is asserted on the device at reset.
Debug session attaching to a running target	Connecting to a device running a valid application with the intent to debug without writing an image (also called a debug attach).
Halting execution immediately following ROM execution	Connecting to a device running a valid application, with the intent to write a new application.
Loading SB3 files over the SWD interface	Creating and loading an OEM provisioning FW SB3 file.

10.3.5.1 Debug session with uninitialized or invalid image or ISP mode

When the chip boots and the boot media does not contain a valid image, the ROM-based program control enters ISP mode. DBGMB disables debug access for security reasons. The chip also enters into ISP mode when the ISP has been asserted as the chip leaves reset. This section describes how to establish a debug session for these scenarios.

To ensure that the state machine controlling debug mailbox commands is in a known state, the debugger can reset this logic via the following steps:

1. Write 1 to CSW[RESYNCH_REQ] to request resynchronization..
2. Write 1 to CSW[CHIP_RESET_REQ] to reset the chip.
3. Read Command and Status Word (CSW).
4. Wait until the chip has completed resynchronization, identified by reading a value of 0 from the lower 16 bits of CSW.

To start a debug session and control the exchange of debug information, the debugger uses the DM-AP commands:

1. Following a successful initial resynchronization, communicate with the chip via 32-bit DM-AP commands to the REQUEST register in the debug mailbox.
2. Read results via the RETURN register. To ensure that the transactions have completed successfully, the debugger must poll the RETURN register as it polled the CSW register following a resynchronization request.
3. View the results in Response packet.

To initiate a debug session over SWD while debug is disabled:

1. The debug system must issue a Start Debug Session command to the ROM-based boot code, following the Debug session protocol.
2. Upon receiving the command, the boot code disables any unwanted peripheral and manages NXP secrets before enabling debug access.
3. After enabling debug access, the ROM enters a while(1) loop.

Once the Start Debug Session command and chip reset have executed successfully, the AP for CPU0 is accessible. You can use it to set breakpoints and perform other tasks, as with other Arm Cortex-M devices.

Following a successful debug connection, a flashloader is loaded into RAM to program the application to be debugged and to set required breakpoints in the code. After this process, a SYSRESET_REQ command must be issued to verify that the ROM fully executes (similar to a deployed end application) before reaching the downloaded application.

The code sample below shows how to initiate a debug session for the scenarios described above:

```
// Pseudo Code Syntax
// ------------------------------------------------------------------------
// WriteDP "register" "value"
// value = ReadDP "register"
// AP transactions presume the DM AP is selected
// WriteAP "register" "value"
// value = ReadAP "register" "value"
// ------------------------------------------------------------------------
```

```
// Read AP ID register to identify DM AP at index 2
WriteDP 2 0x020000F0
// The returned AP ID should be 0x002A0000
value = ReadAP 3
print "AP ID: ", value
```

```
// Select DM AP index 2
WriteDP 2 0x02000000
// Write DM RESYNC_REQ + CHIP_RESET_REQ
WriteAP 0 0x21
```

```
// Poll CSW register (0) for zero return, indicating success
value = -1
while value != 0 {
value = ReadAP 0
}
print "RESYNC_REQ + CHIP_RESET_REQ: ", value
// Write DM START_DBG_SESSION to REQUEST register (1)
WriteAP 1 7
// Poll RETURN register (2) for zero return
value = -1
while value != 0 {
value = (ReadAP 2 & 0xFFFF)
}
print "DEBUG_SESSION_REQ: ", value
```

10.3.5.2 Debug session with valid application

In this case, the application has not disabled debug. You can access the CPU0 AP and set breakpoints without resynchronizing the mailbox hardware or issuing a Debug Session Request. You can use the methods described in Debug session with uninitialized or invalid image or ISP mode to simplify debug support implementations.

10.3.5.3 Debug session attaching to a running target

In this scenario, the device has booted and is running an application that has not disabled debug. The host system is attempting to connect to the device without resetting it and without updating the application. In this case, the CPU0 AP should be accessible and you can set breakpoints without resynchronizing the mailbox hardware.

10.3.5.4 Halting execution immediately following ROM execution

Traditionally, debug systems can set a vector catch at the reset vector to break code execution, but the chip ROM prevents this method.

The debugger must use this reset sequence:

Table 219. Reset procedure

Step	Purpose	Programming	Notes
1	Allow the debug system to halt execution immediately after the ROM completes the preparations associated with a debug session request.	Set the data watchpoint on a read to address location 5000_0040h.	—
2	—	If all data watchpoint comparators are occupied, backup one of the watchpoint settings and replace it with the above watchpoint location.	—
3	Reset the core and peripherals.	Write 1 to the Arm Cortex-M33 field AIRCR[SYSRESETREQ].	—
4	Allow the ROM to reenable debug access.	Wait for 100 ms.	—
5	Verify that the core has halted at the watchpoint.	Read the Arm Cortex-M33 register DHCSR.	If the DHCSR read times out or returns an error response, the ROM has entered an ISP command-handling loop due to

Table continues on the next page...

Table 219. Reset procedure (continued)

Step	Purpose	Programming	Notes
			an invalid image in boot media. Proceed to Step 6, otherwise, proceed to Step 10.
6	—	Execute the start debug session sequence described in Debug session with uninitialized or invalid image or ISP mode.	—
7	—	Wait for 10 ms.	—
8	—	Enable the Cortex-M33 AP.	—
9	—	Read DHCSR and issue a HALT command to Cortex-M33 AP.	—
10	—	Clear data watchpoint added in step 1. If you set the watchpoint as described in step 2, restore the watchpoint configuration.	—

10.3.5.5 Loading SB3 files over the SWD interface

Use these steps to create an OEM provisioning framework that is loadable using DBGMB_NXP_EXEC_RROV_FW:

Table 220. Creating a loadable provisioning framework

Step	Purpose	Programming	Notes
1	—	Power on the board.	—
2	—	Write 21h to CSW.	—
3	Enable debug access.	Issue a "Start debug session" mailbox command in Jlink Script Function InitTarget().	—
4	Force the chip into ISP mode.	Reset the chip (pad reset) while holding the ISP button.	—
5	—	Use ISP provisioning commands to generate the OEM provisioning FW SB3.1 file.	—

Use these steps to load a provisioning framework:

Table 221. Loading a provisioning framework

Step	Purpose	Programming	Notes
1	—	Power on the board.	—
2	—	Write 21h to CSW.	—
3	Enable debug access.	Issue a "Start debug session" mailbox command.	—

Table continues on the next page...

Table 221. Loading a provisioning framework (continued)

Step	Purpose	Programming	Notes
4	—	Load the byte size of the SB file binary to address 30018000h.	—
5	—	Load the SB binary data starting from address 30018004h.	—
6	—	Write 21h to CSW.	—
7	—	Issue DBGMB_NXP_EXEC_PROV_FW mailbox command.	—

10.3.6 Reset handling

The debug domain (DP, Cortex-M33 AP, DM-AP) is reset upon POR or pin reset (assertion of external nRESET). On other resets, the debug domain retains its state. The defined breakpoints and watchpoints persist even when the debugging tool issues a reset to the device.

10.3.7 Mailbox commands

This section describes the request and response message formats and available mailbox commands.

10.3.7.1 Request packet layout

The first word transmitted in a request is a header word containing the command ID and the number of following data words. The command packet is sent to the device by writing 32 bits at a time to the REQUEST register. When sending command packets greater than 32 bits, the debugger must read an ACK_TOKEN in the RETURN register before writing the next 32 bits.

The 32-bit words quantified by the header follow the header itself.

Table 222. Request register byte description

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	commandID[7:0]	commandID[15:8]	dataWordCount[7:0]	dataWordCount[15:8]
1	*data*	—	—	—

The C structure definition for a request is:

```
struct dm_request {
        uint16_t commandID;
        uint16_t dataWordCount;
        uint32_t data[ ];
        };
```

10.3.7.1.1 DM-AP commands

DM-AP commands are written to the REQUEST register. An Exit DM_AP command follows one or more of the DM-AP commands in the table below to resume the normal device boot flow. This sequence does not occur after the Enter ISP Mode and Start Debug Session commands.

Table 223. DM-AP commands

Name	Command code	Parameter and response	Description
Start DM-AP (legacy command)	01h	Parameters: None Response: 32-bit status	Causes the device to enter DM-AP command mode. If used, this command must occur before sending other commands. This command is provided for backward compatibility and is not required.
Get CRP Level	02h	Parameters: None Response: 32-bit status	Returns CRP levels. Returns the device life-cycle state using the following mapping: • 0 – In "Develop" life-cycle state. • 1 – In "Develop2" life-cycle state. • 2 – In "In-field" life-cycle state. • 3 – In "Field Return OEM" life-cycle state.
Mass Erase (legacy command)	03h	Parameters: None Response: 32-bit status	Erase the entire on-chip flash memory, excluding protected flash regions (called either IFR or PFR). This command erases the entire user flash memory. If the life-cycle state value in OTP fuses is "Develop" (3h), both CMPA and CFPA regions are erased. **NOTE** If the CFPA_LC_STATE tests deployed life-cycle states without programming fuses, you must change the CFPA_LC_STATE back to 0h or 3h to perform the mass erase.
Exit DM_AP (legacy command)	04h	Parameters: None Response: 32-bit status	Causes the device to continue normal debug flow.
Enter ISP Mode	05h	Parameters: • dataWordCount: 1h • data[0]: ISP mode enum • FFFFFFFFh0635441A1 Auto detection See the Boot ROM chapter for supported ISP modes. Response: 32-bit status	Enters the specified ISP mode. By default, the state of the ISP boot selection pins at time of reset determines the ISP mode entry. The protected flash region (called either IFR or PFR) configuration usually disables this functionality prior to field deployment. You can use this command to enter ISP mode in those situations or when a different board function uses the ISP boot selection pins. You can restrict support of this command through the

Table continues on the next page...

Table 223. DM-AP commands (continued)

Name	Command code	Parameter and response	Description
			DCFG_SOCU field in the protected flash region (called either IFR or PFR), as described in Debug authentication.
Set FA Mode	06h	Parameters: None Response: 32-bit status	Sets the part permanently in Fault Analysis (FA) mode to return to an NXP factory. Upon receiving this command boot code in ROM, customer-sensitive assets (key codes), stored in the protected flash region (called either IFR or PFR), are erased. Also, the FA or RMA mode bit in the protected flash region becomes 1, so suspect parts can be sent to NXP for FA/RMA testing. You can restrict support of this command through the DCFG_SOCU field in the protected flash region (called either IFR or PFR), as described in Debug authentication.
Start DBG Session	07h	Parameters: None Response: 32-bit status	Starts debug session. Program control is in a ROM memory context when DBGMB fetches instructions from a ROM memory address range. When program control is in a ROM memory context during boot, debug access is disabled regardless of the device life-cycle state or DCFG_SOCU settings. This command instructs the boot code in the ROM to clean up the memory and peripherals initialized by the boot code: SysTick, UART, SPI, I2C, QSPI, SD/MMC, USB, MPU, and SAU. It also instructs the boot code to enter a safe processor execution context for external debuggers to attach. When the boot media contains no valid image, the program control enters an ISP command handler loop (still in a ROM memory context) and debug access is disabled. As a result, an external debug system must use this command to indicate to the ROM its intention to connect to the debugger. Upon receiving the command, the ROM cleans all peripheral configurations and NXP secrets before enabling debug access. After enabling debug access, the ROM enters a while(1) loop.
Debug Auth. Start	10h	Parameters: None Response: dataWordCount: 12Ch or 22Ch data[]: DAC	Start the debug authentication Protocol. ROM responds to the debugger with a debug authentication challenge (DAC) message.

Table continues on the next page...

Table 223. DM-AP commands (continued)

Name	Command code	Parameter and response	Description
Debug Auth. Response	11h	Parameters: dataWordCount: 12Ch data[]: DAR Response: 32-bit status	Debug authentication response
DBGMB_NXP_EXEC_PROV_FW	13h	—	ATE uses a hardware-unlock mechanism and downloads an SB3 image (NXP provisioning framework) to SRAM (C, D, E, F, G, H), other than SRAMX, which remains powered across the reset.

10.3.7.2 Response packet layout

The first word transmitted in a response is a header word containing the command status and the number of following data words. The command response can be read 32 bits at a time through Return Value (RETURN). The initial 32 bits contains the response header, as shown in the table below. When reading a response longer than 32 bits, the debugger writes the ACK_TOKEN to REQUEST after every read until the full response packet is received.

NOTE

To support legacy LPC command and response values, Bit_31 in the header indicates that the response follows the new protocol structure (see Table 225). This bit is 1 when using the new protocol.

Table 224. Response register byte description

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	bits[7:0]:commandStatus[7:0]	bits[7:0]:commandStatus[15:8]	bits[7:0]:dataWordCount[7:0]	bits[6:0]:dataWordCount[14:8] bits[7]:new_protocol[15]
1	*data*	—	—	—

The C structure definition for a response is:

```
struct dm_response {
        uint16_t commandStatus;
        uint16_t dataWordCount;
        uint32_t data[];
        };
```

10.3.7.2.1 Response packet

You can read the command response 32 bits at a time through Return Value (RETURN). The initial 32 bits contain the response header, as shown in the table below. When reading a response greater than 32 bits, the debugger writes the ACK_TOKEN to Request Value (REQUEST) after every read of RETURN until the full response packet is received.

Table 225. Response Packet

Bits	Field	Description
31	LONG_RESP	Long response packet indicator; also called the new protocol indicator
[30:16]	REMAIN_TRANS	Number of times the debugger must read RETURN to receive remaining response data. The debugger writes ACK_TOKEN to REQUEST after every read of RETURN until the full response packet is received. **NOTE** REMAIN_TRANS is valid only when ERROR_SRESP is 0 and LONG_RESP is 1.
[15:0]	ERROR_SRESP	Short response data or error code If bit_20 is not 1, this field is interpreted as short response data. For example, this field returns the CRP level for a GET_CRP_LEVEL command. Error codes • 0000h: Command succeeded. • 0001h: Debug mode not entered. This value is returned when other commands are sent prior to the Enter DM-AP command. • 0002h: Command is not supported. • 0003h: Communication failure. ACK_TOKEN is missing during data transactions.

10.3.7.3 ACK_TOKEN

The ACK_TOKEN provides an acknowledgment to the sender during debug mailbox data transactions. DBGMB uses the acknowledgment in the following ways:

- When the debugger issues a command with parameter data, it waits for the ACK_TOKEN (read through Return Value (RETURN)) before writing the next 32-bit value to Request Value (REQUEST).
- When the debugger receives a long response packet, it writes the ACK_TOKEN to REQUEST before reading the next 32-bit value RETURN.

Table 226. ACK_TOKEN

Bits	Field	Description
[31:16]	REMAIN_TRANS	Number of remaining data transactions
[15:0]	ACK_MARKER	Acknowledgment marker; must always be A5A5h

The C structure definition for the ACK_TOKEN is:

```
struct dm_ack_token {
        uint16_t token; /* always set to A5A5h */
        uint16_t remainCount; /* count of remaining word */
        };
```

10.3.7.4 Error handling

When an overrun occurs from either side of the communication, DBGMB sets the appropriate error flag in Command and Status Word (CSW). The state machine hardware prevents further communication in either direction. The debugger must start with a new resynchronization request to clear the error flag.

10.3.8 Debug authentication

The fundamental principles of debugging, which require access to the system state and system information, conflict with the principles of security, which require the restriction of access to assets. Thus, many products disable debug access completely before deploying the product. This causes challenges for product design teams to do proper Return Material Analysis (RMA). To address these challenges, the chip offers a debug authentication protocol as a mechanism to authenticate the debugger (an external entity) has the credentials approved by the product manufacturer before granting debug access to the device.

The debug authentication is a challenge-response scheme and assures that only the debugger in possession of the required debug credentials can successfully authenticate over the debug interface and access restricted parts of the device. This protocol is divided into steps as described below:

1. The debugger initiates the Debug Mailbox message exchange by setting the CSW[RESYNCH_REQ] bit and CSW[CHIP_RESET_REQ] bit of DM-AP.
2. The debugger waits (minimum 30 ms) for the devices to restart and enter debug mailbox request handling loop.
3. The debugger sends Debug Authentication Start command (command code 10h) to the device.
4. The device responds back with Debug Authentication Challenge (DAC) packet based on the debug access rights pre-configured in CMPA fields , which are collectively referred as Device Credential Constraints Configuration (DCFG_CC). The response packet also contains a 32 bytes random challenge vector.
5. The debugger responds to the challenge with a Debug Authentication Response (DAR) message by using an appropriate debug certificate, matching the device identifier in the DAC. The DAR packet contains the debug access permission certificate, also referred as Debug Credential (DC), and a cryptographic signature binding the DC and the challenge vector provided in the DAC.
6. The device on receiving the DAR, validates the contents by verifying the cryptographic signature of the message using the debugger's public key present in the embedded the Debug Credential (DC). On successful validation of DAR, the device enables access to the debug domains permitted in the DC.

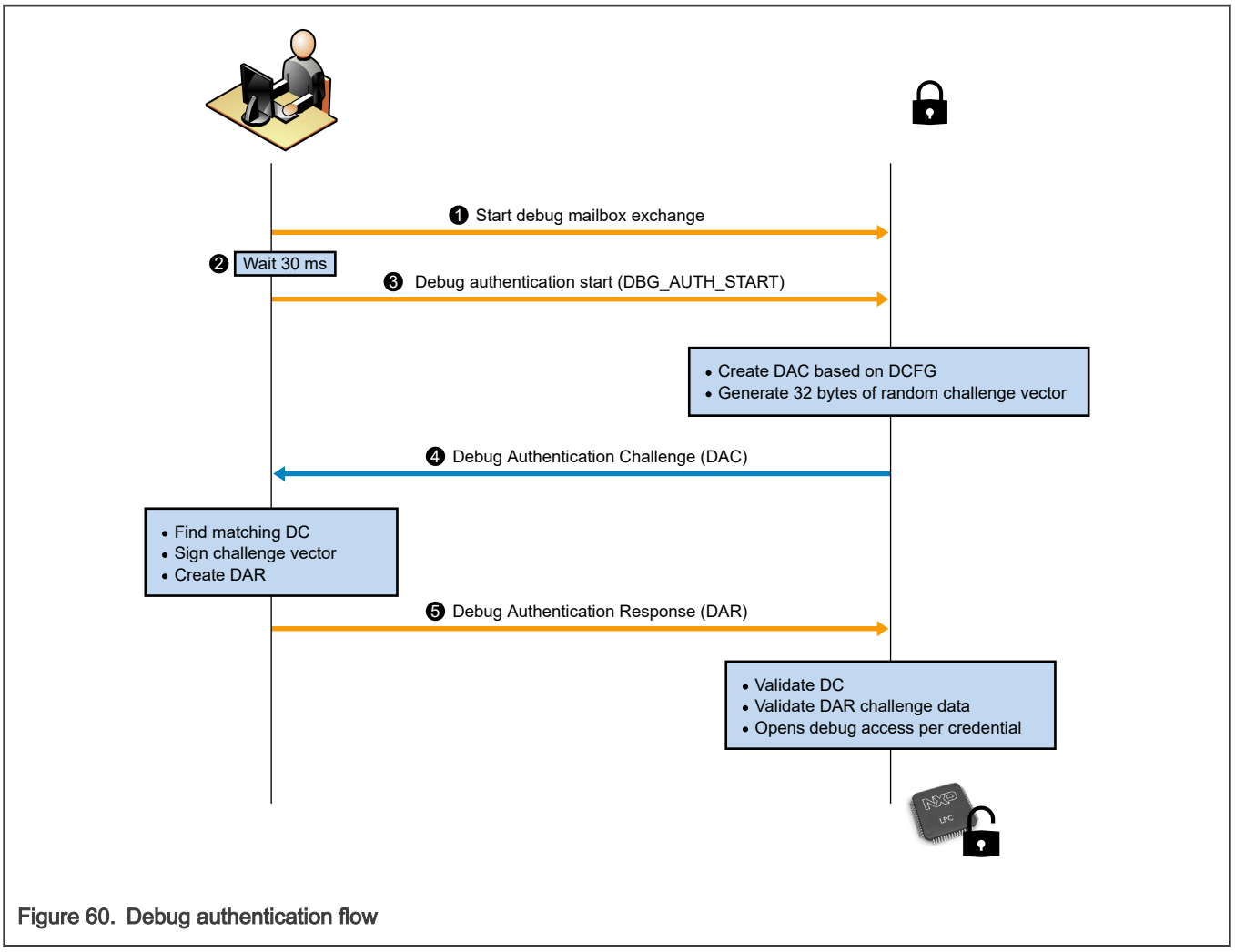


Figure 60. Debug authentication flow

10.3.8.1 Debug Access Control Configuration

The boot code present in the device ROM handles the device side of Debug authentication process. However, the debug access control rights and security policies can be configured by programming the following configuration fields which are collectively referred to as Device Configuration for Credential Constraints (DCFG_CC), present in Customer Manufacturing Programmable Area (CMPA) and Customer Field Programmable Area (CFPA).

- **DCFG_VER:** This field controls the cryptographic primitives used during authentication.
- **DCFG_ROTID:** This field defines the Root of trust identifier (ROTID). The ROTID field is used to bind the devices to specific Certificate Authority (CA) keys issuing the debug credentials. These CA keys are also referred as Root of Trust (RoTK) keys.
- **DCFG_UUID:** Controls whether to enforce UUID check during Debug Credential (DC) validation. If this field is set only DC with matching device UUID can unlock the debug access.
- **DCFG_CC_SOCU:** This configuration field specifies access rights to various debug domains.
- **DCFG_VENDOR_USAGE :** This field can be used to define vendor specific debug policy use case such as DC revocations or department identifier. It is recommended to use field for revocation of already issued debug certificates.

These fields should be programmed as part of the OEM provisioning process.

10.3.8.1.1 Protocol Version (DCFG_VER)

The device supports multiple instantiations of Debug authentication protocol versions, which are defined based on cryptographic algorithms and key sizes.

- Version 2.0: Uses ECDSA P-256 signature verification using ECC keys.
- Version 2.1: Uses ECDSA P-384 signature verification using ECC keys.

To enforce usage of ECDSA P-384, set the ENF_CNSA field in the SECURE_BOOT_CFG word (0100_4050h) to 1h, 2h or 3h in CMPA. Otherwise ECDSA P-256 algorithm will be used for debug authentication.

Note, both debug authentication certificates and image signing certificates use same Root of Trust keys (RoTK). Therefore when this field is set the secure boot image signing key certificate chain should also use P-384 keys.

10.3.8.1.2 Root of Trust Identifier (DCFG_ROTID)

The Root of Trust Identifier used in debug authentication protocol is composed of two elements.

1. A 256-bit cryptographic hash (SHA256) for version 2.0 or 384-bit cryptographic hash (SHA384) for version 2.1 over the Root of Trust Keys Table. This is same as the Root Keys Table Hash (RKTH) or Root of Trust Keys Hash (ROTKH) field in the Secure boot ROM chapter. RKTH is a 32-byte/48-byte SHA-256/SHA-384 of SHA-256/SHA-384 hashes of up to four root public keys.

   This field is located in CMPA at address 0100_4060h - 0100_408Ch.

2. A four 2-bit fields RoTKx_EN, where x = 0 - 3, containing revocation bits for the four Root of Trust keys in the table.

   These four fields are located in CFPA word SEC_BOOTFLAGS (at offset 14h).

10.3.8.1.3 Enforce UUID checking (DCFG_UUID)

Controls whether to enforce UUID check during Debug Credential (DC) validation. If this field is set then only DC containing a UUID attribute that is an exact match to the device can unlock the debug access.

This field can be set by programming UUID_CHECK (bit 15) in CC_SOCU_PIN word in CMPA field or same bit position 15 in DCFG_CC_SOCU_NS_PIN word in CFPA pages, see Table 229.

This device-specific constraint, if enabled, is in addition to all the other constraints defined and enforced by the authentication protocol.

10.3.8.1.4 Credential Constraints (DCFG_CC_SOCU)

The DCFG_CC_SOCU is a configuration that specifies debug access restrictions per debug domain. These access restrictions are also referred as constraint attributes in this section. The debug subsystem on this device is sub-divided into multiple debug domains to allow finer access control. Table 227 shows debug domains and their corresponding control bit position in the CC_SOCU field used in the Debug certificate (that is, the Debug Credential (DC)) and the OTP fuse location. The DCFG_CC_SOCU is logically composed of two components:

- **SOCU_PIN:** A bitmask that specifies which debug domains are predetermined by device configuration.
- **SOCU_DFLT:** Provides the final access level for those bits that the SOCU_PIN field indicated are predetermined by device configuration.

The following table shows the restriction levels:

Table 227. Access restriction levels

Restriction Level	SOCU_PIN [n]	SOCU_DFLT [n]	Description
0	1	1	Access to the sub-domain is always enabled. This setting is provided for module use case scenario where DCFG_CC_SOCU_NS would be used to define further access restrictions before final deployment of the product. See Module use case with Develop and Develop 2 Lifecycle states for details.

Table continues on the next page...

Table 227. Access restriction levels (continued)

Restriction Level	SOCU_PIN [n]	SOCU_DFLT [n]	Description
1	0	0	Access to the sub-domain is disabled at startup. But the access can be enabled through the debug authentication process by providing an appropriate Debug Credential (DC) certificate. **NOTE** Non-S parts without security features do not support the debug authentication process. Hence this option is not available, but other options can be used to permanently enable/ disable debug access on those devices.
-	0	1	Illegal setting. Part may lock-up if this setting is selected.
2	1	0	Access to the sub-domain is permanently disabled and cannot be reversed. This setting offers the highest level of restriction.

Debug domains supported by the device are shown below:

Table 228. CC_LIST_Table

Bit	Symbol	Description
0	NIDEN	Controls non-Invasive debugging of TrustZone for Arm8-M defined non-secure domain of CPU0.
1	DBGEN	Controls invasive debugging of TrustZone for Arm8-M defined non-secure domain of CPU0.
2	SPNIDEN	Controls non-Invasive debugging of TrustZone for Arm8-M defined secure domain of CPU0.
3	SPIDEN	Controls invasive debugging of TrustZone for Arm8-M defined secure domain of CPU0.
4	-	Reserved
5	DSP_DBG_DBGEN	Controls debugging of DSP.
6	ISP_CMD_EN	Controls whether ISP boot flow DM-AP command (command code: 05h) can be issued after authentication.
7	FA_CMD_EN	Controls whether DM-AP Set FA Mode command (command code: 06h) can be issued after authentication.
8	ME_CMD_EN	Controls whether DM-AP Mass Erase command (command code: 02h) can be issued after authentication.
10:9	-	Reserved
31:11	-	Reserved

Table 229. Layout of CC_SOCU_PIN (CMPA 1004_0040h) and CC_SOCU_PIN_NS (CFPA 100_0024h)

Bits	Symbol	Description
14:0	SOCU_PIN[n]	Defines whether the restriction level for the sub-domains is fixed or controlled by debug authentication process. The bit encoding of this field is defined as per the CC_LIST_Table.
15	UUID_CHECK	Controls whether to enforce UUID check during Debug Credential (DC) validation. If this bit is set then the device will only accept Debug Credential (DC) certificate containing UUID attribute that is an exact match to the device's UUID.
31:16	INV_SOCU_PIN[n]	Inverse value of the above bitfield [15:0].

Table 230. Layout of CC_SOCU_DFLT (CMPA 1004_0044h) and CC_SOCU_DFLT_NS 100_0028h

Bits	Symbol	Description
15:0	SOCU_DFLT[n]	Defines the restriction level for the sub-domains which are configured as predetermined in SOCU_PIN[n] field. The bit encoding of this field is defined as per the CC_LIST_Table.
31:16	INV_SOCU_DFLT[n]	Inverse value of the above bitfield [15:0].

10.3.8.1.5 DCFG_CC_SOCU authentication and life cycle dependency

Following table shows the behavior of debug domains based on current life cycle state and debug authentication status. Domain can be either enabled, disabled, or DCFG_SOCU & DCFG_SOCU_NS configuration determines the settings for given life cycle and debug authentication status.

Table 231. Life cycle status for NIDEN/DBGEN and SPNIDEN/SPIDEN

Life cycle	NIDEN/DBGEN		SPNIDEN/SPIDEN	
	not authenticated	authenticated	not authenticated	authenticated
Blank	enabled	enabled	enabled	enabled
FAB	enabled	enabled	enabled	enabled
Develop	enabled	enabled	enabled	enabled
Develop 2	enabled	enabled	disabled	DCFG_SOCU & CC_SOCU
In-field	disabled	DCFG_SOCU & DCFG_SOCU_NS & CC_SOCU	disabled	DCFG_SOCU & DCFG_SOCU_NS & CC_SOCU
In-field Locked	disabled	disabled	disabled	disabled
Field Return OEM	enabled	enabled	enabled	enabled

Table continues on the next page...

Table 231. Life cycle status for NIDEN/DBGEN and SPNIDEN/SPIDEN (continued)

Life cycle	NIDEN/DBGEN		SPNIDEN/SPIDEN	
	not authenticated	authenticated	not authenticated	authenticated
Failure Analysis (FA)	enabled	enabled	enabled	enabled
Bricked	disabled	disabled	disabled	disabled

Table 232. Life cycle status for ISP_CMD_EN

Life cycle	ISP_CMD_EN	
	not authenticated	authenticated
Blank	enabled	enabled
FAB	enabled	enabled
Develop	enabled	enabled
Develop 2	disabled	DCFG_SOCU & CC_SOCU
In-field	disabled	DCFG_SOCU & DCFG_SOCU_NS & CC_SOCU
In-field Locked	disabled	disabled
Field Return OEM	disabled	disabled
Failure Analysis (FA)	disabled	disabled
Bricked	disabled	disabled

Table 233. Life cycle status for FA_CMD_EN and ME_CMD_EN

Life cycle	FA_CMD_EN		ME_CMD_EN	
	not authenticated	authenticated	not authenticated	authenticated
Develop	enabled	enabled	enabled	enabled
Develop 2	disabled	DCFG_SOCU & CC_SOCU	disabled	DCFG_SOCU & CC_SOCU
In-field	disabled	DCFG_SOCU & DCFG_SOCU_NS & CC_SOCU	disabled	DCFG_SOCU & DCFG_SOCU_NS & CC_SOCU

Table continues on the next page...

Table 233. Life cycle status for FA_CMD_EN and ME_CMD_EN (continued)

Life cycle	FA_CMD_EN		ME_CMD_EN	
	not authenticated	authenticated	not authenticated	authenticated
In-field Locked	disabled	disabled	disabled	disabled
Field Return OEM	disabled	disabled	disabled	disabled
Failure Analysis (FA)	disabled	disabled	disabled	disabled
Bricked	disabled	disabled	disabled	disabled

10.3.8.1.6 DCFG_VENDOR_USAGE

This field can be used to define a vendor-specific debug policy use case such as Debug Credential (DC) certificate revocations or department identifier or model identifier. It is recommended to use the field for revocation of already issued debug certificates. During Debug Authentication Response (DAR) processing the device checks that the value specified in the Vendor Usage field of DC matches exactly the value programmed in DCFG_VENDOR_USAGE fields of device configurations.

The device provides a 4-byte length DCFG_VENDOR_USAGE field, composed from the following fields:

- Upper 2 bytes from CMPA.VENDOR_USAGE (protected flash region (called either IFR or PFR) address 1004_0048h)
  - Recommended using this field to identify the department or model number. So that Debug Credential (DC) Certificates can be issued on a class basis instead of issuing UUID-specific certificates.
- Lower 2 bytes from CFPA.VENDOR_USAGE (protected flash region (called either IFR or PFR) address 100_0020h)
  - In CFPA, this field is implemented as a monotonic counter field. Whenever a new version of the CFPA page is written this field can have the same value or a higher value.
  - It is recommended to use this field to revoke DC certificates issued until that point.

10.3.9 Debug Credential Certificate (DC)

By prior construction, the debugger should already have a Debug Credential Key (DCK). The public key part of this key pair represents the identity of the debugger through the creation of a DC. It binds that public key to usage attributes, and is then authorized or signed by the vendor's RoT key.

Total DC size depends on protocol version and number of used root keys.

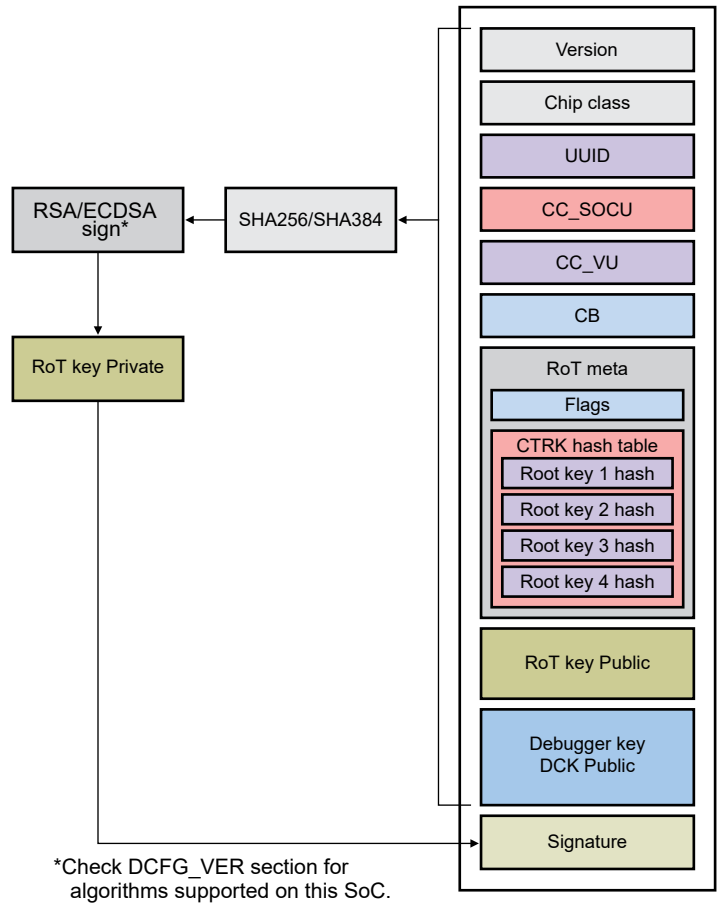


Figure 61. Debug Credential certificate fields

The data structure is a represented as a packed binary concatenation of its component fields as shown in the list below.

Table 234. Debug Credential Certificate fields

Name	Offset	Description	Size in bytes
VERSION	0000h	Identifies the Debug Authentication protocol version Version determined by ENF_CNSA field (CMPA). • 00020000h for v2.0, which uses ECDSA P-256 (ENF_CNSA = 00) • 00020001h for v2.1, which uses ECDSA P-384 (ENF_CNSA = 01, 10, 11)	4
SOCC	0004h	SoC class specifier Always set this field to 0000_0004h.	4
UUID	0008h	Unique device identifier, if this certificate is used on a device configured for UUID-matching. If UUID matching is enabled, the certificate is restricted to a specific device, otherwise the certificate is enabled for whole SoC Class.	16
CC_SOCU	0018h	SoC specific Credential Constraints Specifies the debug access rights allowed for this certificate holder.	4

Table continues on the next page...

Table 234. Debug Credential Certificate fields (continued)

Name	Offset	Description	Size in bytes
CC_VU	001Ch	Vendor Usage Should match DCFG_VENDOR_USAGE field in Device Configuration for Credential Constraints (DCFG_CC). See DCFG_VENDOR_USAGE . It can be used to revoke Debug Certificates.	4
CB	0020h	Credential beacon that vendor has associated with this DC. Only the lower 16-bits of this field are effective. This field can extend the Debug Authentication process. When a nonzero value is used in this field, ROM defers opening debug access to the user application. The result of the authentication process is written to DBG_FEATURES register. The user application, after performing extended processing such as cleaning up critical keys and secrets, should copy the value to DBG_FEATURES_DP register to enable debug access. To aid user application, ROM stores beacon values in DEBUG_AUTH_BEACON register.	4
ROTMETA_FLAGS	0024h	See Table 235	4
ROTMETA_CTRK_HASH_TABLE	0028h	SHA-256 or SHA-384 of root public key calculated based on root certificate EC size. Between two and four root certificates can be specified. In the case where only one root certificate is used, ctrkHashTable is not present.	variable (0 - 192)
ROT_KEY_PUB	variable	X and Y coordinates of Root of Trust vendor public key used for signing this certificate.	64 / 96
DCK_KEY_PUB	variable	X and Y coordinates of Debugger public key	64 / 96
SIGNATURE	variable	R and S portion of ECDSA cryptographic signature (P-256 or P-384) by the RoT over the previous fields. This signature ensures that the DC is approved by the vendor for use by the debugger.	64 / 96

Table 235. ROTMETA_FLAGS

Field	Description
3:0	Reserved
7:4	Number of records in ctrkHashTable [1-4]. When equal to 1, ctrkHashTable is not present and the device computes hash from ROT_KEY_PUB and compares it to RKTH stored in CMPA.
11:8	Determine which root cert number [0-3] was used for signing. Used only when more than one root certificate is specified.
30:12	Reserved

Table continues on the next page...

Table 235. ROTMETA_FLAGS (continued)

Field	Description
31	CA flag Should be always set to 1b in DC

10.3.9.1 Debug Authentication Challenge (DAC)

The debug authentication protocol begins with a DAC message, issued by the device to the debugger. The total message size is 104 bytes.

From a protocol perspective, the debugger must select a Debug Credential (DC) certificate that successfully authenticates based on the constraints provided in the DAC. This DC provides the required debug behavior post-authentication (for example, whether to debug secure world, with the desired credential beacon). If such a credential cannot be found, the debugger should report an error to the user.

NOTE

The debugger must also be able to produce signatures using the private key corresponding to the selected DC. This requirement exists so that any credential store can manage this association between credentials and corresponding private keys.

The named elements of this message are shown below.

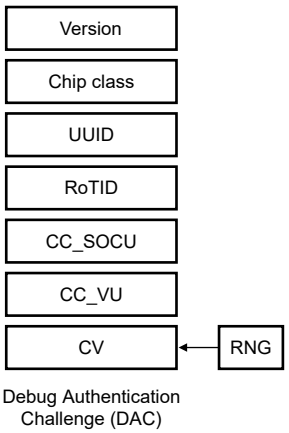


Figure 62. DAC fields

Table 236. DAC fields

Name	Offset	Description	Size in bytes
VERSION	000h	Identifies the Debug Authentication protocol version. Version is determined by ENF_CNSA field (CMPA) • 00020000h for v2.0, which uses ECDSA P-256 (ENF_CNSA = 00) • 00020001h for v2.1, which uses ECDSA P-384 (ENF_CNSA = 01, 10, 11)	4
SOCC	004h	SoC class specifier This field is always set to 0000_0004h.	4

Table continues on the next page...

Table 236. DAC fields (continued)

Name	Offset	Description	Size in bytes
UUID	008h	Unique device identifier If UUID matching is enabled in DCFG_CC_SoCU, then the device includes its UUID in the challenge packet. Otherwise, this field is set to zeroes.	16
ROTID	018h	Metadata used to authenticate Certificate Authority key (RoT key) for signing DC certificate. SHA256 or SHA384 hashes of four RoT public keys are set in this field. See Root of Trust Identifier (DCFG_ROTID) for details. **NOTE** On this device, the same RoT keys are used for certifying image signing keys and debug keys. 32-byte or 48-byte RKTH value. 4 bytes containing revocation flags. Only least significant 4 bits are used.	36 or 52
CC_SOCU and CC_VU	3Ch or 4Ch	Credential Constraints Specifies the debug access rights for this certificate holder. A 32-bit SOCU_PIN value. See the CC_LIST_Table in Credential Constraints (DCFG_CC_SOCU) for bit encoding of this field. A 32-bit SOCU_DFLT value. See the CC_LIST_Table in Credential Constraints (DCFG_CC_SOCU) for bit encoding of this field. A 32-bit DCFG_VENDOR_USAGE value.	12
CV	48h or 58h	Challenge Vector generated by the device. Device provides a random 32-byte value generated using the TRNG block.	32

10.3.9.2 Debug Authentication Response (DAR)

Before the debugger can formulate a response to the challenge, it should perform some checks to confirm the correctness of VER, SoCC, UUID, RoTID, and CC. It should find a matching DC.

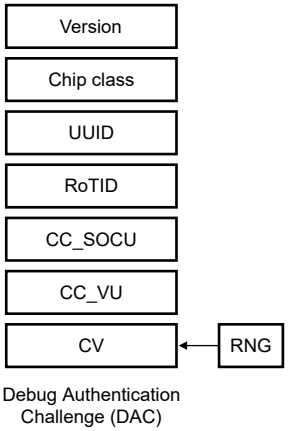


Figure 63. DAR fields

Table 237. DAR fields

Name	Description	Size in bytes
DC	Provides the credential, RoT public key, and more for the debugger as described in Debug Credential Certificate (DC).	variable
AB	The Authentication Beacon provided and signed by the debugger during the authentication process. See the Credential Beacon (CB) field in Debug Credential Certificate (DC) structure for details.	4
SIG	A cryptographic signature by the debugger that binds the previous two fields with the challenge vector from the DAC. SIG = ECDSA_SIGN (DCKpriv,SHA(DAR::DC II DAR::AB II DAR::CV)) • Uses the private key corresponding to the public key (DCK) of the selected DC, proving that debugger has possession of debugger private key. • Depending on which Debug Authentication protocol is used, ECDSA secp256r1 or secp384r1 is used for SIG function. SHA is either SHA256 or SHA384 based on the ECDSA curve type.	64 or 96

10.3.9.3 Device processing the DAR

The device Boot ROM processes the DAR received from the debugger. As part of the validation process, the device will:

- Verify the DC by validating the DC version, SoCC, UUID, RoT, VU, and DC signature.
- Verify that the DAR has a valid signature that binds it to the CV from the DAC.

If valid, it means that:

- The debugger possesses the private key corresponding to the vendor or RoT-signed credential.
- The credential satisfies the constraints specified in the device configuration.
- The response of the debugger to the challenge is produced and signed in response to the challenge (because of its cryptographic dependency on the challenge vector). The response is not replayed from a previous authentication where a different challenge vector is used.

After completion of processing the DAR:

- If authentication succeeds, debug access is granted.

- If authentication fails, no special response is issued. Further debug access requests are ignored and device enters a failure loop (an infinite loop waiting for debug attach).

10.3.9.3.1 Successful authentication

The ROM executes the following steps after successful debug authentication:

1. The ROM determines the final enable states of the debug ports based on pinned state from DCFG_CC_SOCU and the DC::CC fields.
2. The ROM evaluates part enables using the following logic:
   - Uses pinned states based on CC_SOCU_PIN. (CMPA) and DCFG_CC_SOCU_NS_PIN (CFPA).
   - Evaluates socu_pinned and socu_default.
   - Evaluates debug port enables for ports which are not pinned using authentication protocol.
   - Debug_State = (SOCU_PIN & SOCU_DFLT) | (!SOCU_PIN & DC:: CC_SOCU)
   - Enables debug ports for bits set in the above evaluation.
3. The Debug Mailbox handler allows the following commands only if the enable bit is set in the final evaluation of DCFG_CC_SOCU.
   - ENTER_ISP_MODE command, only if default ISP_CMD_EN bit is set in DCFG_CC_SOCU.
   - SET_FA_MODE and Mass Erase commands, only if default FA_CMD_EN bit is set in DCFG_CC_SOCU.
4. The ROM stores the beacons in the write-lockable register DBG_AUTH_SCRATCH.
   - DBG_AUTH_SCRATCH [15:0] = DAR::DC::CB[15:0]
   - DBG_AUTH_SCRATCH [31:16] = DAR::AB[15:0]
5. On receiving an EXIT_DBG_MB command, the ROM exits the debug mailbox handler loop and continues normal boot flow.

10.3.9.4 Debug authentication use cases

This section describes use cases for debug authentication.

10.3.9.4.1 Return Material Analysis (RMA) use case

The diagram shows the RMA flow using debug authentication, where a debug credential certificate is issued for each field technician.

1. Vendor generates RoT key pairs and programs the device with a hash of RoT public key hashes before shipping.
2. Field technician generates own key pair and provides public key to vendor for authorization.
3. Vendor attests public key for field technician. In the debug credential certificate, vendor assigns access rights.
4. End customer with a locked product takes it to field technician.
5. Field technician uses credentials to authenticate with device and unlocks the product for debugging.

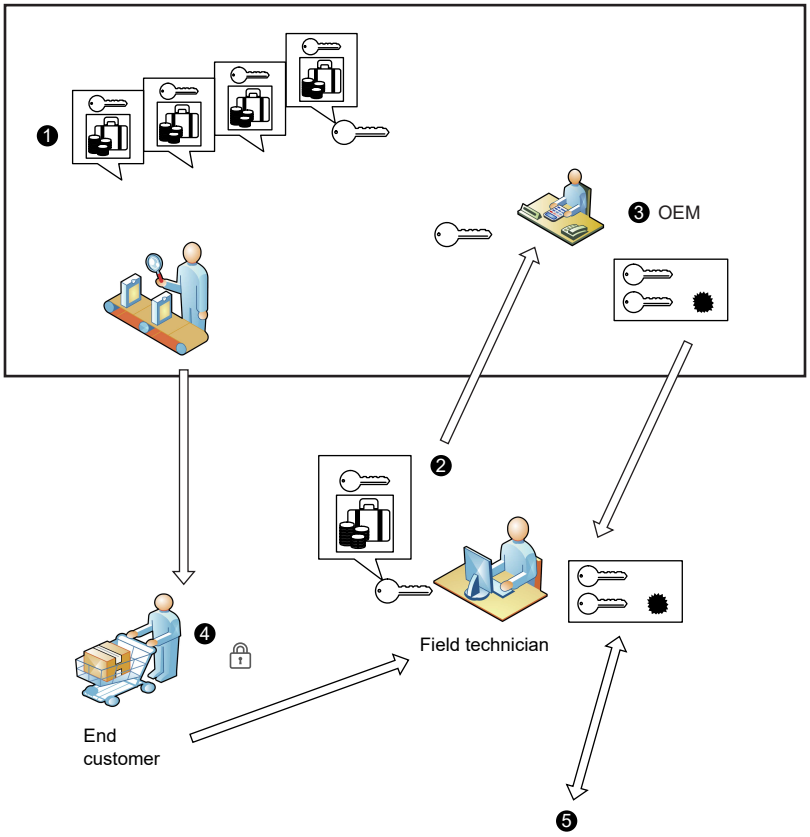


Figure 64. Debug Authentication protocol usage example

10.3.9.4.2 Module use case with Develop and Develop 2 Lifecycle states

CC_SOCU_PIN_NS and CC_SOCU_DFLT_NS allow module makers and OEMs to implement a tiered protection approach.

- The module maker, a tier-1 developer, develops secure code and define access rights to the module using CC_SOCU_PIN and CC_SOCU_DFLT.
- The code is configured to block access to the secure module but allow access to non-secure debug.
- Once the module is ready, the tier-1 developer releases the module to a tier-2 developer (for example, an OEM). The developer blocks debug access to secure mode and enables debug access to non-secure mode.
- A tier-2 developer develops a non-secure module and extends access rights configuration to that module using CC_SOCU_DFLT_NS and CC_SOCU_PIN_NS.

10.3.10 Fault Analysis (FA) mode

The ROM on this chip has an FA mode command (SET_FA_MODE) to enable the deletion of sensitive information (such as keys) before giving the device to NXP for fault analysis. The ROM allows the SET_FA_MODE command only when a corresponding flag in Debug_State is set.

When activated by the boot ROM, the FA_MODE sequence is:

1. Lifecycle fuses are advanced to "Field Return OEM" state.
2. Erase all KEYS and IVs in the KEYSTORE Flash page.
3. Flush all temporary key registers.
4. Block PUF indexes.

5. Open all debug ports.
6. Enter while(1) loop.

10.4 External signals

Table 238 shows the signals related to the debug process. A trace using the serial wire output has limited bandwidth.

Table 238. Serial wire debug signals

Signal	I/O	Description
SWCLK	I	Serial wire clock. Provides the clock for the SWD debug logic in Serial Wire Debug (SWD) mode. SWCLK is the default signal for its pin. At the release of reset, the pin is pulled down internally.
SWDIO	I/O	Serial wire debug data input and output. Used by an external debug tool to communicate with and control the part. SWDIO is the default signal of its pin. At the release of reset, the pin is pulled up internally.
SWO	O	Serial wire output. Optionally provides data from the Instrumentation Trace Macrocell (ITM) for an external debug tool to evaluate. See the chip-specific DBGMB information for the clocking required to enable SWO.

10.5 Memory map and register definition

10.5.1 DBGMB register descriptions

10.5.1.1 DBGMB memory map

DM0 base address: 400B_D000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Command and Status Word (CSW)	32	RW	0000_0000h
4h	Request Value (REQUEST)	32	RW	0000_0000h
8h	Return Value (RETURN)	32	RW	0000_0000h
FCh	Identification (ID)	32	R	002A_0000h

10.5.1.2 Command and Status Word (CSW)

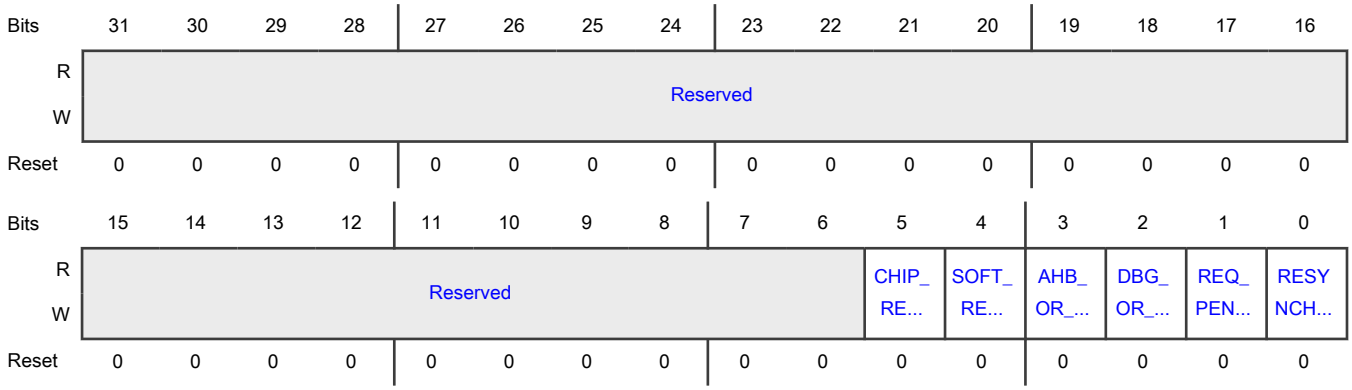
Offset

Register	Offset
CSW	0h

Function

Contains command and status bits to facilitate communication between DBGMB and the chip.

Diagram



Fields

Field	Function
31-6 —	Reserved
5 CHIP_RESET_REQ	Chip Reset Request Causes the chip (but not the DM-AP) to be reset by generating SYSRESET_REQ. This field is write-only. 0b - No effect 1b - Reset
4 SOFT_RESET	Soft Reset Resets the DM-AP. This field is write-only by the chip. 0b - No effect 1b - Reset
3 AHB_OR_ERR	AHB Overrun Error Indicates whether an AHB overrun has occurred: the chip has overwritten a RETURN value before DBGMB read the RETURN value. 0b - No overrun 1b - Overrun occurred
2 DBG_OR_ERR	DBGMB Overrun Error Indicates whether a DBGMB overrun has occurred: DBGMB has overwritten a REQUEST value before the chip read the REQUEST value. 0b - No overrun 1b - Overrun occurred
1 REQ_PENDING	Request Pending Indicates a pending request for DBGMB: a value is waiting to be read from Request Value (REQUEST).

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - No request pending 1b - Request for resynchronization pending
0 RESYNCH_REQ	Resynchronization Request Requests a resynchronization. 0b - No request 1b - Request for resynchronization

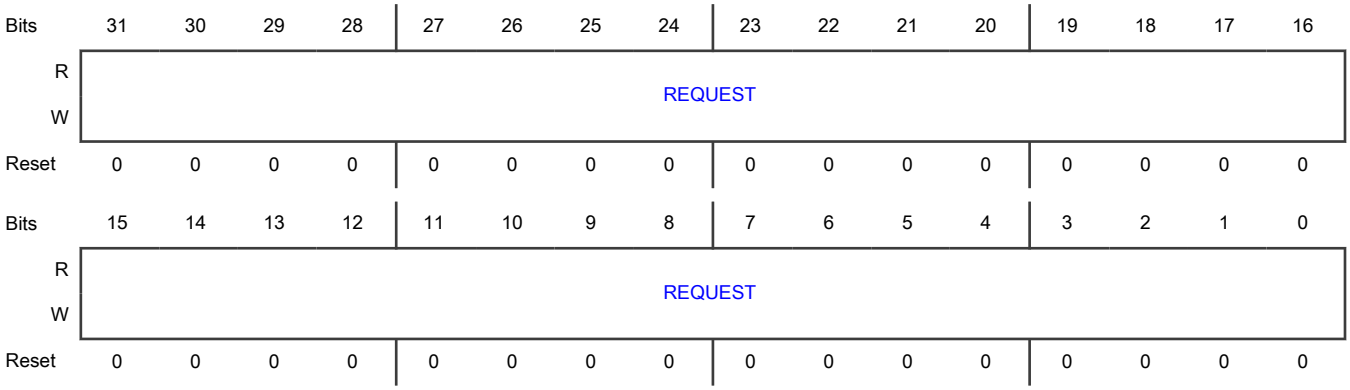
10.5.1.3 Request Value (REQUEST)

Offset

Register	Offset
REQUEST	4h

Function
Used by DBGMB to send action requests to the chip.

Diagram



Fields

Field	Function
31-0 REQUEST	Request Value Indicates the request value. This field reads 0 when no new request is present. The chip clears this field. DBGMB can read back this field to confirm communication.

10.5.1.4 Return Value (RETURN)

Offset

Register	Offset
RETURN	8h

Function
Provides the responses from the chip to DBGMB.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	RET															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	RET															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 RET	Return Value Indicates the return value, which is a response from the chip to DBGMB. If no new data is present, DBGMB reading is stalled until new data is available.

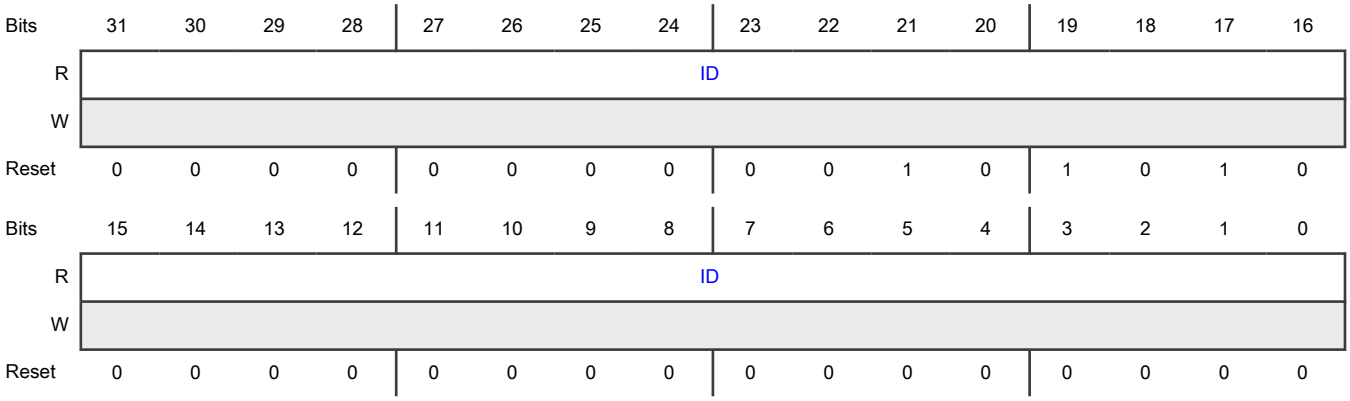
10.5.1.5 Identification (ID)

Offset

Register	Offset
ID	FCh

Function
Provides an identification of the DM-AP interface.

Diagram



Fields

Field	Function
31-0 ID	Identification Value Provides an identification of the DM-AP interface. The Arm Debug Interface Specification version 5.0 (ADIv5) requires every AP to implement an AP identification register, at offset FCh. This register is the last register in the AP register space. This ID register is only readable by external tools (a debug dongle) when identifying the Access Port (AP). For the debug mailbox AP, the identification value is 002A_0000h. This register is not readable from on-chip software. If you attempt a read, you receive a value of 0000_0000h.

Chapter 11
System Controller (SYSCON)

11.1 Chip-specific SYSCON information

Table 239. Reference links to related information[1]

Topic	Related module	Reference
Full description	SYSCON	SYSCON
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

NOTE

The reset values of some registers may change depending on the device's boot settings.

11.1.1 Module instances

This device has one instance of SYSCON module, SYSCON0.

11.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

11.1.3 Configuration

Configure the SYSCON block as follows:

- No clock configuration is needed. The clock to the SYSCON block is always enabled.
- The SYSCON block controls use of the CLKOUT pin, which must also be configured through Port control module.

11.2 Overview

The SYSCON module provides controls and configurations on the system and peripherals for the multiple functions.

11.2.1 Features

The SYSCON module supports the following features and configurations:

- System and bus configuration
  - AHB matrix priority
  - CPUs control and status
  - CPU and DSP debug access
  - CPU LPCAC control
  - NMI source select
  - Calibrate system tick timer

- Boot control
- System/Peripherals - clock select and control
  - Allows enabling and selection of clocks to individual peripherals and memories
  - Allows configuration of clock dividers
- Memory configuration
  - ROM access
  - RAM ECC
  - FLASH cache, bus error
- Gray-2-Binary converter
  - Allows decoding gray value coming from OS Event Timer
- Peripherals configuration
  - SmartDMA interrupt hijack
  - PWM0/1 control
  - Ethernet PHY selection and flow control
  - CTIMER global start
- Security configuration
  - Security control - ELS, GDET0/1
  - Security boot
  - Security attestation
- Reset control
  - Monitors and release resets to individual peripherals
- Device ID register

11.3 Signals

Table 240. SYSCON pin description

Function	Type	Pin	Description
CLKOUT	O	PIO2_2, PIO3_6	CLKOUT clock output. Refer to the Pinouts information.

11.4 Memory map and register definition

This section includes the SYSCON module memory map and detailed descriptions of all registers.

11.4.1 SYSCON register descriptions

11.4.1.1 SYSCON memory map

SYSCON0 base address: 4000_0000h

Offset	Register	Width (In bits)	Access	Reset value
10h	AHB Matrix Priority Control (AHBMATPRIO)	32	RW	See section
38h	Secure CPU0 System Tick Calibration (CPU0STCKCAL)	32	RW	See section
3Ch	Non-Secure CPU0 System Tick Calibration (CPU0NSTCKCAL)	32	RW	See section
40h	System tick calibration for CPU1 (CPU1STCKCAL)	32	RW	See section
48h	NMI Source Select (NMISRC)	32	RW	See section
100h	Peripheral Reset Control 0 (PRESETCTRL0)	32	RW	See section
104h	Peripheral Reset Control 1 (PRESETCTRL1)	32	RW	See section
108h	Peripheral Reset Control 2 (PRESETCTRL2)	32	RW	0000_0000h
10Ch	Peripheral Reset Control 3 (PRESETCTRL3)	32	RW	0000_0000h
120h - 12Ch	Peripheral Reset Control Set (PRESETCTRLSET0 - PRESETCTRLSET3)	32	W	0000_0000h
140h - 14Ch	Peripheral Reset Control Clear (PRESETCTRLCLR0 - PRESETCTRLCLR3)	32	W	0000_0000h
200h	AHB Clock Control 0 (AHBCLKCTRL0)	32	RW	0000_0603h
204h	AHB Clock Control 1 (AHBCLKCTRL1)	32	RW	0000_0000h
208h	AHB Clock Control 2 (AHBCLKCTRL2)	32	RW	0000_0000h
20Ch	AHB Clock Control 3 (AHBCLKCTRL3)	32	RW	0000_0000h
220h - 22Ch	AHB Clock Control Set (AHBCLKCTRLSET0 - AHBCLKCTRLSET3)	32	W	0000_0000h
240h - 24Ch	AHB Clock Control Clear (AHBCLKCTRLCLR0 - AHBCLKCTRLCLR3)	32	W	0000_0000h
260h	CPU0 System Tick Timer Source Select (SYSTICKCLKSEL0)	32	RW	0000_0007h
264h	CPU1 System Tick Timer Source Select (SYSTICKCLKSEL1)	32	RW	See section
268h	Trace Clock Source Select (TRACECLKSEL)	32	RW	See section
26Ch - 27Ch	CTIMER Clock Source Select (CTIMERCLKSEL0 - CTIMERCLKSEL4)	32	RW	See section
288h	CLKOUT Clock Source Select (CLKOUTSEL)	32	RW	See section
2A4h	ADC0 Clock Source Select (ADC0CLKSEL)	32	RW	See section
2A8h	USB-FS Clock Source Select (USB0CLKSEL)	32	RW	See section
2B0h - 2D4h	LP_FLEXCOMM Clock Source Select for Fractional Rate Divider (FCCLKSEL0 - FCCLKSEL9)	32	RW	See section
2F0h	SCTimer/PWM Clock Source Select (SCTCLKSEL)	32	RW	See section
300h	CPU0 System Tick Timer Divider (SYSTICKCLKDIV0)	32	RW	4000_0000h
304h	CPU1 System Tick Timer Divider (SYSTICKCLKDIV1)	32	RW	4000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
308h	TRACE Clock Divider (TRACECLKDIV)	32	RW	4000_0000h
350h	TSI Function Clock Source Select (TSICLKSEL)	32	RW	See section
360h	SINC FILTER Function Clock Source Select (SINCFILTCLKSEL)	32	RW	See section
378h	SLOW_CLK Clock Divider (SLOWCLKDIV)	32	RW	See section
37Ch	TSI Function Clock Divider (TSICLKDIV)	32	RW	4000_0000h
380h	System Clock Divider (AHBCLKDIV)	32	RW	0000_0000h
384h	CLKOUT Clock Divider (CLKOUTDIV)	32	RW	4000_0000h
388h	FRO_HF_DIV Clock Divider (FROHFDIV)	32	RW	4000_0000h
38Ch	WDT0 Clock Divider (WDT0CLKDIV)	32	RW	See section
394h	ADC0 Clock Divider (ADC0CLKDIV)	32	RW	See section
398h	USB-FS Clock Divider (USB0CLKDIV)	32	RW	4000_0000h
3B4h	SCT/PWM Clock Divider (SCTCLKDIV)	32	RW	4000_0000h
3C4h	PLL Clock Divider (PLLCLKDIV)	32	RW	4000_0000h
3D0h - 3E0h	CTimer Clock Divider (CTIMER0CLKDIV - CTIMER4CLKDIV)	32	RW	4000_0000h
3E4h	PLL1 Clock 0 Divider (PLL1CLK0DIV)	32	RW	4000_0000h
3E8h	PLL1 Clock 1 Divider (PLL1CLK1DIV)	32	RW	4000_0000h
3FCh	Clock Configuration Unlock (CLKUNLOCK)	32	RW	0000_0000h
400h	NVM Control (NVM_CTRL)	32	RW	0002_0410h
404h	ROM Wait State (ROMCR)	32	RW	0000_0000h
414h	SmartDMA Interrupt Hijack (SmartDMAINT)	32	RW	See section
464h	ADC1 Clock Source Select (ADC1CLKSEL)	32	RW	See section
468h	ADC1 Clock Divider (ADC1CLKDIV)	32	RW	See section
470h	Control PKC RAM Interleave Access (RAM_INTERLEAVE)	32	RW	0000_0000h
490h	DAC0 Functional Clock Selection (DAC0CLKSEL)	32	RW	0000_0007h
494h	DAC0 functional clock divider (DAC0CLKDIV)	32	RW	4000_0000h
498h	DAC1 Functional Clock Selection (DAC1CLKSEL)	32	RW	0000_0007h
49Ch	DAC1 functional clock divider (DAC1CLKDIV)	32	RW	4000_0000h
4A0h	DAC2 Functional Clock Selection (DAC2CLKSEL)	32	RW	0000_0007h
4A4h	DAC2 functional clock divider (DAC2CLKDIV)	32	RW	4000_0000h
4A8h	FlexSPI Clock Selection (FlexSPICLKSEL)	32	RW	0000_000Fh
4ACh	FlexSPI Clock Divider (FlexSPICLKDIV)	32	RW	4000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
52Ch	PLL Clock Divider Clock Selection (PLLCLKDIVSEL)	32	RW	0000_0007h
530h	I3C0 Functional Clock Selection (I3C0FCLKSEL)	32	RW	0000_0007h
534h	I3C0 FCLK_STC Clock Selection (I3C0FCLKSTCSEL)	32	RW	0000_0007h
538h	I3C0 FCLK_STC Clock Divider (I3C0FCLKSTCDIV)	32	RW	4000_0000h
53Ch	I3C0 FCLK Slow Clock Divider (I3C0FCLKSDIV)	32	RW	4000_0000h
540h	I3C0 Functional Clock FCLK Divider (I3C0FCLKDIV)	32	RW	4000_0000h
544h	I3C0 FCLK Slow Selection (I3C0FCLKSSEL)	32	RW	0000_0000h
548h	MICFIL Clock Selection (MICFILFCLKSEL)	32	RW	0000_000Fh
54Ch	MICFIL Clock Division (MICFILFCLKDIV)	32	RW	4000_0000h
558h	uSDHC Clock Selection (uSDHCCLKSEL)	32	RW	0000_0007h
55Ch	uSDHC Function Clock Divider (uSDHCCLKDIV)	32	RW	4000_0000h
560h	FLEXIO Clock Selection (FLEXIOCLKSEL)	32	RW	0000_0007h
564h	FLEXIO Function Clock Divider (FLEXIOCLKDIV)	32	RW	4000_0000h
5A0h	FLEXCAN0 Clock Selection (FLEXCAN0CLKSEL)	32	RW	0000_0007h
5A4h	FLEXCAN0 Function Clock Divider (FLEXCAN0CLKDIV)	32	RW	0000_0000h
5A8h	FLEXCAN1 Clock Selection (FLEXCAN1CLKSEL)	32	RW	0000_0007h
5ACh	FLEXCAN1 Function Clock Divider (FLEXCAN1CLKDIV)	32	RW	4000_0000h
5B0h	Ethernet RMII Clock Selection (ENETRMIICLKSEL)	32	RW	0000_0007h
5B4h	Ethernet RMII Function Clock Divider (ENETRMIICLKDIV)	32	RW	4000_0000h
5B8h	Ethernet PTP REF Clock Selection (ENETPTPREFCLKSEL)	32	RW	0000_0007h
5BCh	Ethernet PTP REF Function Clock Divider (ENETPTPREFCLKDIV)	32	RW	4000_0000h
5C0h	Ethernet PHY Interface Select (ENET_PHY_INTF_SEL)	32	RW	0000_0000h
5C4h	Sideband Flow Control (ENET_SBD_FLOW_CTRL)	32	RW	0000_0000h
5D4h	EWM0 Clock Selection (EWM0CLKSEL)	32	RW	0000_0001h
5D8h	WDT1 Clock Selection (WDT1CLKSEL)	32	RW	0000_0003h
5DCh	WDT1 Function Clock Divider (WDT1CLKDIV)	32	RW	4000_0000h
5E0h	OSTIMER Clock Selection (OSTIMERCLKSEL)	32	RW	0000_0003h
5F0h	CMP0 Function Clock Selection (CMP0FCLKSEL)	32	RW	0000_0007h
5F4h	CMP0 Function Clock Divider (CMP0FCLKDIV)	32	RW	4000_0000h
5F8h	CMP0 Round Robin Clock Selection (CMP0RRCLKSEL)	32	RW	0000_0007h
5FCh	CMP0 Round Robin Clock Divider (CMP0RRCLKDIV)	32	RW	4000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
600h	CMP1 Function Clock Selection (CMP1FCLKSEL)	32	RW	0000_0007h
604h	CMP1 Function Clock Divider (CMP1FCLKDIV)	32	RW	4000_0000h
608h	CMP1 Round Robin Clock Source Select (CMP1RRCLKSEL)	32	RW	0000_0007h
60Ch	CMP1 Round Robin Clock Division (CMP1RRCLKDIV)	32	RW	4000_0000h
610h	CMP2 Function Clock Source Select (CMP2FCLKSEL)	32	RW	0000_0007h
614h	CMP2 Function Clock Division (CMP2FCLKDIV)	32	RW	4000_0000h
618h	CMP2 Round Robin Clock Source Select (CMP2RRCLKSEL)	32	RW	0000_0007h
61Ch	CMP2 Round Robin Clock Division (CMP2RRCLKDIV)	32	RW	4000_0000h
800h	CPU Control for Multiple Processors (CPUCTRL)	32	RW	0000_0028h
804h	Coprocessor Boot Address (CPBOOT)	32	RW	0000_0000h
80Ch	CPU Status (CPUSTAT)	32	R	See section
824h	LPCAC Control (LPCAC_CTRL)	32	RW	0000_0031h
850h - 874h	LP_FLEXCOMM Clock Divider (FLEXCOMM0CLKDIV - FLEXCOMM9CLKDIV)	32	RW	4000_0000h
880h	SAI0 Function Clock Source Select (SAI0CLKSEL)	32	RW	0000_0007h
884h	SAI1 Function Clock Source Select (SAI1CLKSEL)	32	RW	0000_0007h
888h	SAI0 Function Clock Division (SAI0CLKDIV)	32	RW	4000_0000h
88Ch	SAI1 Function Clock Division (SAI1CLKDIV)	32	RW	4000_0000h
890h	EMVSIM0 Clock Source Select (EMVSIM0CLKSEL)	32	RW	0000_0007h
894h	EMVSIM1 Clock Source Select (EMVSIM1CLKSEL)	32	RW	0000_0007h
898h	EMVSIM0 Function Clock Division (EMVSIM0CLKDIV)	32	RW	4000_0000h
89Ch	EMVSIM1 Function Clock Division (EMVSIM1CLKDIV)	32	RW	4000_0000h
950h	Key Retain Control (KEY_RETAIN_CTRL)	32	RW	0000_0000h
960h	FRO 48MHz Reference Clock Control (REF_CLK_CTRL)	32	RW	0000_0000h
964h	FRO 48MHz Reference Clock Control Set (REF_CLK_CTRL_SET)	32	RW	0000_0000h
968h	FRO 48MHz Reference Clock Control Clear (REF_CLK_CTRL_CLR)	32	RW	0000_0000h
96Ch - 970h	GDET Control Register (GDET0_CTRL - GDET1_CTRL)	32	RW	0000_0000h
974h	ELS Asset Protection Register (ELS_ASSET_PROT)	32	RW	0000_0002h
978h	ELS Lock Control (ELS_LOCK_CTRL)	32	RW	0000_0001h
97Ch	ELS Lock Control DP (ELS_LOCK_CTRL_DP)	32	RW	0000_0002h
980h	Life Cycle State Register (ELS_OTP_LC_STATE)	32	R	0000_0055h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
984h	Life Cycle State Register (Duplicate) (ELS_OTP_LC_STATE_DP)	32	R	0000_0055h
988h	ELS Temporal State (ELS_TEMPORAL_STATE)	32	RW	0000_0000h
98Ch	Key Derivation Function Mask (ELS_KDF_MASK)	32	RW	0000_0000h
9D0h	ELS AS Configuration (ELS_AS_CFG0)	32	R	000C_0255h
9D4h	ELS AS Configuration1 (ELS_AS_CFG1)	32	R	0014_0002h
9D8h	ELS AS Configuration2 (ELS_AS_CFG2)	32	R	FFFF_FFFFh
9DCh	ELS AS Configuration3 (ELS_AS_CFG3)	32	R	See section
9E0h	ELS AS State Register (ELS_AS_ST0)	32	R	0000_0000h
9E4h	ELS AS State1 (ELS_AS_ST1)	32	R	0005_024Fh
9E8h	Boot state captured during boot: Main ROM log (ELS_AS_BOOT_LOG0)	32	R	0000_0000h
9ECh	Boot state captured during boot: Library log (ELS_AS_BOOT_LOG1)	32	R	0000_0000h
9F0h	Boot state captured during boot: Hardware status signals log (ELS_AS_BOOT_LOG2)	32	R	0000_0000h
9F4h	Boot state captured during boot: Security log (ELS_AS_BOOT_LOG3)	32	R	0000_0000h
9F8h	ELS AS Flag0 (ELS_AS_FLAG0)	32	R	0000_0000h
9FCh	ELS AS Flag1 (ELS_AS_FLAG1)	32	R	0000_0000h
A18h	Clock Control (CLOCK_CTRL)	32	RW	See section
B30h	I3C1 Functional Clock Selection (I3C1FCLKSEL)	32	RW	0000_0007h
B34h	Selects the I3C1 Time Control clock (I3C1FCLKSTCSEL)	32	RW	0000_0007h
B38h	I3C1 FCLK_STC Clock Divider (I3C1FCLKSTCDIV)	32	RW	4000_0000h
B3Ch	I3C1 FCLK Slow clock Divider (I3C1FCLKSDIV)	32	RW	4000_0000h
B40h	I3C1 Functional Clock FCLK Divider (I3C1FCLKDIV)	32	RW	4000_0000h
B44h	I3C1 FCLK Slow Selection (I3C1FCLKSSEL)	32	RW	0000_0000h
B50h	ETB Counter Status Register (ETB_STATUS)	32	RW	0000_0000h
B54h	ETB Counter Control Register (ETB_COUNTER_CTRL)	32	RW	0000_0000h
B58h	ETB Counter Reload Register (ETB_COUNTER_RELOAD)	32	RW	0000_0000h
B5Ch	ETB Counter Value Register (ETB_COUNTER_VALUE)	32	R	0000_0000h
B60h	Gray to Binary Converter Gray code_gray[31:0] (GRAY_CODE_LSB)	32	RW	0000_0000h
B64h	Gray to Binary Converter Gray code_gray[41:32] (GRAY_CODE_MSB)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
B68h	Gray to Binary Converter Binary Code [31:0] (BINARY_CODE_LSB)	32	R	0000_0000h
B6Ch	Gray to Binary Converter Binary Code [41:32] (BINARY_CODE_MSB)	32	R	0000_0000h
E04h	Control Automatic Clock Gating (AUTOCLKGATEOVERRIDE)	32	RW	0000_FFFFh
E2Ch	Control Automatic Clock Gating C (AUTOCLKGATEOVERRIDEC)	32	RW	0000_0000h
E38h	PWM0 Submodule Control (PWM0SUBCTL)	32	RW	0000_0000h
E3Ch	PWM1 Submodule Control (PWM1SUBCTL)	32	RW	0000_0000h
E40h	CTIMER Global Start Enable (CTIMERGLOBALSTARTEN)	32	RW	0000_0000h
E44h	RAM ECC Enable Control (ECC_ENABLE_CTRL)	32	RW	0000_0003h
FA0h	Control Write Access to Security (DEBUG_LOCK_EN)	32	RW	See section
FA4h	Cortex Debug Features Control (DEBUG_FEATURES)	32	RW	See section
FA8h	Cortex Debug Features Control (Duplicate) (DEBUG_FEATURES_DP)	32	RW	See section
FB4h	CPU0 Software Debug Access (SWD_ACCESS_CPU0)	32	RW	0000_0000h
FB8h	CPU1 Software Debug Access (SWD_ACCESS_CPU1)	32	W	0000_0000h
FC0h	Debug Authentication BEACON (DEBUG_AUTH_BEACON)	32	RW	0000_0000h
FC4h	DSP Software Debug Access (SWD_ACCESS_DSP)	32	RW	0000_0000h
FF0h	JTAG Chip ID (JTAG_ID)	32	R	See section
FF4h	Device Type (DEVICE_TYPE)	32	R	See section
FF8h	Device ID (DEVICE_ID0)	32	R	See section
FFCh	Chip Revision ID and Number (DIEID)	32	R	See section

11.4.1.2 AHB Matrix Priority Control (AHBMATPRIO)

Offset

Register	Offset
AHBMATPRIO	10h

Function

The Multilayer AHB Matrix arbitrates between masters, when they attempt to access the same matrix slave port at the same time. The priority values are 3 = highest, 0 = lowest. When the priority is the same, the master with the lower master number is given priority.

NOTE

Be careful when modifying this register as improper settings can seriously degrade the performance.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved		PRI_USDHC		PRI_USB_HS		PRI_USB_FS_ENET		PRI_NPU_D		PRI_COOLFLUX_Y_...		PRI_COOLFLUX_X		PRI_COOLFLUX_I	
Reset	u	u	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PRI_NPU_PQ		PRI_PKC_ELS		DMA1		DMA0		PRI_CPU1_CBUS_S...		PRI_CPU1_SBUS_S...		PRI_CPU0_SBUS		PRI_CPU0_CBUS	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved
29-28 PRI_USDHC	USDHC bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
27-26 PRI_USB_HS	USB-HS bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
25-24 PRI_USB_FS_ENET	USB-FS and ENET bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
23-22 PRI_NPU_D	NPU D bus master priority level 00b - level 0 01b - level 1

Table continues on the next page...

Table continued from the previous page...

Field	Function
	10b - level 2 11b - level 3
21-20 PRI_COOLFLUX_Y_ESPI	CoolFlux Y bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
19-18 PRI_COOLFLUX_X	CoolFlux X bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
17-16 PRI_COOLFLUX_I	CoolFlux I bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
15-14 PRI_NPU_PQ	NPU O bus and Powerquad bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
13-12 PRI_PKC_ELS	PKC and ELS bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
11-10 DMA1	DMA1 controller bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3

Table continues on the next page...

Table continued from the previous page...

Field	Function
9-8 DMA0	DMA0 controller bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
7-6 PRI_CPU1_CBUS_SmartDMA_I	CPU1 C-AHB/SmartDMA-I bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
5-4 PRI_CPU1_SBUS_SmartDMA_D	CPU1 S-AHB/SmartDMA-D bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
3-2 PRI_CPU0_SBUS	CPU0 S-AHB bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3
1-0 PRI_CPU0_CBUS	CPU0 C-AHB bus master priority level 00b - level 0 01b - level 1 10b - level 2 11b - level 3

11.4.1.3 Secure CPU0 System Tick Calibration (CPU0STCKCAL)

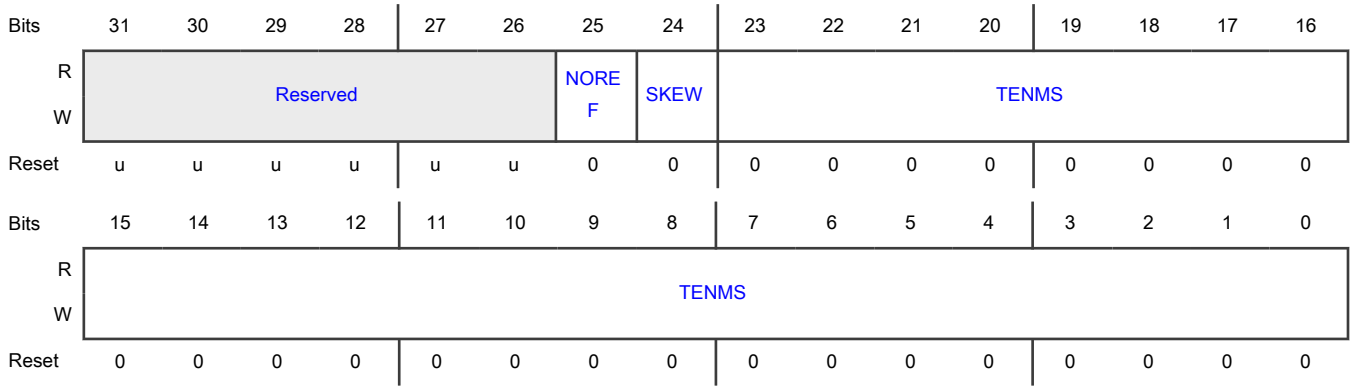
Offset

Register	Offset
CPU0STCKCAL	38h

Function

The CPU0STCKCAL register allows software to set up a default value for the SYST_CALIB register (refer to Arm documentation) in the System Tick Timer of secure part of the CPU0.

Diagram



Fields

Field	Function
31-26 —	Reserved
25 NOREF	Whether the device provides a reference clock to the processor. 0b - Reference clock is provided 1b - No reference clock is provided
24 SKEW	Whether the TENMS value is exact. 0b - TENMS value is exact 1b - TENMS value is not exact or not given
23-0 TENMS	Reload value for 10 ms (100 Hz) timing, subject to system clock skew errors. If the value reads as zero, the calibration value is not known.

11.4.1.4 Non-Secure CPU0 System Tick Calibration (CPU0NSTCKCAL)

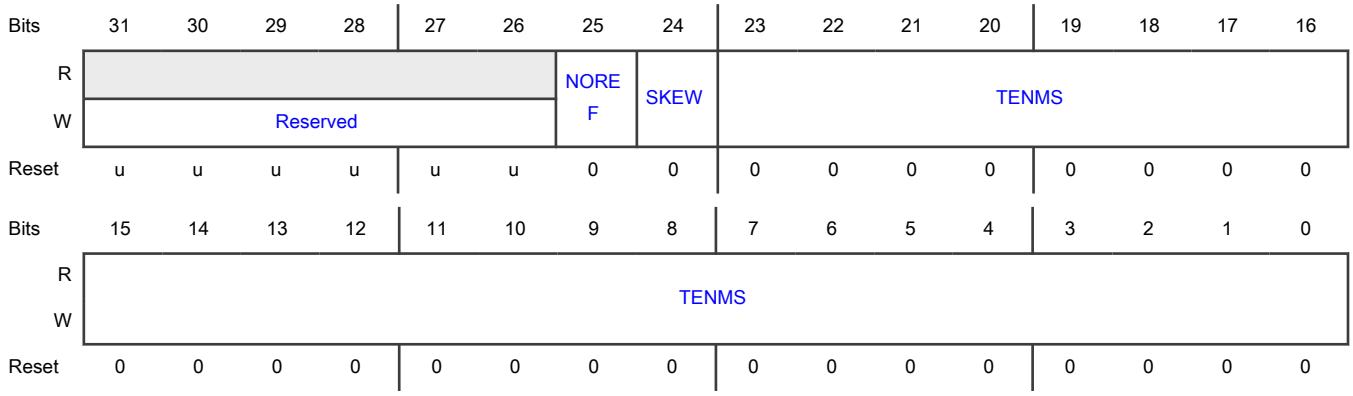
Offset

Register	Offset
CPU0NSTCKCAL	3Ch

Function

The CPU0NSTCKCAL register allows software to set up a default value for the SYST_CALIB register in the System Tick Timer of non-secure part of the CPU0.

Diagram



Fields

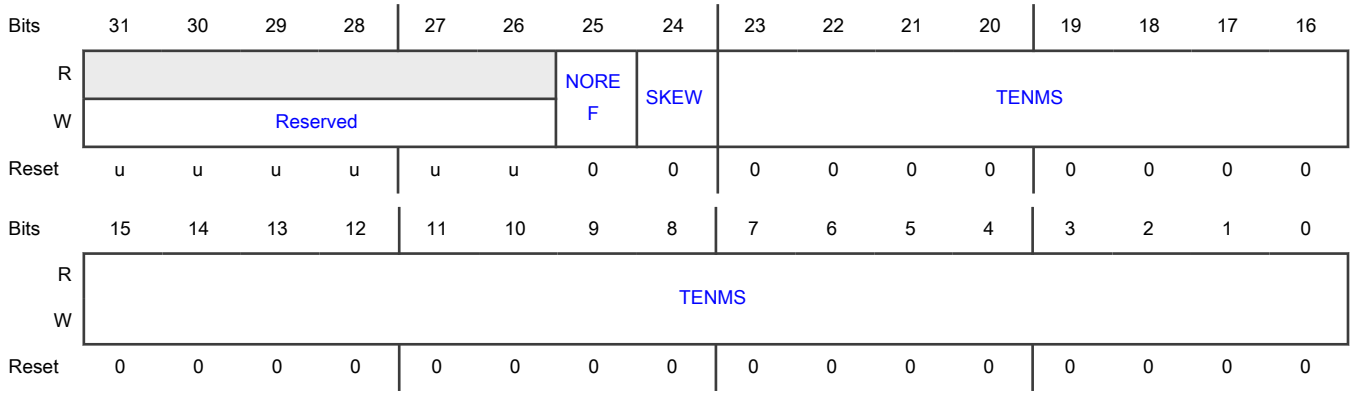
Field	Function
31-26 —	Reserved
25 NOREF	Indicates whether the device provides a reference clock to the processor. 0b - Reference clock is provided 1b - No reference clock is provided
24 SKEW	Indicates whether the TENMS value is exact. 0b - TENMS value is exact 1b - TENMS value is not exact or not given
23-0 TENMS	Reload value for 10 ms (100 Hz) timing, subject to system clock skew errors. If the value reads as zero, the calibration value is not known.

11.4.1.5 System tick calibration for CPU1 (CPU1STCKCAL)

Offset

Register	Offset
CPU1STCKCAL	40h

Diagram



Fields

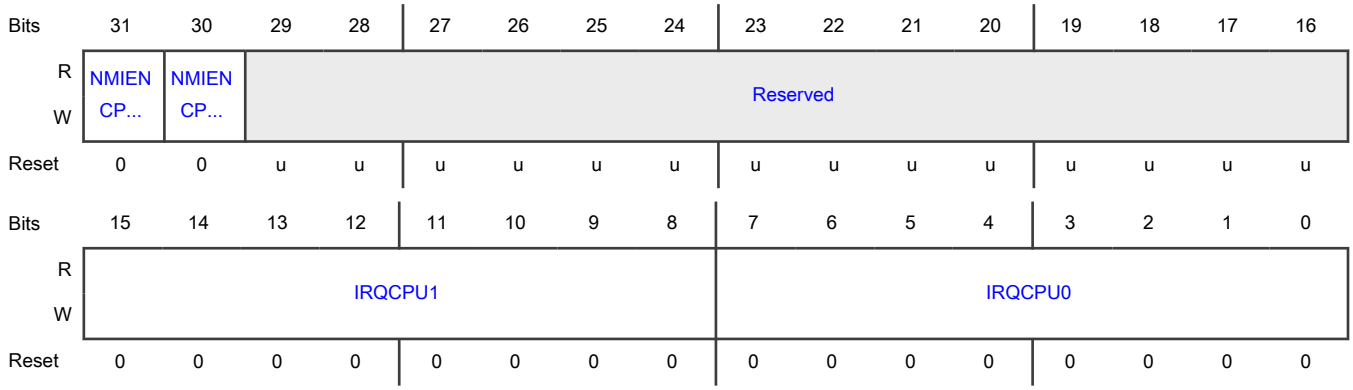
Field	Function
31-26 —	Reserved
25 NOREF	Indicates whether the device provides a reference clock to the processor. 0b - Reference clock is provided 1b - No reference clock is provided
24 SKEW	Indicates whether the TENMS value is exact. 0b - TENMS value is exact 1b - TENMS value is not exact or not given
23-0 TENMS	Reload value for 10 ms (100 Hz) timing, subject to system clock skew errors. If the value reads as zero, the calibration value is not known.

11.4.1.6 NMI Source Select (NMISRC)

Offset

Register	Offset
NMISRC	48h

Diagram



Fields

Field	Function
31 NMIENCPU0	Enables the Non-Maskable Interrupt (NMI) source selected by IRQCPU0. 0b - Disable. 1b - Enable.
30 NMIENCPU1	Enables the Non-Maskable Interrupt (NMI) source selected by IRQCPU1. 0b - Disable. 1b - Enable.
29-16 —	Reserved
15-8 IRQCPU1	The IRQ number of the interrupt that acts as the Non-Maskable Interrupt (NMI) for CPU1, if enabled by NMIENCPU1.
7-0 IRQCPU0	The IRQ number of the interrupt that acts as the Non-Maskable Interrupt (NMI) for CPU0, if enabled by NMIENCPU0.

11.4.1.7 Peripheral Reset Control 0 (PRESETCTRL0)

Offset

Register	Offset
PRESETCTRL0	100h

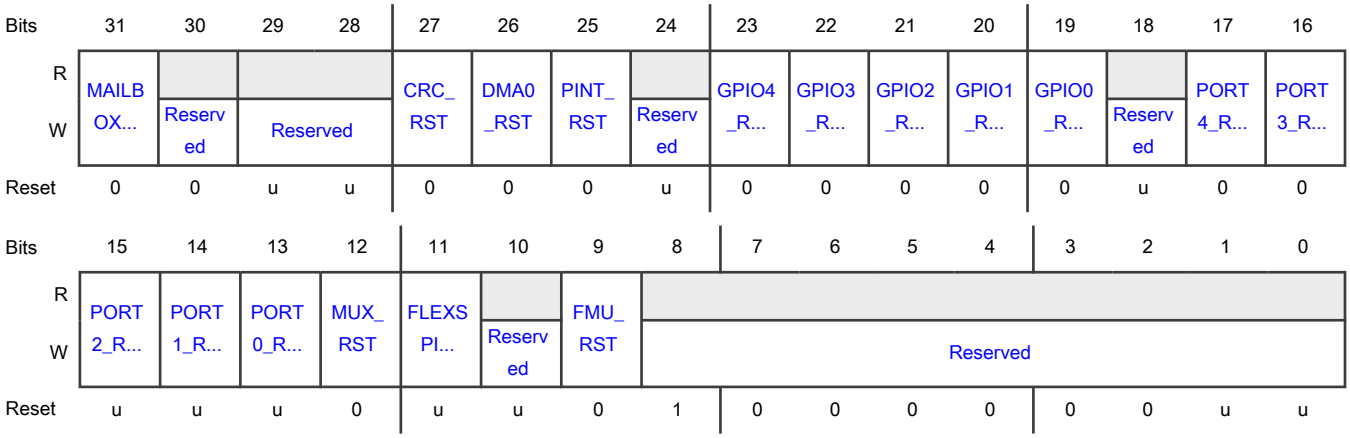
Function

The PRESETCTRL0 register allows software to reset specific peripherals. Writing a 0 to any assigned bit in this register clears the reset and allows the specified peripheral to operate. Writing a 1 asserts the reset.

NOTE

When modifying the PRESETCTRL registers, use the related PRESETCTRLSET and PRESETCTRLCLR registers to avoid setting or clearing bits unintentionally.

Diagram



Fields

Field	Function
31 MAILBOX_RST	Inter-CPU communication Mailbox reset control 0b - Block is not reset 1b - Block is reset
30 —	Reserved Read value is undefined, only zero should be written.
29-28 —	Reserved
27 CRC_RST	CRC reset control 0b - Block is not reset 1b - Block is reset
26 DMA0_RST	DMA0 reset control 0b - Block is not reset 1b - Block is reset
25 PINT_RST	PINT reset control 0b - Block is not reset 1b - Block is reset
24 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
23 GPIO4_RST	GPIO4 reset control 0b - Block is not reset 1b - Block is reset
22 GPIO3_RST	GPIO3 reset control 0b - Block is not reset 1b - Block is reset
21 GPIO2_RST	GPIO2 reset control 0b - Block is not reset 1b - Block is reset
20 GPIO1_RST	GPIO1 reset control 0b - Block is not reset 1b - Block is reset
19 GPIO0_RST	GPIO0 reset control 0b - Block is not reset 1b - Block is reset
18 —	Reserved
17 PORT4_RST	PORT4 reset control 0b - Block is not reset 1b - Block is reset
16 PORT3_RST	PORT3 reset control 0b - Block is not reset 1b - Block is reset
15 PORT2_RST	PORT2 reset control 0b - Block is not reset 1b - Block is reset
14 PORT1_RST	PORT1 reset control 0b - Block is not reset 1b - Block is reset
13 PORT0_RST	PORT0 controller reset control 0b - Block is not reset 1b - Block is reset

Table continues on the next page...

Table continued from the previous page...

Field	Function
12 MUX_RST	INPUTMUX reset control 0b - Block is not reset 1b - Block is reset
11 FLEXSPI_RST	FlexSPI reset control 0b - Block is not reset 1b - Block is reset
10 —	Reserved
9 FMU_RST	Flash management unit reset control 0b - Block is not reset 1b - Block is reset
8-0 —	Reserved

11.4.1.8 Peripheral Reset Control 1 (PRESETCTRL1)

Offset

Register	Offset
PRESETCTRL1	104h

Function

The PRESETCTRL1 register allows software to reset specific peripherals. Writing a 0 to any assigned bit in this register clears the reset and allows the specified peripheral to operate. Writing a 1 asserts the reset.

NOTE

When modifying the PRESETCTRL registers, use the related PRESETCTRLSET and PRESETCTRLCLR registers to avoid setting or clearing bits unintentionally.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Smart DM...				TIMER 1_...	TIMER 0_...	USB0_ FS...	USB0_ FS...	Reserv ed	TIMER 2_...	MICFI L_...	FC9_ RST	FC8_ RST	FC7_ RST	FC6_ RST	FC5_ RST
W		Reserv ed	Reserv ed	Reserv ed												
Reset	0	u	0	u	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FC4_ RST	FC3_ RST	FC2_ RST	FC1_ RST	FC0_ RST	UTICK _R...	EVSIM 1_...	EVSIM 0_...		RTC_ RST	DAC0_ RST	ADC1_ RST	ADC0_ RST	SCT_ RST	OSTIM ER...	MRT_ RST
W									Reserv ed							
Reset	0	0	0	0	0	0	0	0	u	u	0	0	0	0	0	0

Fields

Field	Function
31 SmartDMA_RST	SmartDMA reset control 0b - Block is not reset 1b - Block is reset
30 —	Reserved
29 —	Reserved
28 —	Reserved
27 TIMER1_RST	CTIMER1 reset control 0b - Block is not reset 1b - Block is reset
26 TIMER0_RST	CTIMER0 reset control 0b - Block is not reset 1b - Block is reset
25 USB0_FS_RST	USB FS reset control 0b - Block is not reset 1b - Block is reset
24 USB0_FS_DCD_RST	USB FS DCD reset control 0b - Block is not reset 1b - Block is reset

Table continues on the next page...

Table continued from the previous page...

Field	Function
23 —	Reserved
22 TIMER2_RST	CTIMER2 reset control 0b - Block is not reset 1b - Block is reset
21 MICFIL_RST	MICFIL reset control 0b - Block is not reset 1b - Block is reset
20 FC9_RST	LP_FLEXCOMM9 reset control 0b - Block is not reset 1b - Block is reset
19 FC8_RST	LP_FLEXCOMM8 reset control 0b - Block is not reset 1b - Block is reset
18 FC7_RST	LP_FLEXCOMM7 reset control 0b - Block is not reset 1b - Block is reset
17 FC6_RST	LP_FLEXCOMM6 reset control 0b - Block is not reset 1b - Block is reset
16 FC5_RST	LP_FLEXCOMM5 reset control 0b - Block is not reset 1b - Block is reset
15 FC4_RST	LP_FLEXCOMM4 reset control 0b - Block is not reset 1b - Block is reset
14 FC3_RST	LP_FLEXCOMM3 reset control 0b - Block is not reset 1b - Block is reset
13 FC2_RST	LP_FLEXCOMM2 reset control 0b - Block is not reset 1b - Block is reset

Table continues on the next page...

Table continued from the previous page...

Field	Function
12 FC1_RST	LP_FLEXCOMM1 reset control 0b - Block is not reset 1b - Block is reset
11 FC0_RST	LP_FLEXCOMM0 reset control 0b - Block is not reset 1b - Block is reset
10 UTICK_RST	UTICK reset control 0b - Block is not reset 1b - Block is reset
9 EVSIM1_RST	EVSIM1 reset control 0b - Block is not reset 1b - Block is reset
8 EVSIM0_RST	EVSIM0 reset control 0b - Block is not reset 1b - Block is reset
7 —	Reserved
6 RTC_RST	RTC reset control RTC and Sub-second counter reset control **NOTE** To reset RTC correctly, you must write 1 to RTC.CTRL[SWR] when writing 1 to RTC_RST, and write 0 to RTC.CTRL[SWR] after writing 0 to RTC_RST. 0b - Block is not reset 1b - Block is reset
5 DAC0_RST	DAC0 reset control 0b - Block is not reset 1b - Block is reset
4 ADC1_RST	ADC1 reset control 0b - Block is not reset 1b - Block is reset
3 ADC0_RST	ADC0 reset control

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Block is not reset 1b - Block is reset
2 SCT_RST	SCT reset control 0b - Block is not reset 1b - Block is reset
1 OSTIMER_RST	OS Event Timer reset control 0b - Block is not reset 1b - Block is reset
0 MRT_RST	MRT reset control 0b - Block is not reset 1b - Block is reset

11.4.1.9 Peripheral Reset Control 2 (PRESETCTRL2)

Offset

Register	Offset
PRESETCTRL2	108h

Function

The PRESETCTRL2 register allows software to reset specific peripherals. Writing a 0 to any assigned bit in this register clears the reset and allows the specified peripheral to operate. Writing a 1 asserts the reset.

NOTE

When modifying the PRESETCTRL registers, use the related PRESETCTRLSET and PRESETCTRLCLR registers to avoid setting or clearing bits unintentionally.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R								PKC_RST	PUF_RST	TIMER4_...	TIMER3_...	PLU_RST	PQ_RST		USB_HS_...	USB_HS_...
W	Reserved	Reserved	Reserved											Reserved		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FLEXCAN...	FLEXCAN...	Reserved					FREQME_...	TRO_RST	SAI1_RST	SAI0_RST	FLEXIO_...	USDHC_R...	ENET_RST	DMA1_RST	Reserved
W				Reserved												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 —	Reserved
30 —	Reserved
29-25 —	Reserved
24 PKC_RST	PKC reset control 0b - Block is not reset 1b - Block is reset
23 PUF_RST	PUF reset control 0b - Block is not reset 1b - Block is reset
22 TIMER4_RST	CTIMER4 reset control 0b - Block is not reset 1b - Block is reset
21 TIMER3_RST	CTIMER3 reset control 0b - Block is not reset 1b - Block is reset
20 PLU_RST	PLU reset control 0b - Block is not reset 1b - Block is reset
19 PQ_RST	PowerQuad reset control 0b - Block is not reset 1b - Block is reset
18 —	Reserved
17 USB_HS_PHY_RST	USB HS PHY reset control 0b - Block is not reset 1b - Block is reset
16 USB_HS_RST	USB HS reset control

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Block is not reset 1b - Block is reset
15 FLEXCAN1_RST	CAN1 reset control 0b - Block is not reset 1b - Block is reset
14 FLEXCAN0_RST	CAN0 reset control 0b - Block is not reset 1b - Block is reset
13 —	Reserved
12-9 —	Reserved
8 FREQME_RST	FREQME reset control 0b - Block is not reset 1b - Block is reset
7 TRO_RST	TRO reset control 0b - Block is not reset 1b - Block is reset
6 SAI1_RST	SAI1 reset control 0b - Block is not reset 1b - Block is reset
5 SAI0_RST	SAI0 reset control 0b - Block is not reset 1b - Block is reset
4 FLEXIO_RST	FLEXIO reset control 0b - Block is not reset 1b - Block is reset
3 USDHC_RST	uSDHC reset control 0b - Block is not reset 1b - Block is reset
2	Ethernet reset control

Table continues on the next page...

Table continued from the previous page...

Field	Function
ENET_RST	0b - Block is not reset 1b - Block is reset
1 DMA1_RST	DMA1 reset control 0b - Block is not reset 1b - Block is reset
0 —	Reserved

11.4.1.10 Peripheral Reset Control 3 (PRESETCTRL3)

Offset

Register	Offset
PRESETCTRL3	10Ch

Function

The PRESETCTRL3 register allows software to reset specific peripherals. Writing a 0 to any assigned bit in this register clears the reset and allows the specified peripheral to operate. Writing a 1 asserts the reset.

NOTE

When modifying the PRESETCTRL registers, use the related PRESETCTRLSET and PRESETCTRLCLR registers to avoid setting or clearing bits unintentionally.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved				SEMA42_...	Reserved		EIM_RST	EWM_RST	TSI_RST	NPU_RST	COOLFLU...	VREF_RST	CMP2_RST	Reserved	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	OPAMP2_...	OPAMP1_...	OPAMP0_...	DAC2_RST	DAC1_RST	Reserved	Reserved	AOI0_RST	PWM1_RST	PWM0_RST	QDC1_RST	QDC0_RST	COOLFLU...	SINC_RST	I3C1_RST	I3C0_RST
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-28 —	Reserved
27 SEMA42_RST	Semaphore reset control 0b - Block is not reset 1b - Block is reset
26-25 —	Reserved
24 EIM_RST	EIM reset control 0b - Block is not reset 1b - Block is reset
23 EWM_RST	EWM reset control 0b - Block is not reset 1b - Block is reset
22 TSI_RST	TSI reset control 0b - Block is not reset 1b - Block is reset
21 NPU_RST	NPU reset control 0b - Block is not reset 1b - Block is reset
20 COOLFLUX_APB_RST	CoolFlux APB reset control 0b - Block is not reset 1b - Block is reset
19 VREF_RST	VREF reset control 0b - Block is not reset 1b - Block is reset
18 CMP2_RST	CMP2 reset control 0b - Block is not reset 1b - Block is reset
17-16 —	Reserved
15	OPAMP2 reset control

Table continues on the next page...

Table continued from the previous page...

Field	Function
OPAMP2_RST	0b - Block is not reset 1b - Block is reset
14 OPAMP1_RST	OPAMP1 reset control 0b - Block is not reset 1b - Block is reset
13 OPAMP0_RST	OPAMP0 reset control 0b - Block is not reset 1b - Block is reset
12 DAC2_RST	DAC2 reset control 0b - Block is not reset 1b - Block is reset
11 DAC1_RST	DAC1 reset control 0b - Block is not reset 1b - Block is reset
10 —	Reserved Read value is undefined, only zero should be written.
9 —	Reserved Read value is undefined, only zero should be written.
8 AOI0_RST	AOI0 reset control 0b - Block is not reset 1b - Block is reset
7 PWM1_RST	PWM1 reset control 0b - Block is not reset 1b - Block is reset
6 PWM0_RST	PWM0 reset control 0b - Block is not reset 1b - Block is reset
5 QDC1_RST	QDC1 reset control 0b - Block is not reset 1b - Block is reset
4	QDC0 reset control

Table continues on the next page...

Table continued from the previous page...

Field	Function
QDC0_RST	0b - Block is not reset 1b - Block is reset
3 COOLFLUX_RST	CoolFlux reset control 0b - Block is not reset 1b - Block is reset
2 SINC_RST	SINC reset control 0b - Block is not reset 1b - Block is reset
1 I3C1_RST	I3C1 reset control 0b - Block is not reset 1b - Block is reset
0 I3C0_RST	I3C0 reset control 0b - Block is not reset 1b - Block is reset

11.4.1.11 Peripheral Reset Control Set (PRESETCTRLSET0 - PRESETCTRLSET3)

Offset

Register	Offset
PRESETCTRLSET0	120h
PRESETCTRLSET1	124h
PRESETCTRLSET2	128h
PRESETCTRLSET3	12Ch

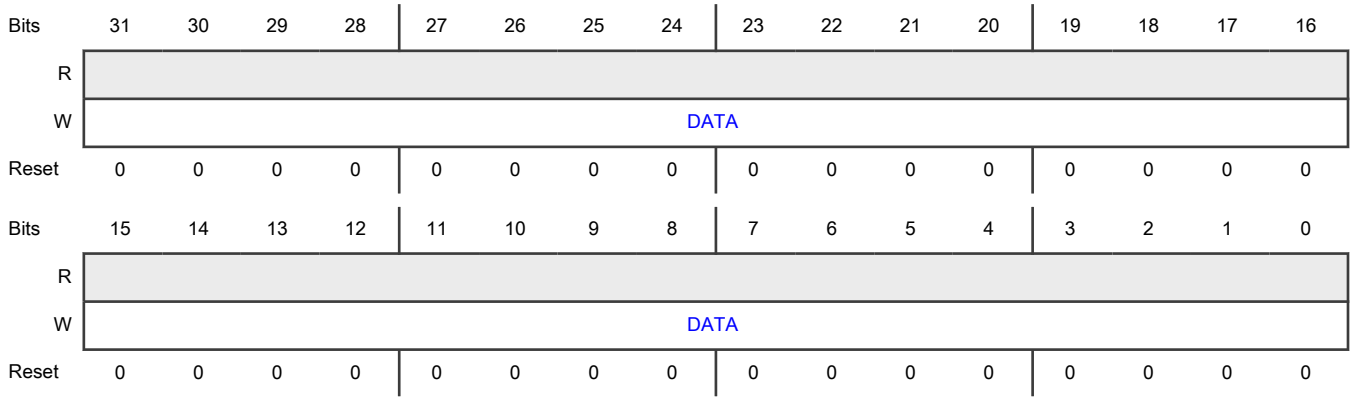
Function

Writing a 1 to a bit position in a write-only PRESETCTRLSETn register sets the corresponding position in PRESETCTRLn.

NOTE

To reset RTC correctly, you must write 1 to RTC.CTRL[SWR] when writing 1 to PRESETCTRL1[RTC_RST], and write 0 to RTC.CTRL[SWR] after writing 0 to PRESETCTRL1[RTC_RST].

Diagram



Fields

Field	Function
31-0 DATA	Data array value, refer to corresponding position in PRESETCTRLn.

11.4.1.12 Peripheral Reset Control Clear (PRESETCTRLCLR0 - PRESETCTRLCLR3)

Offset

Register	Offset
PRESETCTRLCLR0	140h
PRESETCTRLCLR1	144h
PRESETCTRLCLR2	148h
PRESETCTRLCLR3	14Ch

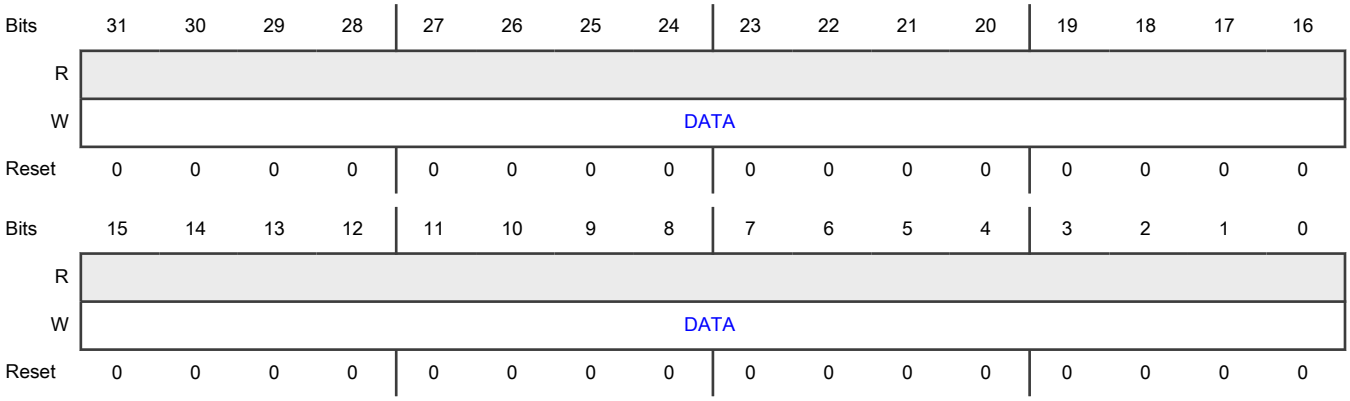
Function

Writing a 1 to a bit position in a write-only PRESETCTRLCLRn register clears the corresponding position in PRESETCTRLn.

NOTE

To reset RTC correctly, you must write 1 to RTC.CTRL[SWR] when writing 1 to PRESETCTRL1[RTC_RST], and write 0 to RTC.CTRL[SWR] after writing 0 to PRESETCTRL1[RTC_RST].

Diagram



Fields

Field	Function
31-0 DATA	Data array value, refer to corresponding position in PRESETCTRLn.

11.4.1.13 AHB Clock Control 0 (AHBCLKCTRL0)

Offset

Register	Offset
AHBCLKCTRL0	200h

Function

The AHBCLKCTRLn registers enable the clocks of the individual modules.

NOTE

When modifying the AHBCLKCTRL registers, use the related AHBCLKCTRLSET and AHBCLKCTRLCLR registers to avoid setting or clearing bits unintentionally.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MAILBOX	Reserved	WWDT1	WWDT0	CRC	DMA0	PINT	Reserved	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0	Reserved	PORT4	PORT3
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PORT2	PORT1	PORT0	MUX	FLEXSPI	FMC	FMU	RAMH_CT...	RAMG_CT...	RAMF_CT...	RAME_CT...	RAMD_CT...	RAMC_CT...	RAMB_CT...	ROM	Reserved
W																
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1

Fields

Field	Function
31 MAILBOX	Enables the clock for the Inter CPU communication Mailbox. 0b - Disables clock 1b - Enables clock
30 —	Reserved Read value is undefined, only zero should be written.
29 WWDT1	Enables the clock for WWDT1 0b - Disables clock 1b - Enables clock
28 WWDT0	Enables the clock for WWDT0 0b - Disables clock 1b - Enables clock
27 CRC	Enables the clock for CRC 0b - Disables clock 1b - Enables clock
26 DMA0	Enables the clock for DMA0 0b - Disables clock 1b - Enables clock
25 PINT	Enables the clock for PINT 0b - Disables clock 1b - Enables clock
24	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
23 GPIO4	Enables the clock for GPIO4 0b - Disables clock 1b - Enables clock
22 GPIO3	Enables the clock for GPIO3 0b - Disables clock 1b - Enables clock
21 GPIO2	Enables the clock for GPIO2 0b - Disables clock 1b - Enables clock
20 GPIO1	Enables the clock for GPIO1 0b - Disables clock 1b - Enables clock
19 GPIO0	Enables the clock for GPIO0 0b - Disables clock 1b - Enables clock
18 —	Reserved
17 PORT4	Enables the clock for PORT4 0b - Disables clock 1b - Enables clock
16 PORT3	Enables the clock for PORT3 0b - Disables clock 1b - Enables clock
15 PORT2	Enables the clock for PORT2 0b - Disables clock 1b - Enables clock
14 PORT1	Enables the clock for PORT1 0b - Disables clock 1b - Enables clock
13	Enables the clock for PORT0 controller

Table continues on the next page...

Table continued from the previous page...

Field	Function
PORT0	0b - Disables clock 1b - Enables clock
12 MUX	Enables the clock for INPUTMUX 0b - Disables clock 1b - Enables clock
11 FLEXSPI	Enables the clock for FlexSPI 0b - Disables clock 1b - Enables clock
10 FMC	Enables the clock for the Flash Memory Controller 0b - Disables clock 1b - Enables clock
9 FMU	Enables the clock for the Flash Management Unit 0b - Disables clock 1b - Enables clock
8 RAMH_CTRL	Enables the clock for the RAMH Controller 0b - Disables clock 1b - Enables clock
7 RAMG_CTRL	Enables the clock for the RAMG Controller 0b - Disables clock 1b - Enables clock
6 RAMF_CTRL	Enables the clock for the RAMF Controller 0b - Disables clock 1b - Enables clock
5 RAME_CTRL	Enables the clock for the RAME Controller 0b - Disables clock 1b - Enables clock
4 RAMD_CTRL	Enables the clock for the RAMD Controller 0b - Disables clock 1b - Enables clock
3 RAMC_CTRL	Enables the clock for the RAMC Controller 0b - Disables clock 1b - Enables clock

Table continues on the next page...

Table continued from the previous page...

Field	Function
2 RAMB_CTRL	Enables the clock for the RAMB Controller 0b - Disables clock 1b - Enables clock
1 ROM	Enables the clock for the ROM 0b - Disables clock 1b - Enables clock
0 —	Reserved Read value is undefined, only zero should be written.

11.4.1.14 AHB Clock Control 1 (AHBCLKCTRL1)

Offset

Register	Offset
AHBCLKCTRL1	204h

Function

The AHBCLKCTRLn registers enable the clocks of the individual modules.

NOTE

When modifying the AHBCLKCTRL registers, use the related AHBCLKCTRLSET and AHBCLKCTRLCLR registers to avoid setting or clearing bits unintentionally.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Smart DMA	Reserved	PKC_RAM	Reserved	TIMER1	TIMER0	USB0_FS	USB0_FS...	Reserved	TIMER2	MICFIL	FC9	FC8	FC7	FC6	FC5
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FC4	FC3	FC2	FC1	FC0	UTICK	EVSIM1	EVSIM0	Reserved	RTC	DAC0	ADC1	ADC0	SCT	OSTIMER	MRT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 SmartDMA	Enables the clock for SmartDMA 0b - Disables clock 1b - Enables clock
30 —	Reserved
29 PKC_RAM	Enables the clock for PKC RAM 0b - Disables clock 1b - Enables clock
28 —	Reserved
27 TIMER1	Enables the clock for CTIMER1 0b - Disables clock 1b - Enables clock
26 TIMER0	Enables the clock for CTIMER0 0b - Disables clock 1b - Enables clock
25 USB0_FS	Enables the clock for USB-FS 0b - Disables clock 1b - Enables clock
24 USB0_FS_DCD	Enables the clock for USB-FS DCD 0b - Disables clock 1b - Enables clock
23 —	Reserved
22 TIMER2	Enables the clock for CTIMER2 0b - Disables clock 1b - Enables clock
21 MICFIL	Enables the clock for MICFIL 0b - Disables clock 1b - Enables clock
20	Enables the clock for LP_FLEXCOMM9

Table continues on the next page...

Table continued from the previous page...

Field	Function
FC9	0b - Disables clock 1b - Enables clock
19 FC8	Enables the clock for LP_FLEXCOMM8 0b - Disables clock 1b - Enables clock
18 FC7	Enables the clock for LP_FLEXCOMM7 0b - Disables clock 1b - Enables clock
17 FC6	Enables the clock for LP_FLEXCOMM6 0b - Disables clock 1b - Enables clock
16 FC5	Enables the clock for LP_FLEXCOMM5 0b - Disables clock 1b - Enables clock
15 FC4	Enables the clock for LP_FLEXCOMM4 0b - Disables clock 1b - Enables clock
14 FC3	Enables the clock for LP_FLEXCOMM3 0b - Disables clock 1b - Enables clock
13 FC2	Enables the clock for LP_FLEXCOMM2 0b - Disables clock 1b - Enables clock
12 FC1	Enables the clock for LP_FLEXCOMM1 0b - Disables clock 1b - Enables clock
11 FC0	Enables the clock for LP_FLEXCOMM0 0b - Disables clock 1b - Enables clock
10 UTICK	Enables the clock for UTICK 0b - Disables clock 1b - Enables clock

Table continues on the next page...

Table continued from the previous page...

Field	Function
9 EVSIM1	Enables the clock for EVSIM1 0b - Disables clock 1b - Enables clock
8 EVSIM0	Enables the clock for EVSIM0 0b - Disables clock 1b - Enables clock
7 —	Reserved
6 RTC	Enables the clock for RTC 0b - Disables clock 1b - Enables clock
5 DAC0	Enables the clock for DAC0 0b - Disables clock 1b - Enables clock
4 ADC1	Enables the clock for ADC1 0b - Disables clock 1b - Enables clock
3 ADC0	Enables the clock for ADC0 0b - Disables clock 1b - Enables clock
2 SCT	Enables the clock for SCT 0b - Disables clock 1b - Enables clock
1 OSTIMER	Enables the clock for the OS Event Timer 0b - Disables clock 1b - Enables clock
0 MRT	Enables the clock for MRT 0b - Disables clock 1b - Enables clock

11.4.1.15 AHB Clock Control 2 (AHBCLKCTRL2)

Offset

Register	Offset
AHBCLKCTRL2	208h

Function

The AHBCLKCTRLn registers enable the clocks of the individual modules.

NOTE

When modifying the AHBCLKCTRL registers, use the related AHBCLKCTRLSET and AHBCLKCTRLCLR registers to avoid setting or clearing bits unintentionally.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	Reserved	Reserved	GDET	Reserved		SCG	Reserved	PKC	PUF	TIMER 4	TIMER 3	PLU_ LUT	PQ	ELS	USB_ HS_...	USB_ HS
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	FLEX CAN1	FLEX CAN0	Reserved	Reserved				FREQ ME	TRO	SAI1	SAI0	FLEXI O	uSDH C	ENET	DMA1	Reserved
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 —	Reserved
30 —	Reserved
29 GDET	Enables the clock for GDET0 and GDET1 0b - Disables clock 1b - Enables clock
28-27 —	Reserved
26	Enables the clock for SCG

Table continues on the next page...

Table continued from the previous page...

Field	Function
SCG	0b - Disables clock 1b - Enables clock
25 —	Reserved Read value is undefined, only zero should be written.
24 PKC	Enables the clock for PKC 0b - Disables clock 1b - Enables clock
23 PUF	Enables the clock for PUF 0b - Disables clock 1b - Enables clock
22 TIMER4	Enables the clock for CTIMER4 0b - Disables clock 1b - Enables clock
21 TIMER3	Enables the clock for CTIMER3 0b - Disables clock 1b - Enables clock
20 PLU_LUT	Enables the clock for PLU_LUT 0b - Disables clock 1b - Enables clock
19 PQ	Enables the clock for Powerquad 0b - Disables clock 1b - Enables clock
18 ELS	Enables the clock for ELS 0b - Disables clock 1b - Enables clock
17 USB_HS_PHY	Enables the clock for USB HS PHY 0b - Disables clock 1b - Enables clock
16 USB_HS	Enables the clock for USB HS 0b - Disables clock 1b - Enables clock

Table continues on the next page...

Table continued from the previous page...

Field	Function
15 FLEXCAN1	Enables the clock for FLEXCAN1 0b - Disables clock 1b - Enables clock
14 FLEXCAN0	Enables the clock for FLEXCAN0 0b - Disables clock 1b - Enables clock
13 —	Reserved
12-9 —	Reserved
8 FREQME	Enables the clock for the Frequency meter 0b - Disables clock 1b - Enables clock
7 TRO	Enables the clock for TRO 0b - Disables clock 1b - Enables clock
6 SAI1	Enables the clock for SAI1 0b - Disables clock 1b - Enables clock
5 SAI0	Enables the clock for SAI0 0b - Disables clock 1b - Enables clock
4 FLEXIO	Enables the clock for FlexIO 0b - Disables clock 1b - Enable clock
3 uSDHC	Enables the clock for uSDHC 0b - Disables clock 1b - Enables clock
2 ENET	Enables the clock for Ethernet 0b - Disables clock 1b - Enables clock

Table continues on the next page...

Table continued from the previous page...

Field	Function
1 DMA1	Enables the clock for DMA1 0b - Disables clock 1b - Enables clock
0 —	Reserved

11.4.1.16 AHB Clock Control 3 (AHBCLKCTRL3)

Offset

Register	Offset
AHBCLKCTRL3	20Ch

Function

The AHBCLKCTRLn registers enable the clocks of the individual modules.

NOTE

When modifying the AHBCLKCTRL registers, use the related AHBCLKCTRLSET and AHBCLKCTRLCLR registers to avoid setting or clearing bits unintentionally.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved				SEMA42	INTM	ERM	EIM	EWM	TSI	NPU	COOLFLU...	VREF	CMP2		
W															Reserved	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	OPAMP2	OPAMP1	OPAMP0	DAC2	DAC1	Reserved	Reserved	EVTG	PWM1	PWM0	QDC1	QDC0	COOLFLUX	SINC	I3C1	I3C0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-28 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
27 SEMA42	Enables the clock for Semaphore 0b - Disables clock 1b - Enables clock
26 INTM	Enables the clock for INTM 0b - Disables clock 1b - Enables clock
25 ERM	Enables the clock for ERM 0b - Disables clock 1b - Enables clock
24 EIM	Enables the clock for EIM 0b - Disables clock 1b - Enables clock
23 EWM	Enables the clock for EWM 0b - Disables clock 1b - Enables clock
22 TSI	Enables the clock for TSI 0b - Disables clock 1b - Enables clock
21 NPU	Enables the clock for NPU 0b - Disables clock 1b - Enables clock
20 COOLFLUX_APB	Enables the clock for CoolFlux APB 0b - Disables clock 1b - Enables clock (CoolFlux needs to be properly programmed before the clock enabled.)
19 VREF	Enables the clock for VREF 0b - Disables clock 1b - Enables clock
18 CMP2	Enables the clock for CMP2 0b - Disables clock 1b - Enables clock
17-16 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
15 OPAMP2	Enables the clock for OPAMP2 0b - Disables clock 1b - Enables clock
14 OPAMP1	Enables the clock for OPAMP1 0b - Disables clock 1b - Enables clock
13 OPAMP0	Enables the clock for OPAMP0 0b - Disables clock 1b - Enables clock
12 DAC2	Enables the clock for DAC2 0b - Disables clock 1b - Enables clock
11 DAC1	Enables the clock for DAC1 0b - Disables clock 1b - Enables clock
10 —	Reserved Read value is undefined, only zero should be written.
9 —	Reserved Read value is undefined, only zero should be written.
8 EVTG	Enables the clock for EVTG 0b - Disables clock 1b - Enables clock
7 PWM1	Enables the clock for PWM1 0b - Disables clock 1b - Enables clock
6 PWM0	Enables the clock for PWM0 0b - Disables clock 1b - Enables clock
5 QDC1	Enables the clock for QDC1 0b - Disables clock 1b - Enables clock

Table continues on the next page...

Table continued from the previous page...

Field	Function
4 QDC0	Enables the clock for QDC0 0b - Disables clock 1b - Enables clock
3 COOLFLUX	Enables the clock for CoolFlux 0b - Disables clock 1b - Enables clock
2 SINC	Enables the clock for SINC 0b - Disables clock 1b - Enables clock
1 I3C1	Enables the clock for I3C1 0b - Disables clock 1b - Enables clock
0 I3C0	Enables the clock for I3C0 0b - Disables clock 1b - Enables clock

11.4.1.17 AHB Clock Control Set (AHBCLKCTRLSET0 - AHBCLKCTRLSET3)

Offset

Register	Offset
AHBCLKCTRLSET0	220h
AHBCLKCTRLSET1	224h
AHBCLKCTRLSET2	228h
AHBCLKCTRLSET3	22Ch

Function
Writing a 1 to a bit position in a write-only AHBCLKCTRLSETn register sets the corresponding position in AHBCLKCTRLn.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 DATA	Data array value

11.4.1.18 AHB Clock Control Clear (AHBCLKCTRLCLR0 - AHBCLKCTRLCLR3)

Offset

Register	Offset
AHBCLKCTRLCLR0	240h
AHBCLKCTRLCLR1	244h
AHBCLKCTRLCLR2	248h
AHBCLKCTRLCLR3	24Ch

Function

Writing a 1 to a bit position in a write-only AHBCLKCTRLCLRn register clears the corresponding position in AHBCLKCTRLn.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

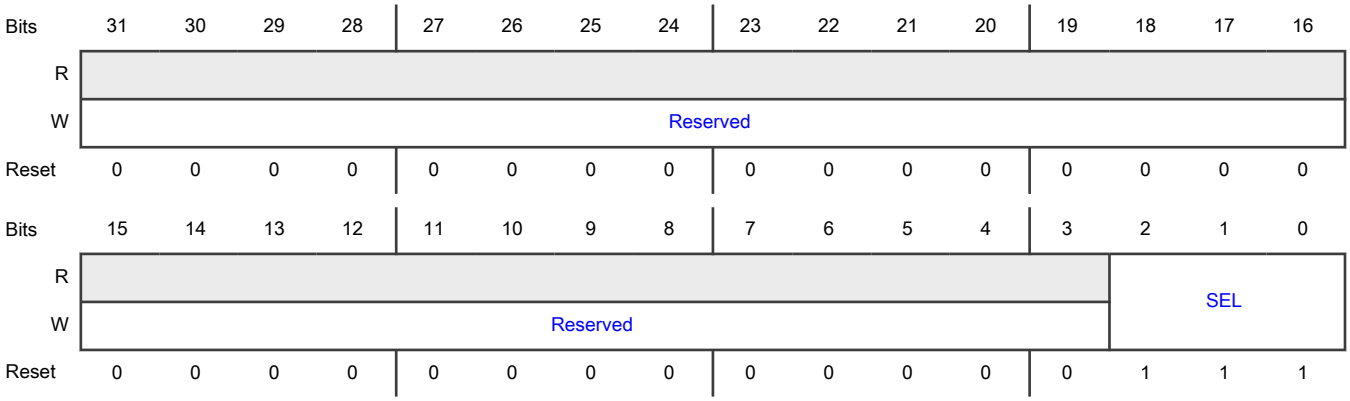
Field	Function
31-0 DATA	Data array value

11.4.1.19 CPU0 System Tick Timer Source Select (SYSTICKCLKSEL0)

Offset

Register	Offset
SYSTICKCLKSEL0	260h

Diagram



Fields

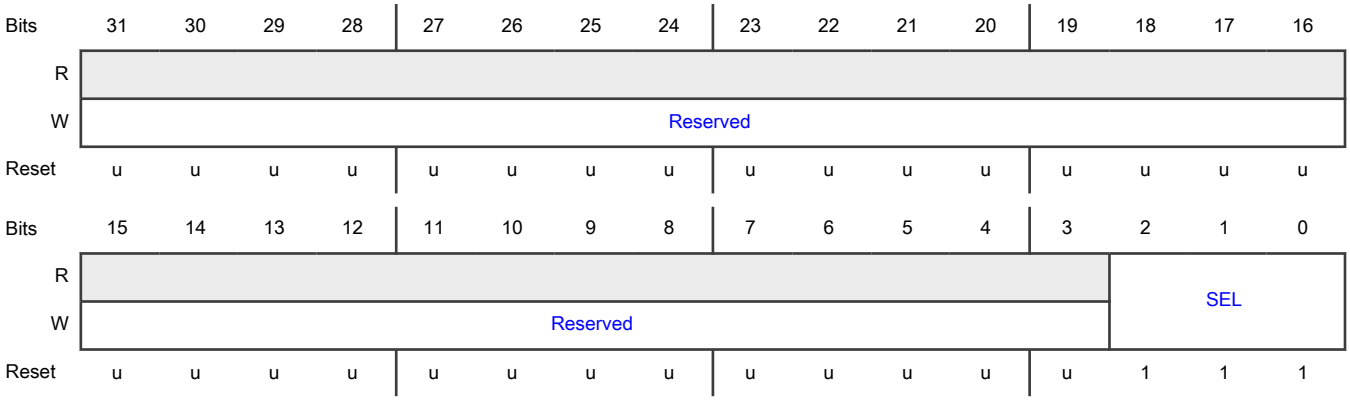
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the System Tick Timer for CPU0 source 000b - SYSTICKCLKDIV0 output 001b - Clk 1 MHz clock 010b - LP Oscillator clock 011b - No clock 100b - No clock 101b - No clock 110b - No clock 111b - No clock

11.4.1.20 CPU1 System Tick Timer Source Select (SYSTICKCLKSEL1)

Offset

Register	Offset
SYSTICKCLKSEL1	264h

Diagram



Fields

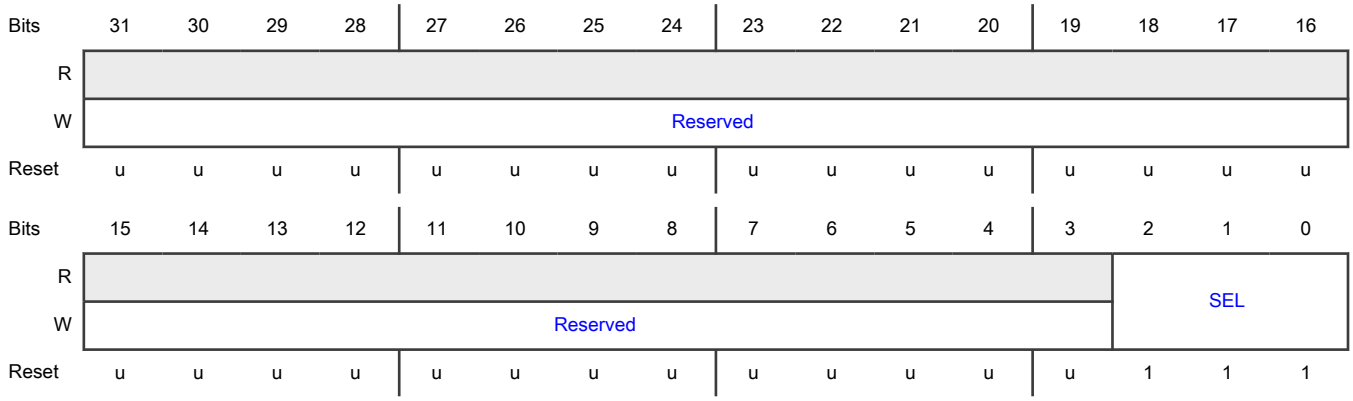
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the System Tick Timer for CPU1 source. 000b - SYSTICKCLKDIV1 output 001b - Clk 1 MHz clock 010b - LP Oscillator clock 011b - No clock 100b - No clock 101b - No clock 110b - No clock 111b - No clock

11.4.1.21 Trace Clock Source Select (TRACECLKSEL)

Offset

Register	Offset
TRACECLKSEL	268h

Diagram



Fields

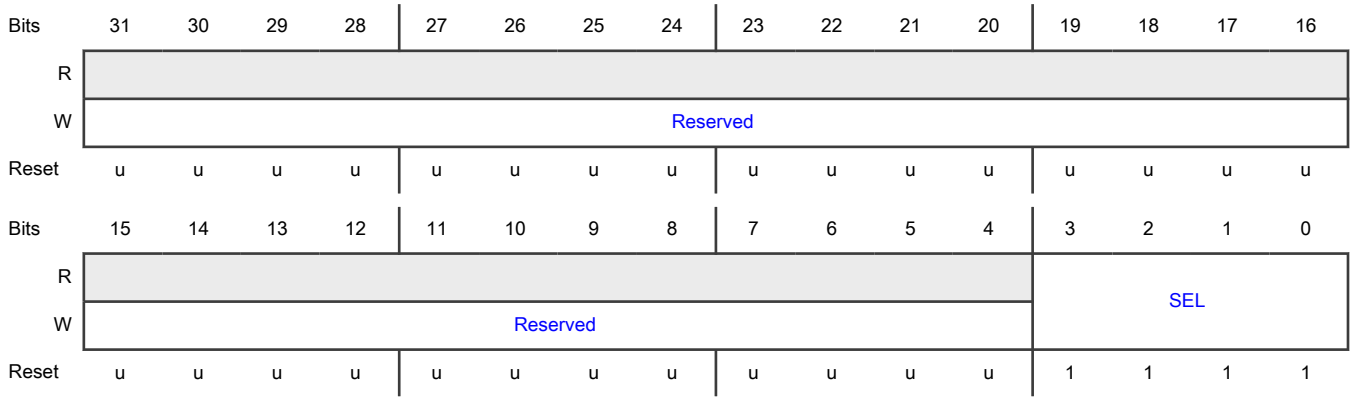
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the trace clock source. 000b - TRACECLKDIV output 001b - Clk 1 MHz clock 010b - LP Oscillator clock 011b - No clock 100b - No clock 101b - No clock 110b - No clock 111b - No clock

11.4.1.22 CTIMER Clock Source Select (CTIMERCLKSEL0 - CTIMERCLKSEL4)

Offset

Register	Offset
CTIMERCLKSEL0	26Ch
CTIMERCLKSEL1	270h
CTIMERCLKSEL2	274h
CTIMERCLKSEL3	278h
CTIMERCLKSEL4	27Ch

Diagram



Fields

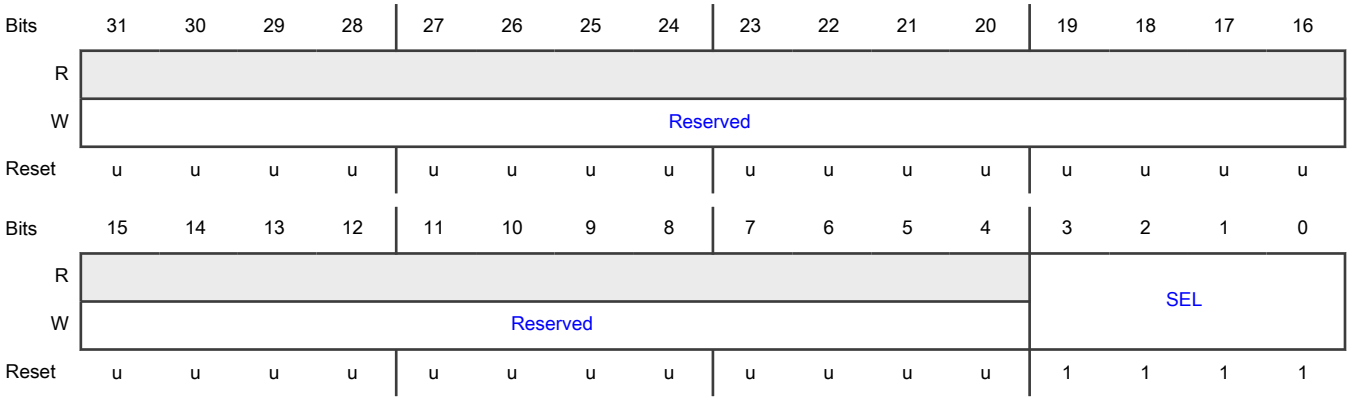
Field	Function
31-4 —	Reserved
3-0 SEL	Selects the CTIMER clock source. 0000b - FRO_1M clock 0001b - PLL0 clock 0010b - PLL1_clk0 clock 0011b - FRO_HF clock 0100b - FRO 12MHz clock 0101b - SAI0 MCLK IN clock 0110b - LP Oscillator clock 0111b - No clock 1000b - SAI1 MCLK IN clock 1001b - SAI0 TX_BCLK clock 1010b - SAI0 RX_BCLK clock 1011b - SAI1 TX_BCLK clock 1100b - SAI1 RX_BCLK clock 1101b - No clock 1110b - No clock 1111b - No clock

11.4.1.23 CLKOUT Clock Source Select (CLKOUTSEL)

Offset

Register	Offset
CLKOUTSEL	288h

Diagram



Fields

Field	Function
31-4 —	Reserved
3-0 SEL	Selects the CLKOUT clock source. 0000b - Main clock (main_clk) 0001b - PLL0 clock (pll0_clk) 0010b - CLKIN clock (clk_in) 0011b - FRO_HF clock (fro_hf) 0100b - FRO 12 MHz clock (fro_12m) 0101b - PLL1_clk0 clock (pll1_clk) 0110b - LP Oscillator clock (lp_osc) 0111b - USB PLL clock (usb_pll_clk) 1000b - No clock 1001b - No clock 1010b - No clock 1011b - No clock 1100b - No clock

Table continues on the next page...

Table continued from the previous page...

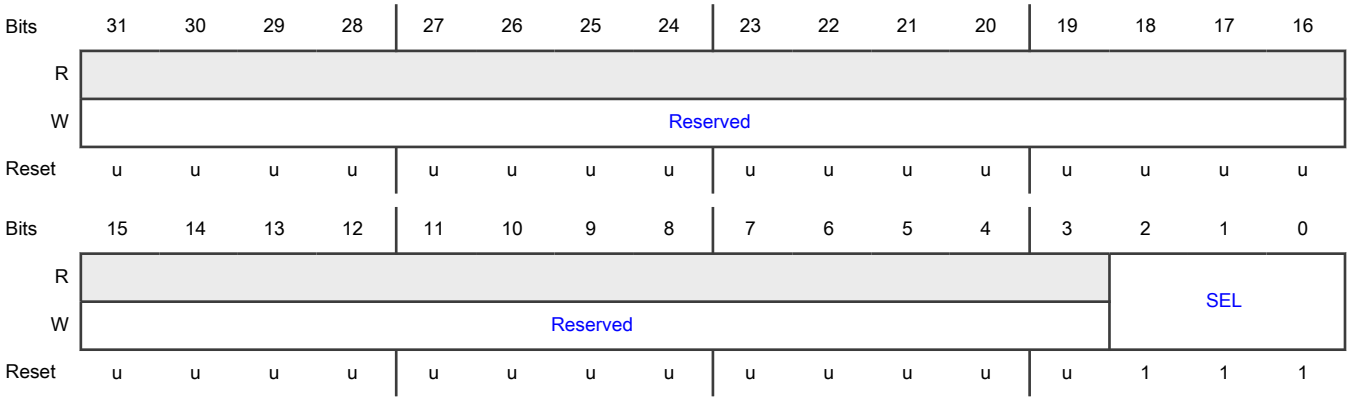
Field	Function
	1101b - No clock 1110b - No clock 1111b - No clock

11.4.1.24 ADC0 Clock Source Select (ADC0CLKSEL)

Offset

Register	Offset
ADC0CLKSEL	2A4h

Diagram



Fields

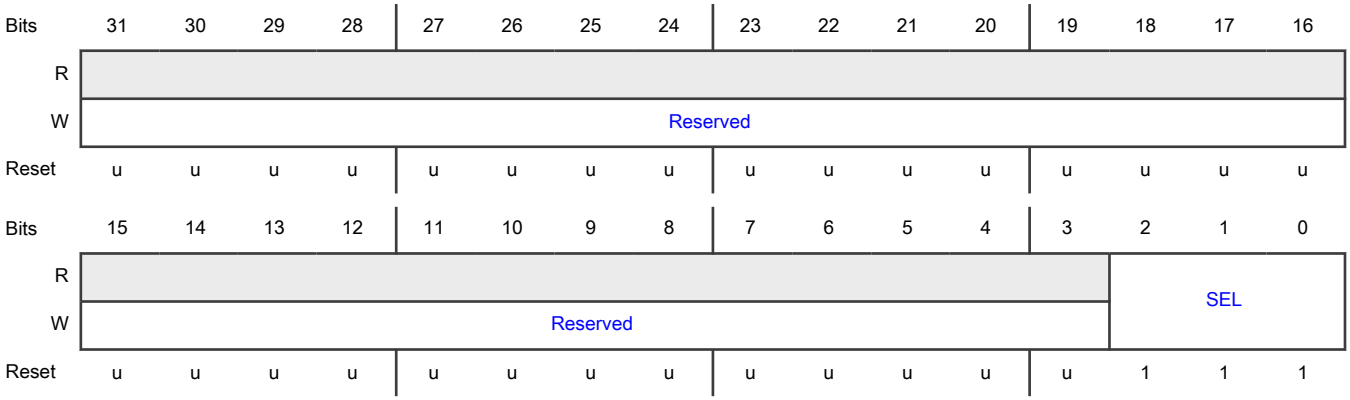
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the ADC0 clock source. 000b - No clock 001b - PLL0 clock 010b - FRO_HF clock 011b - FRO 12 MHz clock 100b - Clk_in 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.25 USB-FS Clock Source Select (USB0CLKSEL)

Offset

Register	Offset
USB0CLKSEL	2A8h

Diagram



Fields

Field	Function
31-3 —	Reserved
2-0 SEL	Selects the USB-FS clock source. 000b - No clock 001b - PLL0 clock 010b - No clock 011b - Clk 48 MHz clock 100b - Clk_in 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

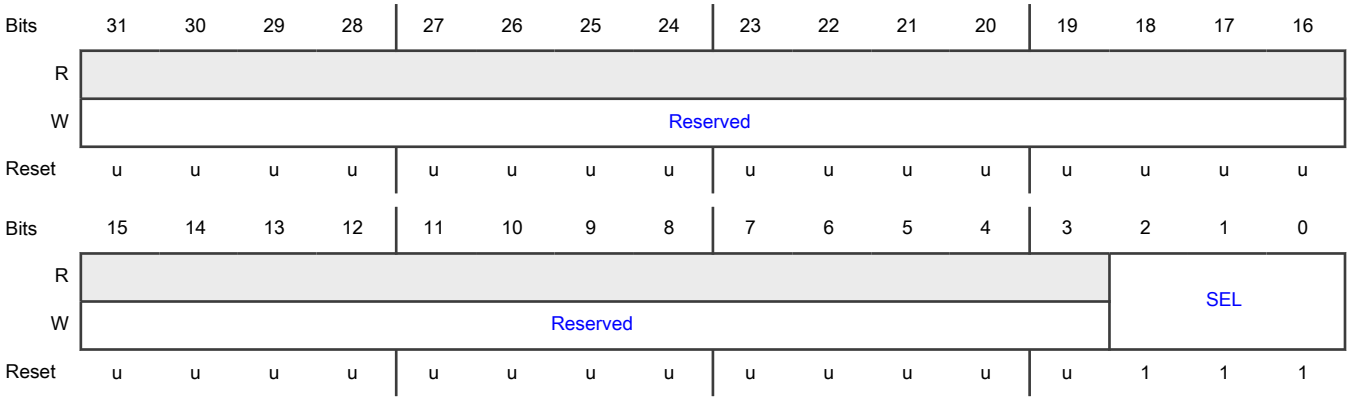
11.4.1.26 LP_FLEXCOMM Clock Source Select for Fractional Rate Divider (FCCLKSEL0 - FCCLKSEL9)

Offset

For n = 0 to 9:

Register	Offset
FCCLKSELn	2B0h + (n × 4h)

Diagram



Fields

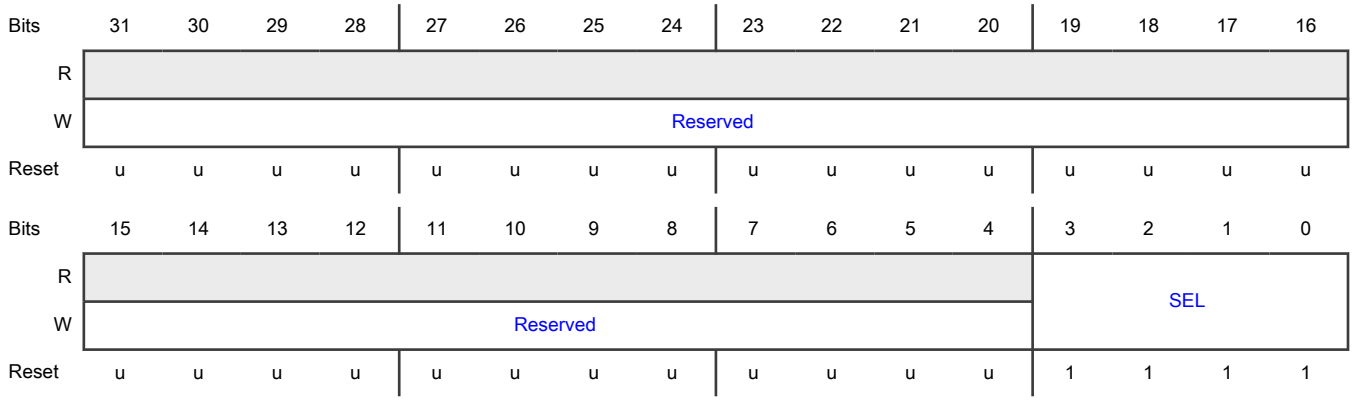
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the LP_FLEXCOMM clock source for Fractional Rate Divider. 000b - No clock 001b - PLL divided clock 010b - FRO 12 MHz clock 011b - fro_hf_div clock 100b - clk_1m clock 101b - USB PLL clock 110b - LP Oscillator clock 111b - No clock

11.4.1.27 SCTimer/PWM Clock Source Select (SCTCLKSEL)

Offset

Register	Offset
SCTCLKSEL	2F0h

Diagram



Fields

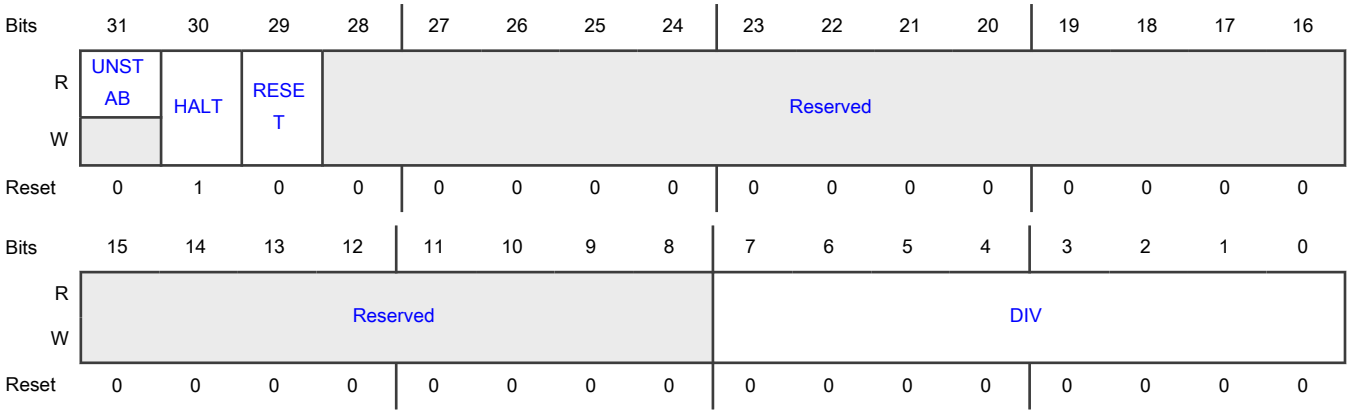
Field	Function
31-4 —	Reserved
3-0 SEL	Selects the SCTimer/PWM clock source. 0000b - No clock 0001b - PLL0 clock 0010b - CLKIN clock 0011b - FRO_HF clock 0100b - PLL1_clk0 clock 0101b - SAI0 MCLK_IN clock 0110b - USB PLL clock 0111b - No clock 1000b - SAI1 MCLK_IN clock 1001b - No clock 1010b - No clock 1011b - No clock 1100b - No clock 1101b - No clock 1110b - No clock 1111b - No clock

11.4.1.28 CPU0 System Tick Timer Divider (SYSTICKCLKDIV0)

Offset

Register	Offset
SYSTICKCLKDIV0	300h

Diagram



Fields

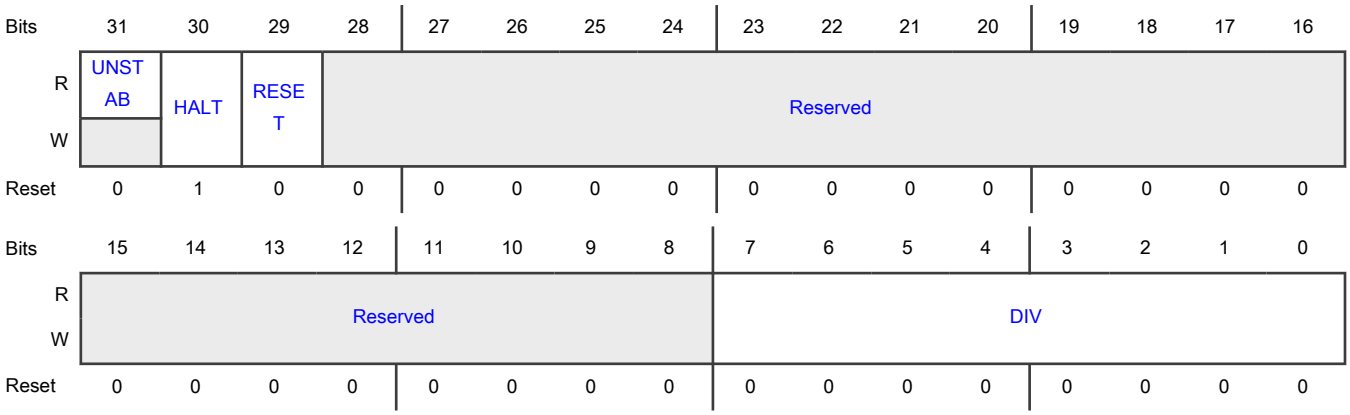
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset.
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.29 CPU1 System Tick Timer Divider (SYSTICKCLKDIV1)

Offset

Register	Offset
SYSTICKCLKDIV1	304h

Diagram



Fields

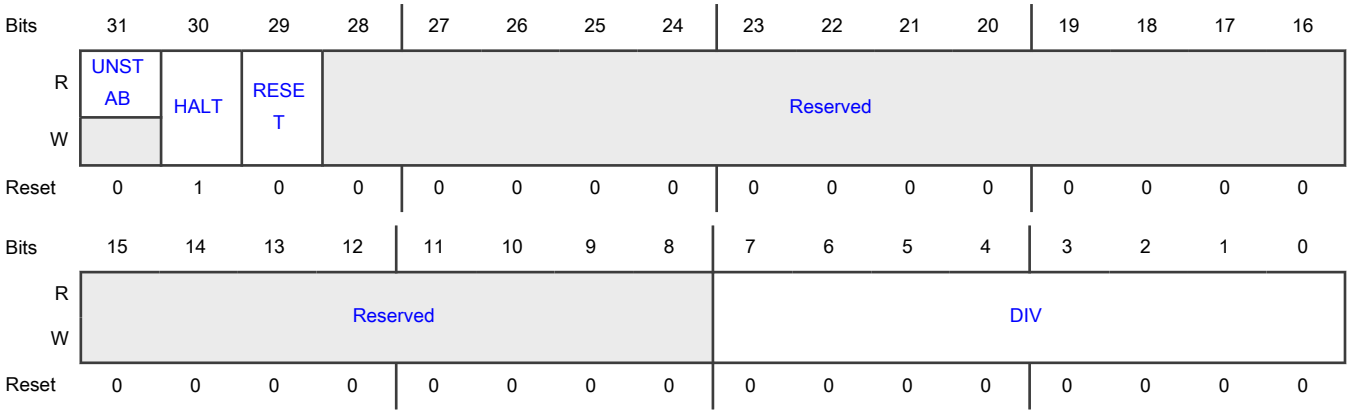
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.30 TRACE Clock Divider (TRACECLKDIV)

Offset

Register	Offset
TRACECLKDIV	308h

Diagram



Fields

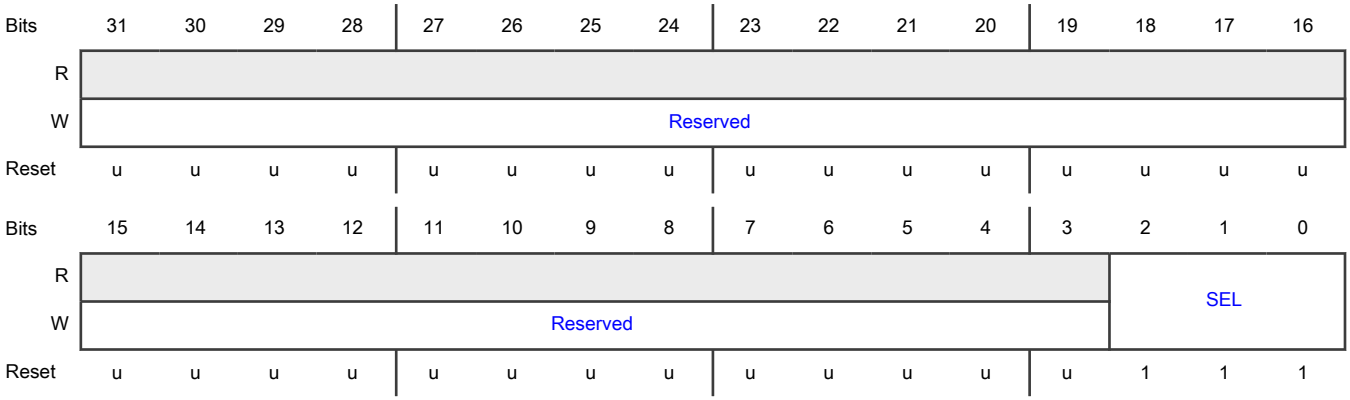
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.31 TSI Function Clock Source Select (TSICLKSEL)

Offset

Register	Offset
TSICLKSEL	350h

Diagram



Fields

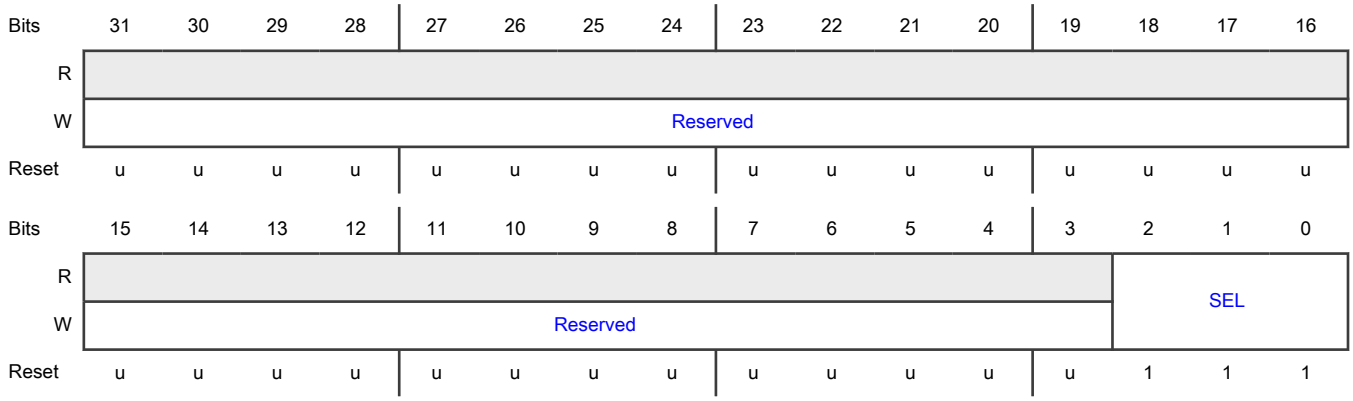
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the TSI function clock source. 000b - No clock 001b - No clock 010b - clk_in 011b - No clock 100b - FRO_12Mhz clock 101b - No clock 110b - No clock 111b - No clock

11.4.1.32 SINC FILTER Function Clock Source Select (SINCFILTCLKSEL)

Offset

Register	Offset
SINCFILTCLKSEL	360h

Diagram



Fields

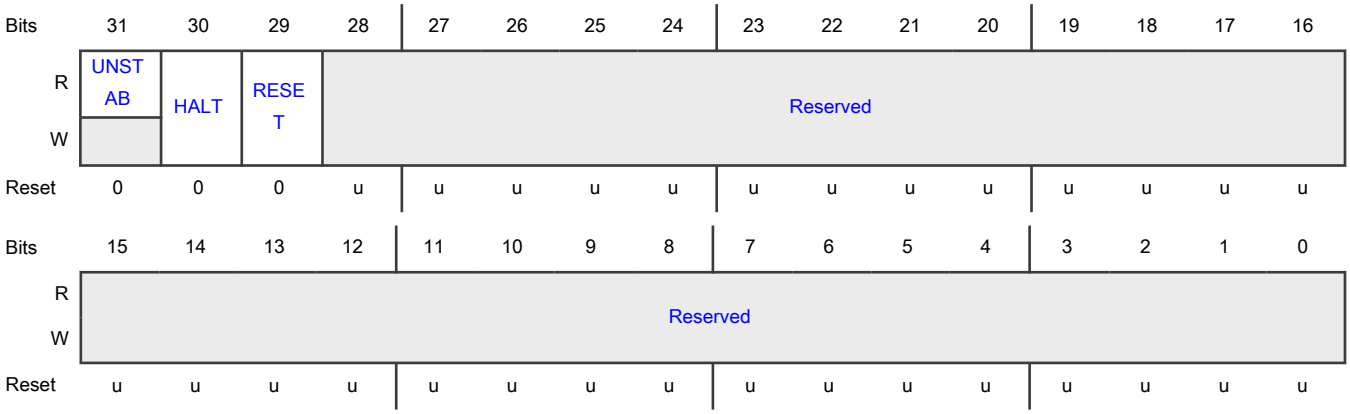
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the SINC FILTER function clock source. 000b - No clock 001b - PLL0 clock 010b - clk_in 011b - FRO_HF clock 100b - FRO_12Mhz clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.33 SLOW_CLK Clock Divider (SLOWCLKDIV)

Offset

Register	Offset
SLOWCLKDIV	378h

Diagram



Fields

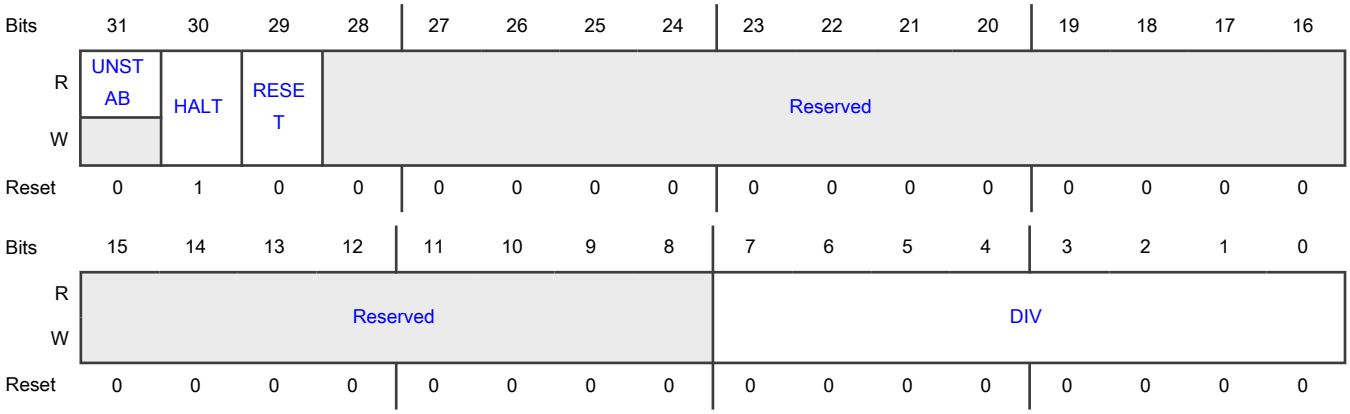
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-0 —	Reserved

11.4.1.34 TSI Function Clock Divider (TSICLKDIV)

Offset

Register	Offset
TSICLKDIV	37Ch

Diagram



Fields

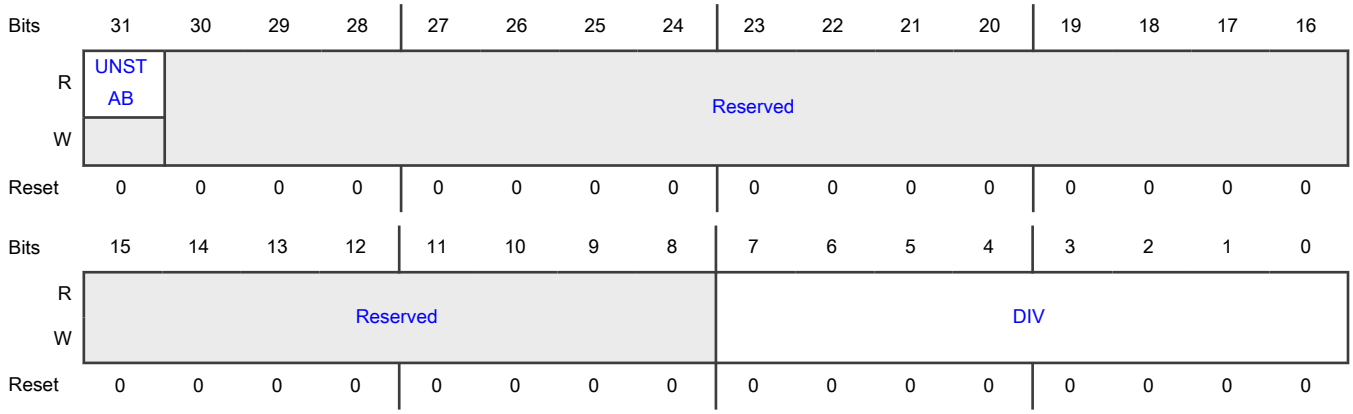
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value: • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.35 System Clock Divider (AHBCLKDIV)

Offset

Register	Offset
AHBCLKDIV	380h

Diagram



Fields

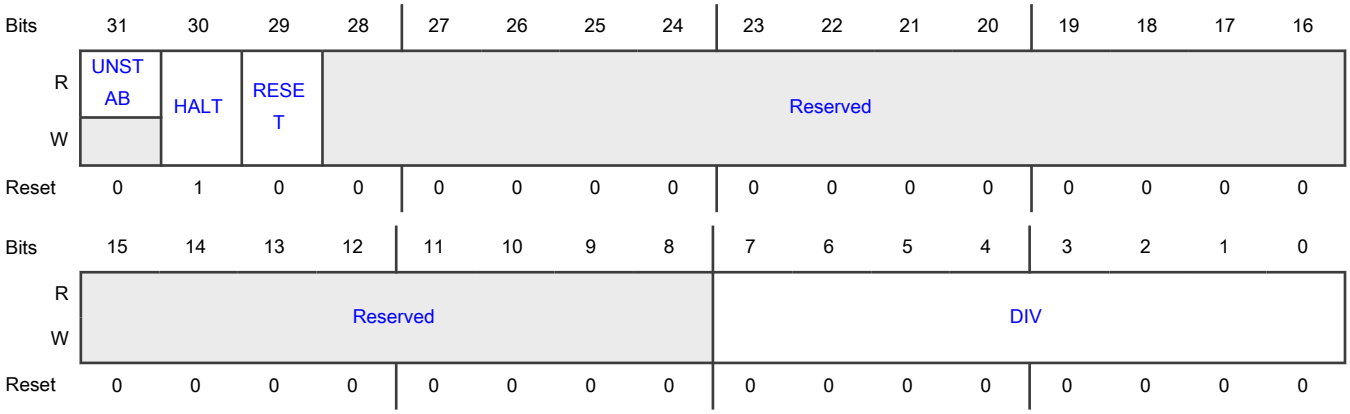
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.36 CLKOUT Clock Divider (CLKOUTDIV)

Offset

Register	Offset
CLKOUTDIV	384h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.37 FRO_HF_DIV Clock Divider (FROHFDIV)

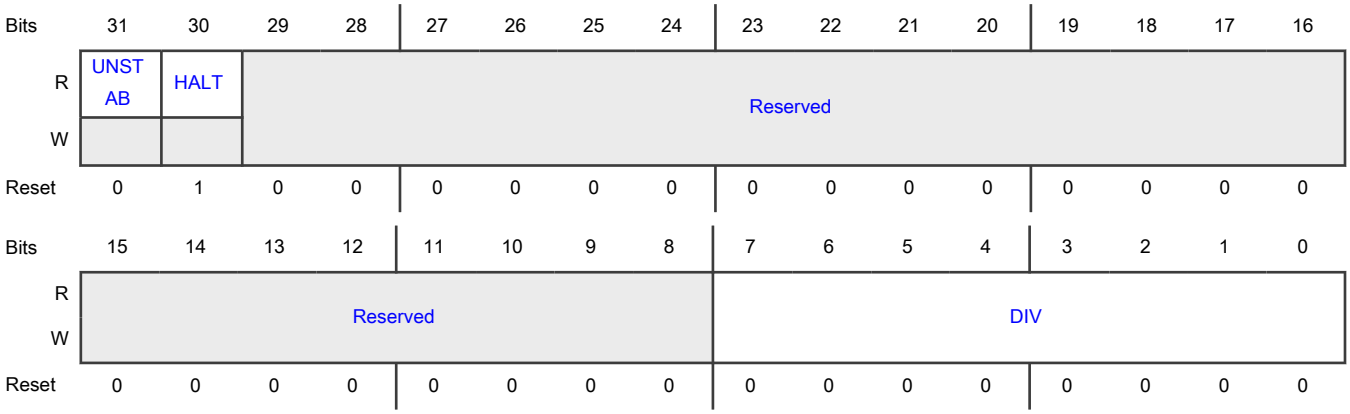
Offset

Register	Offset
FROHFDIV	388h

Function

This register is used to generate fro_hf_div clock from fro_hf clock.

Diagram



Fields

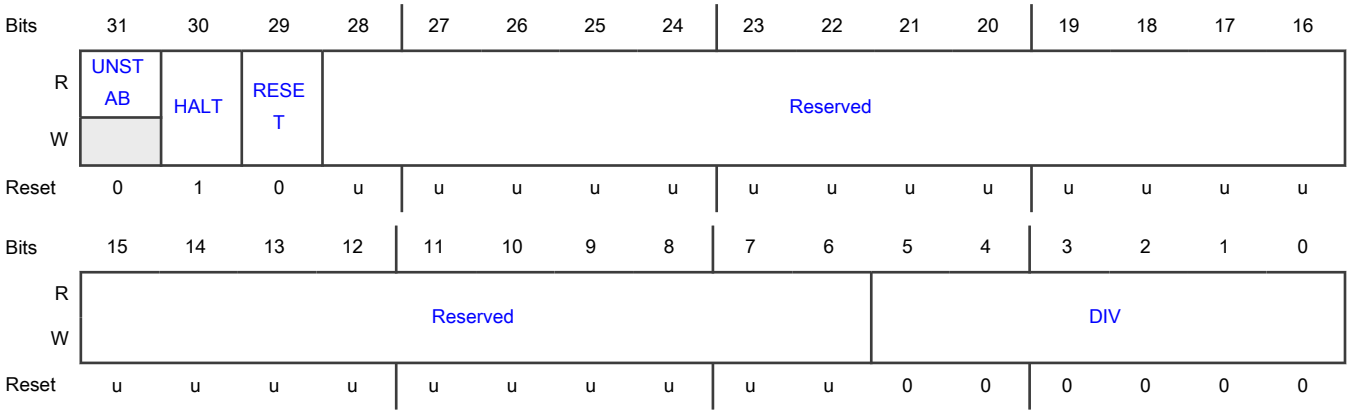
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running, this bit is set to 0 when the register is written. 1b - Divider clock is stopped
29-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.38 WDT0 Clock Divider (WDT0CLKDIV)

Offset

Register	Offset
WDT0CLKDIV	38Ch

Diagram



Fields

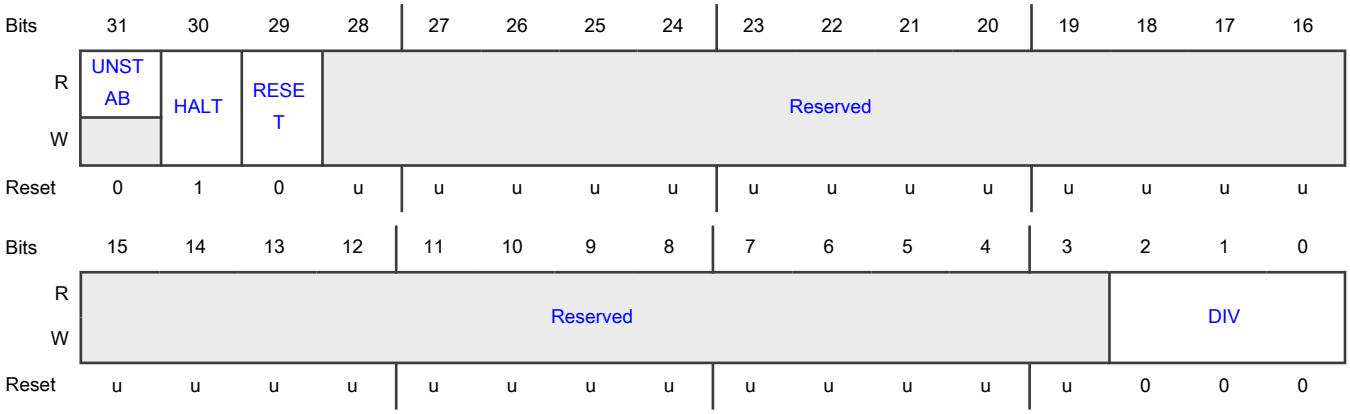
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-6 —	Reserved
5-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.39 ADC0 Clock Divider (ADC0CLKDIV)

Offset

Register	Offset
ADC0CLKDIV	394h

Diagram



Fields

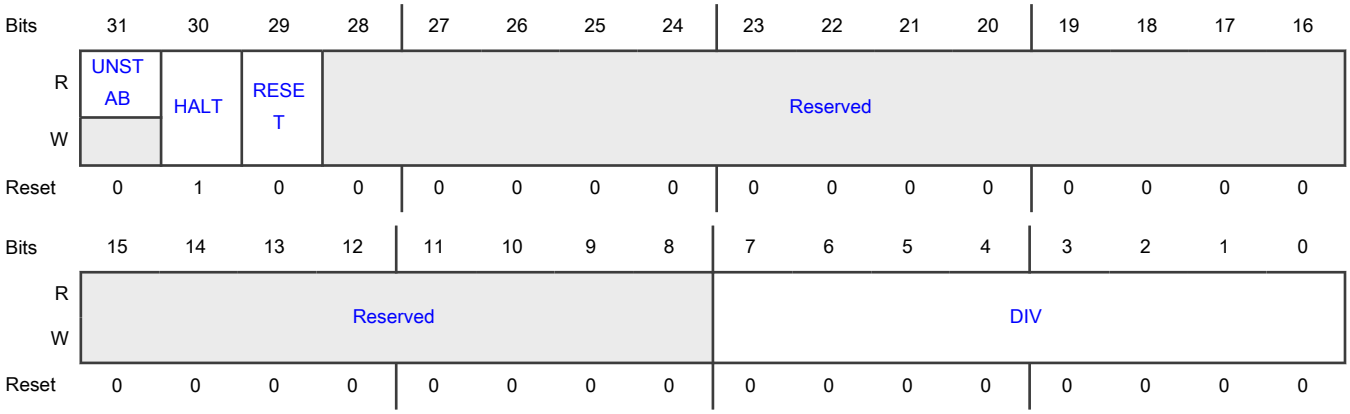
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-3 —	Reserved
2-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.40 USB-FS Clock Divider (USB0CLKDIV)

Offset

Register	Offset
USB0CLKDIV	398h

Diagram



Fields

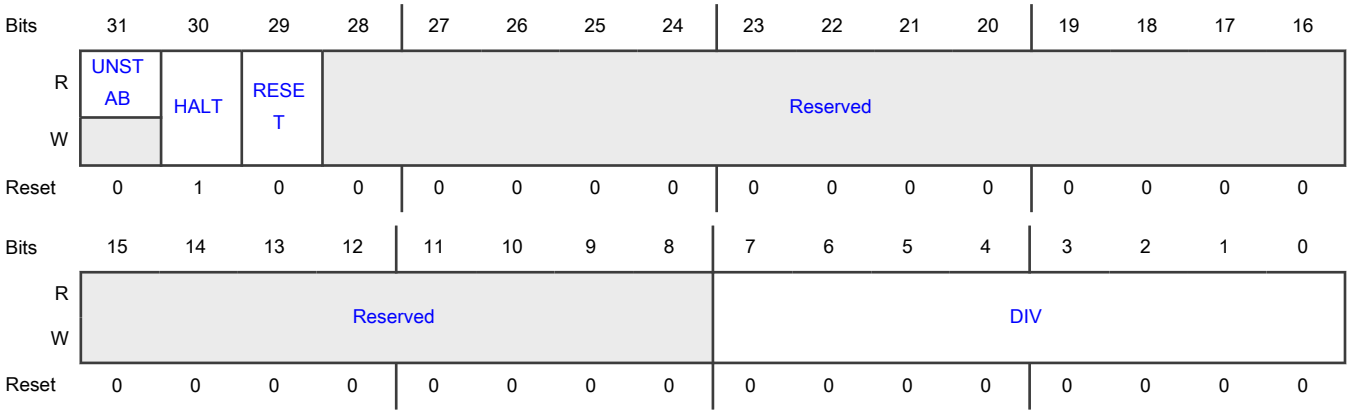
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.41 SCT/PWM Clock Divider (SCTCLKDIV)

Offset

Register	Offset
SCTCLKDIV	3B4h

Diagram



Fields

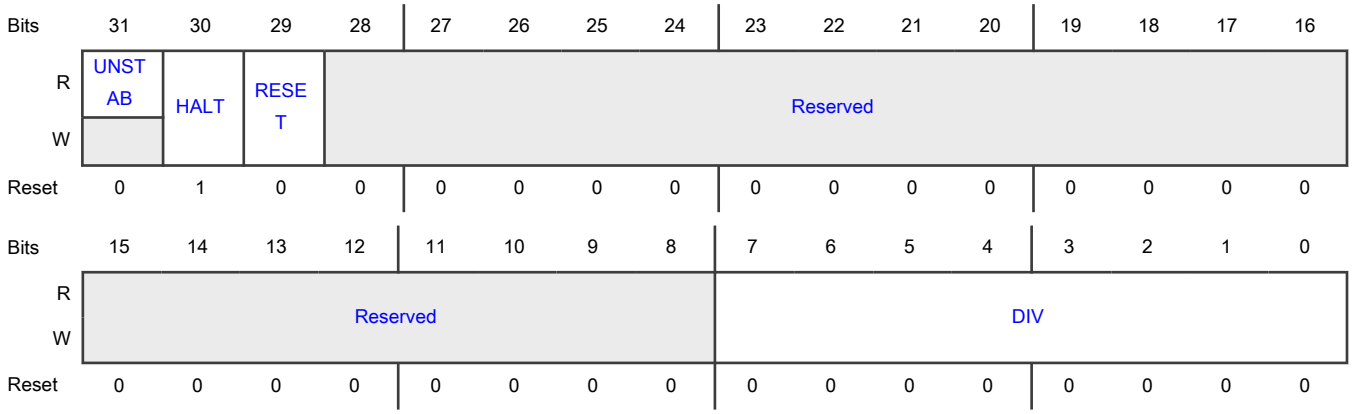
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.42 PLL Clock Divider (PLLCLKDIV)

Offset

Register	Offset
PLLCLKDIV	3C4h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.43 CTimer Clock Divider (CTIMER0CLKDIV - CTIMER4CLKDIV)

Offset

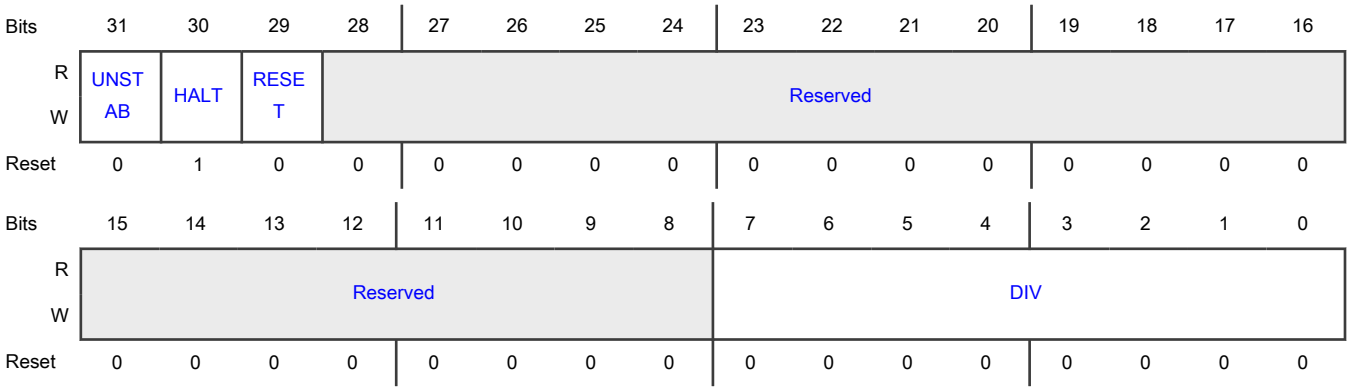
Register	Offset
CTIMER0CLKDIV	3D0h
CTIMER1CLKDIV	3D4h
CTIMER2CLKDIV	3D8h

Table continues on the next page...

Table continued from the previous page...

Register	Offset
CTIMER3CLKDIV	3DCh
CTIMER4CLKDIV	3E0h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Stable divider clock 1b - Unstable clock frequency
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock has stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.44 PLL1 Clock 0 Divider (PLL1CLK0DIV)

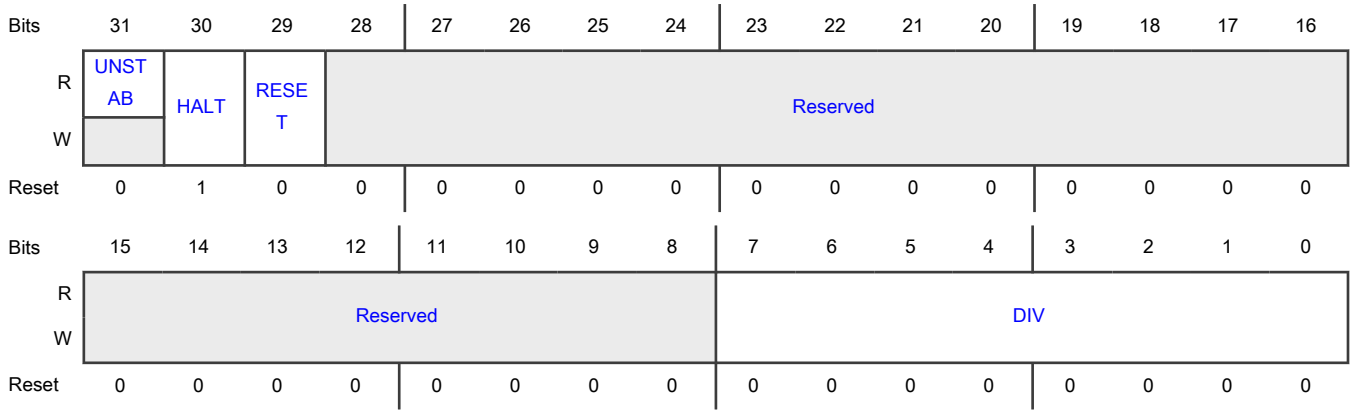
Offset

Register	Offset
PLL1CLK0DIV	3E4h

Function

This register is used to generate pll1_clk0 clock from pll1_clk clock.

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.45 PLL1 Clock 1 Divider (PLL1CLK1DIV)

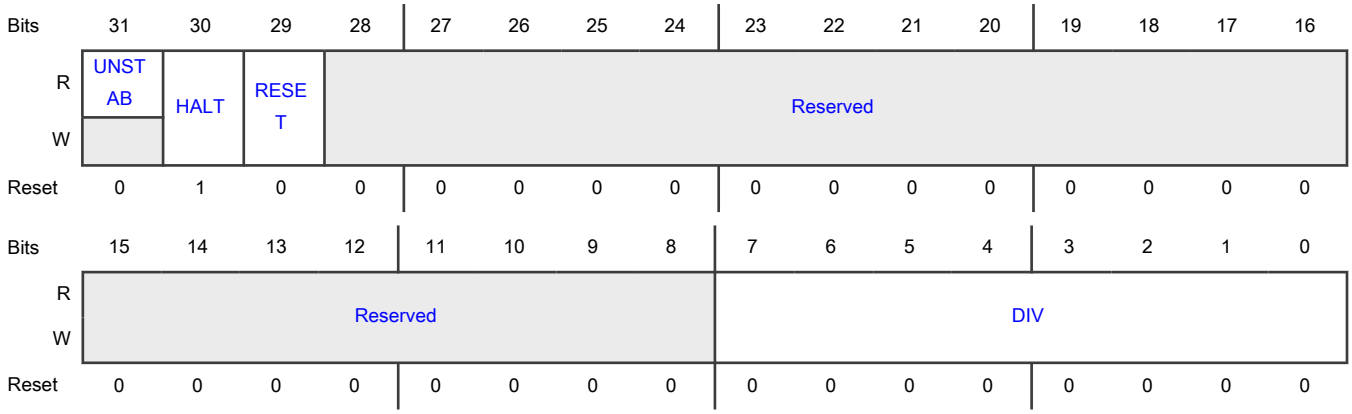
Offset

Register	Offset
PLL1CLK1DIV	3E8h

Function

This register is used to generate pll1_clk1 clock from pll1_clk clock.

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

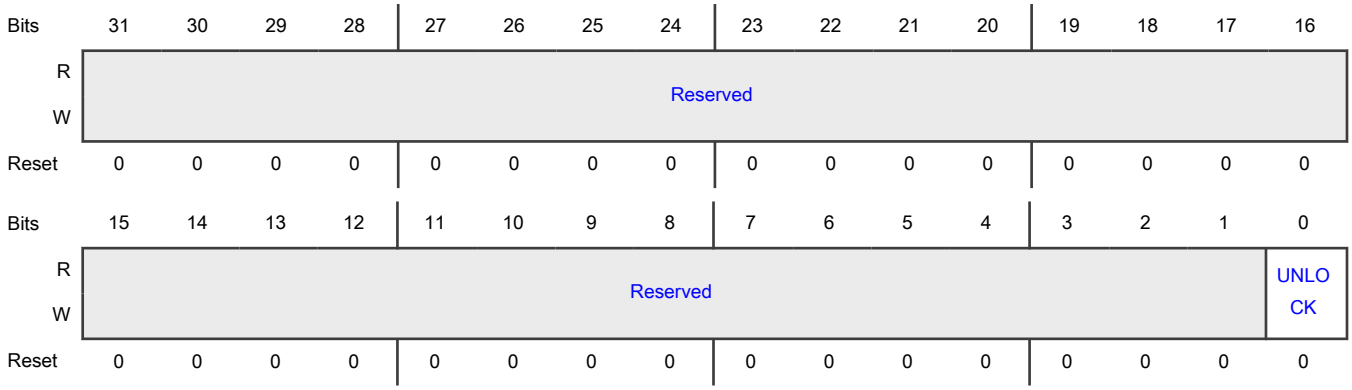
11.4.1.46 Clock Configuration Unlock (CLKUNLOCK)

Offset

Register	Offset
CLKUNLOCK	3FCh

Function
This register controls access to the clock select and divider configuration registers.

Diagram



Fields

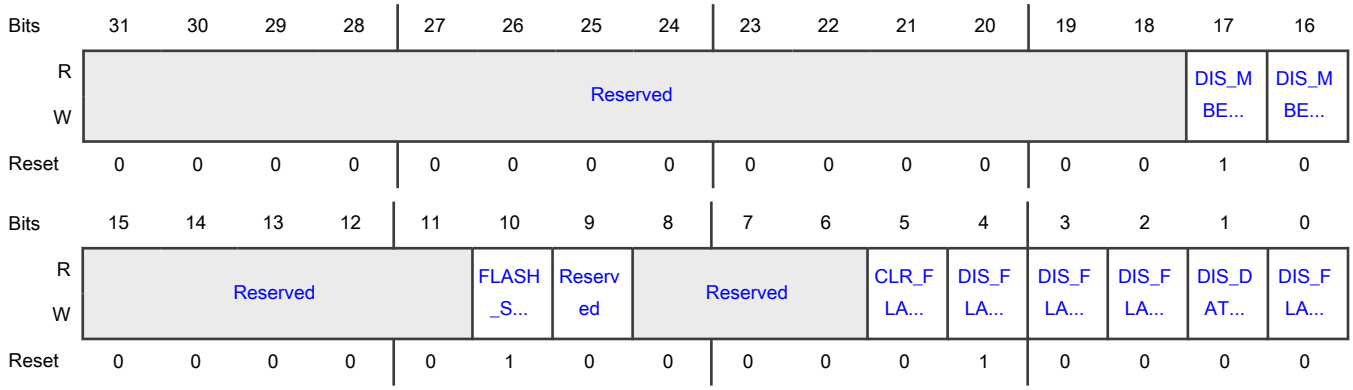
Field	Function
31-1 —	Reserved
0 UNLOCK	Controls clock configuration registers access (for example, xxxDIV, xxxSEL) 0b - Updates are allowed to all clock configuration registers 1b - Freezes all clock configuration registers update

11.4.1.47 NVM Control (NVM_CTRL)

Offset

Register	Offset
NVM_CTRL	400h

Diagram



Fields

Field	Function
31-18 —	Reserved
17 DIS_MBECC_ERR_DATA	Bus error on data multi-bit ECC error control 0b - Enables bus error on multi-bit ECC error for data 1b - Disables bus error on multi-bit ECC error for data
16 DIS_MBECC_ERR_INST	Bus error on instruction multi-bit ECC error control 0b - Enables bus error on multi-bit ECC error for instruction 1b - Disables bus error on multi-bit ECC error for instruction
15-11 —	Reserved
10 FLASH_STALL_EN	FLASH stall on busy control 0b - No stall on FLASH busy 1b - Stall on FLASH busy
9 —	Reserved Keep the default value.
8-6 —	Reserved
5 CLR_FLASH_CACHE	Clear flash cache control 0b - No clear flash cache 1b - Clears flash cache
4	Flash data cache control

Table continues on the next page...

Table continued from the previous page...

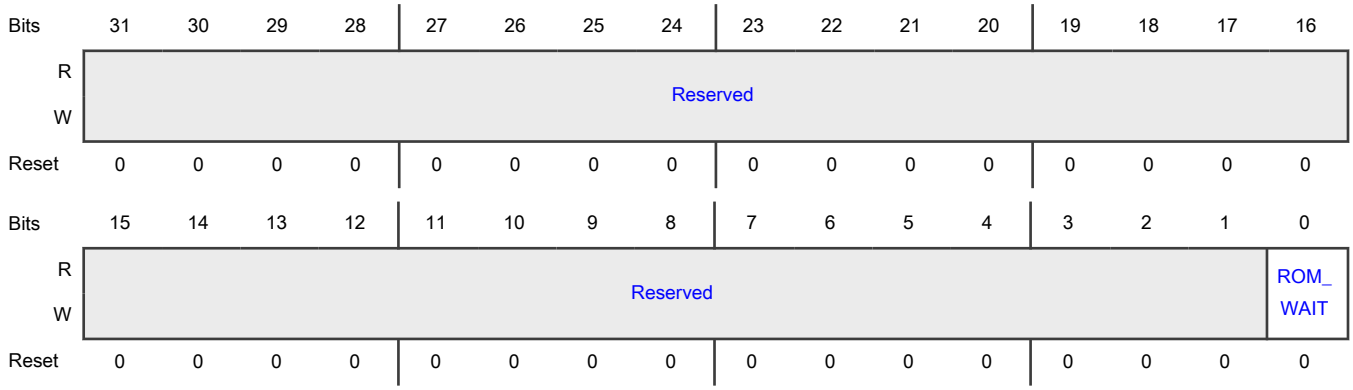
Field	Function
DIS_FLASH_DATA	0b - Enables flash data cache when DIS_FLASH_CACHE=0 1b - Disables flash data cache
3 DIS_FLASH_INST	Flash instruction cache control 0b - Enables flash instruction cache when DIS_FLASH_CACHE=0 1b - Disables flash instruction cache
2 DIS_FLASH_CACHE	Flash cache control 0b - Enables flash cache 1b - Disables flash cache
1 DIS_DATA_SPEC	Flash data speculation control **NOTE** If DIS_MBECC_ERR_DATA and/or DIS_MBECC_ERR_INST are set, then speculation will not be enabled, even if this bit is cleared. 0b - Enables data speculation 1b - Disables data speculation
0 DIS_FLASH_SPEC	Flash speculation control **NOTE** If DIS_MBECC_ERR_DATA and/or DIS_MBECC_ERR_INST are set, then speculation will not be enabled, even if this bit is cleared. 0b - Enables flash speculation 1b - Disables flash speculation

11.4.1.48 ROM Wait State (ROMCR)

Offset

Register	Offset
ROMCR	404h

Diagram



Fields

Field	Function
31-1 —	Reserved
0 ROM_WAIT	ROM waiting Arm core and other masters for one cycle 0b - Disabled 1b - Enabled

11.4.1.49 SmartDMA Interrupt Hijack (SmartDMAINT)

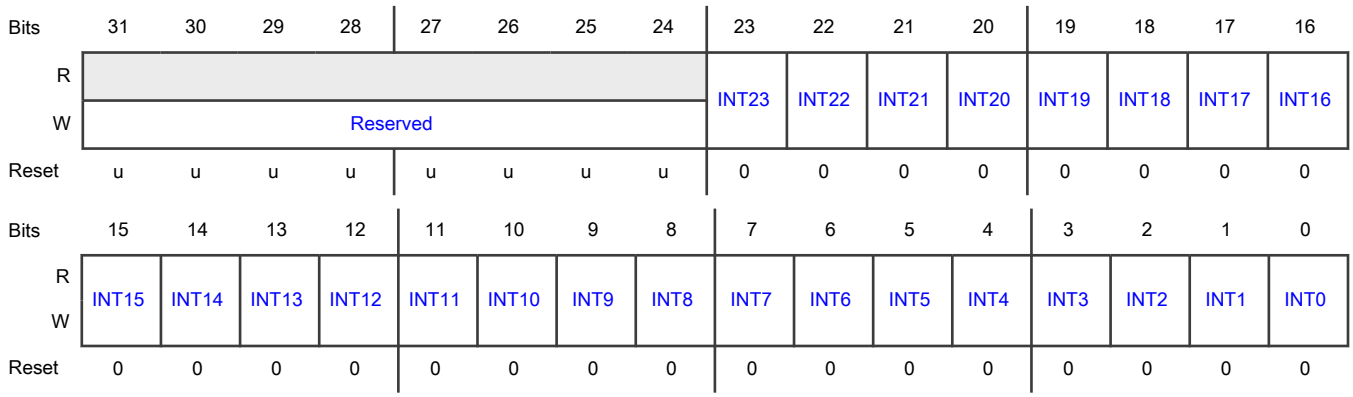
Offset

Register	Offset
SmartDMAINT	414h

Function

Bits directly control the SmartDMA hijacking the system interrupts.

Diagram



Fields

Field	Function
31-24 —	Reserved
23 INT23	SmartDMA hijack NVIC IRQ77 0b - Disable 1b - Enable
22 INT22	SmartDMA hijack NVIC IRQ67 0b - Disable 1b - Enable
21 INT21	SmartDMA hijack NVIC IRQ66 0b - Disable 1b - Enable
20 INT20	SmartDMA hijack NVIC IRQ51 0b - Disable 1b - Enable
19 INT19	SmartDMA hijack NVIC IRQ50 0b - Disable 1b - Enable
18 INT18	SmartDMA hijack NVIC IRQ47 0b - Disable 1b - Enable
17 INT17	SmartDMA hijack NVIC IRQ45 0b - Disable 1b - Enable
16 INT16	SmartDMA hijack NVIC IRQ42 0b - Disable 1b - Enable
15 INT15	SmartDMA hijack NVIC IRQ41 0b - Disable 1b - Enable
14 INT14	SmartDMA hijack NVIC IRQ40 0b - Disable 1b - Enable

Table continues on the next page...

Table continued from the previous page...

Field	Function
13 INT13	SmartDMA hijack NVIC IRQ39 0b - Disable 1b - Enable
12 INT12	SmartDMA hijack NVIC IRQ38 0b - Disable 1b - Enable
11 INT11	SmartDMA hijack NVIC IRQ37 0b - Disable 1b - Enable
10 INT10	SmartDMA hijack NVIC IRQ36 0b - Disable 1b - Enable
9 INT9	SmartDMA hijack NVIC IRQ35 0b - Disable 1b - Enable
8 INT8	SmartDMA hijack NVIC IRQ34 0b - Disable 1b - Enable
7 INT7	SmartDMA hijack NVIC IRQ33 0b - Disable 1b - Enable
6 INT6	SmartDMA hijack NVIC IRQ32 0b - Disable 1b - Enable
5 INT5	SmartDMA hijack NVIC IRQ31 0b - Disable 1b - Enable
4 INT4	SmartDMA hijack NVIC IRQ30 0b - Disable 1b - Enable
3 INT3	SmartDMA hijack NVIC IRQ29

Table continues on the next page...

Table continued from the previous page...

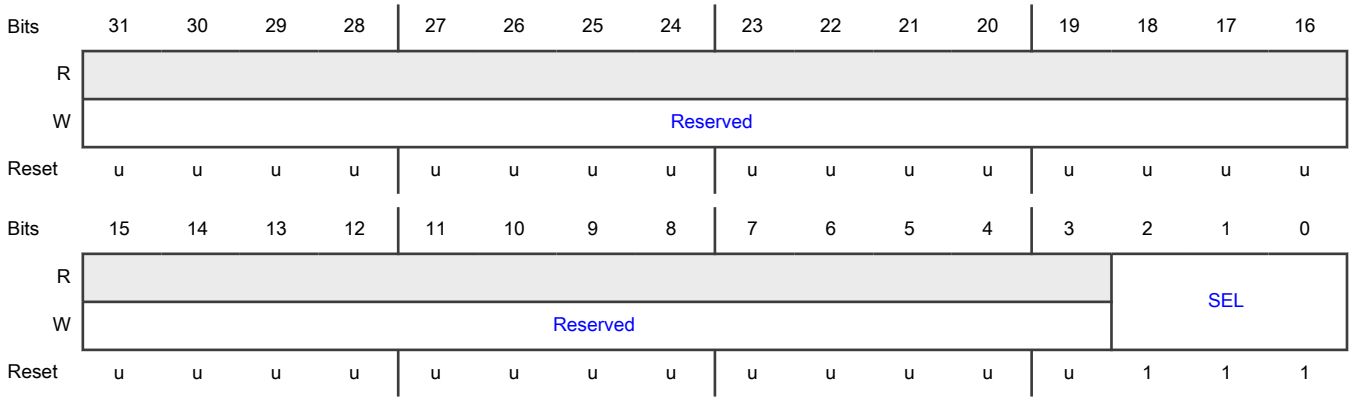
Field	Function
	0b - Disable 1b - Enable
2 INT2	SmartDMA hijack NVIC IRQ18 0b - Disable 1b - Enable
1 INT1	SmartDMA hijack NVIC IRQ17 0b - Disable 1b - Enable
0 INT0	SmartDMA hijack NVIC IRQ1 0b - Disable 1b - Enable

11.4.1.50 ADC1 Clock Source Select (ADC1CLKSEL)

Offset

Register	Offset
ADC1CLKSEL	464h

Diagram



Fields

Field	Function
31-3	Reserved

Table continues on the next page...

Table continued from the previous page...

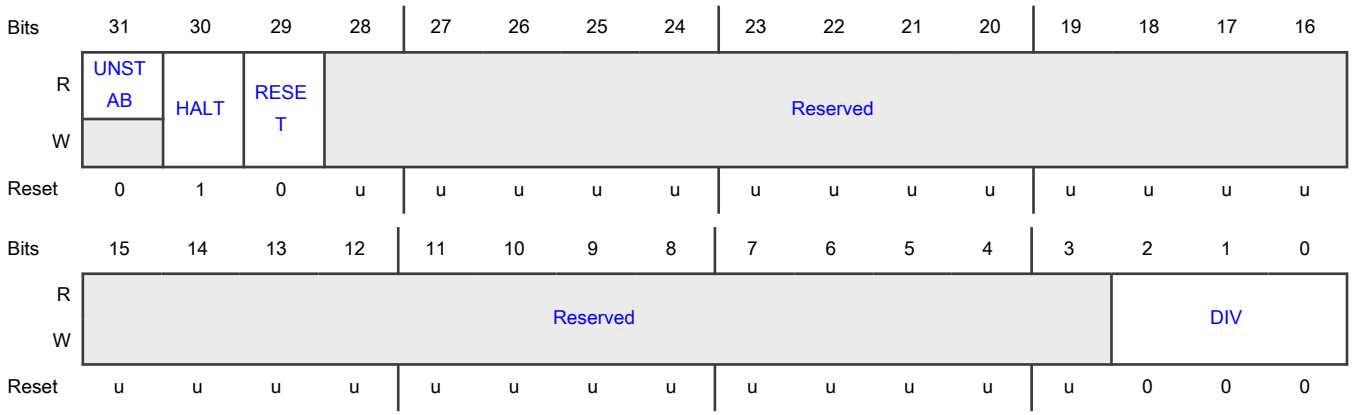
Field	Function
—	
2-0 SEL	Selects the ADC1 clock source 000b - No clock 001b - PLL0 clock 010b - FRO_HF clock 011b - FRO 12 MHz clock 100b - Clk_in clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.51 ADC1 Clock Divider (ADC1CLKDIV)

Offset

Register	Offset
ADC1CLKDIV	468h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable

Table continues on the next page...

Table continued from the previous page...

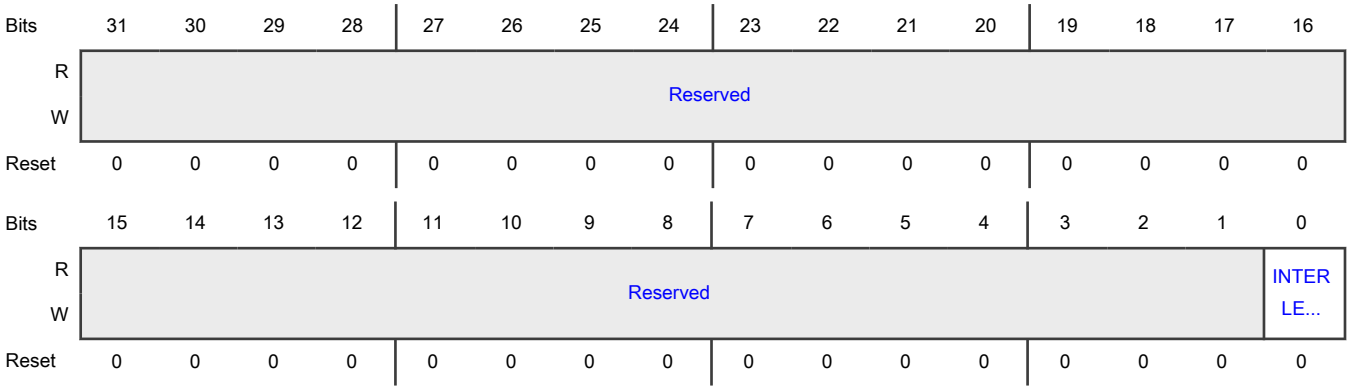
Field	Function
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-3 —	Reserved
2-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.52 Control PKC RAM Interleave Access (RAM_INTERLEAVE)

Offset

Register	Offset
RAM_INTERLEAVE	470h

Diagram



Fields

Field	Function
31-1 —	Reserved

Table continues on the next page...

Table continued from the previous page...

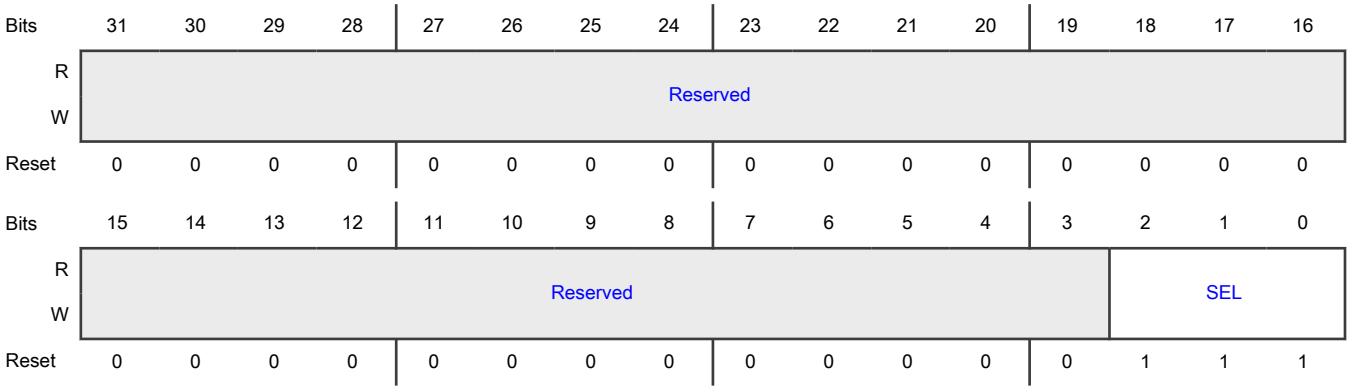
Field	Function
0 INTERLEAVE	Controls PKC RAM access for PKC RAM 0 and PKC RAM 1 0b - RAM access to PKC RAM 0 and PKC RAM 1 is consecutive. 1b - RAM access to PKC RAM 0 and PKC RAM 1 is interleaved. This setting is need for PKC L0 memory access.

11.4.1.53 DACn Functional Clock Selection (DAC0CLKSEL - DAC2CLKSEL)

Offset

Register	Offset
DAC0CLKSEL	490h
DAC1CLKSEL	498h
DAC2CLKSEL	4A0h

Diagram



Fields

Field	Function
31-3 —	Reserved
2-0 SEL	Selects the DAC clock source 000b - No clock 001b - PLL0 clock 010b - Clk_in 011b - FRO_HF

Table continues on the next page...

Table continued from the previous page...

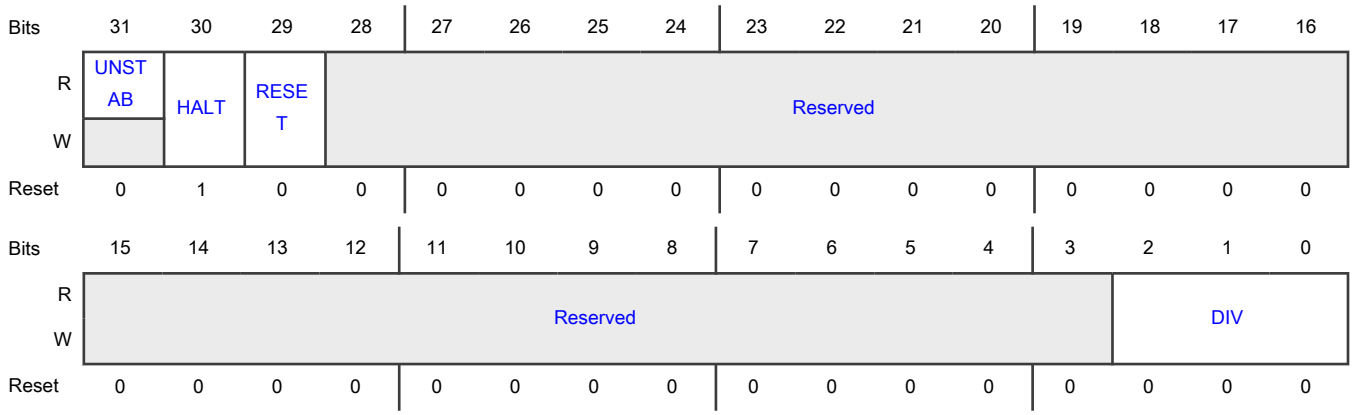
Field	Function
	100b - FRO_12M 101b - PLL1_clk0 clock 110b - No clock 111b - No clock

11.4.1.54 DACn functional clock divider (DAC0CLKDIV - DAC2CLKDIV)

Offset

Register	Offset
DAC0CLKDIV	494h
DAC1CLKDIV	49Ch
DAC2CLKDIV	4A4h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped

Table continues on the next page...

Table continued from the previous page...

Field	Function
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-3 —	Reserved
2-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.55 FlexSPI Clock Selection (FlexSPICLKSEL)

Offset

Register	Offset
FlexSPICLKSEL	4A8h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved												SEL			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Fields

Field	Function
31-4 —	Reserved
3-0 SEL	Selects the FlexSPI clock 0000b - No clock 0001b - PLL0 clock

Table continues on the next page...

Table continued from the previous page...

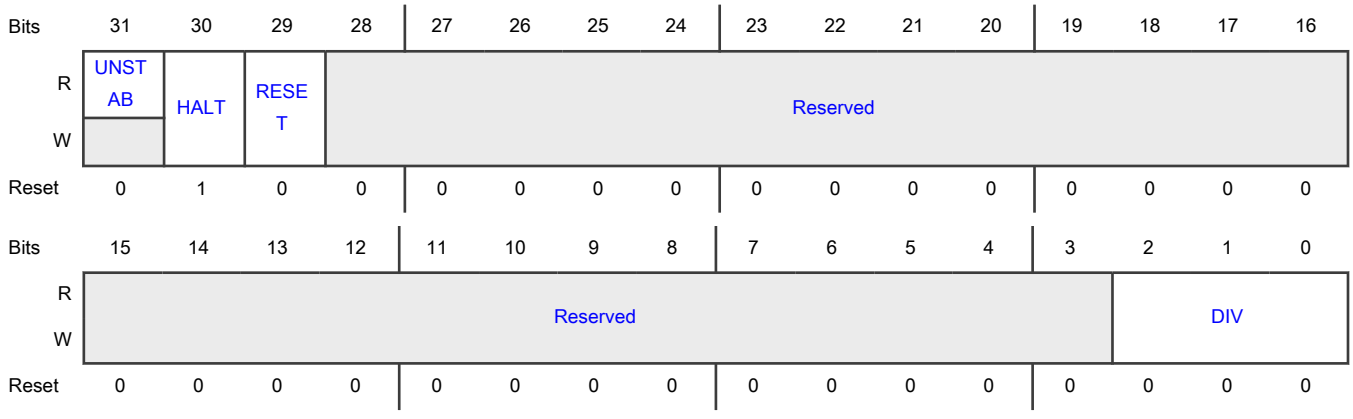
Field	Function
	0010b - No clock 0011b - FRO_HF 0100b - No clock 0101b - pll1_clock 0110b - USB PLL clock 0111b - No clock 1000b - No clock 1001b - No clock 1010b - No clock 1011b - No clock 1100b - No clock 1101b - No clock 1110b - No clock 1111b - No clock

11.4.1.56 FlexSPI Clock Divider (FlexSPICLKDIV)

Offset

Register	Offset
FlexSPICLKDIV	4ACh

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-3 —	Reserved
2-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.57 PLL Clock Divider Clock Selection (PLLCLKDIVSEL)

Offset

Register	Offset
PLLCLKDIVSEL	52Ch

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

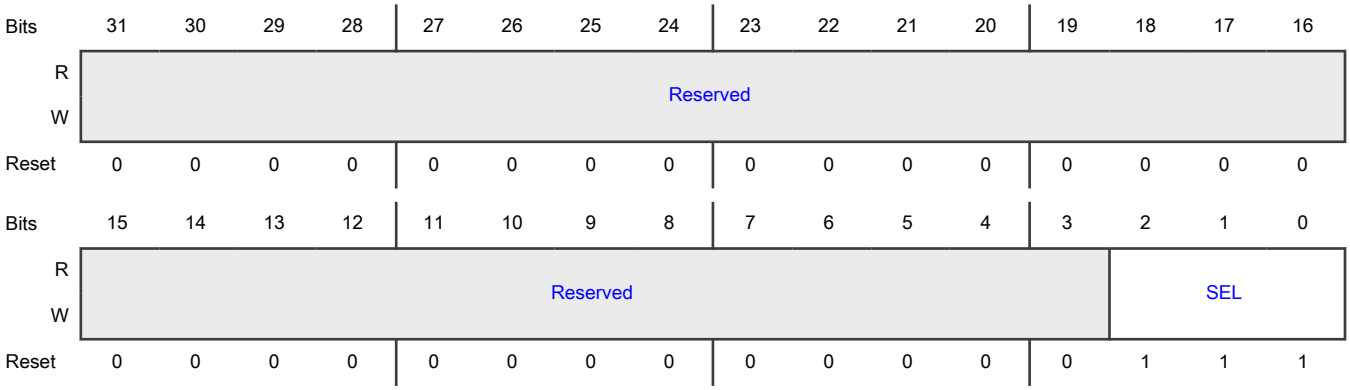
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the PLL Clock Divider source clock 000b - PLL0 clock 001b - pll1_clk0 010b - No clock 011b - No clock 100b - No clock 101b - No clock 110b - No clock 111b - No clock

11.4.1.58 I3C0 Functional Clock Selection (I3C0FCLKSEL)

Offset

Register	Offset
I3C0FCLKSEL	530h

Diagram



Fields

Field	Function
31-3 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
2-0 SEL	Selects the I3C0 clock 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - FRO_HF clock 100b - No clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.59 I3C0 FCLK_STC Clock Selection (I3C0FCLKSTCSEL)

Offset

Register	Offset
I3C0FCLKSTCSEL	534h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

Field	Function
31-3 —	Reserved
2-0 SEL	Selects the I3C0 Time Control clock 000b - I3C0 functional clock I3C0FCLK

Table continues on the next page...

Table continued from the previous page...

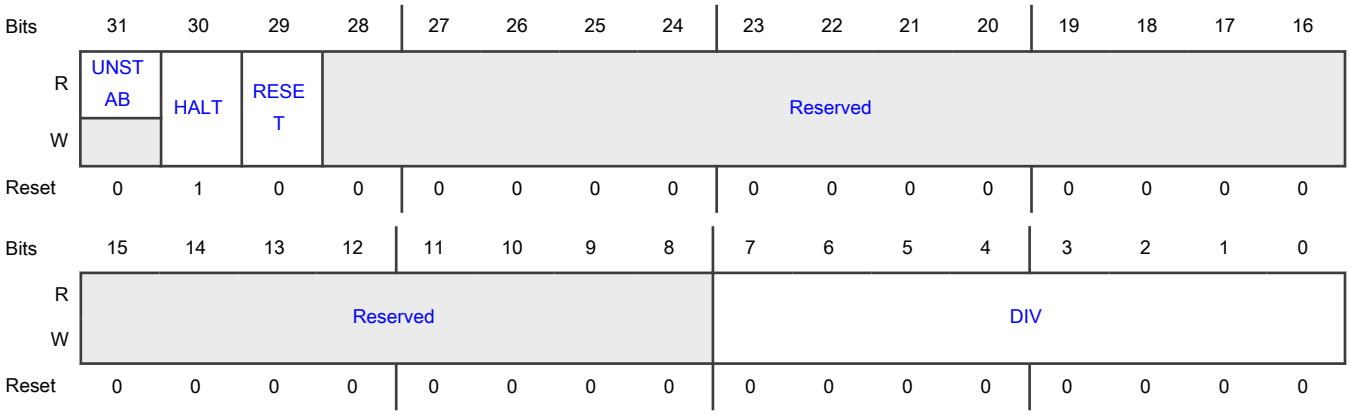
Field	Function
	001b - FRO_1M clock 010b - No clock 011b - No clock 100b - No clock 101b - No clock 110b - No clock 111b - No clock

11.4.1.60 I3C0 FCLK_STC Clock Divider (I3C0FCLKSTCDIV)

Offset

Register	Offset
I3C0FCLKSTCDIV	538h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped

Table continues on the next page...

Table continued from the previous page...

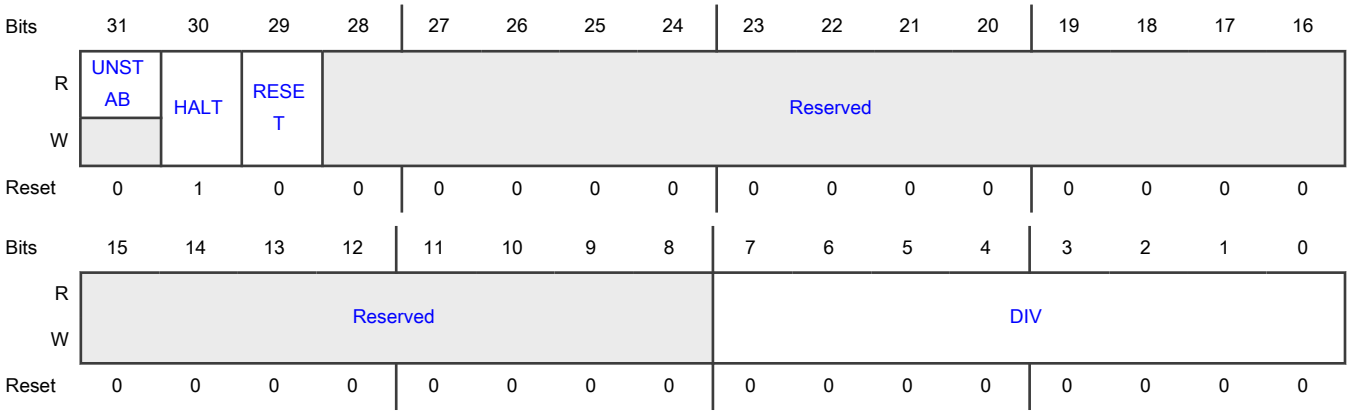
Field	Function
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.61 I3C0 FCLK Slow Clock Divider (I3C0FCLKSDIV)

Offset

Register	Offset
I3C0FCLKSDIV	53Ch

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running

Table continues on the next page...

Table continued from the previous page...

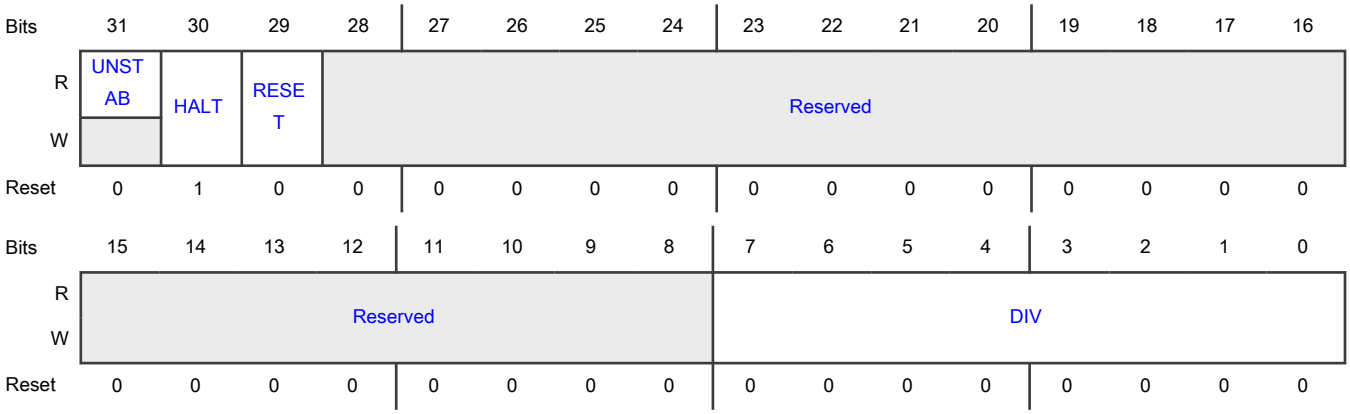
Field	Function
	1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.62 I3C0 Functional Clock FCLK Divider (I3C0FCLKDIV)

Offset

Register	Offset
I3C0FCLKDIV	540h

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30	Halts the divider counter

Table continues on the next page...

Table continued from the previous page...

Field	Function
HALT	0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.63 I3C0 FCLK Slow Selection (I3C0FCLKSSEL)

Offset

Register	Offset
I3C0FCLKSSEL	544h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved														SEL	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-3 —	Reserved
2-0	Selects the I3C FCLK Slow clock

Table continues on the next page...

Table continued from the previous page...

Field	Function
SEL	000b - FRO_1M clock 001b - No clock 010b - No clock 011b - No clock 100b - No clock 101b - No clock 110b - No clock 111b - No clock

11.4.1.64 MICFIL Clock Selection (MICFILFCLKSEL)

Offset

Register	Offset
MICFILFCLKSEL	548h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved												SEL			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Fields

Field	Function
31-4 —	Reserved
3-0 SEL	Selects the MICFIL clock 0000b - FRO_12M clock

Table continues on the next page...

Table continued from the previous page...

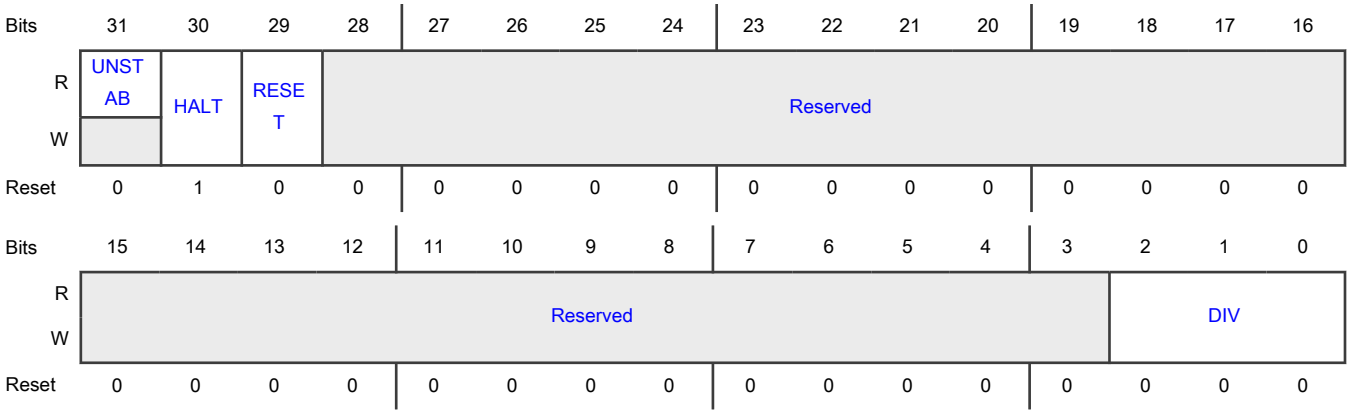
Field	Function
	0001b - PLL0 clock 0010b - CLKIN clock 0011b - FRO_HF clock 0100b - PLL1_clk0 clock 0101b - SAI0_MCLK clock 0110b - USB PLL clock 0111b - No clock 1000b - SAI1_MCLK clock 1001b - No clock 1010b - No clock 1011b - No clock 1100b - No clock 1101b - No clock 1110b - No clock 1111b - No clock

11.4.1.65 MICFIL Clock Division (MICFILFCLKDIV)

Offset

Register	Offset
MICFILFCLKDIV	54Ch

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-3 —	Reserved
2-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.66 uSDHC Clock Selection (uSDHCCLKSEL)

Offset

Register	Offset
uSDHCCLKSEL	558h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

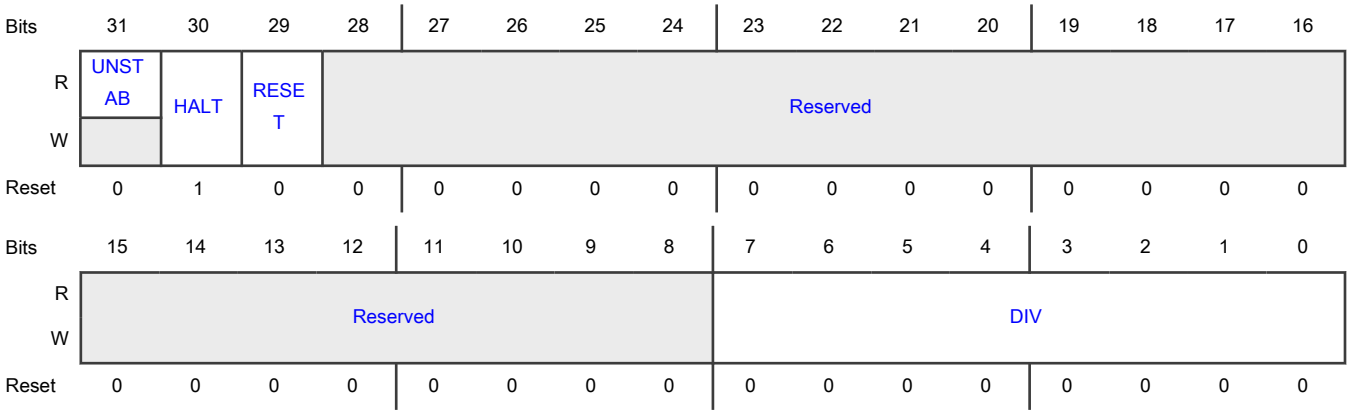
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the uSDHC clock 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - FRO_HF clock 100b - FRO_12M clock 101b - pll1_clk1 clock 110b - USB PLL clock 111b - No clock

11.4.1.67 uSDHC Function Clock Divider (uSDHCCLKDIV)

Offset

Register	Offset
uSDHCCLKDIV	55Ch

Diagram



Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

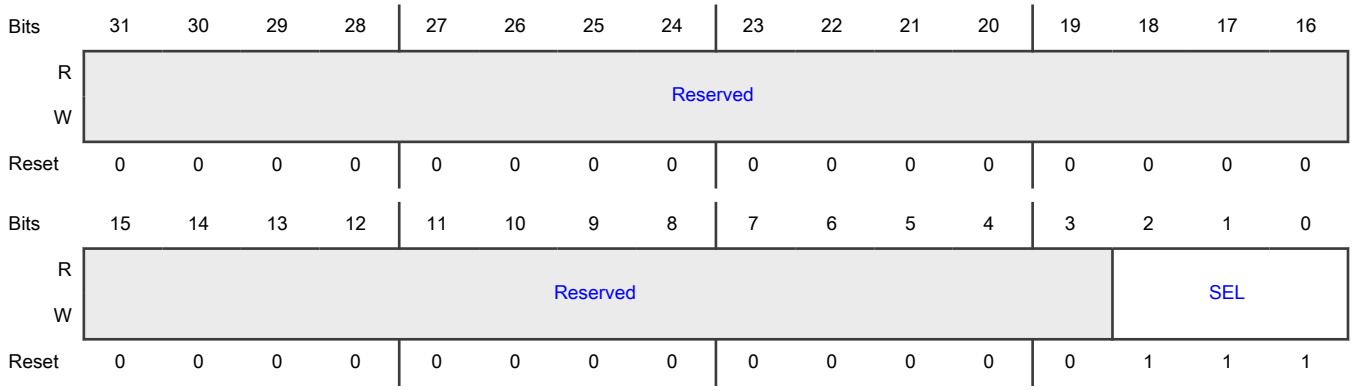
Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.68 FLEXIO Clock Selection (FLEXIOCLKSEL)

Offset

Register	Offset
FLEXIOCLKSEL	560h

Diagram



Fields

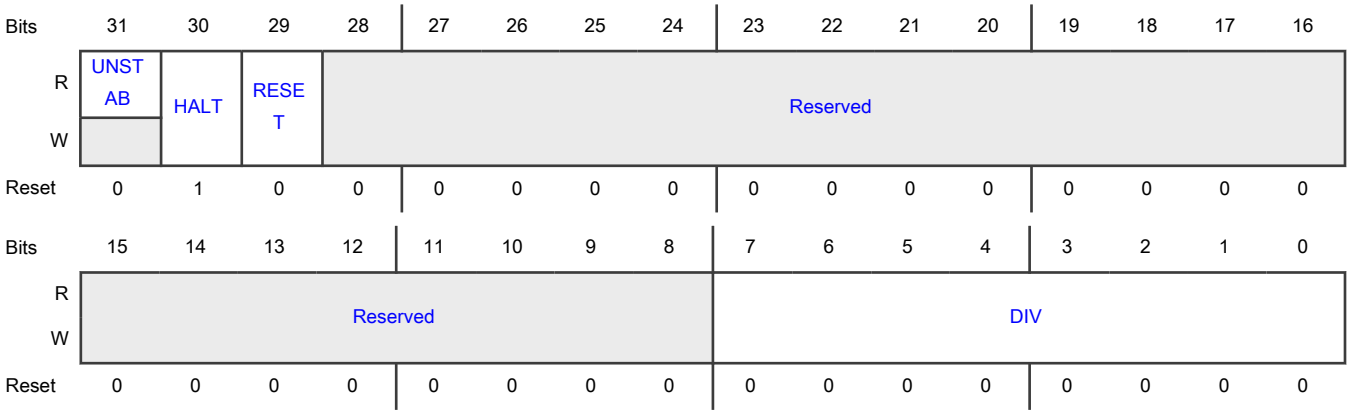
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the FLEXIO clock 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - FRO_HF clock 100b - FRO_12M clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.69 FLEXIO Function Clock Divider (FLEXIOCLKDIV)

Offset

Register	Offset
FLEXIOCLKDIV	564h

Diagram



Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.70 FLEXCAN0 Clock Selection (FLEXCAN0CLKSEL)

Offset

Register	Offset
FLEXCAN0CLKSEL	5A0h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

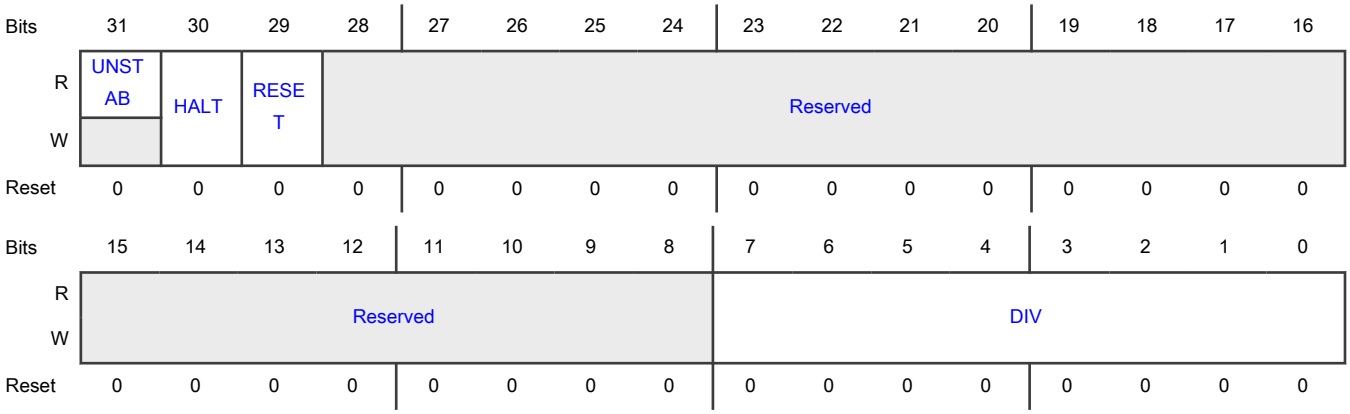
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the FLEXCAN0 clock 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - FRO_HF clock 100b - No clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.71 FLEXCAN0 Function Clock Divider (FLEXCAN0CLKDIV)

Offset

Register	Offset
FLEXCAN0CLKDIV	5A4h

Diagram



Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.72 FLEXCAN1 Clock Selection (FLEXCAN1CLKSEL)

Offset

Register	Offset
FLEXCAN1CLKSEL	5A8h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

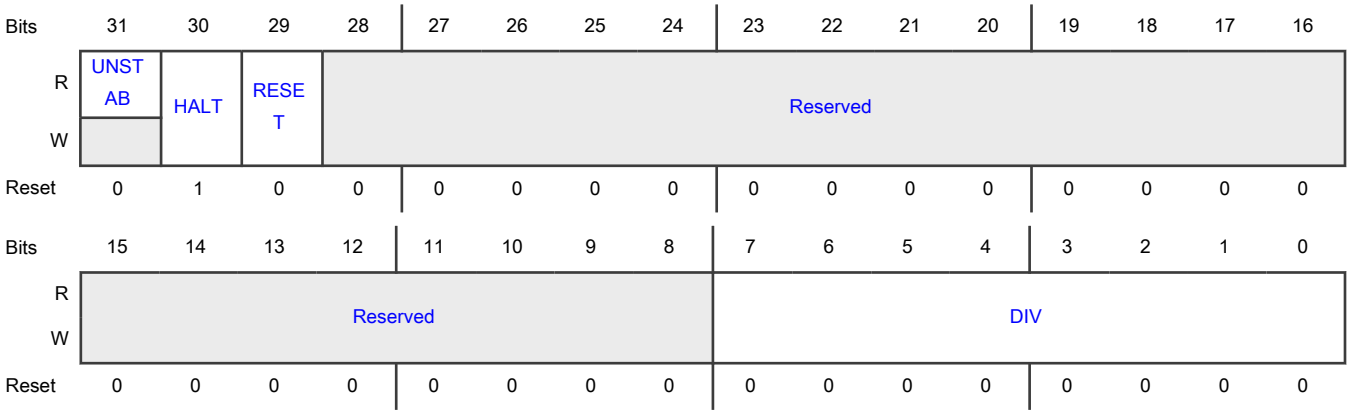
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the FLEXCAN1 clock 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - FRO_HF clock 100b - No clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.73 FLEXCAN1 Function Clock Divider (FLEXCAN1CLKDIV)

Offset

Register	Offset
FLEXCAN1CLKDIV	5ACh

Diagram



Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.74 Ethernet RMII Clock Selection (ENETRMIICLKSEL)

Offset

Register	Offset
ENETRMIICLKSEL	5B0h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

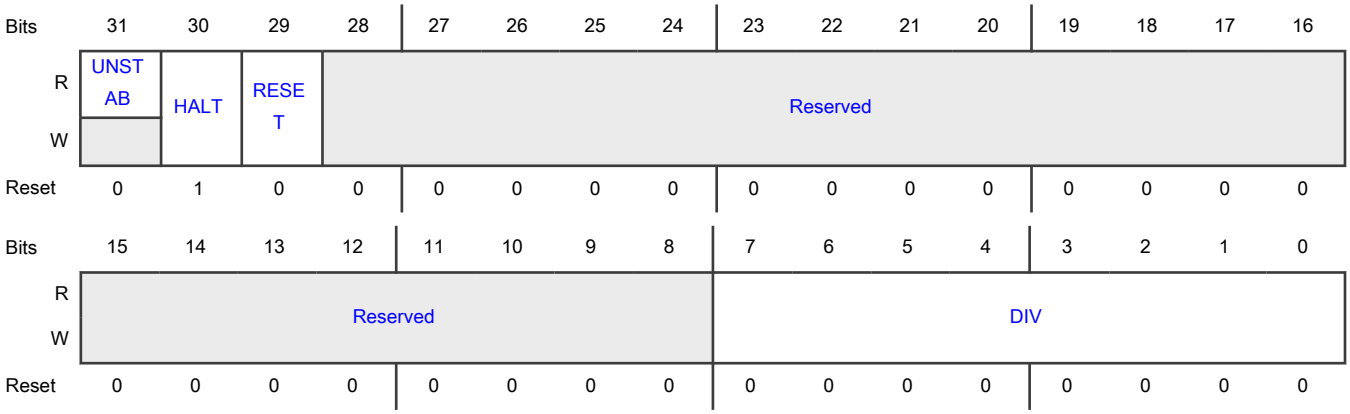
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the Ethernet RMII clock 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - No clock 100b - No clock 101b - PLL1_clk0 clock 110b - No clock 111b - No clock

11.4.1.75 Ethernet RMII Function Clock Divider (ENETRMIICLKDIV)

Offset

Register	Offset
ENETRMIICLKDIV	5B4h

Diagram



Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.76 Ethernet PTP REF Clock Selection (ENETPTPREFCLKSEL)

Offset

Register	Offset
ENETPTPREFCLKSEL	5B8h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

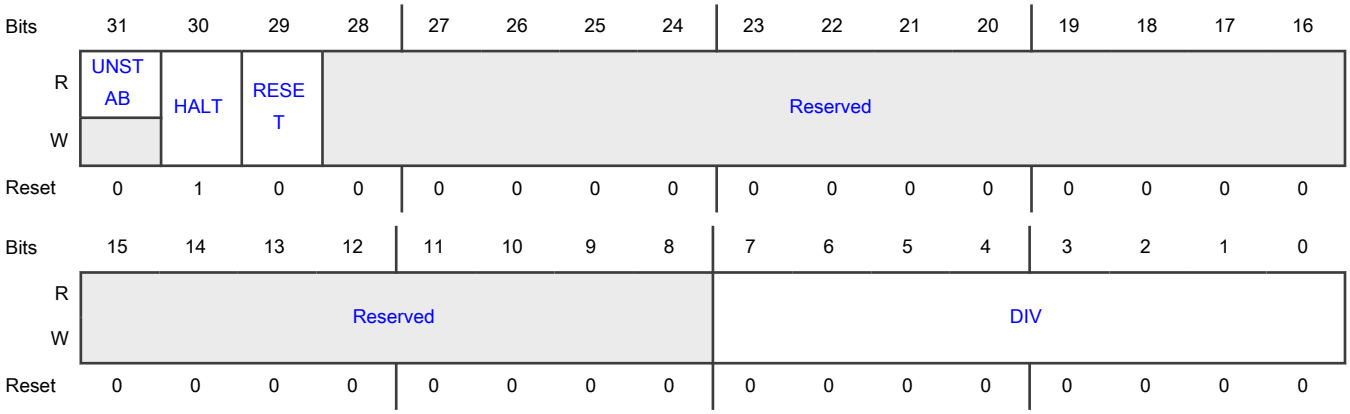
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the Ethernet PTP REF clock 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - No clock 100b - enet0_tx_clk clock 101b - pll1_clk1 clock 110b - No clock 111b - No clock

11.4.1.77 Ethernet PTP REF Function Clock Divider (ENETPTPREFCLKDIV)

Offset

Register	Offset
ENETPTPREFCLKDIV	5BCh

Diagram



Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

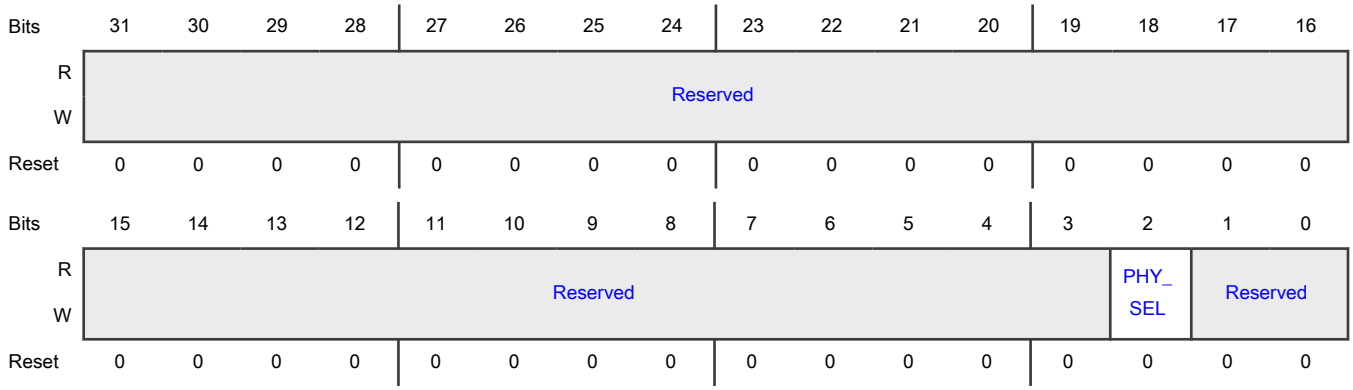
Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.78 Ethernet PHY Interface Select (ENET_PHY_INTF_SEL)

Offset

Register	Offset
ENET_PHY_INTF_SEL	5C0h

Diagram



Fields

Field	Function
31-3 —	Reserved
2 PHY_SEL	Selects the PHY interface 0b - Selects MII PHY Interface 1b - Selects RMII PHY Interface
1-0 —	Reserved

11.4.1.79 Sideband Flow Control (ENET_SBD_FLOW_CTRL)

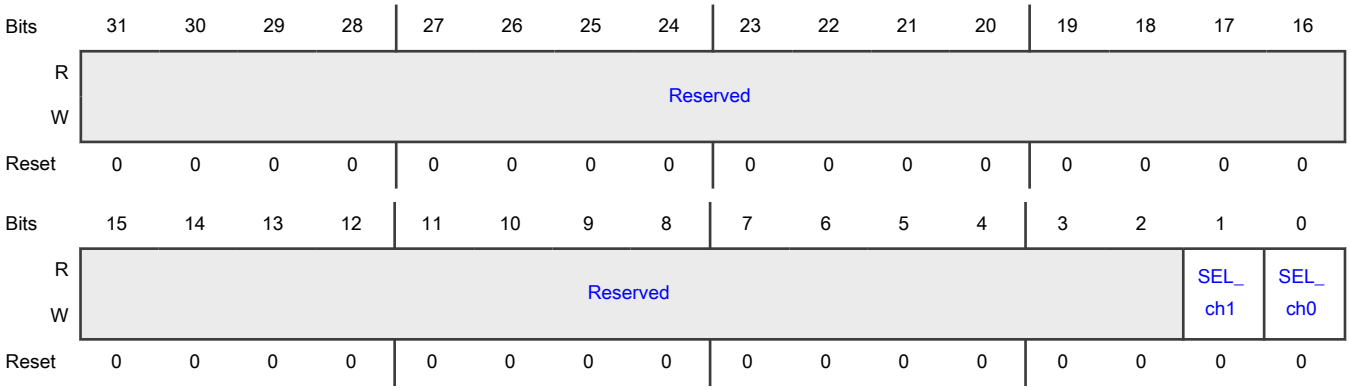
Offset

Register	Offset
ENET_SBD_FLOW_CTRL	5C4h

Function

When set high, instructs the MAC to transmit Pause frames in the Full-duplex mode. In the half-duplex mode, the MAC enables the back-pressure function until this signal is made low again.

Diagram



Fields

Field	Function
31-2 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
1 SEL_ch1	Sideband Flow Control for channel1 0b - No trigger flow control 1b - Trigger flow control
0 SEL_ch0	Sideband Flow Control for channel0 0b - No trigger flow control 1b - Trigger flow control

11.4.1.80 EWM0 Clock Selection (EWM0CLKSEL)

Offset

Register	Offset
EWM0CLKSEL	5D4h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved															SEL
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Fields

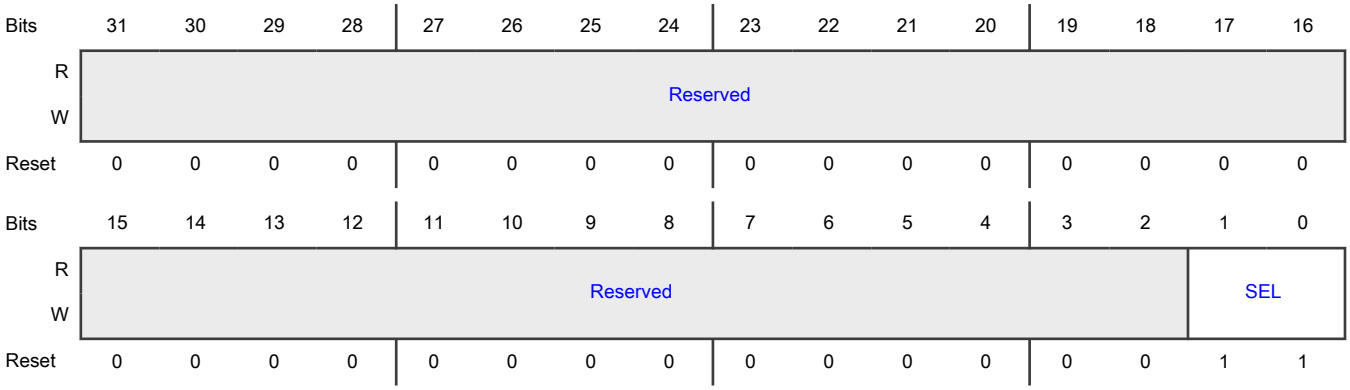
Field	Function
31-1 —	Reserved
0 SEL	Selects the EWM0 clock 0b - clk_16k[2] 1b - xtal32k[2]

11.4.1.81 WDT1 Clock Selection (WDT1CLKSEL)

Offset

Register	Offset
WDT1CLKSEL	5D8h

Diagram



Fields

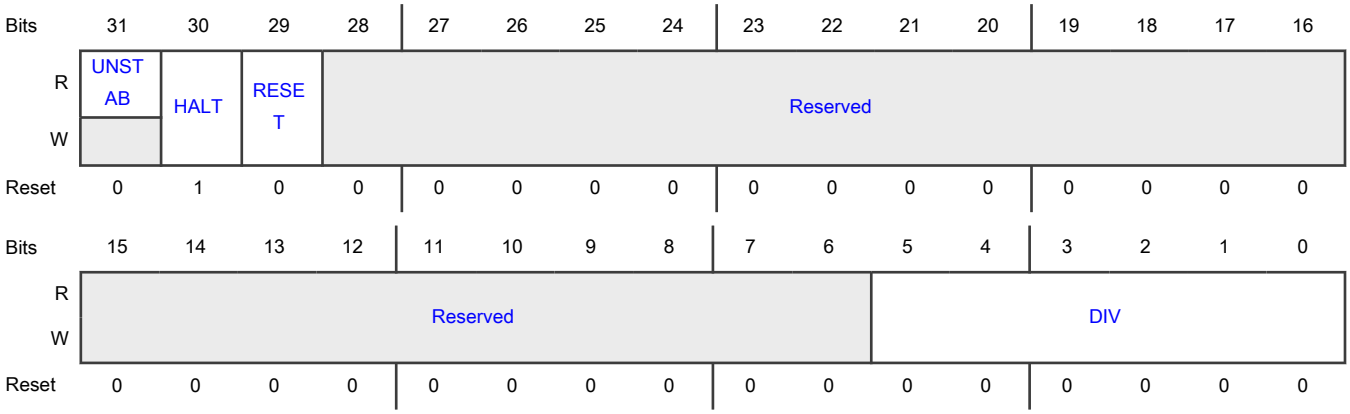
Field	Function
31-2 —	Reserved
1-0 SEL	Selects the WDT1 clock 00b - FRO16K clock 2 01b - fro_hf_div clock 10b - clk_1m clock 11b - clk_1m clock

11.4.1.82 WDT1 Function Clock Divider (WDT1CLKDIV)

Offset

Register	Offset
WDT1CLKDIV	5DCh

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-6 —	Reserved
5-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.83 OSTIMER Clock Selection (OSTIMERCLKSEL)

Offset

Register	Offset
OSTIMERCLKSEL	5E0h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved														SEL	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Fields

Field	Function
31-2 —	Reserved
1-0 SEL	Selects the OS Event Timer clock 00b - clk_16k[2] 01b - xtal32k[2] 10b - clk_1m clock 11b - No clock

11.4.1.84 CMP0 Function Clock Selection (CMP0FCLKSEL)

Offset

Register	Offset
CMP0FCLKSEL	5F0h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved														SEL	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

Field	Function
31-3 —	Reserved
2-0 SEL	Selects the CMP0 function clock 000b - No clock 001b - PLL0 clock 010b - FRO_HF clock 011b - FRO_12M clock 100b - CLKIN clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.85 CMP0 Function Clock Divider (CMP0FCLKDIV)

Offset

Register	Offset
CMP0FCLKDIV	5F4h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	UNSTAB	HALT	RESET	Reserved												
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved												DIV			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-4 —	Reserved
3-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.86 CMP0 Round Robin Clock Selection (CMP0RRCLKSEL)

Offset

Register	Offset
CMP0RRCLKSEL	5F8h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

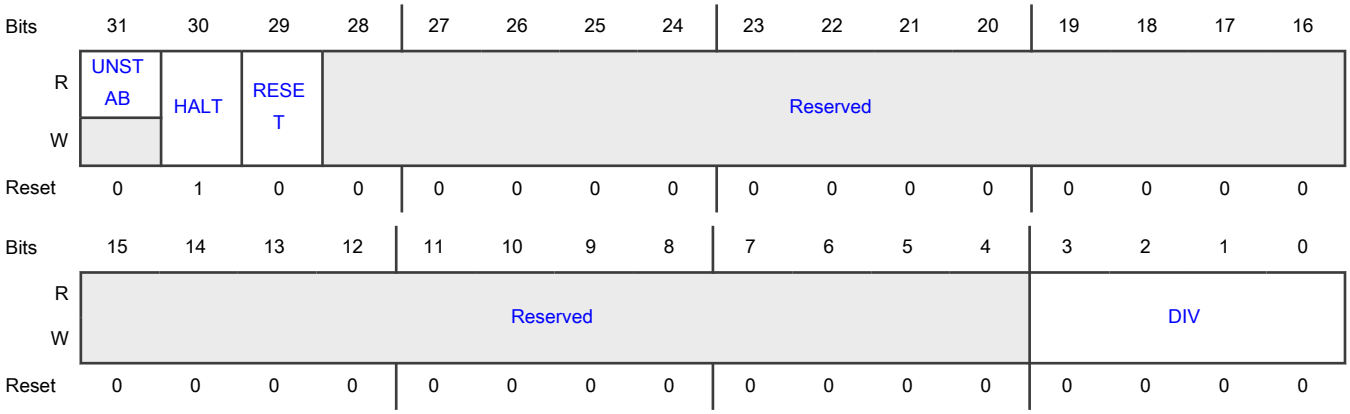
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the CMP0 round robin clock 000b - No clock 001b - PLL0 clock 010b - FRO_HF clock 011b - FRO_12M clock 100b - CLKIN clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.87 CMP0 Round Robin Clock Divider (CMP0RRCLKDIV)

Offset

Register	Offset
CMP0RRCLKDIV	5FCh

Diagram



Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-4 —	Reserved
3-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.88 CMP1 Function Clock Selection (CMP1FCLKSEL)

Offset

Register	Offset
CMP1FCLKSEL	600h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

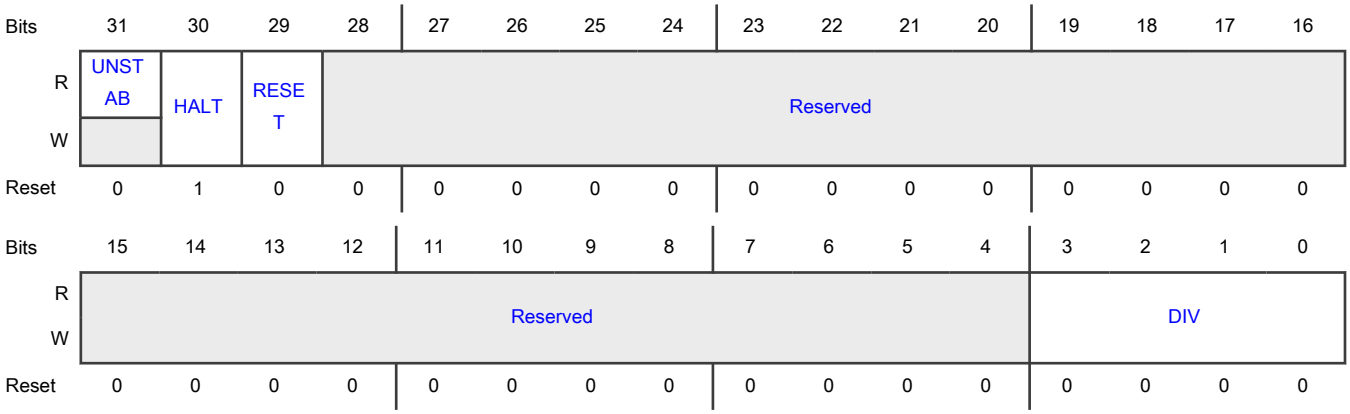
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the CMP1 function clock 000b - No clock 001b - PLL0 clock 010b - FRO_HF clock 011b - FRO_12M clock 100b - CLKIN clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.89 CMP1 Function Clock Divider (CMP1FCLKDIV)

Offset

Register	Offset
CMP1FCLKDIV	604h

Diagram



Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-4 —	Reserved
3-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.90 CMP1 Round Robin Clock Source Select (CMP1RRCLKSEL)

Offset

Register	Offset
CMP1RRCLKSEL	608h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

Field	Function
31-3 —	Reserved
2-0 SEL	Selects the CMP1 round robin clock 000b - No clock 001b - PLL0 clock 010b - FRO_HF clock 011b - FRO_12M clock 100b - CLKIN clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.91 CMP1 Round Robin Clock Division (CMP1RRCLKDIV)

Offset

Register	Offset
CMP1RRCLKDIV	60Ch

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	UNSTAB	HALT	RESET	Reserved												
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved												DIV			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-4 —	Reserved
3-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.92 CMP2 Function Clock Source Select (CMP2FCLKSEL)

Offset

Register	Offset
CMP2FCLKSEL	610h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved													SEL		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

Field	Function
31-3 —	Reserved
2-0 SEL	Selects the CMP2 function clock 000b - No clock 001b - PLL0 clock 010b - FRO_HF clock 011b - FRO_12M clock 100b - CLKIN clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.93 CMP2 Function Clock Division (CMP2FCLKDIV)

Offset

Register	Offset
CMP2FCLKDIV	614h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	UNSTAB	HALT	RESET	Reserved												
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved												DIV			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

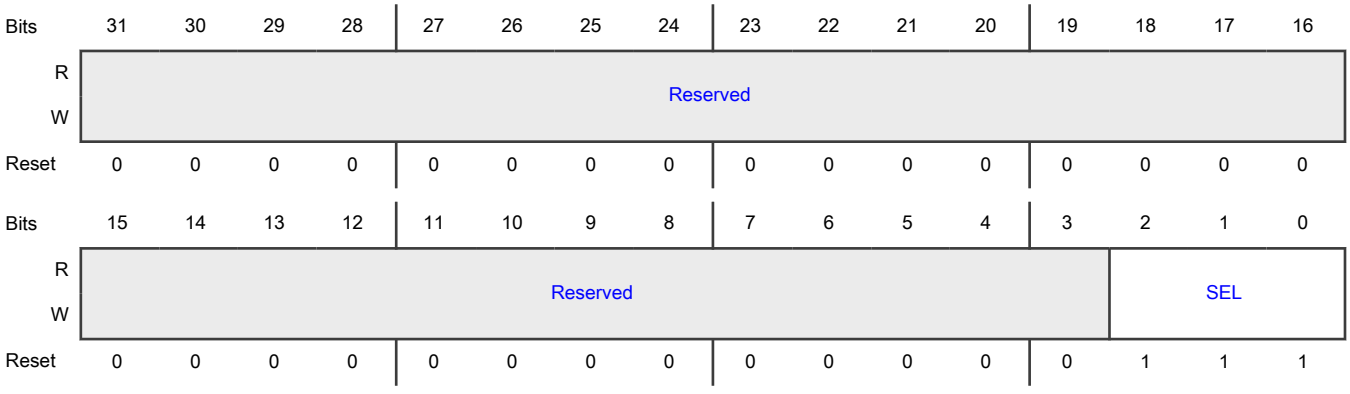
Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-4 —	Reserved
3-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.94 CMP2 Round Robin Clock Source Select (CMP2RRCLKSEL)

Offset

Register	Offset
CMP2RRCLKSEL	618h

Diagram



Fields

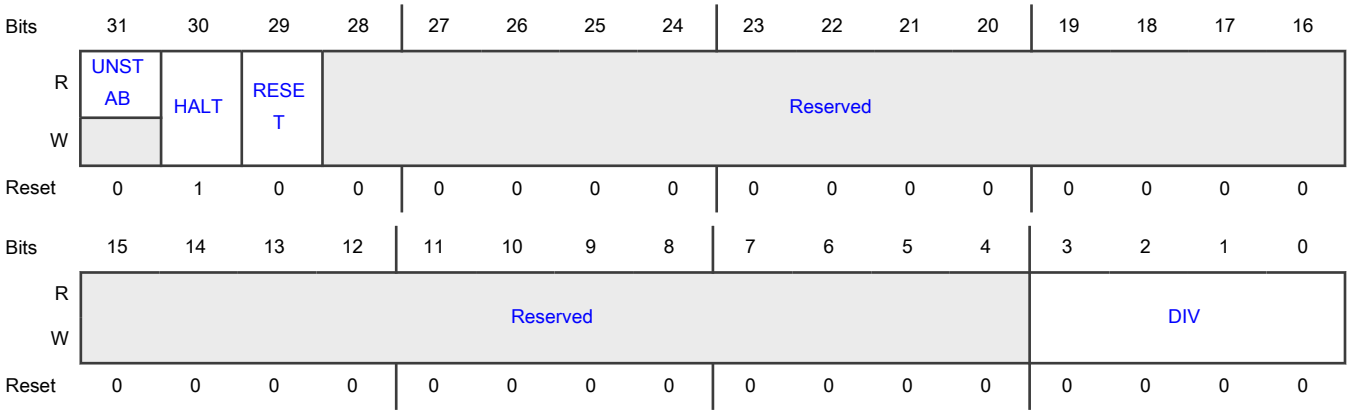
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the CMP2 round robin clock 000b - No clock 001b - PLL0 clock 010b - FRO_HF clock 011b - FRO_12M clock 100b - CLKIN clock 101b - PLL1_clk0 clock0 110b - USB PLL clock 111b - No clock

11.4.1.95 CMP2 Round Robin Clock Division (CMP2RRCLKDIV)

Offset

Register	Offset
CMP2RRCLKDIV	61Ch

Diagram



Fields

Field	Function
31	Divider status flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
UNSTAB	0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-4 —	Reserved
3-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.96 CPU Control for Multiple Processors (CPUCTRL)

Offset

Register	Offset
CPUCTRL	800h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	PROT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved										CPU1 RST...	Reserved	CPU1 CLK...	Reserved		
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0

Fields

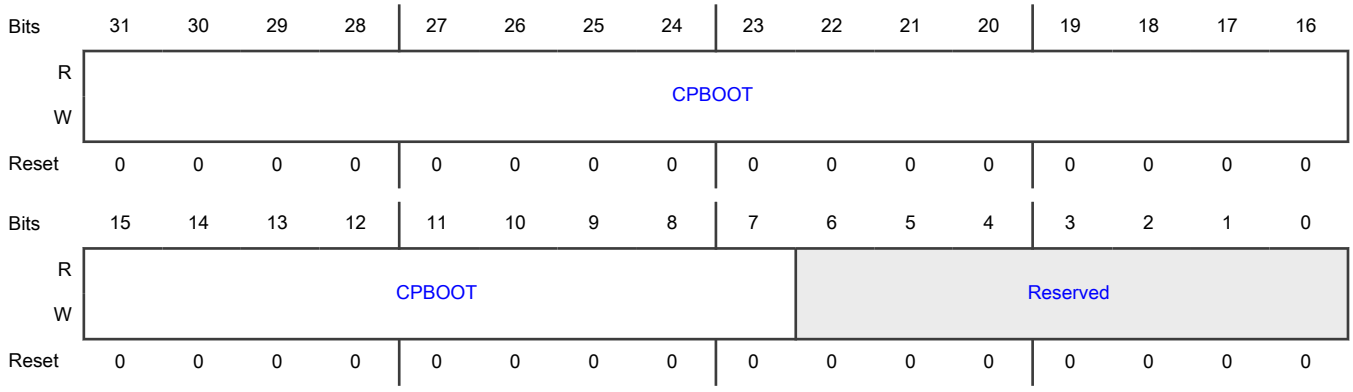
Field	Function
31-16 PROT	Write Protect Must be written as 0xC0C4 for the write to have an effect 1100_0000_1100_0100b - For write operation to have an effect.
15-6 —	Reserved
5 CPU1RSTEN	CPU1 reset Controls the reset signal to CPU1. This bit can be cleared to release CPU1 from reset after a valid boot address has been written to CPBOOT. 0b - The CPU1 is not reset. 1b - The CPU1 is reset.
4 —	Reserved
3 CPU1CLKEN	Enables the CPU1 clock 0b - The CPU1 clock is not enabled 1b - The CPU1 clock is enabled
2-0 —	Reserved

11.4.1.97 Coprocessor Boot Address (CPBOOT)

Offset

Register	Offset
CPBOOT	804h

Diagram



Fields

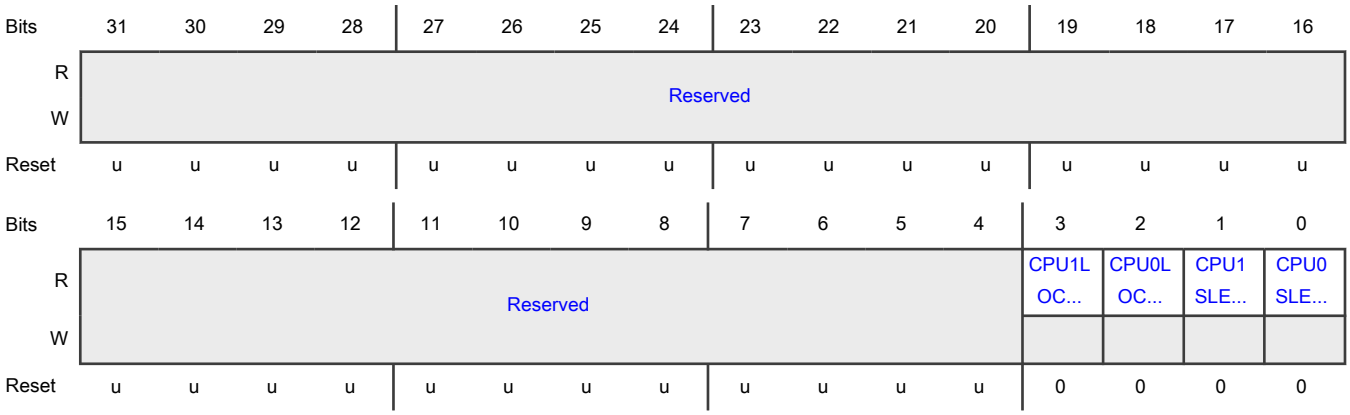
Field	Function
31-7 CPBOOT	Coprocessor Boot VTOR Address [31:7] for CPU1
6-0 —	Reserved

11.4.1.98 CPU Status (CPUSTAT)

Offset

Register	Offset
CPUSTAT	80Ch

Diagram



Fields

Field	Function
31-4 —	Reserved
3 CPU1LOCKUP	CPU1 lockup state 0b - CPU is not in lockup 1b - CPU is in lockup
2 CPU0LOCKUP	CPU0 lockup state 0b - CPU is not in lockup 1b - CPU is in lockup

Table continues on the next page...

Table continued from the previous page...

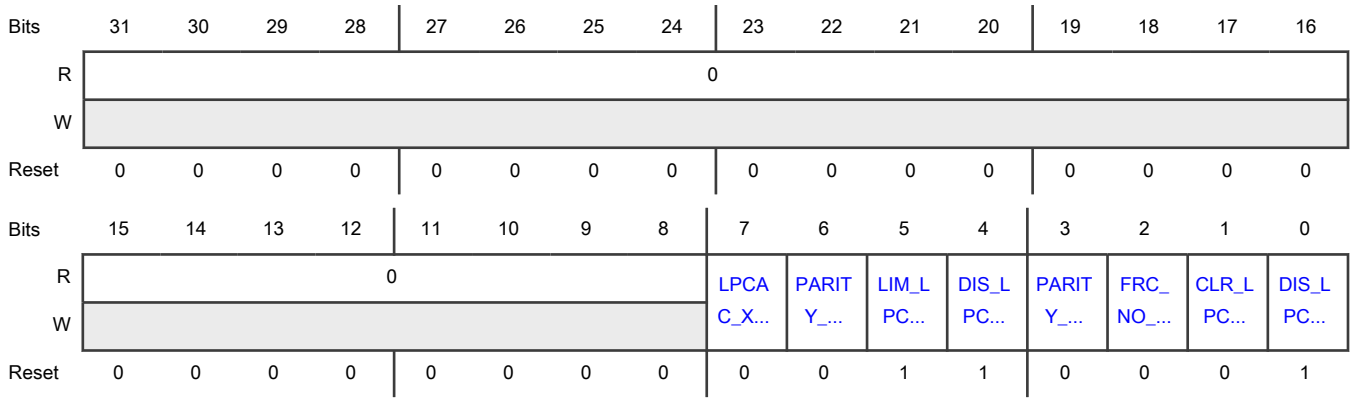
Field	Function
1 CPU1SLEEPING	CPU1 sleeping state 0b - CPU is not sleeping 1b - CPU is sleeping
0 CPU0SLEEPING	CPU0 sleeping state 0b - CPU is not sleeping 1b - CPU is sleeping

11.4.1.99 LPCAC Control (LPCAC_CTRL)

Offset

Register	Offset
LPCAC_CTRL	824h

Diagram



Fields

Field	Function
31-8 —	Reserved
7 LPCAC_XOM	LPCAC XOM(eXecute-Only-Memory) attribute control Controls if the instruction fetch attribute is used as part of the address input to the LPCAC. When XOM regions in the internal flash are not configured at the MBC, then this option should be disabled so that instructions and data can be stored within the same cache line. This provides the best cache efficiency for non-XOM applications. When XOM areas in the internal flash are configured at the MBC, then this bit must be set so that instructions and data are cached using separate lines within the LPCAC.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Disabled. 1b - Enabled.
6 PARITY_FAULT_EN	Enable parity error report. 0b - Disables parity error report 1b - Enables parity error report
5 LIM_LPCAC_WTBF	Limit LPCAC Write Through Buffer. 0b - Write buffer enabled when transaction is bufferable. 1b - Write buffer enabled when transaction is cacheable and bufferable
4 DIS_LPCAC_WTBF	Disable LPCAC Write Through Buffer. 0b - Enables write through buffer 1b - Disables write through buffer
3 PARITY_MISS_EN	Enables parity miss. 0b - Disabled 1b - Enables parity, miss on parity error
2 FRC_NO_ALLOC	Forces no allocation. 0b - Forces allocation 1b - Forces no allocation
1 CLR_LPCAC	Clears the cache function. 0b - Unclears the cache 1b - Clears the cache
0 DIS_LPCAC	Disables/enables the cache function. 0b - Enabled 1b - Disabled

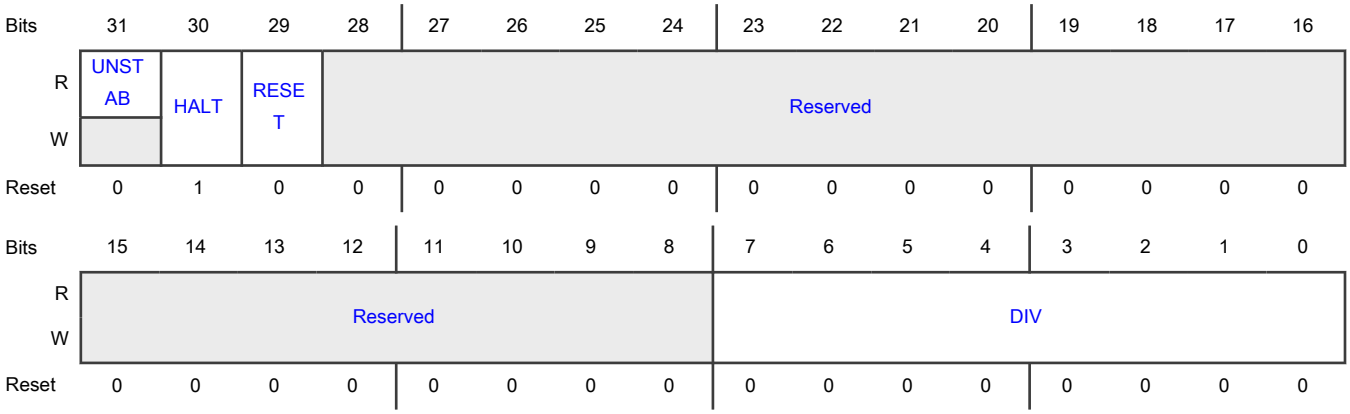
11.4.1.100 LP_FLEXCOMM Clock Divider (FLEXCOMM0CLKDIV - FLEXCOMM9CLKDIV)

Offset

For n = 0 to 9:

Register	Offset
FLEXCOMMnCLKDIV	850h + (n × 4h)

Diagram



Fields

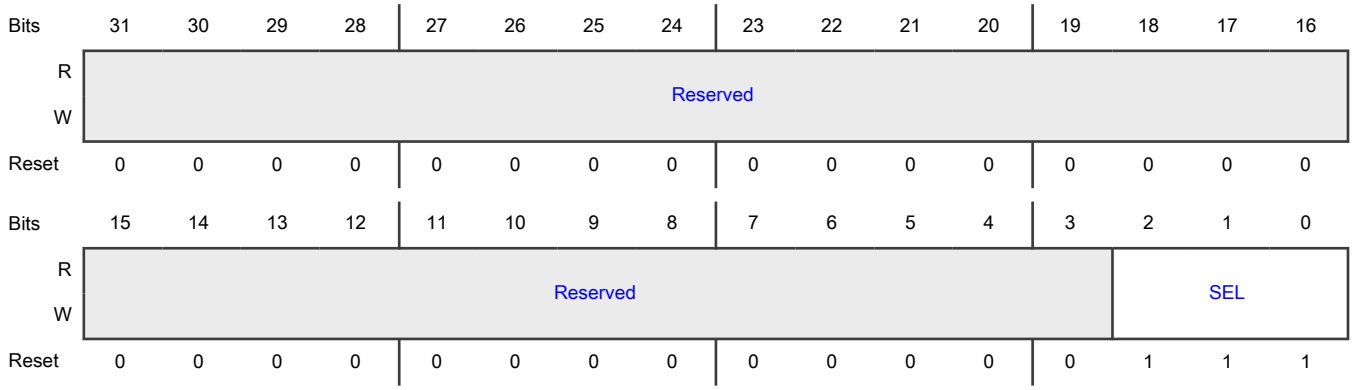
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value • 0 - Divide by 1 • 1 - Divide by 2 • ... • value - Divide by (DIV+1)

11.4.1.101 SAI0 Function Clock Source Select (SAI0CLKSEL)

Offset

Register	Offset
SAI0CLKSEL	880h

Diagram



Fields

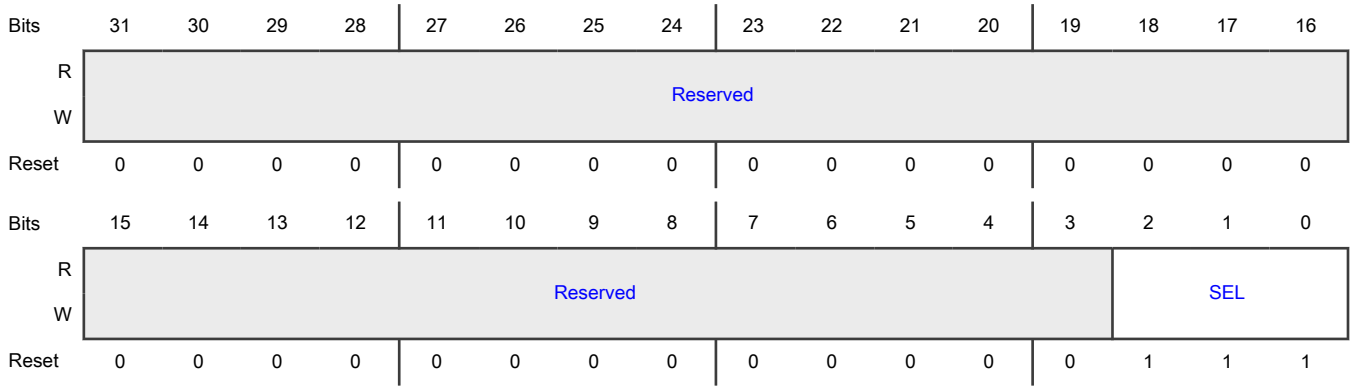
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the clock source 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - FRO_HF clock 100b - PLL1_CLK0 clock 101b - No clock 110b - USB PLL clock 111b - No clock

11.4.1.102 SAI1 Function Clock Source Select (SAI1CLKSEL)

Offset

Register	Offset
SAI1CLKSEL	884h

Diagram



Fields

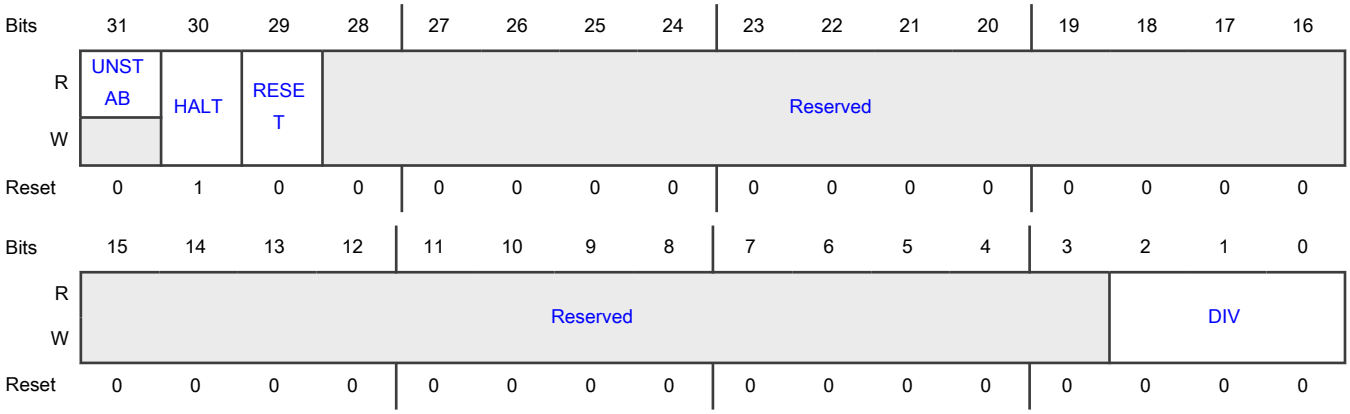
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the clock source 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - FRO_HF clock 100b - PLL1_CLK0 clock 101b - No clock 110b - USB PLL clock 111b - No clock

11.4.1.103 SAI0 Function Clock Division (SAI0CLKDIV)

Offset

Register	Offset
SAI0CLKDIV	888h

Diagram



Fields

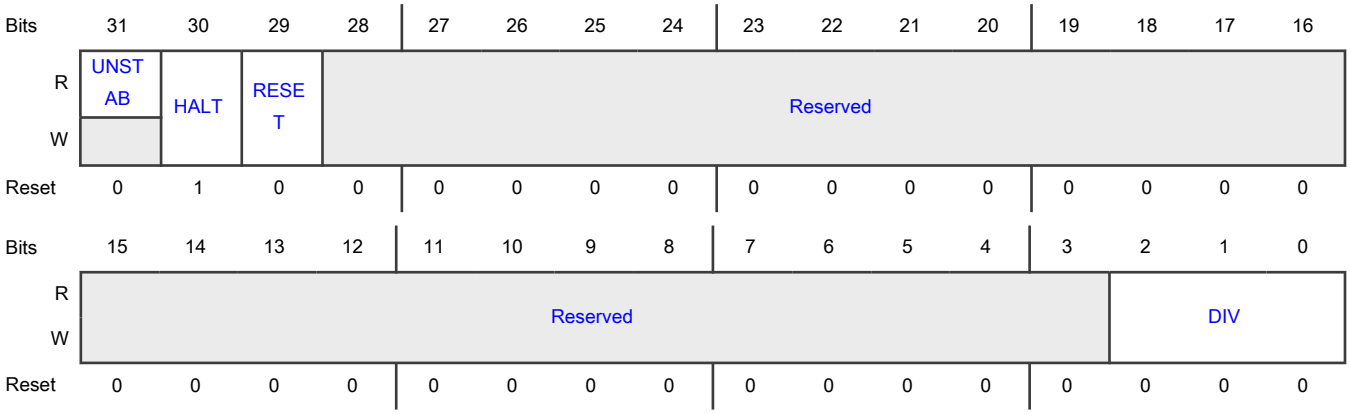
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-3 —	Reserved
2-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.104 SAI1 Function Clock Division (SAI1CLKDIV)

Offset

Register	Offset
SAI1CLKDIV	88Ch

Diagram



Fields

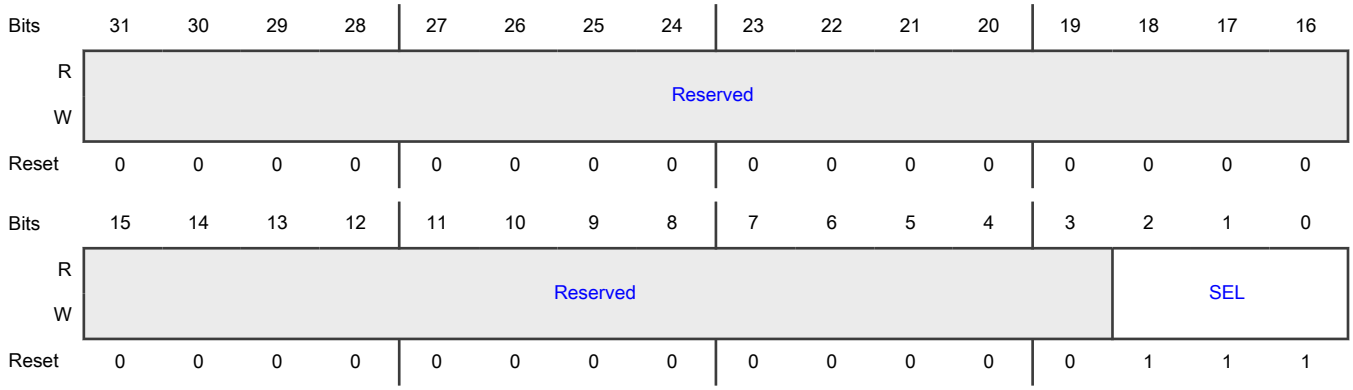
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-3 —	Reserved
2-0 DIV	Clock divider value The divider value = (DIV + 1).

11.4.1.105 EMVSIM0 Clock Source Select (EMVSIM0CLKSEL)

Offset

Register	Offset
EMVSIM0CLKSEL	890h

Diagram



Fields

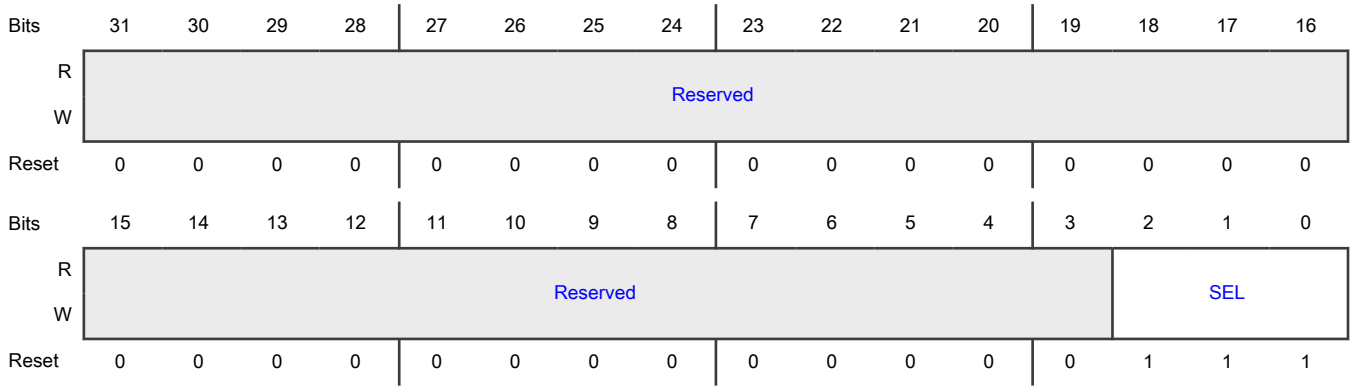
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the EMVSIM0 function clock source 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - FRO_HF clock 100b - FRO_12M clock 101b - PLL1_clk0 clock0 110b - No clock 111b - No clock

11.4.1.106 EMVSIM1 Clock Source Select (EMVSIM1CLKSEL)

Offset

Register	Offset
EMVSIM1CLKSEL	894h

Diagram



Fields

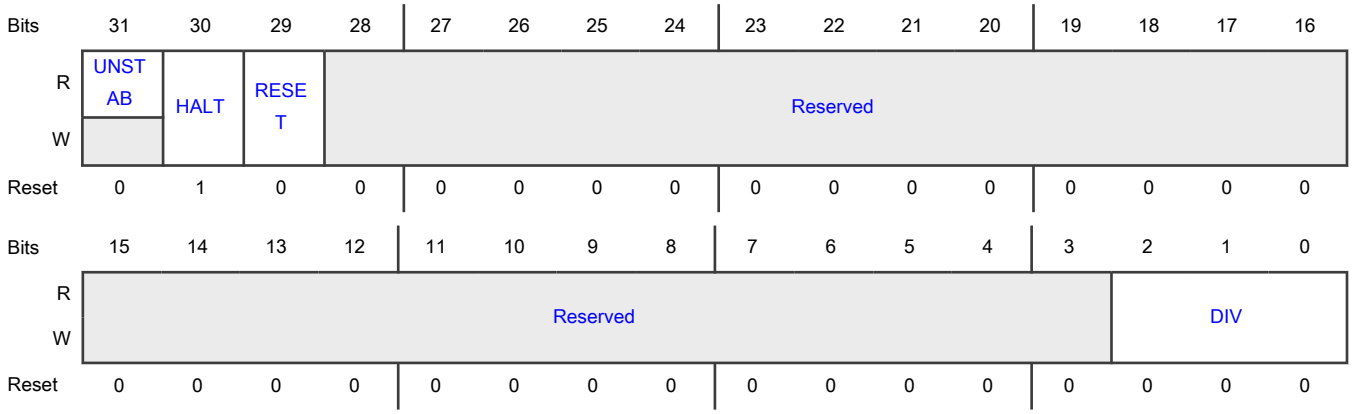
Field	Function
31-3 —	Reserved
2-0 SEL	Selects the EMVSIM1 function clock source 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - FRO_HF clock 100b - FRO_12M clock 101b - PLL1_clk0 clock0 110b - No clock 111b - No clock

11.4.1.107 EMVSIM0 Function Clock Division (EMVSIM0CLKDIV)

Offset

Register	Offset
EMVSIM0CLKDIV	898h

Diagram



Fields

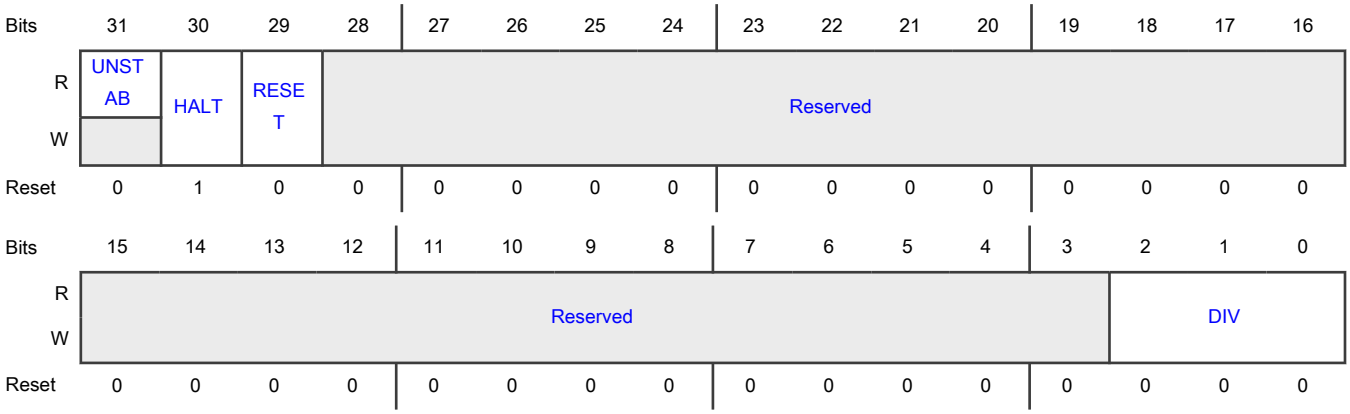
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-3 —	Reserved
2-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.108 EMVSIM1 Function Clock Division (EMVSIM1CLKDIV)

Offset

Register	Offset
EMVSIM1CLKDIV	89Ch

Diagram



Fields

Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-3 —	Reserved
2-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.109 Key Retain Control (KEY_RETAIN_CTRL)

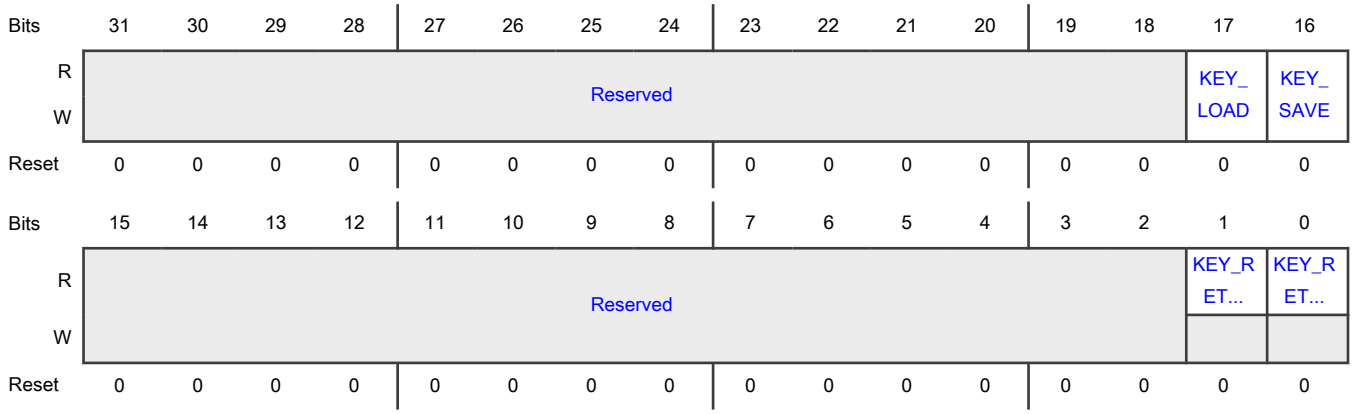
Offset

Register	Offset
KEY_RETAIN_CTRL	950h

Function

All functionality for this register requires the ELS clock to be enabled and not be reset.

Diagram



Fields

Field	Function
31-18 —	Reserved
17 KEY_LOAD	Do not set both KEY_SAVE and KEY_LOAD at the same time. Software should only set KEY_LOAD if KEY_RETAIN_DONE is set and to initiate a load of the key retained in the VBAT domain into ELS. After software sets KEY_LOAD, then wait for KEY_RETAIN_DONE to set before clearing KEY_LOAD. 0b - Key load sequence is disabled. 1b - Key load sequence is enabled.
16 KEY_SAVE	Do not set both KEY_SAVE and KEY_LOAD at the same time. Software should set KEY_SAVE before configuring the PUF to load device unique key into ELS. After software sets KEY_SAVE and the PUF had loaded the key into ELS, then wait for KEY_RETAIN_DONE to set before clearing KEY_SAVE. 0b - Key save sequence is disabled. 1b - Key save sequence is enabled.
15-2 —	Reserved
1 KEY_RETAIN_ DONE	Indicates the successful completion of the key_save or key_load routine. Once set, to clear the key_retain_done flag, both key_save and key_load should be cleared by software. 0b - Key save / load sequence has not completed. 1b - Key save / load sequence has completed.
0 KEY_RETAIN_ VALID	Indicates if the PUF key has been retained in the VBAT domain and has not been reset or otherwise invalidated by software. 0b - PUF key is not retained in VBAT domain. 1b - PUF key is retained in VBAT domain.

11.4.1.110 FRO 48MHz Reference Clock Control (REF_CLK_CTRL)

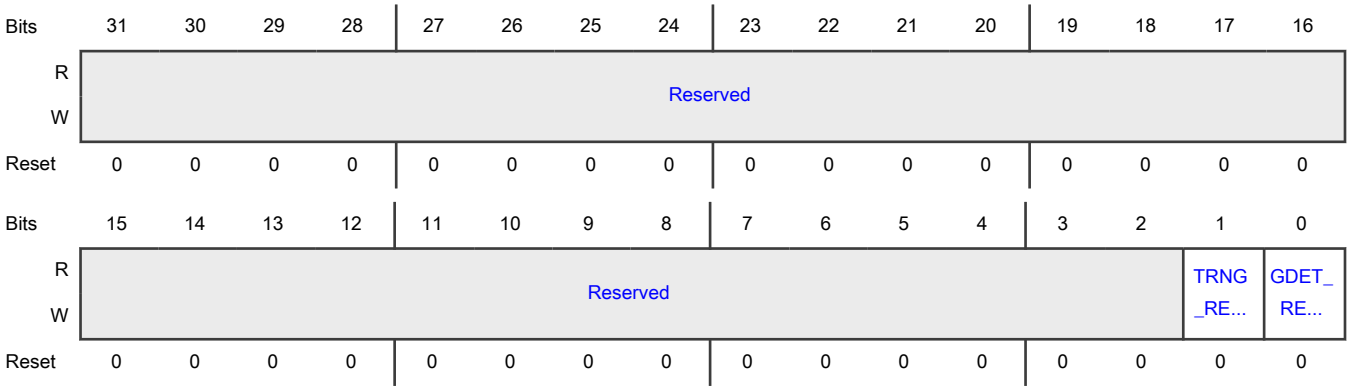
Offset

Register	Offset
REF_CLK_CTRL	960h

Function

This register controls the reference clock gate on/off and it is glitch free.

Diagram



Fields

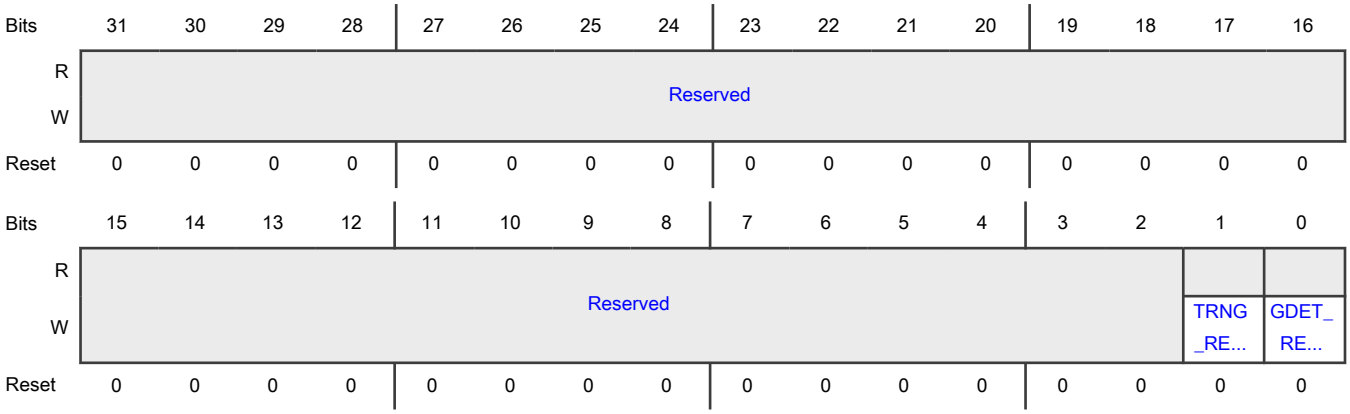
Field	Function
31-2 —	Reserved
1 TRNG_REFCLK_EN	ELS TRNG reference clock enable bit A write 1 to this bit enables the FRO 48 MHz clock as reference clock to TRNG module. Prior to write 1, the FRO 48 MHz distribution must be activated. 0b - Disabled. 1b - Enabled
0 GDET_REFCLK_EN	GDET reference clock enable bit A write 1 to this bit enables the FRO 48 MHz clock as reference clock to GDET module. Prior to write 1, the FRO 48 MHz distribution must be activated. 0b - Disabled. 1b - Enabled

11.4.1.111 FRO 48MHz Reference Clock Control Set (REF_CLK_CTRL_SET)

Offset

Register	Offset
REF_CLK_CTRL_SET	964h

Diagram



Fields

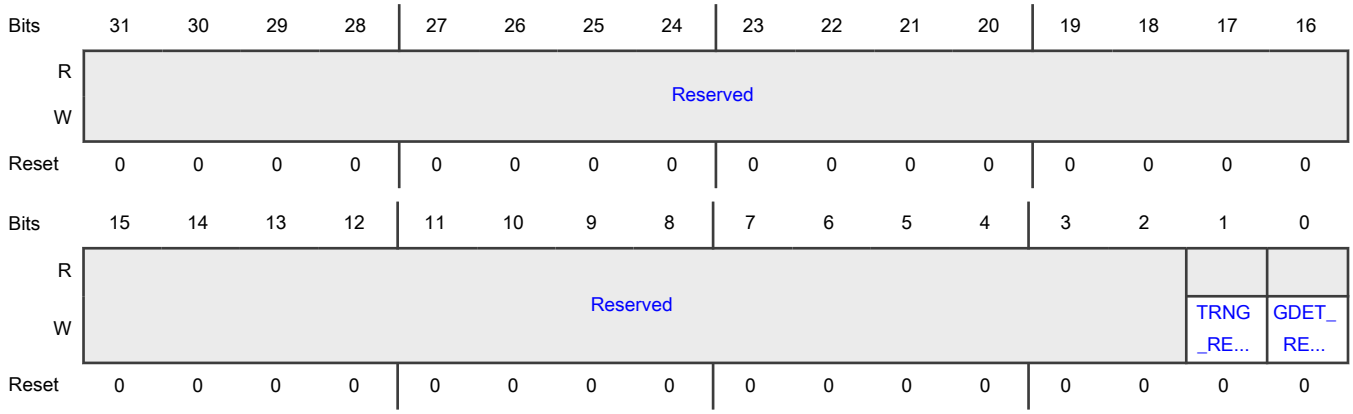
Field	Function
31-2 —	Reserved
1 TRNG_REFCLK_EN_SET	ELS TRNG reference clock enable set bit A write 1 to this bit sets the TRNG_REFCLK_EN to 1. A write 0 has no effect. 0b - No effect. 1b - Set to 1
0 GDET_REFCLK_EN_SET	GDET reference clock enable set bit A write 1 to this bit sets the GDET_REFCLK_EN to 1. A write 0 has no effect. 0b - No effect. 1b - Set to 1

11.4.1.112 FRO 48MHz Reference Clock Control Clear (REF_CLK_CTRL_CLR)

Offset

Register	Offset
REF_CLK_CTRL_CLR	968h

Diagram



Fields

Field	Function
31-2 —	Reserved
1 TRNG_REFCLK_EN_CLR	ELS TRNG reference clock enable clear bit A write 1 to this bit clears the TRNG_REFCLK_EN to 0. A write 0 has no effect. 0b - No effect. 1b - Set to 0
0 GDET_REFCLK_EN_CLR	GDET reference clock enable clear bit A write 1 to this bit clears the GDET_REFCLK_EN to 0. A write 0 has no effect. 0b - No effect. 1b - Set to 0

11.4.1.113 GDET Control Register (GDET0_CTRL - GDET1_CTRL)

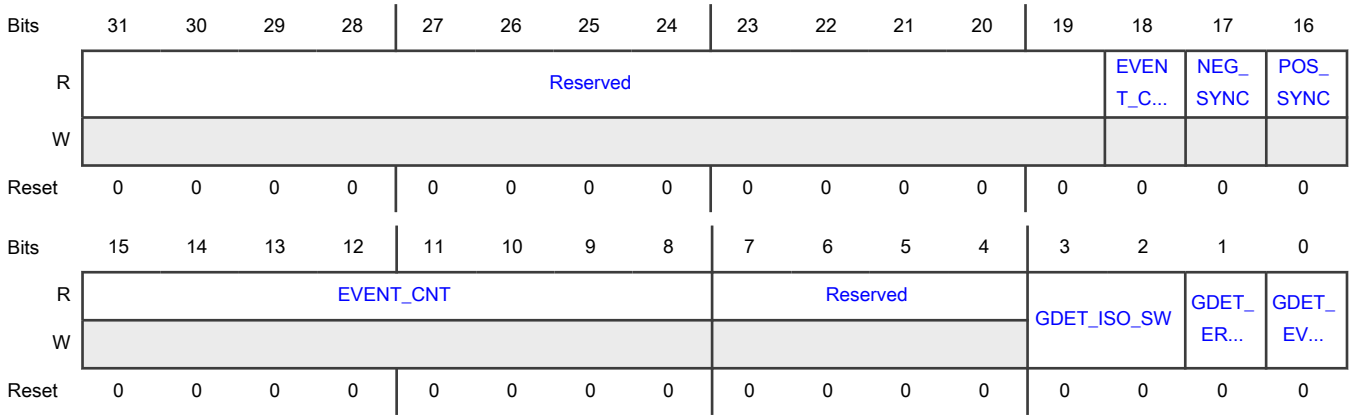
Offset

Register	Offset
GDET0_CTRL	96Ch
GDET1_CTRL	970h

Function

GDET control register

Diagram



Fields

Field	Function
31-19 —	Reserved
18 EVENT_CLR_FLAG	Event counter cleared 0b - Event counter not cleared 1b - Event counter cleared
17 NEG_SYNC	Negative glitch detected 0b - Negative glitch not detected 1b - Negative glitch detected
16 POS_SYNC	Positive glitch detected 0b - Positive glitch not detected 1b - Positive glitch detected
15-8 EVENT_CNT	Event count value Gray-based counter value of the glitch events detected
7-4 —	Reserved
3-2 GDET_ISO_SW	GDET isolation control 00b - Isolation is disabled 01b - Isolation is disabled 10b - Isolation is enabled. When both GDET0_CTRL/GDET1_CTRL GDET_ISO_SW are "10", isolation_on is asserted. 11b - Isolation is disabled

Table continues on the next page...

Table continued from the previous page...

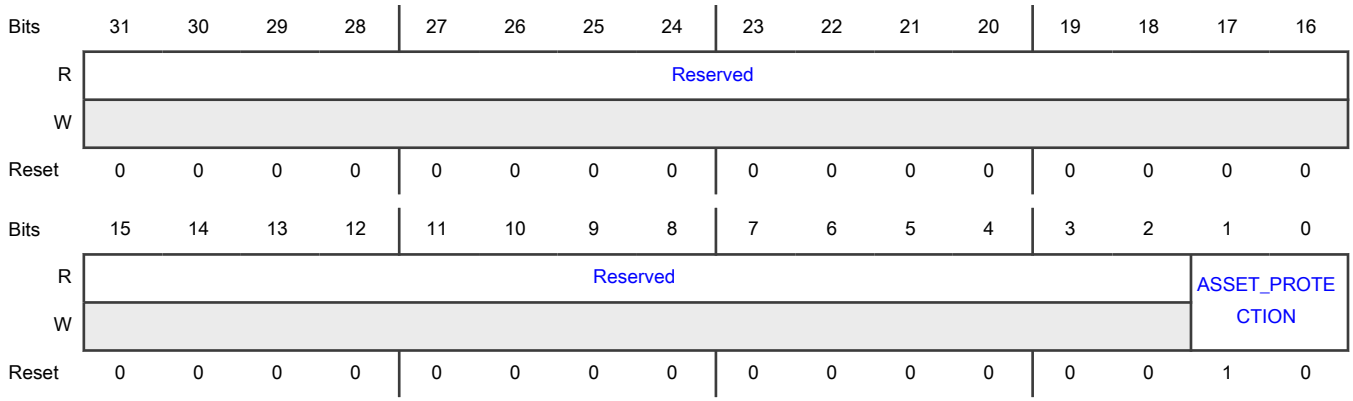
Field	Function
1 GDET_ERR_CLR	Clears GDET error status 0b - Error status not cleared 1b - Clears error status
0 GDET_EVTCNT_CLR	Controls the GDET clean event counter 0b - Event counter not cleared 1b - Clears event counter

11.4.1.114 ELS Asset Protection Register (ELS_ASSET_PROT)

Offset

Register	Offset
ELS_ASSET_PROT	974h

Diagram



Fields

Field	Function
31-2 —	Reserved
1-0 ASSET_PROTECTION	ELS asset protection. This field controls the asset protection port to the ELS module. Refer to the ELS chapter in the SRM for more details. 00b - ELS asset is protected 01b - ELS asset is not protected 10b - ELS asset is protected 11b - ELS asset is protected

11.4.1.115 ELS Lock Control (ELS_LOCK_CTRL)

Offset

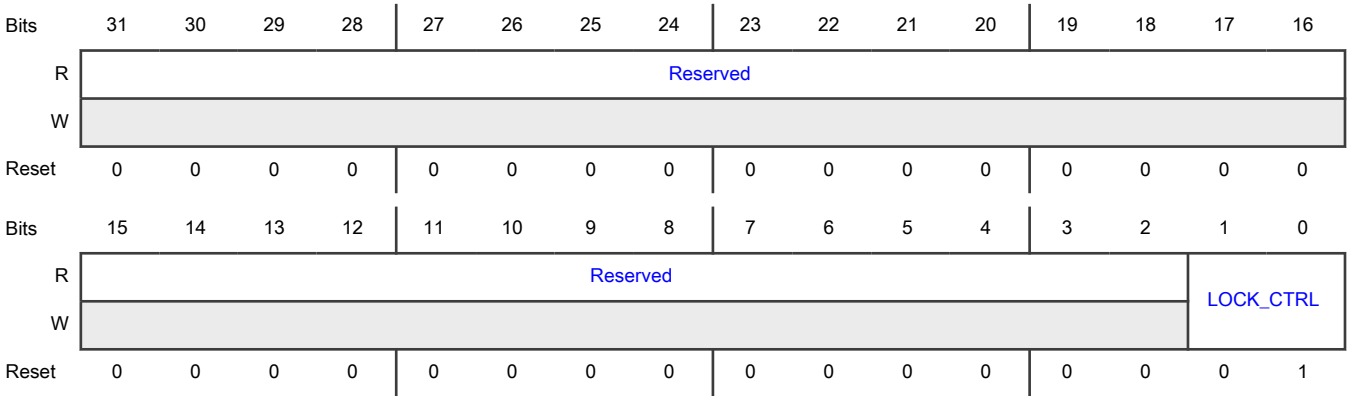
Register	Offset
ELS_LOCK_CTRL	978h

Function

When the slice in this register and the same slice located in ELS_LOCK_CTRL_DP are set to 10, the corresponding ELS input is driven 1 to enable write to the ELS_ASSET_PROT register.

ELS_ASSET_PROT is writable only when ELS_LOCK_CTRL[1:0] = 10 & ELS_LOCK_CTRL_DP[1:0] = 10.

Diagram



Fields

Field	Function
31-2 —	Reserved
1-0 LOCK_CTRL	ELS Lock Control

LOCK_CTRL[1:0]	LOCK_CTRL_DP[1:0]	ELS_ASSET_PROT is writable	LOCK_CTRL & LOCK_CTRL_DP are writable
"10"	"10"	YES	YES
"01"	"10"	NO	YES
NA	!= "10"	NO	NO
"00" or "11"	NA	NO	NO

11.4.1.116 ELS Lock Control DP (ELS_LOCK_CTRL_DP)

Offset

Register	Offset
ELS_LOCK_CTRL_DP	97Ch

Function

Refer to ELS_LOCK_CTRL for more details

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved														LOCK_CTRL_DP	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Fields

Field	Function
31-2 —	Reserved
1-0 LOCK_CTRL_DP	Refer to ELS_LOCK_CTRL[1:0]

11.4.1.117 Life Cycle State Register (ELS_OTP_LC_STATE)

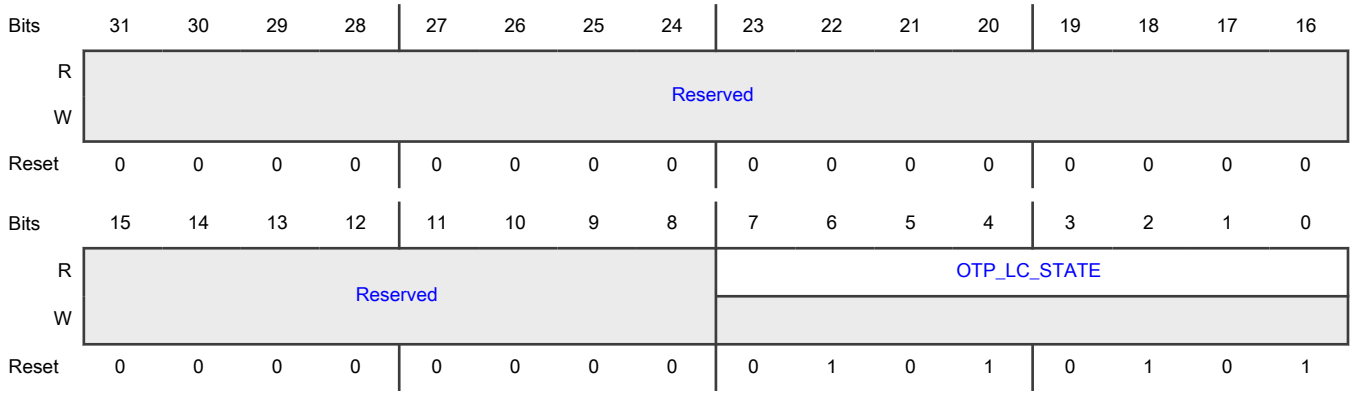
Offset

Register	Offset
ELS_OTP_LC_STATE	980h

Function

Current device lifecycle status

Diagram



Fields

Field	Function
31-8 —	Reserved
7-0 OTP_LC_STATE	OTP life cycle state

11.4.1.118 Life Cycle State Register (Duplicate) (ELS_OTP_LC_STATE_DP)

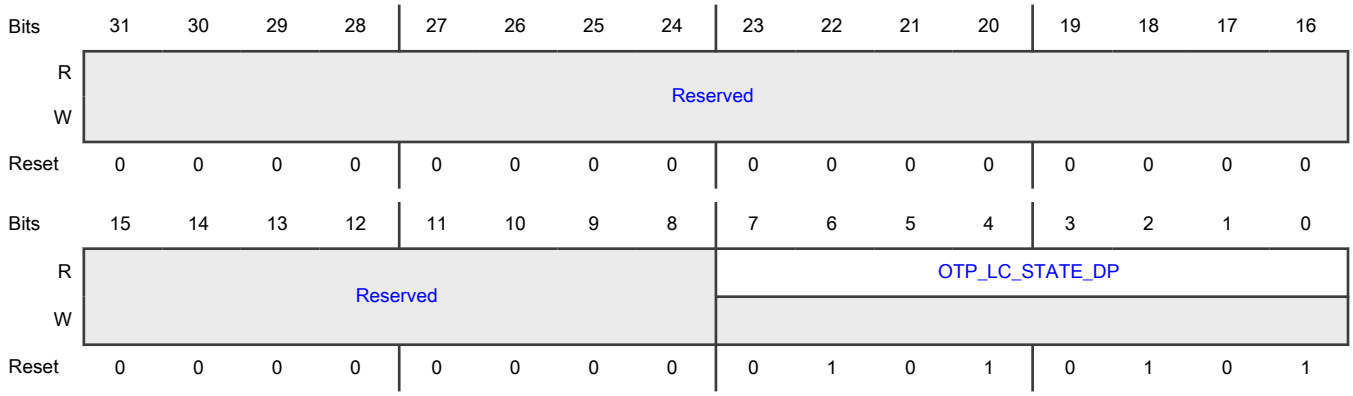
Offset

Register	Offset
ELS_OTP_LC_STATE_DP	984h

Function

Current device lifecycle status

Diagram



Fields

Field	Function
31-8 —	Reserved
7-0 OTP_LC_STATE_DP	OTP life cycle state

11.4.1.119 ELS Temporal State (ELS_TEMPORAL_STATE)

Offset

Register	Offset
ELS_TEMPORAL_STATE	988h

Function

This is boot process milestone counter initially incremented by Boot-ROM then, carried over to application for continuation of increment when needed. This 4-bit counter is expanded to 16-bit value (TEMPORAL_STATE_DEC) as shown in description before connecting to key management modules to provide isolation of keys per milestone stage.

(1) PUF key context management: Check bits [23:8] in 4.5.1.18 Hardware Restrict User Context 0 (HW_RUC0) section and bits [23:8] of context word 2 in 5.4. PUF Context key Management section.

(2) ELS key management: Check Hardware derivation data (hw_drv_data[63:0]) field in CKDF data structures table in 3.5.2.10 CKDF Command section.

hw_drv_data[7:0] = OTP_LC_STATE or ELS_KDF_MASK[7:0]

hw_drv_data[23:8] = TEMPORAL_STATE_DEC[15:0] or ELS_KDF_MASK[23:8]

hw_drv_data[24]" <= "FLAG_AP_ENABLE_CPU0" on "ELS_AS_FLAG0" or " ELS_KDF_MASK[24]"

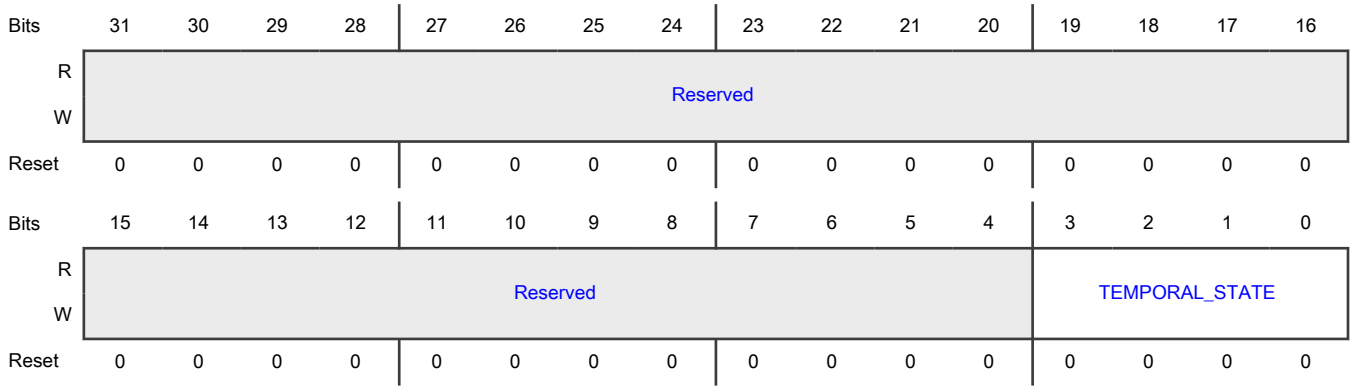
hw_drv_data[25]" <= "FLAG_AP_ENABLE_CPU1" on "ELS_AS_FLAG0" or " ELS_KDF_MASK[25]"

hw_drv_data[26]" <= "FLAG_AP_ENABLE_DSP" on "ELS_AS_FLAG0" or " ELS_KDF_MASK[26]"

hw_drv_data[31:27]" <= KDF_MASK[31:27]

hw_drv_data[63:32] = 0

Diagram



Fields

Field	Function
31-4 —	Reserved
3-0 TEMPORAL_STATE	Temporal state

TEMPORAL_STATE[3:0]	TEMPORAL_STATE_DEC[15:0]
0x0	0x0001
0x1	0x0003
0x2	0x0007
0x3	0x000F
0x4	0x001F
0x5	0x003F
0x6	0x007F
0x7	0x00FF
0x8	0x01FF
0x9	0x03FF
0x9	0x03FF
0xA	0x07FF
0xB	0x0FFF
0xC	0x1FFF
0xD	0x3FFF
0xE	0x7FFF
0xF	0xFFFF

11.4.1.120 Key Derivation Function Mask (ELS_KDF_MASK)

Offset

Register	Offset
ELS_KDF_MASK	98Ch

Function

ELS "hw_drv_data[31:0]" are OR masked by this register.

"hw_drv_data[7:0]" <= "OTP_LC_STATE[7:0]" or "KDF_MASK[7:0]"

"hw_drv_data[23:8]" <= A decoded version of "TEMPORAL_STATE[3:0]" or "KDF_MASK[23:8]"

"hw_drv_data[24]" <= "FLAG_AP_ENABLE_CPU0" on "ELS_AS_FLAG0" or "KDF_MASK[24]"

"hw_drv_data[25]" <= "FLAG_AP_ENABLE_CPU1" on "ELS_AS_FLAG0" or "KDF_MASK[25]"

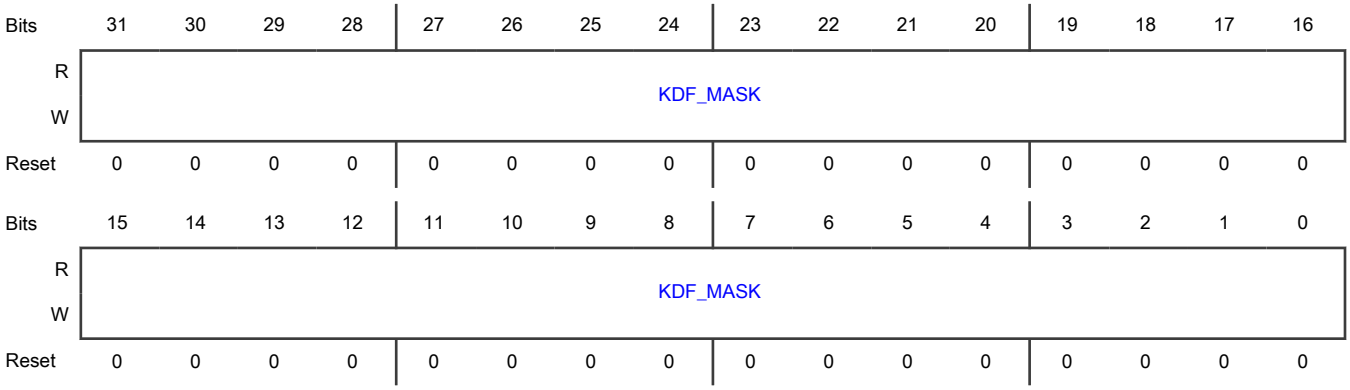
"hw_drv_data[26]" <= "FLAG_AP_ENABLE_DSP" on "ELS_AS_FLAG0" or "KDF_MASK[26]"

"hw_drv_data[31:27]" <= KDF_MASK[31:27]

NOTE

"hw_drv_data[63:32]" are always driven by all zeros {32{1'b0}}

Diagram



Fields

Field	Function
31-0 KDF_MASK	Key derivation function mask

11.4.1.121 ELS AS Configuration (ELS_AS_CFG0)

Offset

Register	Offset
ELS_AS_CFG0	9D0h

Function
ELS AS configuration.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		CFG_QK_...	CFG_QK_...	CFG_CLK...	CFG_LHT...	CFG_VOL...	CFG_TEM...	CFG_CWD...	CFG_WDT...	CFG_LVD...	CFG_LVD...	CFG_LVD...	CFG_LVD...	CFG_TAM...	CFG_ANA...
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CFG_ANA...	CFG_ELS...	CFG_CWD...	CFG_WDT...	CFG_LVD...	Reserved	CFG_LVD...	Reserved	CFG_LC_STATE							
W																
Reset	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	1

Fields

Field	Function
31-30 —	Reserved
29 CFG_QK_DISABLE_WRAP	When QK PUF "qk_disable_wrap" input is driven 1, this bit indicates state 1
28 CFG_QK_DISABLE_ENROLL	When QK PUF "qk_disable_enroll" input is driven 1, this bit indicates state 1
27 CFG_CLKTAMPER_DET_ENABLED	When clk tamper detector is enabled in VBAT, this bit indicates state 1.
26	When light tamper detector is enabled in VBAT, this bit indicates state 1.

Table continues on the next page...

Table continued from the previous page...

Field	Function
CFG_LHTTAMPER_DET_ENABLED	
25 CFG_VOLTAMPER_DET_ENABLED	When voltage tamper detector is enabled in VBAT, this bit indicates state 1.
24 CFG_TEMPTAMPER_DET_ENABLED	When temperature tamper detector is enabled in VBAT, this bit indicates state 1.
23 CFG_CWDT1_ENABLED	When Code WatchDog Timer 1 is activated, this bit indicates state 1.
22 CFG_WDT1_ENABLED	When WatchDog Timer 1 is activated, this bit indicates state 1.
21 CFG_LVD_VDDIO_IRQ_ENABLED	When SPC VDDIO LVD analog detector are turned on and VDDIO LVD irq are enabled, this bit indicates state 1.
20 CFG_LVD_VSYS_IRQ_ENABLED	When SPC VSYS LVD analog detector are turned on and VSYS LVD irq are enabled, this bit indicates state 1.
19 CFG_LVD_VDDIO_RESET_ENABLED	When SPC VDDIO LVD analog detector are turned on and VDDIO LVD reset are enabled, this bit indicates state 1.
18 CFG_LVD_VSYS_RESET_ENABLED	When SPC VSYS LVD analog detector are turned on and VSYS LVD reset are enabled, this bit indicates state 1.
17	When tamper detector is enabled in TDET, this bit indicates state 1.

Table continues on the next page...

Table continued from the previous page...

Field	Function
CFG_TAMPER_DET_ENABLED	
16 CFG_ANA_GDET_IRQ_ENABLED	When SPC analog glitch detect IRQ is enabled, this bit indicates state 1
15 CFG_ANA_GDET_RESET_ENABLED	When SPC analog glitch detect reset is enabled, this bit indicates state 1
14 CFG_ELS_GDET_ENABLED	When either GDET is enabled, this bit indicates state 1.
13 CFG_CWDT0_ENABLED	When Code WatchDog Timer 0 is activated, this bit indicates state 1
12 CFG_WDT0_ENABLED	When WatchDog Timer 0 is activated, this bit indicates state 1
11 CFG_LVD_CORE_IRQ_ENABLED	When SPC CORE LVD analog detector are turned on, and CORE LVD IRQ are enabled, this bit indicates state 1.
10 —	Reserved
9 CFG_LVD_CORE_RESET_ENABLED	When SPC CORE LVD analog detector are turned on, and CORE LVD reset are enabled, this bit indicates state 1.
8 —	Reserved
7-0	LC state configuration bit OTP_LC_STATE_FINAL[7:0] described in the ELS_OTP_LC_STATE register reflects to this bit field

Table continues on the next page...

Table continued from the previous page...

Field	Function
CFG_LC_STATE	

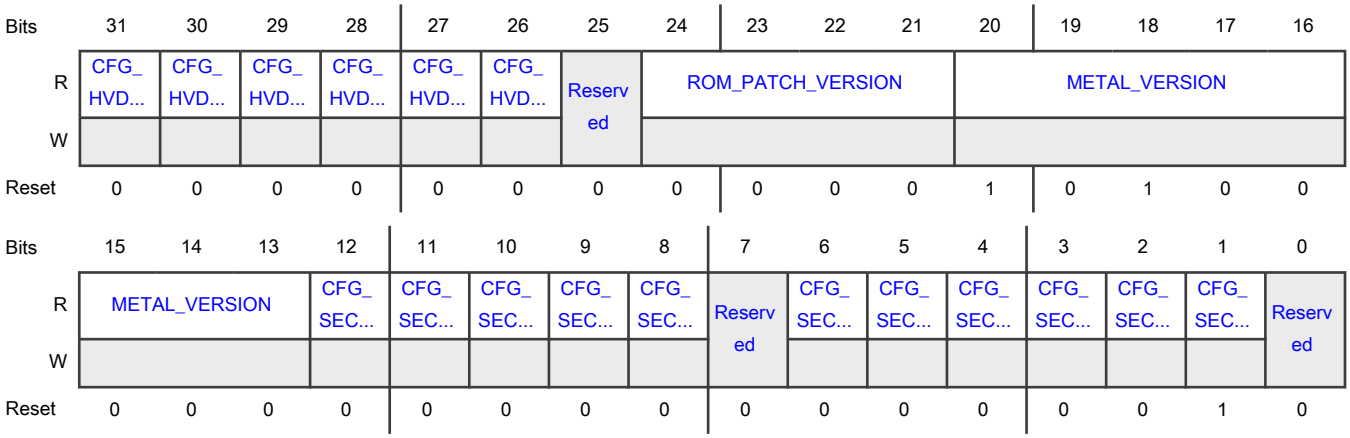
11.4.1.122 ELS AS Configuration1 (ELS_AS_CFG1)

Offset

Register	Offset
ELS_AS_CFG1	9D4h

Function
ELS AS configuration1.

Diagram



Fields

Field	Function
31 CFG_HVD_VDDIO_IRQ_ENABLED	When SPC VDDIO HVD analog detector are turned on and VDDIO HVD irq are enabled, this bit indicates state 1.
30 CFG_HVD_VSYS_IRQ_ENABLED	When SPC VSYS HVD analog detector are turned on and VSYS HVD irq are enabled, this bit indicates state 1.

Table continues on the next page...

Table continued from the previous page...

Field	Function
29 CFG_HVD_VDDIO_RESET_ENABLED	When SPC VDDIO HVD analog detector are turned on and VDDIO HVD reset are enabled, this bit indicates state 1.
28 CFG_HVD_VSYS_RESET_ENABLED	When SPC VSYS HVD analog detector are turned on and VSYS HVD reset are enabled, this bit indicates state 1.
27 CFG_HVD_CORE_IRQ_ENABLED	When SPC CORE HVD analog detector are turned on, and CORE HVD IRQ are enabled, this bit indicates state 1.
26 CFG_HVD_CORE_RESET_ENABLED	When SPC CORE HVD analog detector are turned on, and CORE HVD reset are enabled, this bit indicates state 1.
25 —	Reserved
24-21 ROM_PATCH_VERSION	ROM patch version
20-13 METAL_VERSION	metal version
12 CFG_SEC_LOCK_SAU	When the LOCK_SAU bits in CPU0_LOCK_REG on the AHB secure controller are not equal to 10, this bit indicates state 1
11 CFG_SEC_LOCK_S_VTAIRCR	When the LOCK_S_VTAIRCR bits in CPU0_LOCK_REG on the AHB secure controller are not equal to 10, this bit indicates state 1
10 CFG_SEC_LOCK_S_MPU	When the LOCK_S_MPU bits in CPU0_LOCK_REG on the AHB secure controller are not equal to 10, this bit indicates state 1
9	When the LOCK_NS_VTOR bits in CPU0_LOCK_REG on the AHB secure controller are not equal to 10, this bit indicates state 1

Table continues on the next page...

Table continued from the previous page...

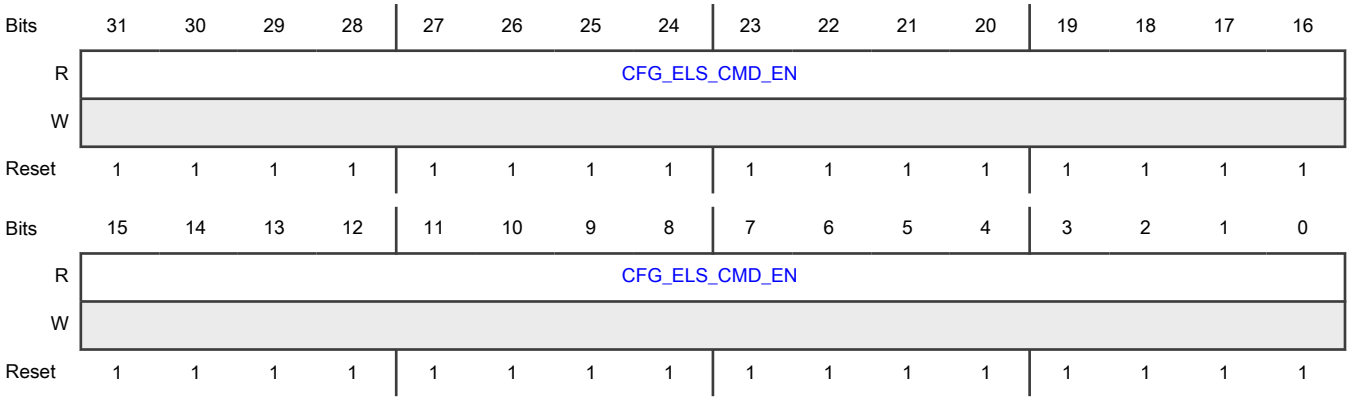
Field	Function
CFG_SEC_LOCK_NS_VTOR	
8 CFG_SEC_LOCK_NS_MPU	When the LOCK_NS_MPU bits in CPU0_LOCK_REG on the AHB secure controller are not equal to 10, this bit indicates state 1
7 —	Reserved
6 CFG_SEC_IDAU_ALLNS	When the IDAU_ALL_NS bits in MISC_CTRL_REG and MISC_CTRL_DP_REG on the AHB secure controller are equal to 01, this bit indicates state 1
5 CFG_SEC_ENA_SEC_CHK	When the ENABLE_SECURE_CHECKING bits in MISC_CTRL_REG and MISC_CTRL_DP_REG on the AHB secure controller are not equal to 10, this bit indicates state 1
4 CFG_SEC_ENA_S_PRIV_CHK	When the ENABLE_S_PRIV_CHECK bits in MISC_CTRL_REG and MISC_CTRL_DP_REG on the AHB secure controller are not equal to 10, this bit indicates state 1
3 CFG_SEC_ENA_NS_PRIV_CHK	When the ENABLE_NS_PRIV_CHECK bits in MISC_CTRL_REG and MISC_CTRL_DP_REG on the AHB secure controller are not equal to 10, this bit indicates state 1
2 CFG_SEC_DIS_VIOL_ABORT	When the DISABLE_VIOLATION_ABORT bits in MISC_CTRL_REG and MISC_CTRL_DP_REG on the AHB secure controller are not equal to 10, this bit indicates state 1
1 CFG_SEC_DIS_STRICT_MODE	When CFG_SEC_ENA_SEC_CHK indicates state 0 or when DISABLE_STRICT_MODE bits in MISC_CTRL_REG and MISC_CTRL_DP_REG on the AHB secure controller are equal to 01, this bit indicates state 1
0 —	Reserved

11.4.1.123 ELS AS Configuration2 (ELS_AS_CFG2)

Offset

Register	Offset
ELS_AS_CFG2	9D8h

Diagram



Fields

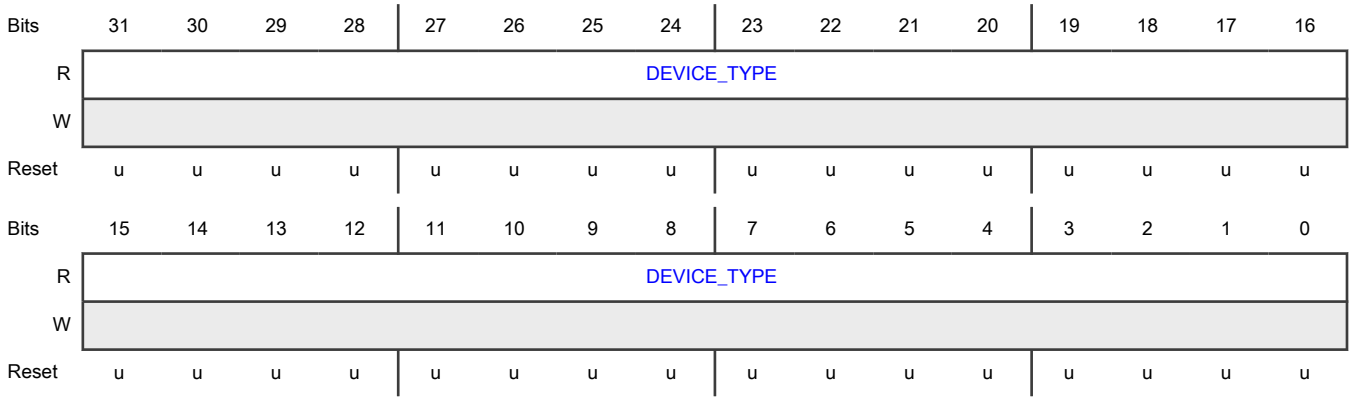
Field	Function
31-0 CFG_ELS_CMD_EN	ELS configuration command enable bit

11.4.1.124 ELS AS Configuration3 (ELS_AS_CFG3)

Offset

Register	Offset
ELS_AS_CFG3	9DCh

Diagram



Fields

Field	Function
31-0 DEVICE_TYPE	Device type identification data

11.4.1.125 ELS AS State Register (ELS_AS_ST0)

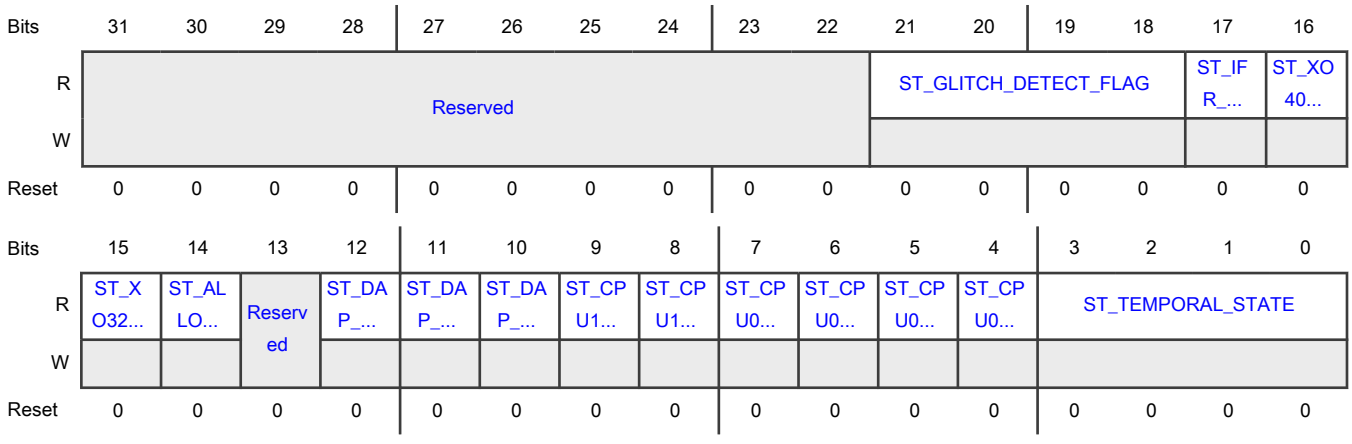
Offset

Register	Offset
ELS_AS_ST0	9E0h

Function

ELS AS state register.

Diagram



Fields

Field	Function
31-22 —	Reserved
21-18 ST_GLITCH_DETECT_FLAG	GLITCH_DETECT_FLAG is state of 4-bit Glitch Ripple Counter output.
17 ST_IFR_LOAD_FAILED	When IFR load fail flag is state 1, this bit indicates state 1
16 ST_XO40M_FAILED	When XO40M oscillation fail flag is state 1, this bit indicates state 1
15 ST_XO32K_FAILED	When XO32K oscillation fail flag is state 1, this bit indicates state 1
14 ST_ALLOW_TEST_ACCESS	When JTAG TAP access is allowed, this bit indicates state 1.
13 —	Reserved
12 ST_DAP_ENABLE_DSP	When DAP to AP3 for DSP (CoolFlux) debug access is allowed, this bit indicates state 1
11 ST_DAP_ENABLE_CPU1	When DAP to AP1 for CPU1 (CM33) debug access is allowed, this bit indicates state 1.
10 ST_DAP_ENABLE_CPU0	When DAP to AP0 for CPU0 (CM33) debug access is allowed, this bit indicates state 1
9 ST_CPU1_NIDEN	When CPU1 (CM33) "niden" input is state 1, this bit indicates state 1.
8	When CPU1 (CM33) "deben" input is state 1, this bit indicates state 1.

Table continues on the next page...

Table continued from the previous page...

Field	Function
ST_CPU1_DBGEN	
7 ST_CPU0_SPNIDEN	When CPU0 (CM33) "spniden" input is state 1, this bit indicates state 1
6 ST_CPU0_SPIDEN	When CPU0 (CM33) "spiden" input is state 1, this bit indicates state 1
5 ST_CPU0_NIDEN	When CPU0 (CM33) "niden" input is state 1, this bit indicates state 1
4 ST_CPU0_DBGEN	When CPU0 (CM33) "deben" input is state 1, this bit indicates state 1
3-0 ST_TEMPORAL_STATE	TEMPORAL_STATE[3:0] in the ELS_TEMPORAL_STATE register reflects this register

11.4.1.126 ELS AS State1 (ELS_AS_ST1)

Offset

Register	Offset
ELS_AS_ST1	9E4h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved												ST_LDO_CORE_DS		ST_LDO_CORE_VOUT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ST_BOOT_RETRY_CNT				ST_BOOT_MODE		ST_DCDC_DS		ST_DCDC_VOUT		ST_MAIN...	ST_QK_Z...	ST_QK_PUF_SCORE			
W																
Reset	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	1

Fields

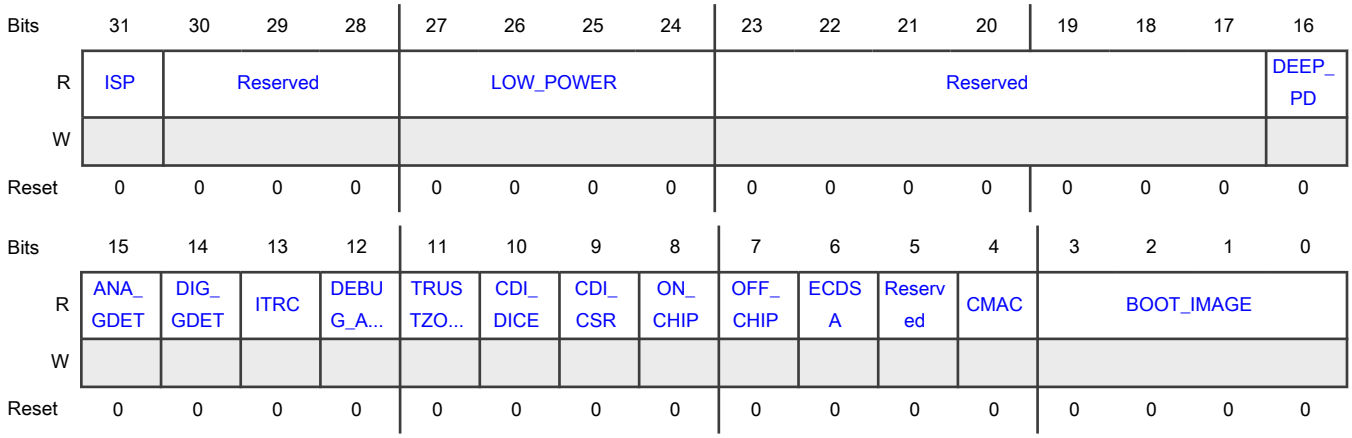
Field	Function
31-20 —	Reserved
19-18 ST_LDO_CORE_DS	LDO_CORE drive strength setting. Default is normal drive.
17-16 ST_LDO_CORE_VOUT	VOUT[1:0] setting on LDO Core register in SPC block will reflect to this register. Default is 1.0V
15-12 ST_BOOT_RETRY_CNT	BOOT_RETRY_CNT[3:0] in the ELS_BOOT_RETRY_CNT register reflects this register
11-10 ST_BOOT_MODE	ISP pin status during boot. By default ISP pin is pulled up. If want to enter ISP mode during boot, ISP pin should be pull down when out of reset.
9-8 ST_DCDC_DS	DCDC drive strength setting. Default is normal drive.
7-6 ST_DCDC_VOUT	VOUT[1:0] setting on DCDC0 register in SPC block will reflect to this register. Default is 1.0V
5 ST_MAIN_CLK_IS_EXT	When MAIN_CLK is running from external clock source either XO32M, XO32K or GPIO CLKIN, this bit indicates state 1
4 ST_QK_ZEROIZED	This register bit indicates the state of "qk_zeroized" output from QK PUF block
3-0 ST_QK_PUF_SCORE	These register bits indicate the state of "qk_puf_score[3:0]" outputs from QK PUF block

11.4.1.127 Boot state captured during boot: Main ROM log (ELS_AS_BOOT_LOG0)

Offset

Register	Offset
ELS_AS_BOOT_LOG0	9E8h

Diagram



Fields

Field	Function
31 ISP	ISP pin state at boot time. ROM copies CMC->MR0[0].
30-28 —	Reserved
27-24 LOW_POWER	Last low-power mode value. ROM copies SPC_LP_MODE field from SPC->SC[7:4].
23-17 —	Reserved
16 DEEP_PD	Boot from deep-power down state.
15 ANA_GDET	Analog glitch detector is enabled during boot.
14 DIG_GDET	Digital glitch detector is enabled during boot.
13 ITRC	ITRC zeroize event is handled in this session of boot.
12 DEBUG_AUTH	Debug authentication done in this session prior to boot.
11	TrustZone preset data is loaded during this boot.

Table continues on the next page...

Table continued from the previous page...

Field	Function
TRUSTZONE	
10 CDI_DICE	CDI per DICE specification is computed on this boot.
9 CDI_CSR	CDI based device keys are derived for CSR harvesting on this boot.
8 ON_CHIP	On-chip Prince is enabled during boot.
7 OFF_CHIP	Off-chip Prince is enabled during boot.
6 ECDSA	ECDSA P-384 verification is done on this boot.
5 —	Reserved
4 CMAC	CMAC verify is used instead of ECDSA verify on this boot.
3-0 BOOT_IMAGE	Boot image source used during this boot. 0000b - Internal flash image 0 0001b - Internal flash image 1 0010b - FlexSPI flash image 0 0011b - FlexSPI flash image 1 0100b - Recovery SPI flash image 0101b - Serial boot image (write-memory and execute ISP command used) 0110b - Receive SB3 containing SB_JUMP command is used. 0111b - Customer SBL/recovery image (Bank1 IFR0). 1000b - NXP MAD recovery image (Bank1 IFR0). 1001b - Reserved. 1010b - Reserved. 1011b - Reserved. 1100b - Reserved. 1101b - Reserved.

Table continues on the next page...

Table continued from the previous page...

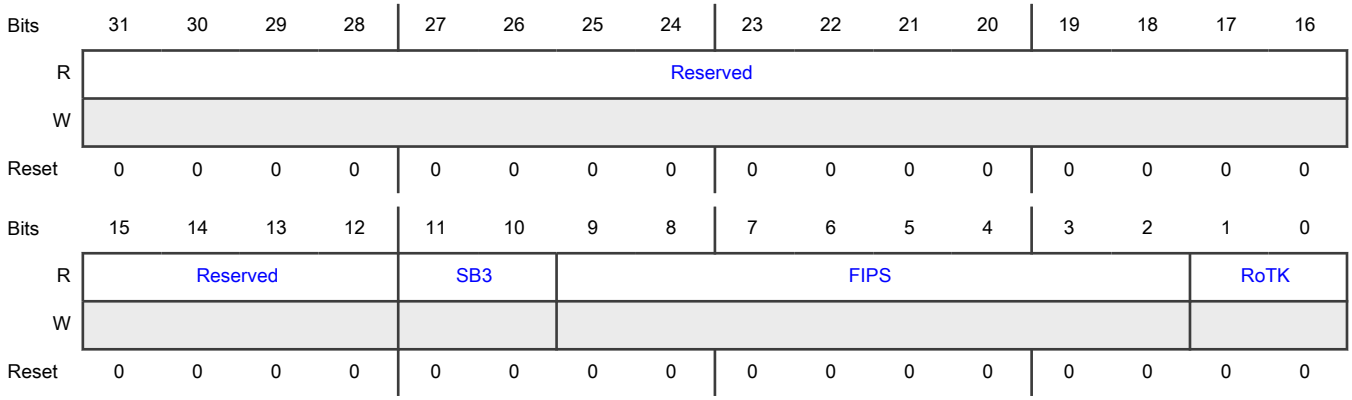
Field	Function
	1110b - Reserved. 1111b - Reserved.

11.4.1.128 Boot state captured during boot: Library log (ELS_AS_BOOT_LOG1)

Offset

Register	Offset
ELS_AS_BOOT_LOG1	9ECh

Diagram



Fields

Field	Function
31-12 —	Reserved
11-10 SB3	SB3 type (valid after nboot_sb3_load_manifest()). 00b - customer fw load/update file. Firmware update, recovery boot, load_to_ram signed and encrypted image/ 01b - NXP Provisioning FW. 10b - ELS signed OEM Provisioning FW.
9-2 FIPS	FIPS self-test is executed and PASS during this boot. When a bit is set, means self-test is executed and it FAILS. When a bit is clear, means corresponding self-test is executed and PASS or it is not executed. bit 9 - FIPS_PUF_STEN bit 8 - FIPS_HKDF_STEN

Table continues on the next page...

Table continued from the previous page...

Field	Function
	bit 7 - FIPS_CKDF_STEN bit 6 - FIPS_CMAC_STEN bit 5 - FIPS_DRBG_STEN bit 4 - FIPS_ECDSA_STEN bit 3 - FIPS_AES_STEN bit 2 - FIPS_SHA_STEN
1-0 RoTK	RoTK index used for this boot.

11.4.1.129 Boot state captured during boot: Hardware status signals log (ELS_AS_BOOT_LOG2)

Offset

Register	Offset
ELS_AS_BOOT_LOG2	9F0h

Function
System reset and VBAT event status log.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CMC_SRS2								VBAT_STATUS1						Reserved	CMC_SRS1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CMC_SRS1								VBAT_STATUS0		CMC_SRS0					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-24 CMC_SRS2	CMC->SRS[31:24]

Table continues on the next page...

Table continued from the previous page...

Field	Function
23-18 VBAT_STATUS1	VBAT->STATUSA[11:6] | ~VBAT->STATUSB[11:6]
17 —	Reserved
16-8 CMC_SRS1	CMC->SRS[16:8]
7-6 VBAT_STATUS0	VBAT->STATUSA[1:0] | ~VBAT->STATUSB[1:0]
5-0 CMC_SRS0	CMC->SRS[5:0]

11.4.1.130 Boot state captured during boot: Security log (ELS_AS_BOOT_LOG3)

Offset

Register	Offset
ELS_AS_BOOT_LOG3	9F4h

Function

Persistent Security error counters (a.k.a. monotonic NVM counters) from active CFPA page.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ERR_ITRC_COUNT								ERR_AUTH_FAIL_COUNT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-16 —	Reserved
15-8 ERR_ITRC_COUNT	CFPA->ERR_ITRC_COUNT[7:0]
7-0 ERR_AUTH_FAIL_COUNT	CFPA->ERR_AUTH_FAIL_COUNT[7:0]

11.4.1.131 ELS AS Flag0 (ELS_AS_FLAG0)

Offset

Register	Offset
ELS_AS_FLAG0	9F8h

Function
ELS AS flag0

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		FLAG_CL...	FLAG_LH...	FLAG_VO...	FLAG_TE...	FLAG_CW...	FLAG_WD...	FLAG_CW...	FLAG_WD...	FLAG_LV...	FLAG_LV...	FLAG_CP...	FLAG_CP...	FLAG_SE...	FLAG_FL...
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FLAG_TA...	FLAG_AN...	FLAG_EL...	FLAG_QK...	FLAG_CW...	FLAG_WD...	FLAG_CW...	FLAG_WD...	Reserved		FLAG_LV...	Reserved	EFUSE_A...	FLAG_AP...	FLAG_AP...	FLAG_AP...
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
29 FLAG_CLKTAMPER_DET_IRQ_OCCURED	This flag bit is set as 1 when clock temper IRQ is enabled and IRQ event is triggered. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
28 FLAG_LHTTAMPER_DET_IRQ_OCCURED	This flag bit is set as 1 when light temper IRQ is enabled and IRQ event is triggered. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
27 FLAG_VOLTAMPER_DET_IRQ_OCCURED	This flag bit is set as 1 when voltage temper IRQ is enabled and IRQ event is triggered. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
26 FLAG_TEMPTAMPER_DET_IRQ_OCCURED	This flag bit is set as 1 when temperature temper IRQ is enabled and IRQ event is triggered. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
25 FLAG_CWDT1_IRQ_OCCURED	This flag bit is set as 1 when Code WatchDog Timer 1 IRQ is enabled and IRQ event is triggered. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
24 FLAG_WDT1_IRQ_OCCURED	This flag bit is set as 1 when WatchDog Timer 1 IRQ is enabled and IRQ event is triggered. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
23 FLAG_CWDT1_RESET_OCCURED	This flag bit is set as 1 when Code WatchDog Timer 1 reset is enabled and reset event is triggered. This register is cleared 0 by AO domain POR. 0b - Not Triggered 1b - Triggered
22 FLAG_WDT1_RESET_OCCURED	This flag bit is set as 1 when WatchDog Timer 1 reset is enabled and reset event is triggered. This register is cleared 0 by AO domain POR. 0b - Not Triggered 1b - Triggered
21	This flag register is set 1 when VDD LVD event is triggered. This register is cleared 0 by PMC reset event.

Table continues on the next page...

Table continued from the previous page...

Field	Function
FLAG_LVD_VDDIO_OCCURED	0b - Not Triggered 1b - Triggered
20 FLAG_LVD_VSYS_OCCURED	This flag register is set 1 when VDD_SYS LVD event is triggered. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
19 FLAG_CPU0_NS_D_ACC_OCCURED	This flag bit is set as 1 when CPU0 (CM33) makes non-secure data transactions. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
18 FLAG_CPU0_NS_C_ACC_OCCURED	This flag bit is set as 1 when CPU0 (CM33) makes non-secure code transactions. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
17 FLAG_SEC_VIOL_IRQ_OCURRED	This flag bit is set as 1 when security violation is indicated from FLASH sub-system or AHB bus matrix. 0b - Not Triggered 1b - Triggered
16 FLAG_FLASH_ECC_INVALID	This flag bit is set as 1 when FLASH controller indicates ECC error. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
15 FLAG_TAMPER_EVENT_DETECTED	This flag bit is set as 1 when tamper event is flagged from TDET. This register is cleared 0 by AO domain POR or by PMC reset event, if tamper detection event is cleared by software. 0b - Not Triggered 1b - Triggered
14 FLAG_ANA_GLITCH_DETECTED	This flag bit is set as 1 when ANALOG GDET error is flagged in SYSCON block. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
13 FLAG_ELS_GLITCH_DETECTED	This flag bit is set as 1 when GDET error is flagged. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered

Table continues on the next page...

Table continued from the previous page...

Field	Function
12 FLAG_QK_ERROR	This flag bit is set as 1 when QK_ERROR is flagged from QK PUF block. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
11 FLAG_CWDT0_IRQ_OCCURED	This flag bit is set as 1 when Code WatchDog Timer 0 IRQ is enabled and IRQ event is triggered. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
10 FLAG_WDT0_IRQ_OCCURED	This flag bit is set as 1 when WatchDog Timer 0 IRQ is enabled and IRQ event is triggered. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
9 FLAG_CWDT0_RESET_OCCURED	This flag bit is set as 1 when Code WatchDog Timer 0 reset is enabled and reset event is triggered. This register is cleared 0 by AO domain POR. 0b - Not Triggered 1b - Triggered
8 FLAG_WDT0_RESET_OCCURED	This flag bit is set as 1 when WatchDog Timer 0 reset is enabled and reset event is triggered. This register is cleared 0 by AO domain POR. 0b - Not Triggered 1b - Triggered
7-6 —	Reserved
5 FLAG_LVD_CORE_OCCURED	This flag register is set 1 when VDD_CORE LVD event is triggered. This register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
4 —	Reserved
3 EFUSE_ATTACK_DETECT	OTPC can output attack_detect signal when it detects attack when load shadow registers. The output will be cleared by reset. ELS_AS_FLAG is reset by PoR, so the status can be recorded. 0b - Not Triggered 1b - Triggered
2	This flag bit is set as 1 when DAP enables AP3 for DSP (CoolFlux) debug access. The register is cleared 0 by PMC reset event.

Table continues on the next page...

Table continued from the previous page...

Field	Function
FLAG_AP_ENABLE_DSP	0b - Not Triggered 1b - Triggered
1 FLAG_AP_ENABLE_CPU1	This flag bit is set as 1 when DAP enables AP1 for CPU1 (CM33) debug access. The register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered
0 FLAG_AP_ENABLE_CPU0	This flag bit is set as 1 when DAP enables AP0 for CPU0 (CM33) debug access. The register is cleared 0 by PMC reset event. 0b - Not Triggered 1b - Triggered

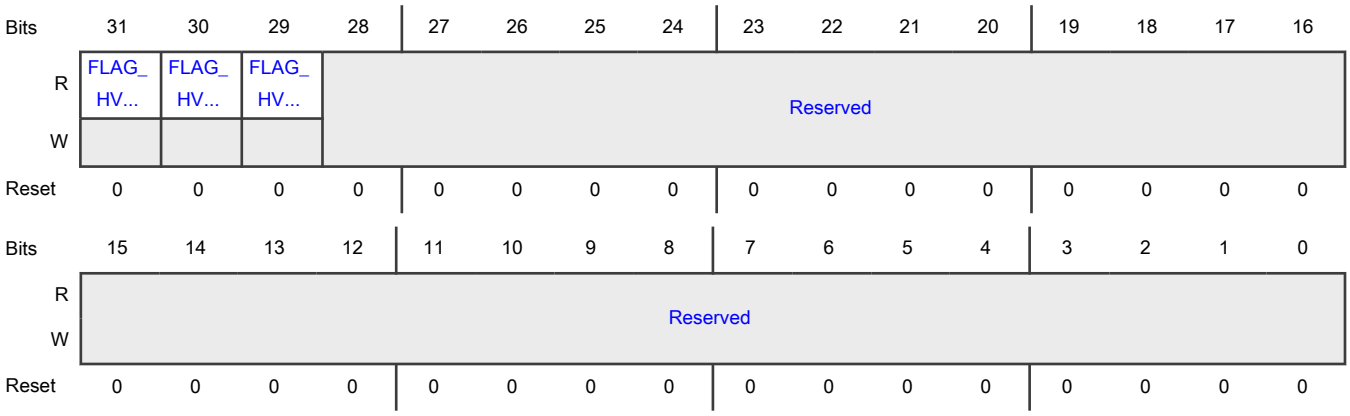
11.4.1.132 ELS AS Flag1 (ELS_AS_FLAG1)

Offset

Register	Offset
ELS_AS_FLAG1	9FCh

Function
ELS AS flag1.

Diagram



Fields

Field	Function
31	This flag bit is set as 1 when HVD from VDD power domain is triggered

Table continues on the next page...

Table continued from the previous page...

Field	Function
FLAG_HVD_VDDIO_OCCURED	0b - Not Triggered 1b - Triggered
30 FLAG_HVD_VSYS_OCCURED	This flag bit is set as 1 when HVD from VDD_SYS power domain is triggered 0b - Not Triggered 1b - Triggered
29 FLAG_HVD_CORE_OCCURED	This flag bit is set as 1 when HVD from VDD_CORE power domain is triggered. 0b - Not Triggered 1b - Triggered
28-0 —	Reserved

11.4.1.133 Clock Control (CLOCK_CTRL)

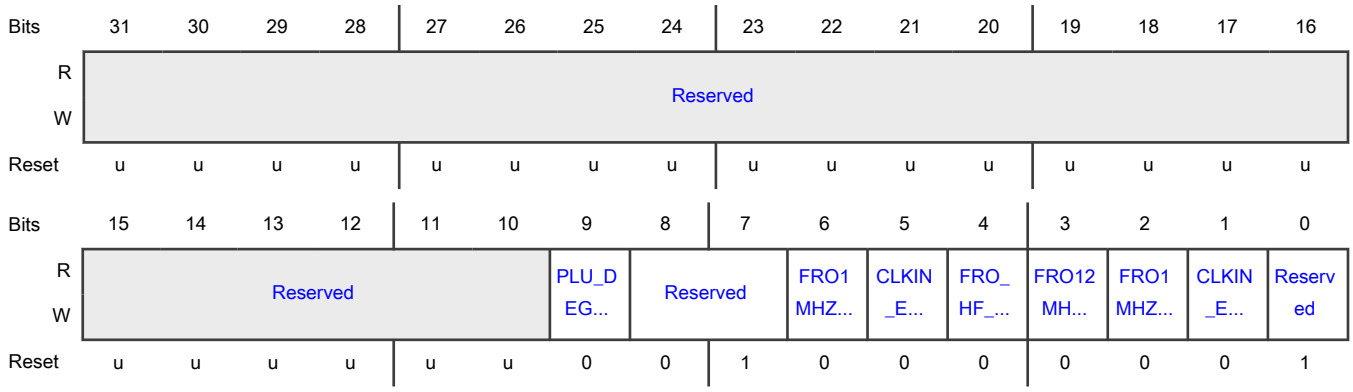
Offset

Register	Offset
CLOCK_CTRL	A18h

Function

Various system clocks enable controls

Diagram



Fields

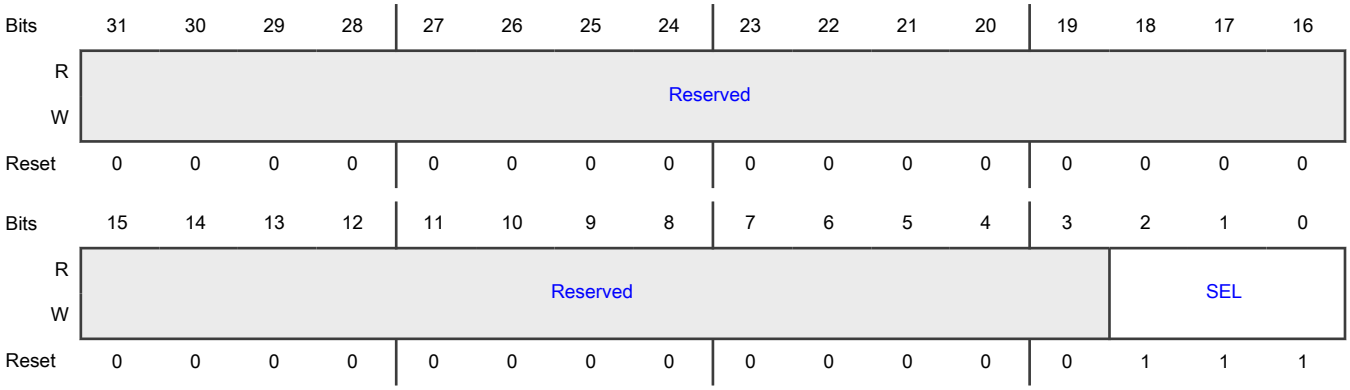
Field	Function
31-10 —	Reserved
9 PLU_DEGLITCH_CLK_ENA	Enables clocks FRO_1MHz and FRO_12MHz for PLU deglitching. 0b - Clock is not enabled 1b - Clock is enabled
8-7 —	Reserved
6 FRO1MHZ_CLK_ENA	Enables FRO_1MHz clock for clock muxing in clock gen 0b - Clock is not enabled 1b - Clock is enabled
5 CLKIN_ENA	Enables clk_in clock for MICFIL, EMVSIM0/1, CAN0/1, I3C0/1, SAI0/1, SINC Filter (SINC), TSI, USBFS, SCT, uSDHC, clkout. 0b - Clock is not enabled 1b - Clock is enabled
4 FRO_HF_ENA	Enables FRO HF clock for the Frequency Measure module 0b - Clock is not enabled 1b - Clock is enabled
3 FRO12MHZ_ENA	Enables the FRO_12MHz clock for the Flash, LPTMR0/1, and Frequency Measurement modules 0b - Clock is not enabled 1b - Clock is enabled
2 FRO1MHZ_ENA	Enables the FRO_1MHz clock for RTC module and for UTICK 0b - Clock is not enabled 1b - Clock is enabled
1 CLKIN_ENA_FM_USBH_LPT	Enables the clk_in clock for the Frequency Measurement, USB HS and LPTMR0/1 modules. 0b - Clock is not enabled 1b - Clock is enabled
0 —	Reserved

11.4.1.134 I3C1 Functional Clock Selection (I3C1FCLKSEL)

Offset

Register	Offset
I3C1FCLKSEL	B30h

Diagram



Fields

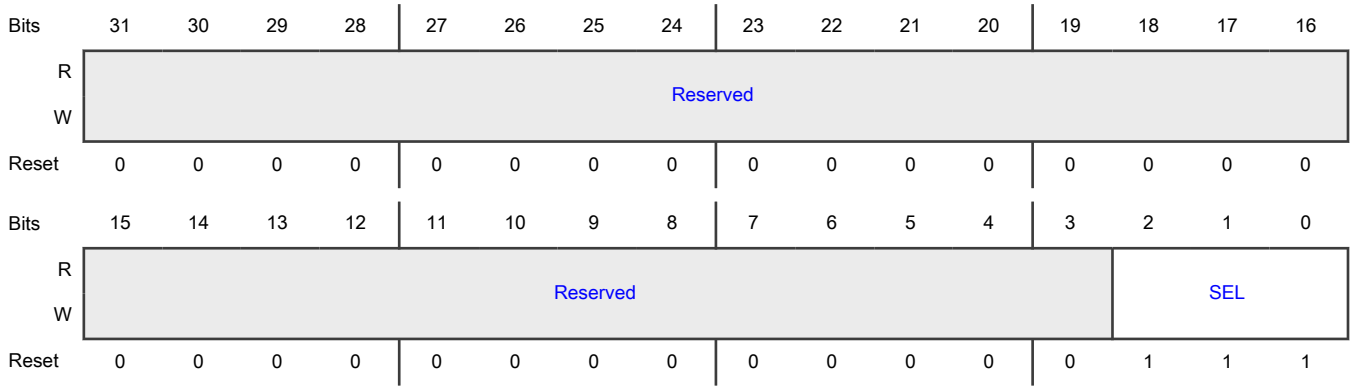
Field	Function
31-3 —	Reserved
2-0 SEL	I3C1 clock select 000b - No clock 001b - PLL0 clock 010b - CLKIN clock 011b - FRO_HF clock 100b - No clock 101b - PLL1_clk0 clock 110b - USB PLL clock 111b - No clock

11.4.1.135 Selects the I3C1 Time Control clock (I3C1FCLKSTCSEL)

Offset

Register	Offset
I3C1FCLKSTCSEL	B34h

Diagram



Fields

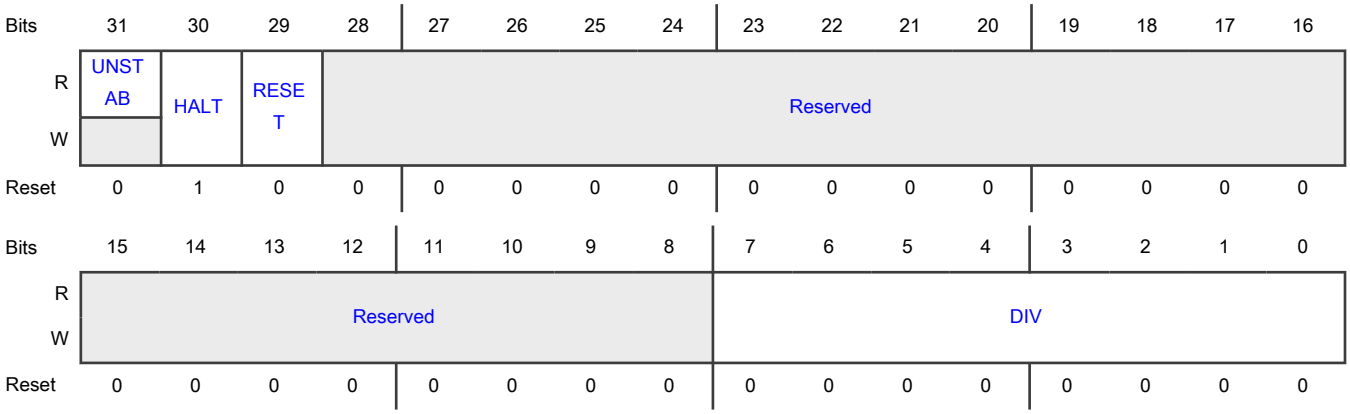
Field	Function
31-3 —	Reserved
2-0 SEL	I3C1 FCLK_STC clock select 000b - I3C1 functional clock I3C1FCLK 001b - FRO_1M clock 010b - No clock 011b - No clock 100b - No clock 101b - No clock 110b - No clock 111b - No clock

11.4.1.136 I3C1 FCLK_STC Clock Divider (I3C1FCLKSTCDIV)

Offset

Register	Offset
I3C1FCLKSTCDIV	B38h

Diagram



Fields

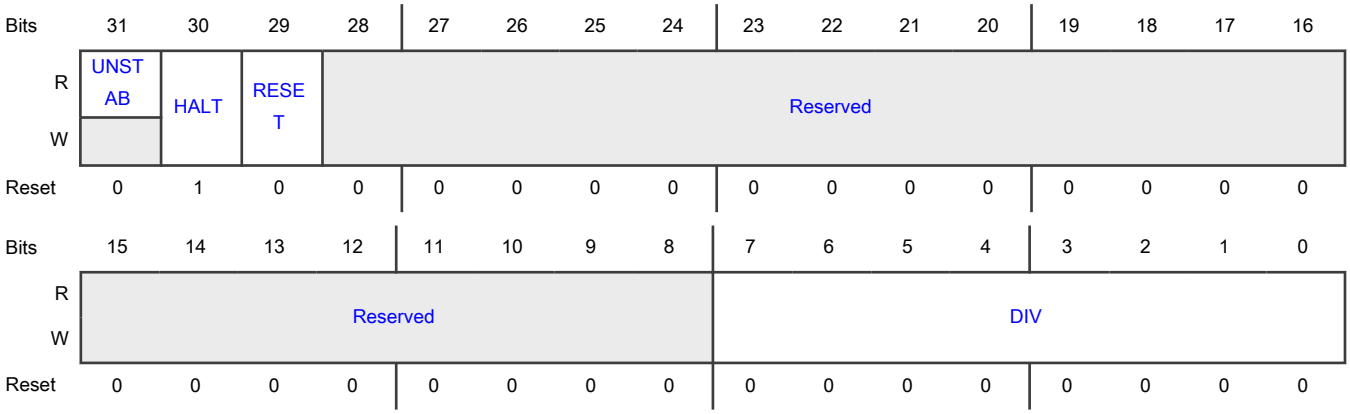
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.137 I3C1 FCLK Slow clock Divider (I3C1FCLKSDIV)

Offset

Register	Offset
I3C1FCLKSDIV	B3Ch

Diagram



Fields

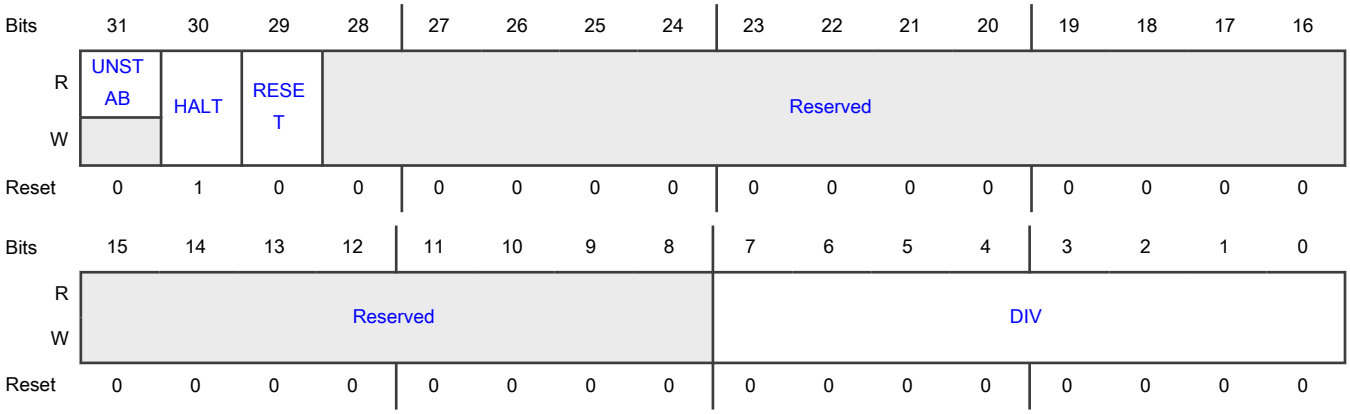
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.138 I3C1 Functional Clock FCLK Divider (I3C1FCLKDIV)

Offset

Register	Offset
I3C1FCLKDIV	B40h

Diagram



Fields

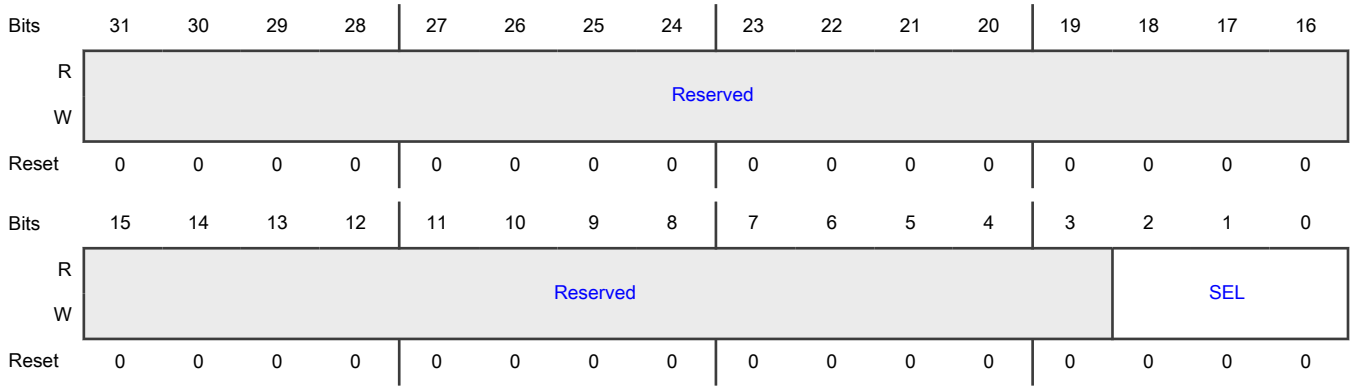
Field	Function
31 UNSTAB	Divider status flag 0b - Divider clock is stable 1b - Clock frequency is not stable
30 HALT	Halts the divider counter 0b - Divider clock is running 1b - Divider clock is stopped
29 RESET	Resets the divider counter 0b - Divider is not reset 1b - Divider is reset
28-8 —	Reserved
7-0 DIV	Clock divider value The divider value = (DIV + 1)

11.4.1.139 I3C1 FCLK Slow Selection (I3C1FCLKSSEL)

Offset

Register	Offset
I3C1FCLKSSEL	B44h

Diagram



Fields

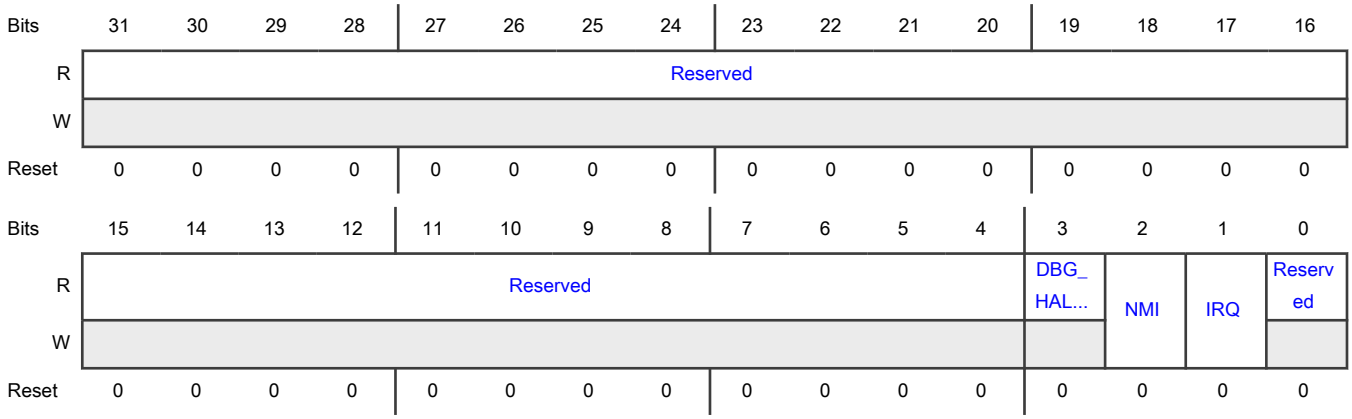
Field	Function
31-3 —	Reserved
2-0 SEL	I3C1 FCLK Slow Clock Select 000b - FRO_1M clock 001b - No clock 010b - No clock 011b - No clock 100b - No clock 101b - No clock 110b - No clock 111b - No clock

11.4.1.140 ETB Counter Status Register (ETB_STATUS)

Offset

Register	Offset
ETB_STATUS	B50h

Diagram



Fields

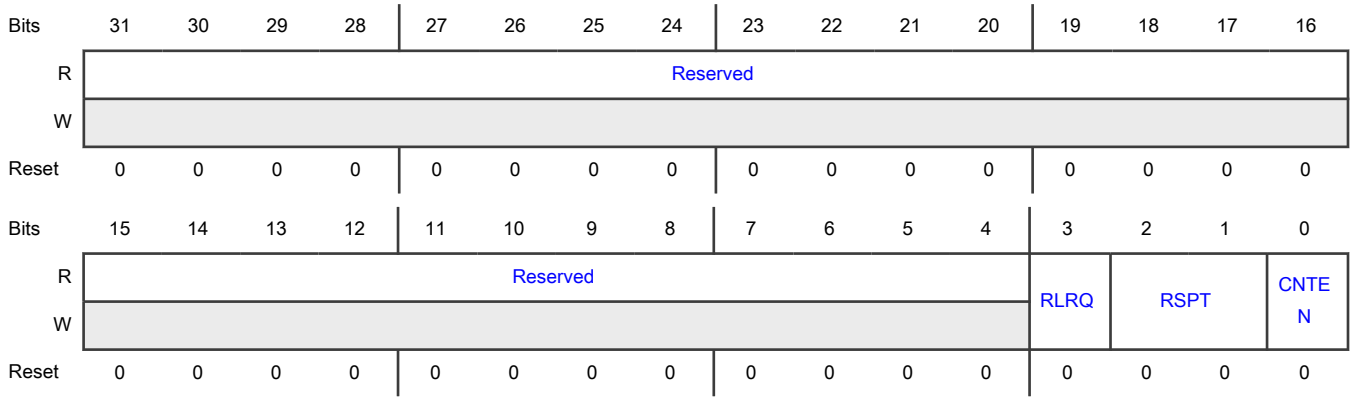
Field	Function
31-4 —	Reserved
3 DBG_HALT_REQ	Debug halt request 0b - The debug halt request signal is not asserted 1b - The debug halt request signal is asserted when the ETB count expires
2 NMI	ETB NMI 0b - ETB NMI is not asserted 1b - ETB NMI is asserted. Write 1 to clear it.
1 IRQ	ETB Interrupt 0b - ETB interrupt is not asserted 1b - ETB interrupt is asserted when ETB count expires. Write 1 to clear it.
0 —	Reserved

11.4.1.141 ETB Counter Control Register (ETB_COUNTER_CTRL)

Offset

Register	Offset
ETB_COUNTER_CTRL	B54h

Diagram



Fields

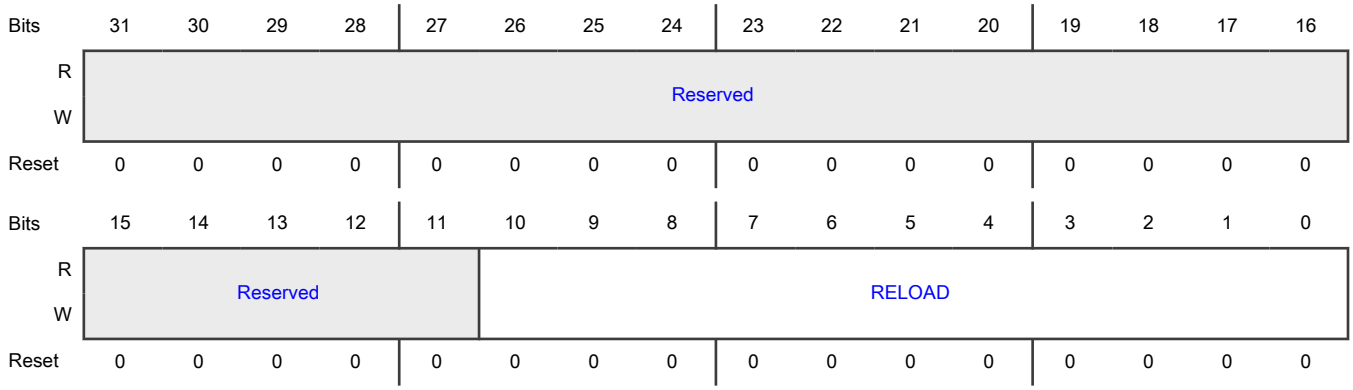
Field	Function
31-4 —	Reserved
3 RLRQ	Reload request Reloads the ETB packet counter with the ETB_COUNTER_RELOAD value. It is always zero when read. If IRQ or NMI interrupts were enabled and an NMI or IRQ interrupt was generated on counter expiration, setting this bit clears the pending NMI or IRQ interrupt request. If debug halt was enabled and a debug halt request was asserted on counter expiration, setting this bit clears the debug halt request. 0b - No effect 1b - Clears pending debug halt, NMI, or IRQ interrupt requests
2-1 RSPT	Response Type 00b - No response when the ETB count expires 01b - Generates a normal interrupt when the ETB count expires 10b - Generates an NMI interrupt when the ETB count expires 11b - Generates a debug halt when the ETB count expires via CPU0 CTICHIN[2]
0 CNTEN	Enables the ETB counter 0b - ETB counter is disabled 1b - ETB counter is enabled

11.4.1.142 ETB Counter Reload Register (ETB_COUNTER_RELOAD)

Offset

Register	Offset
ETB_COUNTER_RELOAD	B58h

Diagram



Fields

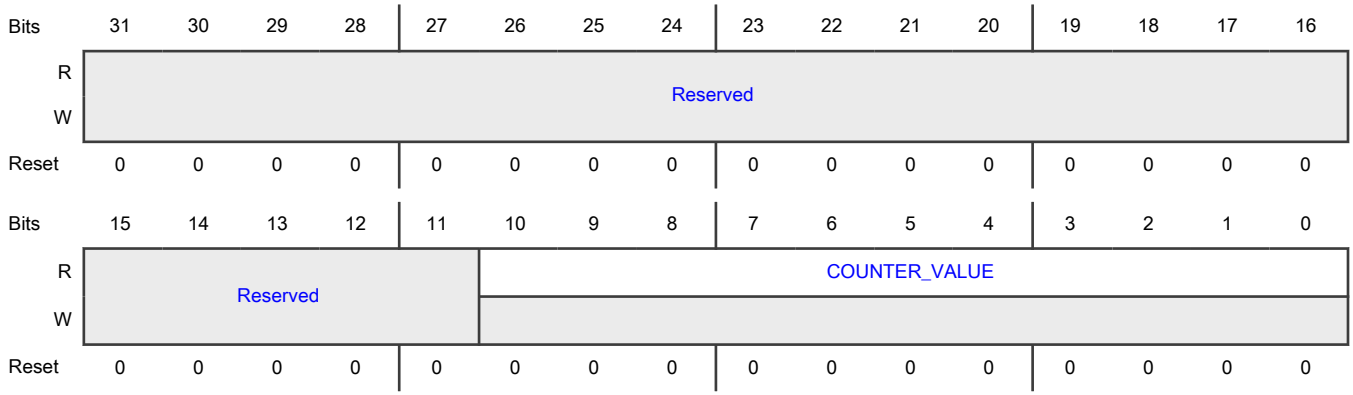
Field	Function
31-11 —	Reserved
10-0 RELOAD	Byte count reload value Indicates the 0-mod-4 value that the counter reloads to. Writing a non-0-mod-4 value to this field results in a bus error.

11.4.1.143 ETB Counter Value Register (ETB_COUNTER_VALUE)

Offset

Register	Offset
ETB_COUNTER_VALUE	B5Ch

Diagram



Fields

Field	Function
31-11 —	Reserved
10-0 COUNTER_VALUE	Byte count counter value Indicates the current 0-mod-4 value of the counter

11.4.1.144 Gray to Binary Converter Gray code_gray[31:0] (GRAY_CODE_LSB)

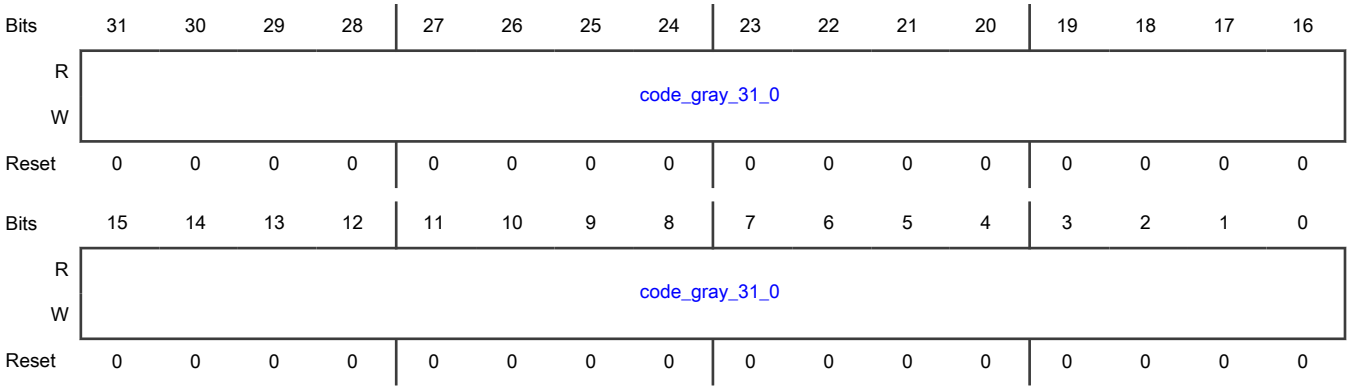
Offset

Register	Offset
GRAY_CODE_LSB	B60h

Function

The Gray Code LSB Input register (CODE_GRAY_LSB) contains the least-significant portion of the Gray code to be converted back to binary.

Diagram



Fields

Field	Function
31-0 code_gray_31_0	Gray code [31:0] CODE_GRAY_LSB is the least-significant 32 bits of the 42-bit Gray code to be converted.

11.4.1.145 Gray to Binary Converter Gray code_gray[41:32] (GRAY_CODE_MSB)

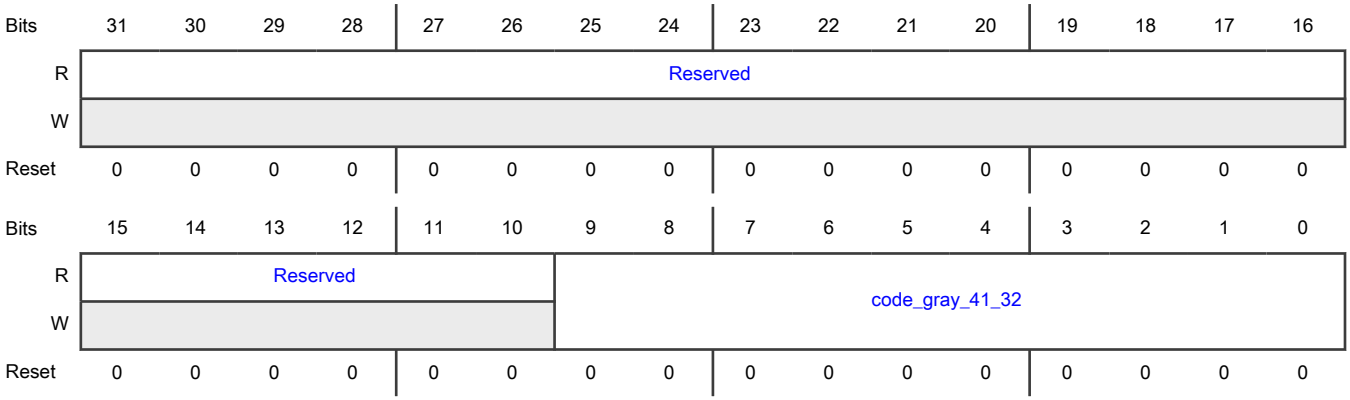
Offset

Register	Offset
GRAY_CODE_MSB	B64h

Function

The Gray Code MSB Input register (CODE_GRAY_MSB) contains the most-significant portion of the Gray code to be converted back to binary.

Diagram



Fields

Field	Function
31-10 —	Reserved
9-0 code_gray_41_32	Gray code [41:32] CODE_GRAY_MSB is the most-significant 10 bits of the 42-bit Gray code to be converted.

11.4.1.146 Gray to Binary Converter Binary Code [31:0] (BINARY_CODE_LSB)

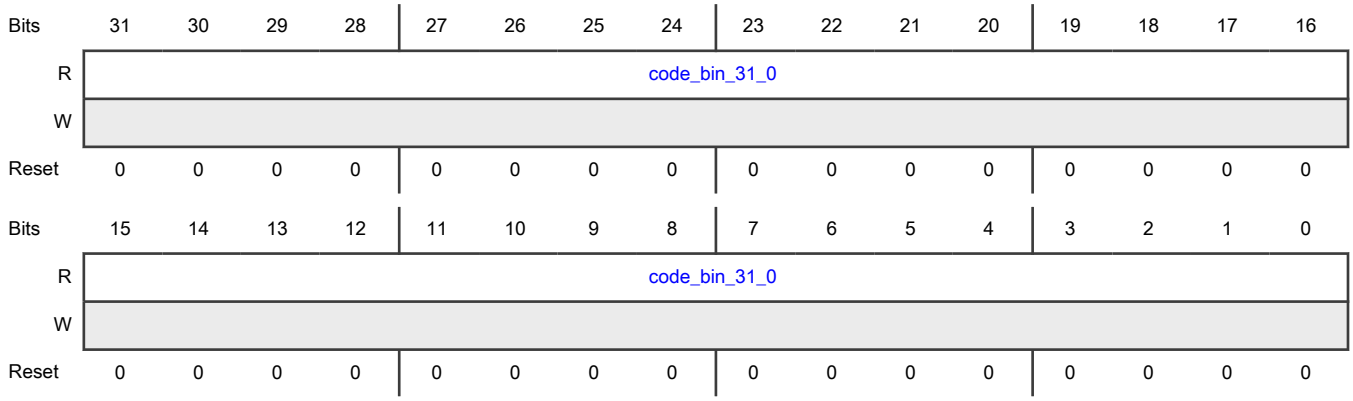
Offset

Register	Offset
BINARY_CODE_LSB	B68h

Function

The Binary Code LSB register (BINARY_CODE_LSB) contains the least-significant portion of the code converted from Gray to binary coding.

Diagram



Fields

Field	Function
31-0 code_bin_31_0	Binary code [31:0] code_bin_31_0 is the least-significant 32 bits of the 42-bit converted code.

11.4.1.147 Gray to Binary Converter Binary Code [41:32] (BINARY_CODE_MSB)

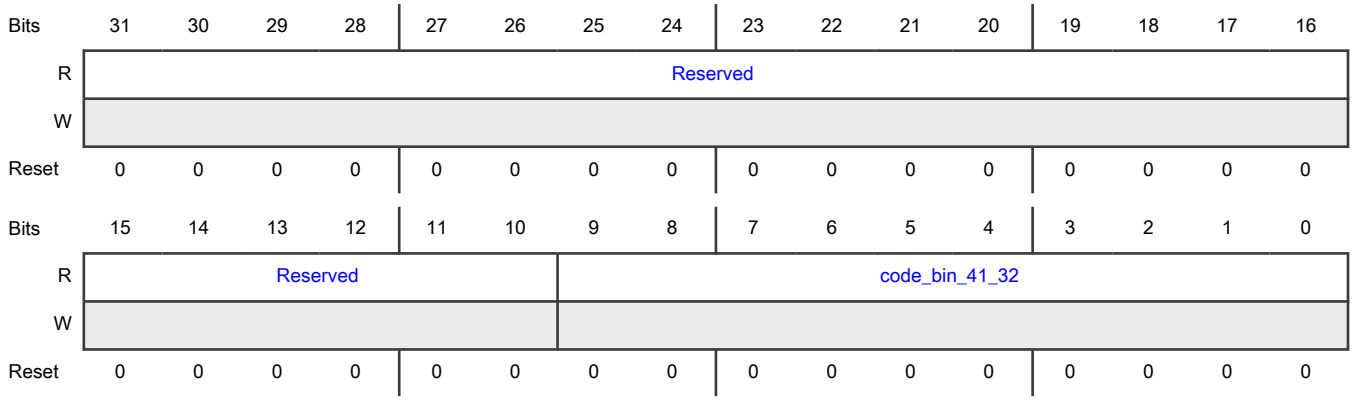
Offset

Register	Offset
BINARY_CODE_MSB	B6Ch

Function

The Binary Code MSB register (BINARY_CODE_MSB) contains the most-significant portion of the code converted from Gray to binary coding.

Diagram



Fields

Field	Function
31-10 —	Reserved
9-0 code_bin_41_32	Binary code [41:32] code_bin_41_32 is the most-significant 10 bits of the 42-bit converted code.

11.4.1.148 Control Automatic Clock Gating (AUTOCLKGATEOVERRIDE)

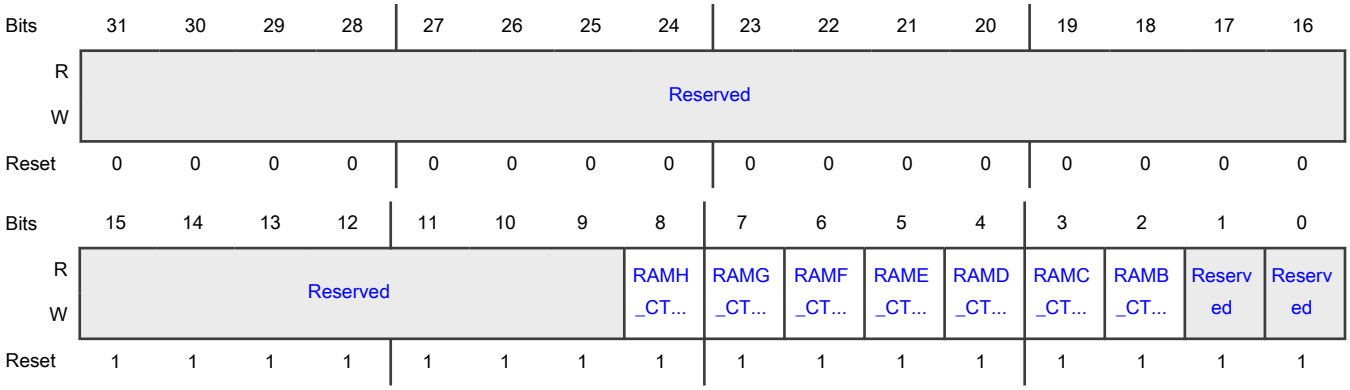
Offset

Register	Offset
AUTOCLKGATEOVERRIDE	E04h

Function

This register allows selectively disabling automatic clock gating for device SRAMs. By default, automatic clock gating turns off clocks to each internal SRAM after 16 bus clocks with no activity. This saves power when the SRAMs are not used for a period of time. When turned off due to automatic clock gating, there is a 1 clock delay for the next access to an SRAM. Automatic clock gating may be disabled for time-critical code, which may typically give a 1 or 2% speed improvement.

Diagram



Fields

Field	Function
31-9 —	Reserved
8	Controls automatic clock gating for the RAMG Controller

Table continues on the next page...

Table continued from the previous page...

Field	Function
RAMH_CTRL	0b - Automatic clock gating is not overridden 1b - Automatic clock gating is overridden (Automatic clock gating is disabled).
7 RAMG_CTRL	Controls automatic clock gating for the RAMG Controller 0b - Automatic clock gating is not overridden 1b - Automatic clock gating is overridden (Automatic clock gating is disabled).
6 RAMF_CTRL	Controls automatic clock gating for the RAMF Controller 0b - Automatic clock gating is not overridden 1b - Automatic clock gating is overridden (Automatic clock gating is disabled).
5 RAME_CTRL	Controls automatic clock gating for the RAMD Controller. 0b - Automatic clock gating is not overridden 1b - Automatic clock gating is overridden (Automatic clock gating is disabled).
4 RAMD_CTRL	Controls automatic clock gating for the RAMD Controller 0b - Automatic clock gating is not overridden 1b - Automatic clock gating is overridden (Automatic clock gating is disabled).
3 RAMC_CTRL	Controls automatic clock gating for the RAMC Controller 0b - Automatic clock gating is not overridden 1b - Automatic clock gating is overridden (Automatic clock gating is disabled).
2 RAMB_CTRL	Controls automatic clock gating for the RAMB Controller 0b - Automatic clock gating is not overridden 1b - Automatic clock gating is overridden (Automatic clock gating is disabled).
1 —	Reserved Keep the default value.
0 —	Reserved

11.4.1.149 Control Automatic Clock Gating C (AUTOCLKGATEOVERRIDEC)

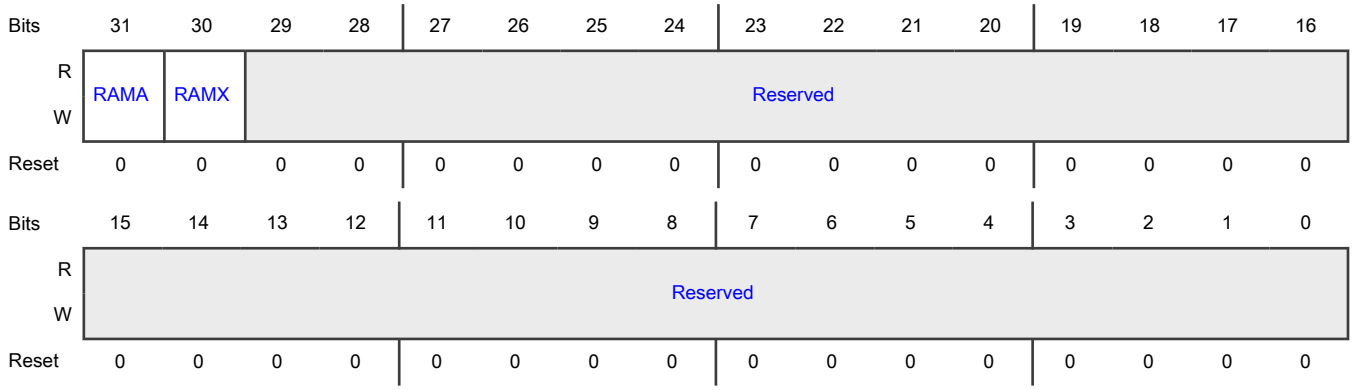
Offset

Register	Offset
AUTOCLKGATEOVERRIDEC	E2Ch

Function

This register allows selectively disabling automatic clock gating for device SRAMs. By default, automatic clock gating turns off clocks to each internal SRAM after 16 bus clocks with no activity. This saves power when the SRAMs are not used for a period of time. When turned off due to automatic clock gating, there is a 1 clock delay for the next access to an SRAM. Automatic clock gating may be disabled for time-critical code, which may typically give a 1 or 2% speed improvement.

Diagram



Fields

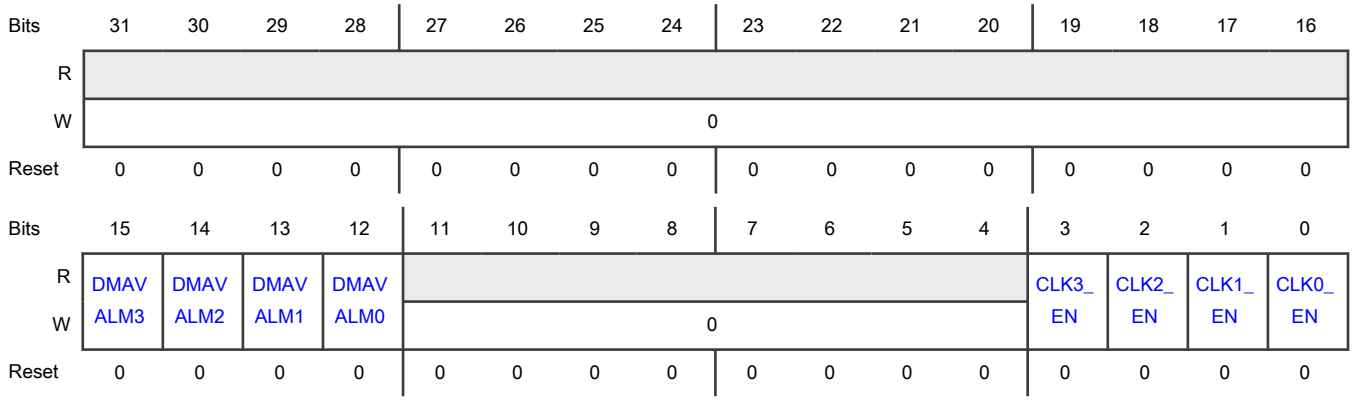
Field	Function
31 RAMA	Controls automatic clock gating of the RAMA controller 0b - Automatic clock gating is not overridden 1b - Automatic clock gating is overridden (Automatic clock gating is disabled).
30 RAMX	Controls automatic clock gating of the RAMX controller 0b - Automatic clock gating is not overridden 1b - Automatic clock gating is overridden (Automatic clock gating is disabled).
29-0 —	Reserved

11.4.1.150 PWM0 Submodule Control (PWM0SUBCTL)

Offset

Register	Offset
PWM0SUBCTL	E38h

Diagram



Fields

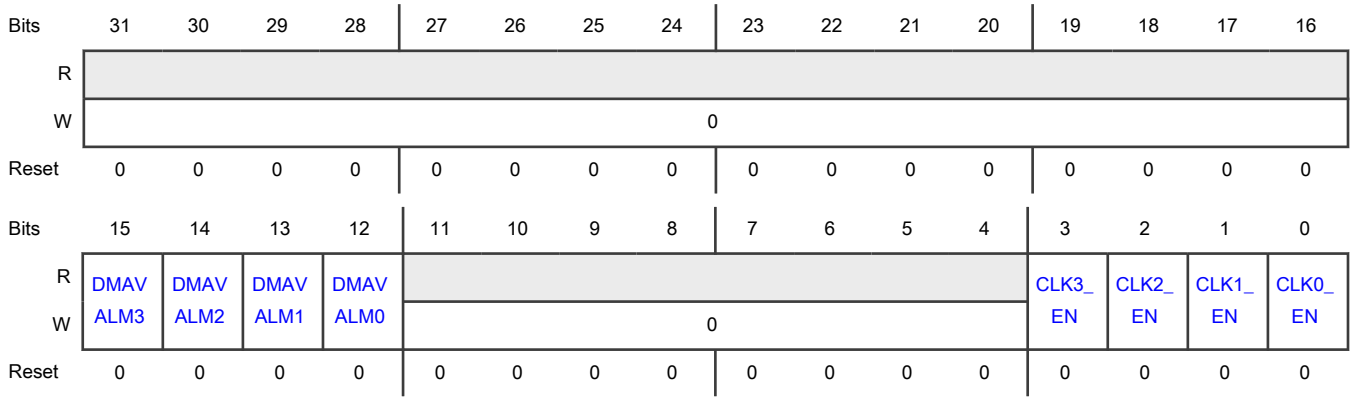
Field	Function
31-16 —	Reserved
15-12 DMAVALMn	PWM0 submodule n DMA compare value done mask When the mask is used, DMA should write the PWM's MCTRL register in the end to set the LDOK bits of relevant submodules at the same time. Since the DMA done signal is blocked, the DMA request is not cleared automatically by the hardware. After the DMA transfer is completed for the current DMA request, the DMA request needs to be cleared manually by clearing SMxDMAEN[VALDE] followed by clearing SMxSTS[RF] with the software.
11-4 —	Reserved
3-0 CLKn_EN	Enables PWM0 SUB Clockn

11.4.1.151 PWM1 Submodule Control (PWM1SUBCTL)

Offset

Register	Offset
PWM1SUBCTL	E3Ch

Diagram



Fields

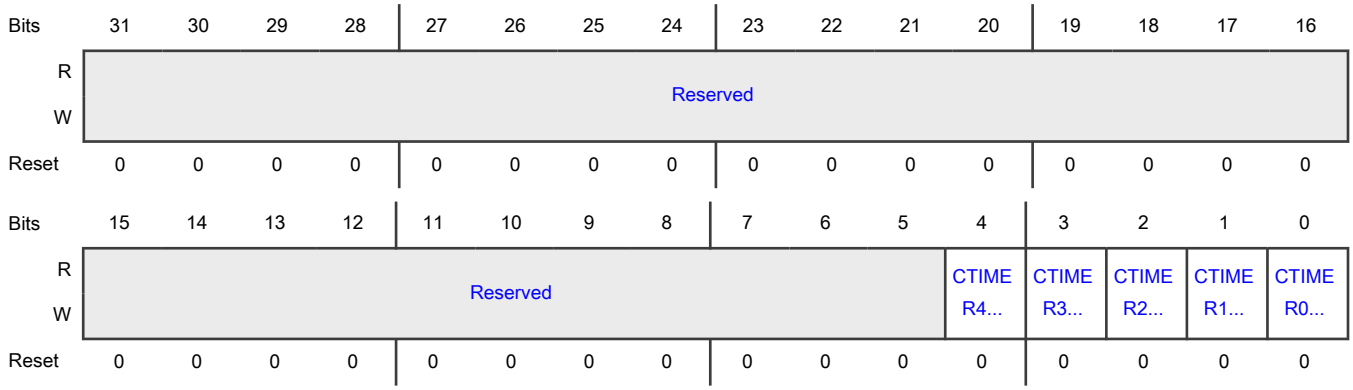
Field	Function
31-16 —	Reserved
15-12 DMAVALMn	PWM1 submodule n DMA compare value done mask When the mask is used, DMA should write the PWM's MCTRL register in the end to set the LDOK bits of relevant submodules at the same time. Since the DMA done signal is blocked, the DMA request is not cleared automatically by the hardware. After the DMA transfer is completed for the current DMA request, the DMA request needs to be cleared manually by clearing SMxDMAEN[VALDE] followed by clearing SMxSTS[RF] with the software.
11-4 —	Reserved
3-0 CLKn_EN	Enables PWM1 SUB Clockn

11.4.1.152 CTIMER Global Start Enable (CTIMERGLOBALSTARTEN)

Offset

Register	Offset
CTIMERGLOBALSTARTEN	E40h

Diagram



Fields

Field	Function
31-5 —	Reserved
4 CTIMER4_CLK_EN	Enables the CTIMER4 function clock 0b - Disable 1b - Enable
3 CTIMER3_CLK_EN	Enables the CTIMER3 function clock 0b - Disable 1b - Enable
2 CTIMER2_CLK_EN	Enables the CTIMER2 function clock 0b - Disable 1b - Enable
1 CTIMER1_CLK_EN	Enables the CTIMER1 function clock 0b - Disable 1b - Enable
0 CTIMER0_CLK_EN	Enables the CTIMER0 function clock 0b - Disable 1b - Enable

11.4.1.153 RAM ECC Enable Control (ECC_ENABLE_CTRL)

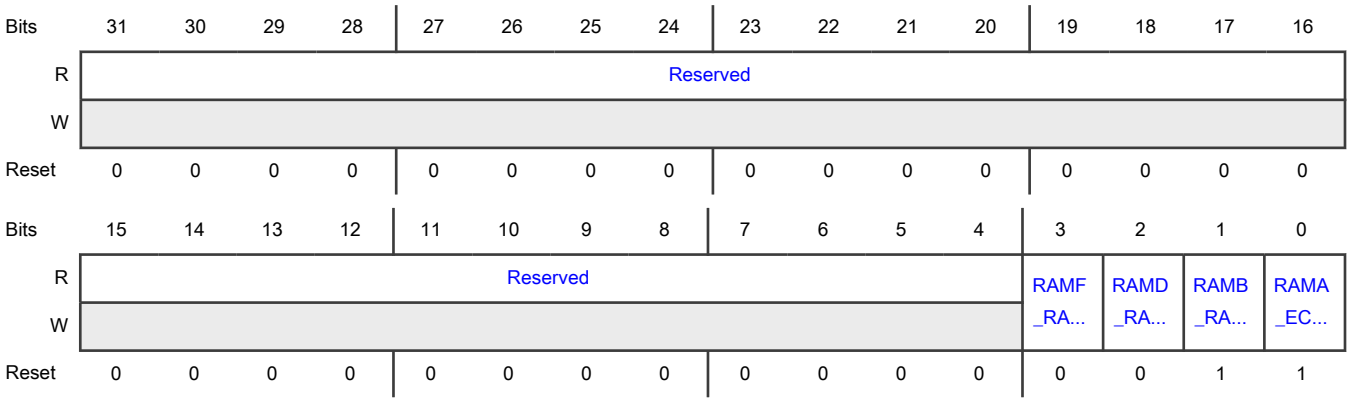
Offset

Register	Offset
ECC_ENABLE_CTRL	E44h

Function

This register is used to enable/disable RAM blocks ECC function. Only combinations 'b0000, 'b0001, 'b0011, 'b0111 and 'b1111 are allowed. For details of RAM blocks and ECC availability for a specific part number refer to the device data sheet .

Diagram



Fields

Field	Function
31-4 —	Reserved
3 RAMF_RAME_ECC_ENABLE	RAMF and RAME ECC enable 0b - ECC is disabled 1b - ECC is enabled
2 RAMD_RAMC_ECC_ENABLE	RAMD and RAMC ECC enable 0b - ECC is disabled 1b - ECC is enabled
1 RAMB_RAMX_ECC_ENABLE	RAMB and RAMX ECC enable 0b - ECC is disabled 1b - ECC is enabled
0 RAMA_ECC_ENABLE	RAMA ECC enable 0b - ECC is disabled 1b - ECC is enabled

11.4.1.154 Control Write Access to Security (DEBUG_LOCK_EN)

Offset

Register	Offset
DEBUG_LOCK_EN	FA0h

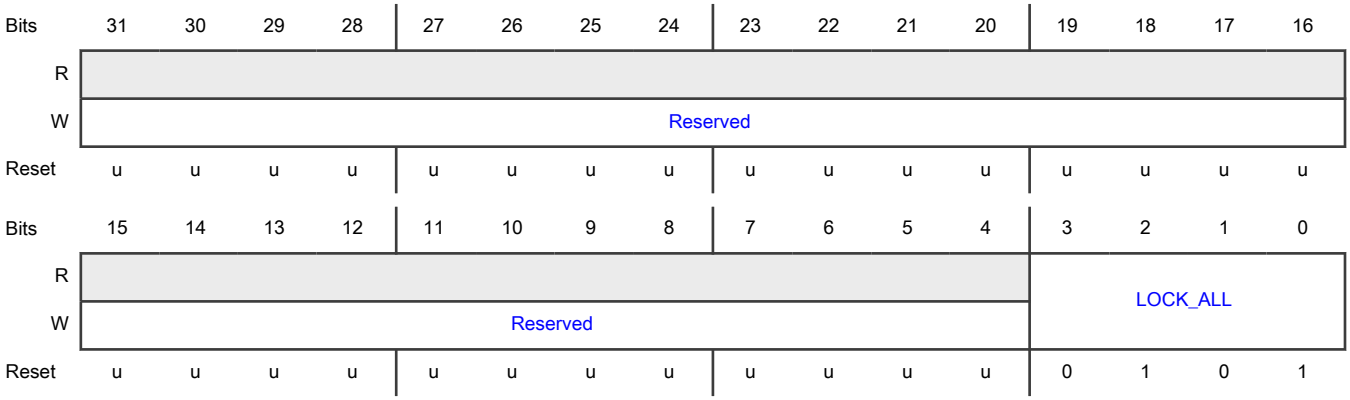
Function

Controls write access to the SWD_ACCESS_CPU0

, SWD_ACCESS_CPU1

, SWD_ACCESS_DSP, DEBUG_FEATURES, DEBUG_FEATURES_DP and DEBUG_AUTH_BEACON registers

Diagram



Fields

Field	Function
31-4 —	Reserved
3-0 LOCK_ALL	Controls write access to the security registers Controls write access to the SWD_ACCESS_CPU0, SWD_ACCESS_CPU1, SWD_ACCESS_DSP, DEBUG_FEATURES, DEBUG_FEATURES_DP and DEBUG_AUTH_BEACON registers 0000b - Any other value than b1010: disables write access to all registers 1010b - Enables write access to all registers

11.4.1.155 Cortex Debug Features Control (DEBUG_FEATURES)

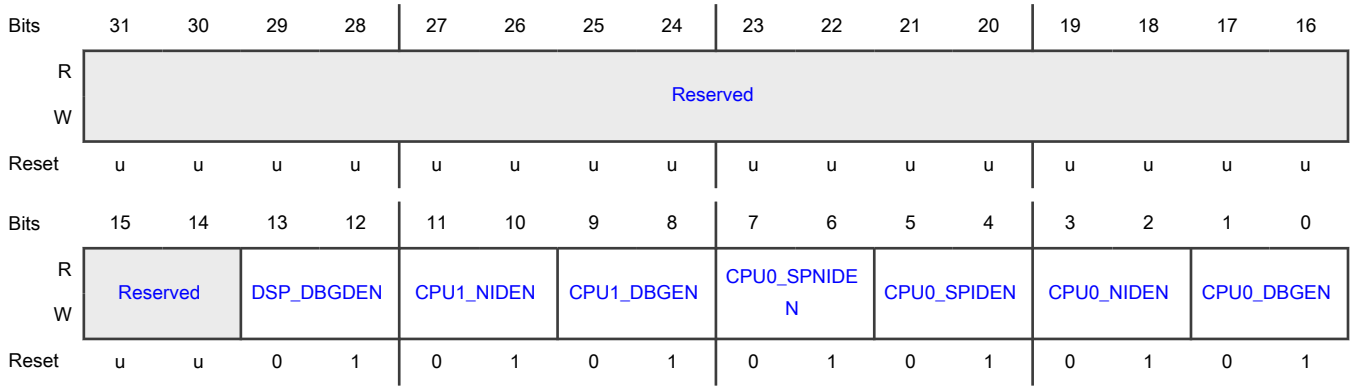
Offset

Register	Offset
DEBUG_FEATURES	FA4h

Function

Cortex M33 (CPU0) , micro Cortex M33 (CPU1) and CoolFlux BSP32 debug features control.

Diagram



Fields

Field	Function
31-14 —	Reserved
13-12 DSP_DBGDEN	DSP invasive debug control Enables invasive debug for DSP. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
11-10 CPU1_NIDEN	CPU1 non-invasive debug control Enables non-invasive debug for CPU1. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
9-8 CPU1_DBGEN	CPU1 invasive debug control Enables invasive debug for CPU1. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
7-6 CPU0_SPNIDEN	CPU0 secure privileged non-invasive debug control Enables privileged secure non-invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
5-4 CPU0_SPIDEN	CPU0 secure privileged invasive debug control Enables privileged secure invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug

Table continues on the next page...

Table continued from the previous page...

Field	Function
3-2 CPU0_NIDEN	CPU0 non-invasive debug control Enables non-invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
1-0 CPU0_DBGEN	CPU0 invasive debug control Enables invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug

11.4.1.156 Cortex Debug Features Control (Duplicate) (DEBUG_FEATURES_DP)

Offset

Register	Offset
DEBUG_FEATURES_DP	FA8h

Function

Cortex M33 (CPU0) micro Cortex M33 (CPU1) and CoolFlux DSP debug features control.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved		DSP_DBGEN		CPU1_NIDEN		CPU1_DBGEN		CPU0_SPNIDEN		CPU0_SPIDEN		CPU0_NIDEN		CPU0_DBGEN	
Reset	u	u	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Fields

Field	Function
31-14 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
13-12 DSP_DBGEN	DSP invasive debug control Enables invasive debug for DSP. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
11-10 CPU1_NIDEN	CPU1 non-invasive debug control Enables non-invasive debug for CPU1. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
9-8 CPU1_DBGEN	CPU1 invasive debug control Enables invasive debug for CPU1. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
7-6 CPU0_SPNIDEN	CPU0 secure privileged non-invasive debug control Enables privileged secure non-invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
5-4 CPU0_SPIDEN	CPU0 secure privileged invasive debug control Enables privileged secure invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
3-2 CPU0_NIDEN	CPU0 non-invasive debug control Enables non-invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug
1-0 CPU0_DBGEN	CPU0 invasive debug control Enables invasive debug for CPU0. Values not listed are reserved. 01b - Disables debug 10b - Enables debug

11.4.1.157 CPU0 Software Debug Access (SWD_ACCESS_CPU0)

Offset

Register	Offset
SWD_ACCESS_CPU0	FB4h

Function
This register is used by ROM during DEBUG authentication mechanism to enable debug access port for CPU0. Write once only.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SEC_CODE															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SEC_CODE															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 SEC_CODE	CPU0 SWD-AP: 0x12345678 0000_0000_0000_0000_0000_0000_0000_0000b - CPU0 DAP is not allowed. Reading back register is read as 0x5. 0001_0010_0011_0100_0101_0110_0111_1000b - Value to write to enable CPU0 SWD access. Reading back register is read as 0xA.

11.4.1.158 CPU1 Software Debug Access (SWD_ACCESS_CPU1)

Offset

Register	Offset
SWD_ACCESS_CPU1	FB8h

Function
This register is used by ROM during DEBUG authentication mechanism to enable debug access port for CPU1.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	SEC_CODE															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	SEC_CODE															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 SEC_CODE	Security code to allow CPU1 DAP: 0x12345678 0000_0000_0000_0000_0000_0000_0000_0000b - CPU1 DAP is not allowed 0001_0010_0011_0100_0101_0110_0111_1000b - Security code to allow CPU1 DAP

11.4.1.159 Debug Authentication BEACON (DEBUG_AUTH_BEACON)

Offset

Register	Offset
DEBUG_AUTH_BEACON	FC0h

Function

This register protected by security. ROM sets register (read-only) with value received in debug credentials before passing control to the user code. This can be used to extend debug authentication control for the customer application.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BEACON															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BEACON															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 BEACON	Sets by the debug authentication code in ROM to pass the debug beacons (Credential Beacon and Authentication Beacon) to the application code.

11.4.1.160 DSP Software Debug Access (SWD_ACCESS_DSP)

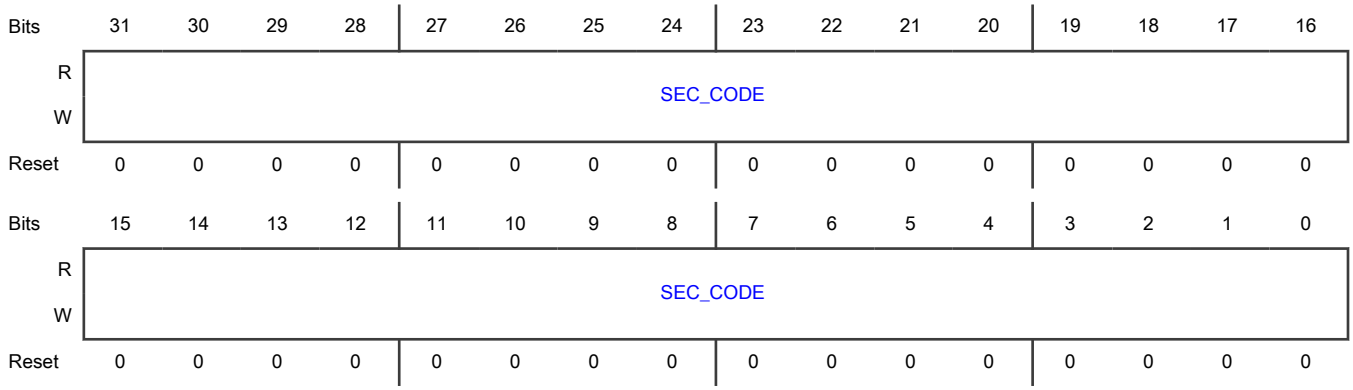
Offset

Register	Offset
SWD_ACCESS_DSP	FC4h

Function

This register is used by ROM during DEBUG authentication mechanism to enable debug access port for DSP.

Diagram



Fields

Field	Function
31-0 SEC_CODE	DSP SWD-AP: 0x12345678 0000_0000_0000_0000_0000_0000_0000_0000b - DSP DAP is not allowed. Reading back register is read as 0x5. 0001_0010_0011_0100_0101_0110_0111_1000b - Value to write to enable DSP SWD access. Reading back register is read as 0xA.

11.4.1.161 JTAG Chip ID (JTAG_ID)

Offset

Register	Offset
JTAG_ID	FF0h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JTAG_ID															
W																
Reset	u	u	u	u	0	1	1	1	0	0	1	0	0	1	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	JTAG_ID															
W																
Reset	0	1	0	0	0	0	0	0	0	0	1	0	1	0	1	1

Fields

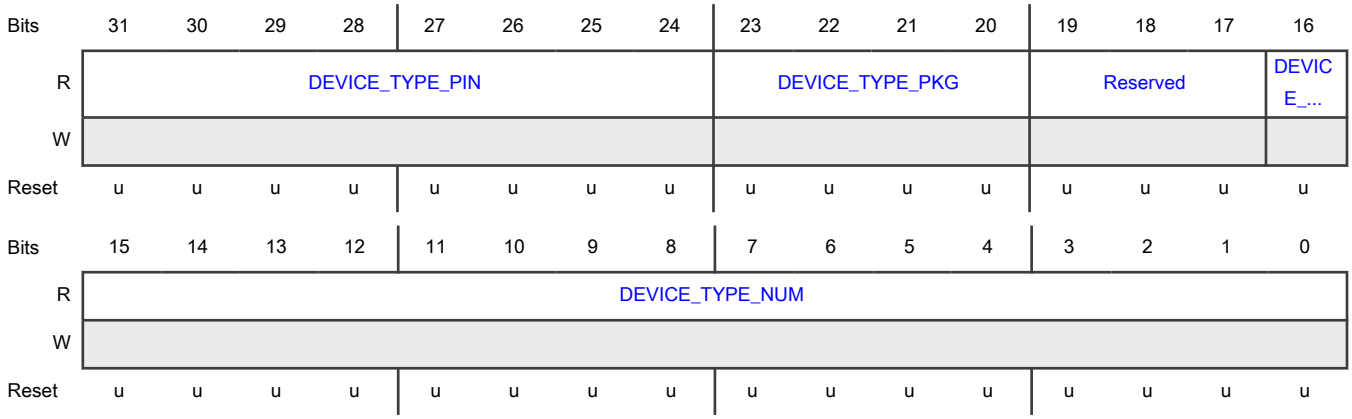
Field	Function
31-0 JTAG_ID	Indicates the device ID

11.4.1.162 Device Type (DEVICE_TYPE)

Offset

Register	Offset
DEVICE_TYPE	FF4h

Diagram



Fields

Field	Function
31-24 DEVICE_TYPE_PIN	Indicates the pin number. For example, 100-pin is 0x64.
23-20 DEVICE_TYPE_PKG	Indicates the device package type 0000b - HLQFP 0001b - HTQFP 0010b - BGA 0011b - MAXQFP (HDQFP) 0100b - QFN 0101b - CSP 0110b - LQFP
19-17 —	Reserved
16 DEVICE_TYPE_SEC	Indicates the device secure type 0b - Non-secure part 1b - Secure part
15-0 DEVICE_TYPE_NUM	Indicates the device part number bit[15:12] is family number • 0001b: MCX N series • 0010b: MCX A series • 0011b: MCX L series • other value: reserved bit[11:0]: 3 digital of the part number. For example, the value for the N947 device is 0x947.

11.4.1.163 Device ID (DEVICE_ID0)

Offset

Register	Offset
DEVICE_ID0	FF8h

Function
This register contains the device ID.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved				CUSTOMER_REVISION_ID				ROM_REV_MINOR				Reserved			
W																
Reset	0	0	0	0	u	u	u	u	u	u	u	u	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved								FLASH_SIZE				RAM_SIZE			
W																
Reset	0	0	0	0	0	0	0	0	u	u	u	u	u	u	u	u

Fields

Field	Function
31-28 —	Reserved
27-24 CUSTOMER_REVISION_ID	CM33_SECURITY_EXTENSION field. 1010b - Non secure version All other values - Any other value represents secure version
23-20 ROM_REV_MINOR	ROM Patch Version.
19-8 —	Reserved
7-4 FLASH_SIZE	Indicates Flash size of the device. 0000b - 32 KB 0001b - 64 KB

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0010b - 128 KB 0011b - 256 KB 0100b - 512 KB 0101b - 768 KB 0110b - 1 MB 0111b - 1.5 MB 1000b - 2 MB
3-0 RAM_SIZE	Indicates RAM size of the device. 0000b - 8 KB 0001b - 16 KB 0010b - 32 KB 0011b - 64 KB 0100b - 96 KB 0101b - 128 KB 0110b - 160 KB 0111b - 192 KB 1000b - 256 KB 1001b - 288 KB 1010b - 352 KB 1011b - 512 KB

11.4.1.164 Chip Revision ID and Number (DIEID)

Offset

Register	Offset
DIEID	FFCh

Function
This register contains the chip number and revision.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved				MCO_NUM_IN_DIE_ID											
W																
Reset	u	u	u	u	u	u	u	u	0	1	0	1	0	0	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MCO_NUM_IN_DIE_ID								MAJOR_REVISION				MINOR_REVISION			
W																
Reset	0	0	0	0	1	0	0	1	1	0	1	0	u	u	u	u

Fields

Field	Function
31-28 —	Reserved
27-8 MCO_NUM_IN_DIE_ID	Chip number
7-4 MAJOR_REVISION	Chip major revision
3-0 MINOR_REVISION	Chip minor revision

Chapter 12
VBAT

12.1 Chip-specific VBAT information

Table 241. Reference links to related information[1]

Topic	Related module	Reference
Full description	VBAT	VBAT
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

12.1.1 Module instances

This device contains one instance of the VBAT module, VBAT0.

12.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

12.1.3 WAKEUP_b signal pinout

VBAT_WAKEUP_b is the default signal for P5_2 after a chip reset.

12.1.4 Connections for OSCCLKE[CLKEn] and FROCLKE[CLKEn]

Table 242. Connections for OSCCLKE[CLKEn] and FROCLKE[CLKEn]

VBAT clock enable	Connects to this power domain
CLKE3	CORE_MAIN
CLKE2	CORE_WAKE
CLKE1	VDD_SYS
CLKE0	VDD_BAT

12.1.5 Clearing the CLOCK_DET flag when the CLKMON is disabled

When the CLKMON is disabled, the following register write sequence is needed to clear the STATUSA[CLOCK_DET] register bit:

1. First write 0x5 to the MONTSTA register,
2. Then write 0x0 to MONTSTA.

12.1.6 VBAT power supply

VBAT is supplied from the chip power supply VDD_BAT.

12.1.7 Connections for VBAT interrupt detect signals [IRQn_DET]

Table 243. Connections for [IRQn_DET]

VBAT interrupt detect	Connects to
IRQ3_DET	TDET interrupt
IRQ2_DET	RTC interrupt
IRQ1_DET	GPIO5 interrupt 1
IRQ0_DET	GPIO5 interrupt 0

12.1.8 VBAT LDORAMC[RET] configuration

LDORAMC.RET[3:0] controls retention of RAMA3-RAMA0 in low-power mode in this chip. Each of the four RAMA arrays are 8 KB in size. For example:

- LDORAMC[RET]=0x0 retains all 32 KB of RAMA
- LDORAMC[RET]=0xE retains 8 KB of RAMA0
- LDORAMC[RET]=0xF does not retain any arrays of RAMA

12.1.9 SEC0_DET connection

Security input in SEC0_DET detects the ITRC SRAM zeroize event.

12.2 Overview

VBAT works with the power management system to implement power-saving mechanisms. It provides bandgap timers and a voltage regulator to supply retention SRAM in low-power modes.

VBAT implements the 16 kHz internal clock source via FRO 16kHz.

12.2.1 Block diagram

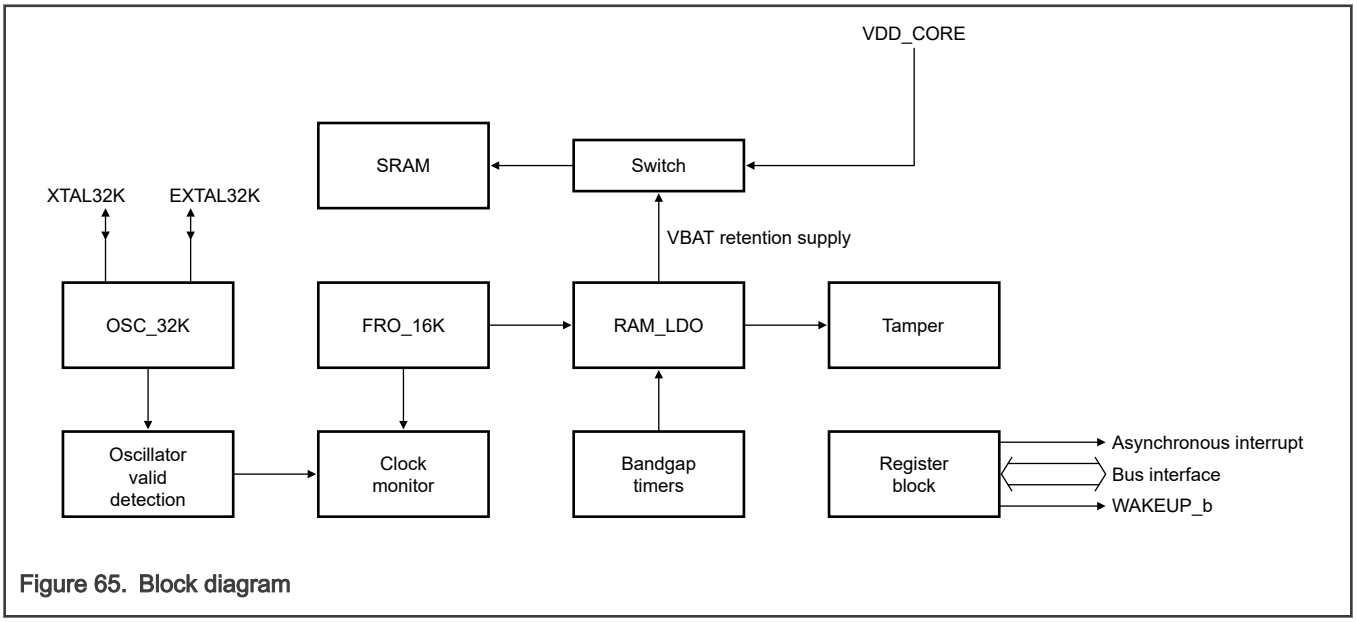


Figure 65. Block diagram

12.2.2 Features

- Includes oscillator for 32.768 kHz crystal (OSC32k)
- Supports internal 16 kHz free-running oscillator (FRO16K)
- Supports backup retention SRAM regulator
- Includes clock monitor (high/low/loss) for the OSC32k clock source
- Supports voltage monitor (high/low) for the VBAT supply
- Monitors temperature (high/low)
- Light tamper detect

12.3 Functional description

The following sections describe the functional details of VBAT.

12.3.1 Operations

This section describes the operations of VBAT.

12.3.1.1 Register protection

The VBAT registers are implemented as separate A and B registers. When configuring an A register, you must write the inverse value to the corresponding B register. The entire B register can be written as the inverse of the A register (for example, LDOCTLB = ~LDOCTLA). The reserved bits in the B register are not writable, and read as 0's. For example, if LDOCTLA = 0x5, LDOCTLB can be written as ~LDOCTLA, and then LDOCTLB will read back as 0x2.

After configuring the A and B registers for a module, it is possible to lock the configuration until the next VBAT POR by configuring the corresponding Lock registers. After the configuration is locked, if any protected A register does not match the inverse of the corresponding B register, then STATUSA[CONFIG_DET] becomes 1. This can trigger an interrupt.

12.3.1.2 FRO 16.384 kHz clock

FRO16K is enabled by default on POR. You can disable it or lock it to prevent any change to the enable field. FRO 16k should be enabled prior to enabling the backup SRAM LDO regulator or VBAT bandgap timers.

12.3.1.3 OSC 32.768 kHz clock

OSC32K is disabled by default on VBAT POR. When enabled, it takes the oscillator start-up time before STATUSA[OSC_RDY] becomes 1, and the clock is output to other modules.

The OSC32k supports 2 modes of operation, a high performance transconductance oscillator mode and a low power switched oscillator mode. A software configurable capacitor bank is supported in normal mode and must be configured when the oscillator is first enabled.

NOTE

For the oscillator characteristics of each mode, refer to the VBAT section of the device datasheet

12.3.1.4 SRAM LDO

SRAM LDO is a retention regulator that retains VBAT retention SRAM in low-power modes, including when VBAT is the only available supply. When IRQENA[LDO_RDY] becomes 1, the chip can enter into a low-power mode where the VBAT supply retains SRAM. Enable the bandgap and LDO before entering a low-power mode where the VBAT supply powers the retention SRAM. Enable Refresh mode for the lowest power consumption.

It takes approximately 8ms for the LDO_RDY flag to set after software enables the bandgap and LDO. When using the SRAM LDO it is recommended to leave it enabled in active modes to avoid this delay before entering a low power mode.

Although SRAM is automatically retained in all low-power modes, you can manually switch SRAM to the VBAT retention supply anytime. Do not access the SRAM when it receives power from the VBAT retention supply. You must manually reverse these steps before accessing the SRAM again.

To manually switch VBAT to power the SRAM:

1. Isolate the SRAM array (write 1 to LDORAMC[ISO]).
2. Switch the supply from VDD_CORE to the VBAT retention LDO (write 1 to LDORAMC[SWI]).

LDORAMC[RETn] can configure how much SRAM is retained in low-power modes. Each bit of the field corresponds to a different SRAM array, and you can configure the number of arrays retained (when SRAM LDO is enabled, retain at least one array).

When SRAM LDO is disabled, LDORAMC[RETn] controls whether the retention of VBAT SRAM happens using VDD_CORE in low-power modes.

In general, using the SRAM LDO to retain the VBAT retention SRAM is a lower power option than using VDD_CORE directly.

NOTE

Enable FRO16K before enabling SRAM LDO or the bandgap.

12.3.1.5 Bandgap timer

Two software-configurable bandgap timers are available when the SRAM LDO bandgap is enabled. These timers share logic with the bandgap timer refresh logic to minimize power, so they use the FRO16K clock source and are only available when the bandgap is enabled. When you change the configured timeout, NXP recommends that you:

1. Disable the timer.
2. Clear the flag.
3. Write the new timeout value.
4. Enable the timeout again.

12.3.1.6 Clock monitor

VBAT has two clock monitors. One uses FRO_16K as its clock source. The other uses OSC_RTC as its clock source. Each monitor checks a divided version of the other clock at the same frequency but divided by 16. The monitor expects a clock edge every 8 cycles, and you can trim the monitor to assert after 10, 12, 14, or 16 cycles with no edge.

Changing the trim makes the clock monitor more sensitive and reduces the latency, but also makes it more susceptible to variation in the duty cycle. The accuracy and latency of the monitor can vary by a cycle due to synchronization differences, so for each trim point:

- At 2 cycles after the expected edge the monitor asserts between 12.5% and 37.5% frequency variation with a maximum latency of 11 cycles
- At 4 cycles after the expected edge the monitor asserts between 37.5% and 62.5% frequency variation with a maximum latency of 13 cycles
- At 6 cycles after the expected edge the monitor asserts between 62.5% and 87.5% frequency variation with a maximum latency of 15 cycles
- At 8 cycles after the expected edge the monitor asserts between 87.5% and 112.5% frequency variation with a maximum latency of 17 cycles

The frequency of the clock monitor can also be trimmed to either 1 kHz or 64 Hz. Configuring to a slower clock increases the detector's latency but reduces power consumption.

NOTE

You must enable both the FRO16k and OSC32k and OSC_RDY sets before enabling the clock monitor.

12.3.1.7 Analog tamper

You can configure the analog tamper module to generate an interrupt or tamper event using the following detectors:

- VBAT voltage is out of range.
- Temperature is out of range.
- Light detector.

NOTE

Enable both the FRO16K and the LDO bandgaps before configuring the analog tamper module.

12.3.2 Low-power modes

VBAT remains functional in all low-power modes.

The VBAT module supports operation in a VBAT only power mode where only the VBAT is powered and the rest of the device is powered off externally. The VBAT_WAKEUP_b output can be used to control an external power switch or to communicate with an external PMIC.The VBAT_WAKEUP_b output drives low to indicate power supplies to the rest of the device should be powered. This occurs when either WAKECFG[OUT] is low or a status flag as enabled by WAKENA/WAKENB has asserted.When software writes WAKECFG[OUT]=1 and there are no configured status flags asserted, the VBAT_WAKEUP_B output drives high indicating that the rest of the device can be powered off.When a configured status flag asserts when only VBAT is powered, this will cause the VBAT_WAKEUP_b to drive low again and to power up the rest of the device. When WAKEUP_b pin is configured for open drain operation, an external wakeup source can drive this pin low, and the falling edge will assert the WAKEUP_FLAG.

12.3.3 Debug mode

Debug modes do not affect VBAT.

12.3.4 Clocks

VBAT implements the following clock domains:

- Bus interface clock - access the registers
- FRO16 kHz internal clock
- Crystal oscillator 32.768 kHz clock

12.3.5 Reset

Only the POR resets VBAT.

12.3.6 Interrupts

VBAT generates one interrupt combining the following sources:

- VBAT POR
- Wake-up pin assertion
- Bandgap timer 0
- Bandgap timer 1
- Crystal oscillator ready
- Clock monitor frequency error
- VBAT configuration error

- Voltage monitor detect
- Temperature monitor detect
- Light monitor detect

12.4 External signals

Table 244. External signals

Signal	Description	Direction
EXTAL32K	Oscillator input for 32.768 kHz crystal.	Input
XTAL32K	Oscillator output for 32.768 kHz crystal.	Output
WAKEUP_b	Active low wake-up pin, falling edge sets the WAKEUP_FLAG. When WAKEUP_b pin is configured for open drain operation, an external wakeup source can drive this pin low, and the falling edge will assert the WAKEUP_FLAG	Input/Output

12.5 Initialization

To enable and lock the FRO16K:

1. Write 1h to FROCTLA[FRO_EN].
2. Write 0h to FROCTLB[INVERSE].
3. Write 1h to FROLCKA[LOCK].
4. Write 0h FROLCKB[LOCK].
5. Alter FROCLKE[CLKE] to clock gate different FRO16K outputs to different peripherals to reduce power consumption.

To enable and lock the LDO and bandgap:

1. Enable the FRO16K.
2. Write 7h to LDO_RAM Control A (LDOCTLA).
3. Write 0h to LDOCTLB[INVERSE].
4. Wait for STATUSA[LDO_RDY] to become 1.
5. Write 1h to LDOLCKA[LOCK].
6. Write 0h to LDOLCKB[LOCK].

To configure and lock OSC32kHz for normal mode operation:

1. Configure OSCCTLA[EXTAL_CAP_SEL], OSCCTLA[XTAL_CAP_SEL] and OSCCTLA[COARSE_AMP_GAIN] as required based on the external crystal component ESR and CL values, and by the PCB parasitics on the EXTAL32K and XTAL32K pins. Configure 0h to OSCCTLA[MODE_EN], 1h to OSCCTLA[CAP_SEL_EN], and 1h to OSCCTLA[OSC_EN].
2. Write inverse of OSCCTLA to OSCCTLB[INVERSE].
3. Wait for STATUSA[OSC_RDY] to become 1.
4. Write 1h to OSCLCKA[LOCK].
5. Write 0h to OSCLCKB[LOCK].

To configure OSC32kHz for low power mode operation:

1. Write 3h to OSCCFGA[INIT_TRIM].
2. Write inverse of OSCCFGA to OSCCTLB[INVERSE].

3. Configure OSCCTLA[EXTAL_CAP_SEL], OSCCTLA[XTAL_CAP_SEL] and OSCCTLA[COARSE_AMP_GAIN] as required based on the external crystal component ESR and CL values, and by the PCB parasitics on the EXTAL32K and XTAL32K pins. Configure 1h to OSCCTLA[MODE_EN], 1h to OSCCTLA[CAP_SEL_EN], and 1h to OSCCTLA[OSC_EN].
4. Write inverse of OSCCTLA to OSCCTLB[INVERSE].
5. Wait for STATUSA[OSC_RDY] to become 1.
6. Write 0h to OSCCFGA[INIT_TRIM].
7. Write inverse of OSCCFGA to OSCCFGB[INVERSE].
8. Configure 3h to OSCCTLA[MODE_EN], 0h to OSCCTLA[EXTAL_CAP_SEL] and 0h to OSCCTLA[XTAL_CAP_SEL]. Configure OSCCTLA[SUPPLY_DET] as required by application.
9. Write inverse of OSCCTLA to OSCCTLB[INVERSE].
10. Wait for STATUSA[OSC_RDY] to become 1.

To configure and lock the clock monitor:

1. Enable and lock the FRO16K and OSC32K
2. Write 1h to MONCTLA[MON_EN]
3. Write 0h to MONCTLB[INVERSE]
4. Write 1h to MONLCKA[LOCK]
5. Write 0h to MONLCKB[LOCK]

To configure and lock the VBAT voltage and temperature monitors:

1. Enable and lock the FRO16K and VBAT LDO bandgap (VBAT LDO is optional)
2. Wait for the STATUSA[LDO_RDY] to set
3. Write 1h to TAMCTLA[VOLT_EN] and 1h to TAMCTLA[TEMP_EN]
4. Write Ch to TAMCTLB[INVERSE]
5. Write 1h to TAMLCKA[LOCK]
6. Write 0h to TAMLCKB[LOCK]

12.6 Application information

Configure the VBAT LDO to retain the state of any required SRAM to guard against an unexpected loss of power to the system:

1. Enable the LDO and bandgap.
2. Configure LDORAMC[RETn] to retain the appropriate arrays.

For a software-controlled power down, follow these steps to ensure robust retention of memory before switching off external power:

1. Write 1b to LDORAMC[ISO].
2. Write 1b to LDORAMC[SWI].

You must reverse the above steps to access the SRAM arrays, which the VBAT LDO retains. That is, follow these steps after external power is again switched on:

1. Write 0b to LDORAMC[SWI].
2. Write 0b to LDORAMC[ISO].

12.7 Memory map and register definition

This section includes VBAT memory map and detailed descriptions of all registers.

NOTE

The VBAT registers are reset on VBAT Cold Reset only (VBAT POR).

12.7.1 VBAT register descriptions

12.7.1.1 VBAT memory map

VBAT0 base address: 4005_9000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Version ID (VERID)	32	R	0101_000Fh
10h	Status A (STATUSA)	32	RW	0000_17C1h
14h	Status B (STATUSB)	32	RW	000F_003Eh
18h	Interrupt Enable A (IRQENA)	32	RW	0000_0000h
1Ch	Interrupt Enable B (IRQENB)	32	RW	000F_FFFFh
20h	Wake-up Enable A (WAKENA)	32	RW	0000_0000h
24h	Wake-up Enable B (WAKENB)	32	RW	000F_FFFFh
28h	Tamper Enable A (TAMPERA)	32	RW	0000_0001h
2Ch	Tamper Enable B (TAMPERB)	32	RW	0000_FFC0h
30h	Lock A (LOCKA)	32	RW	0000_0000h
34h	Lock B (LOCKB)	32	RW	0000_0001h
38h	Wake-up Configuration (WAKECFG)	32	RW	0000_0000h
100h	Oscillator Control A (OSCCTLA)	32	RW	0000_0000h
104h	Oscillator Control B (OSCCTLB)	32	RW	000F_FFFFh
108h	Oscillator Configuration A (OSCCFGA)	32	RW	0000_0000h
10Ch	Oscillator Configuration B (OSCCFGB)	32	RW	0000_0FFFh
118h	Oscillator Lock A (OSCLCKA)	32	RW	0000_0000h
11Ch	Oscillator Lock B (OSCLCKB)	32	RW	0000_0001h
120h	Oscillator Clock Enable (OSCCLKE)	32	RW	0000_0000h
200h	FRO16K Control A (FROCTLA)	32	RW	0000_0001h
204h	FRO16K Control B (FROCTLB)	32	RW	0000_0000h
218h	FRO16K Lock A (FROLCKA)	32	RW	0000_0000h
21Ch	FRO16K Lock B (FROLCKB)	32	RW	0000_0001h
220h	FRO16K Clock Enable (FROCLKE)	32	RW	0000_0000h
300h	LDO_RAM Control A (LDOCTLA)	32	RW	0000_0000h
304h	LDO_RAM Control B (LDOCTLB)	32	RW	0000_0007h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
318h	LDO_RAM Lock A (LDOLCKA)	32	RW	0000_0000h
31Ch	LDO_RAM Lock B (LDOLCKB)	32	RW	0000_0001h
320h	RAM Control (LDORAMC)	32	RW	0000_0000h
330h	Bandgap Timer 0 (LDOTIMER0)	32	RW	0000_0000h
338h	Bandgap Timer 1 (LDOTIMER1)	32	RW	0000_0000h
400h	CLKMON Control A (MONCTLA)	32	RW	0000_0000h
404h	CLKMON Control B (MONCTLB)	32	RW	0000_0001h
408h	CLKMON Configuration A (MONCFGA)	32	RW	0000_00FFh
40Ch	CLKMON Configuration B (MONCFGB)	32	RW	0000_0000h
410h	CLKMON Test A (MONTSTA)	32	RW	0000_0000h
414h	CLKMON Test B (MONTSTB)	32	RW	0000_000Fh
418h	CLKMON Lock A (MONLCKA)	32	RW	0000_0000h
41Ch	CLKMON Lock B (MONLCKB)	32	RW	0000_0001h
500h	TAMPER Control A (TAMCTLA)	32	RW	0000_0000h
504h	TAMPER Control B (TAMCTLB)	32	RW	0000_000Fh
518h	TAMPER Lock A (TAMLCKA)	32	RW	0000_0000h
51Ch	TAMPER Lock B (TAMLCKB)	32	RW	0000_0001h
600h	Switch Control A (SWICTLA)	32	RW	0000_0000h
604h	Switch Control B (SWICTLB)	32	RW	0000_0003h
618h	Switch Lock A (SWILCKA)	32	RW	0000_0000h
61Ch	Switch Lock B (SWILCKB)	32	RW	0000_0001h
700h	Wakeup 0 Register A (WAKEUP0A)	32	RW	0000_0000h
704h	Wakeup 0 Register B (WAKEUP0B)	32	RW	FFFF_FFFFh
708h	Wakeup 0 Register A (WAKEUP1A)	32	RW	0000_0000h
70Ch	Wakeup 0 Register B (WAKEUP1B)	32	RW	FFFF_FFFFh
7F8h	Wakeup Lock A (WAKLCKA)	32	RW	0000_0000h
7FCh	Wakeup Lock B (WAKLCKB)	32	RW	0000_0001h

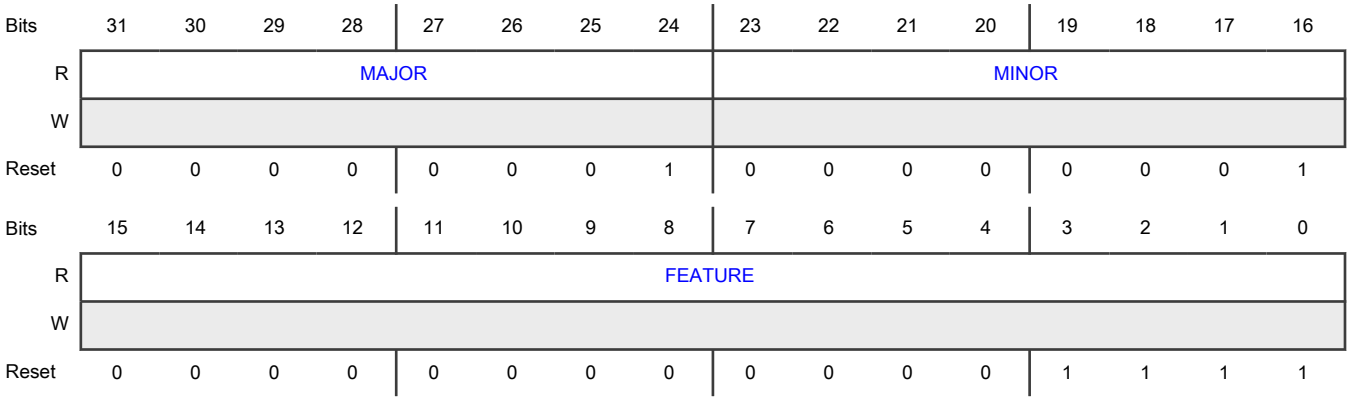
12.7.1.2 Version ID (VERID)

Offset

Register	Offset
VERID	0h

Function
Records the specific version of VBAT in the chip.

Diagram



Fields

Field	Function
31-24 MAJOR	Major Version Number Returns the major version number for the specification.
23-16 MINOR	Minor Version Number Returns the minor version number for the specification.
15-0 FEATURE	Feature Specification Number Returns the feature set number, indicating the feature set present in this instance of VBAT.

12.7.1.3 Status A (STATUSA)

Offset

Register	Offset
STATUSA	10h

Function
Contains all status flags for VBAT.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0												IRQ3_DET	IRQ2_DET	IRQ1_DET	IRQ0_DET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	SEC0_DET	0	LIGHT_D...	TEMP_DET	VOLT_DET	CONFIG_...	CLOCK_D...	OSC_RDY	LDO_RDY	TIMER1_...	TIMER0_...	WAKEUP_...	POR_DET
W				W1C		W1C	W1C	W1C	W1C	W1C			W1C	W1C	W1C	W1C
Reset	0	0	0	1	0	1	1	1	1	1	0	0	0	0	0	1

Fields

Field	Function
31-20 —	Reserved
19 IRQ3_DET	Interrupt 3 Detect Indicates that interrupt 3 has asserted. **NOTE** See the chip-specific VBAT information section for IRQn_DET connections. 0b - Not asserted 1b - Asserted
18 IRQ2_DET	Interrupt 2 Detect Indicates that interrupt 2 has asserted. 0b - Not asserted 1b - Asserted
17 IRQ1_DET	Interrupt 1 Detect Indicates that interrupt 1 has asserted. 0b - Not asserted 1b - Asserted
16 IRQ0_DET	Interrupt 0 Detect Indicates that interrupt 0 has asserted. 0b - Not asserted 1b - Asserted
15	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
14 —	Reserved
13 —	Reserved
12 SEC0_DET	Input 0 Detect Security input 0 has detected an error. 0b - Security input 0 not detected 1b - Security input 0 detected
11 —	Reserved
10 LIGHT_DET	Light Detect The light monitor has detected an error. 0b - Light error not detected 1b - Light error detected
9 TEMP_DET	Temperature Detect The temperature monitor has detected an error. 0b - Temperature error not detected 1b - Temperature error detected
8 VOLT_DET	Voltage Detect The voltage monitor has detected an error with the VBAT supply. **NOTE** STATUSA[VOLT_DET] will be set to 1 every time the voltage detector is enabled by setting TAMCTLA[VOLT_EN] to 1, software must clear the flag after enabling the voltage detector. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Not detected 1b - Detected When writing 0b - No effect

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - Clear the flag
7 CONFIG_DET	Configuration Detect Flag Indicates a configuration error in the VBAT registers where one or more A registers are not equal to the inverse of corresponding B registers and the configuration is locked. This may indicate a loss of state or attack. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Not detected 1b - Detected When writing 0b - No effect 1b - Clear the flag
6 CLOCK_DET	Clock Detect The clock monitor has detected an error. 0b - Clock error not detected 1b - Clock error detected
5 OSC_RDY	OSC32k Ready Indicates whether OSC32k is enabled and the clock output is stable. 0b - Disabled (clock not ready) 1b - Enabled (clock ready)
4 LDO_RDY	LDO Ready Indicates whether LDO is enabled and ready to enter Low-Power mode. 0b - Disabled (not ready) 1b - Enabled (ready)
3 TIMER1_FLAG	Bandgap Timer 1 Flag Asserts periodically according to the bandgap timer 1 period. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Not reached 1b - Reached

Table continues on the next page...

Table continued from the previous page...

Field	Function
	When writing 0b - No effect 1b - Clear the flag
2 TIMER0_FLAG	Bandgap Timer 0 Flag Asserts periodically according to the bandgap timer 0 period. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Not reached 1b - Reached When writing 0b - No effect 1b - Clear the flag
1 WAKEUP_FLAG	Wakeup Pin Flag Asserts whenever VBAT detects a falling edge on the wake-up pin. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Not asserted 1b - Asserted When writing 0b - No effect 1b - Clear the flag
0 POR_DET	POR Detect Flag Indicates if the VBAT POR has asserted. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Not reset 1b - Reset When writing

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - No effect 1b - Clear the flag

12.7.1.4 Status B (STATUSB)

Offset

Register	Offset
STATUSB	14h

Function
Contains the inverted status flags for VBAT.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0												INVERSE			
W													W0C			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INVERSE															
W	W0C															
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0

Fields

Field	Function
31-20 —	Reserved
19-0 INVERSE	Inverse value Contains the inverted contents of Status A (STATUSA).

12.7.1.5 Interrupt Enable A (IRQENA)

Offset

Register	Offset
IRQENA	18h

Function

Contains all interrupt enables for VBAT. When configuring the IRQ enables, both enable A and enable B registers need to be programmed to the appropriate values.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0												IRQ3_DET	IRQ2_DET	IRQ1_DET	IRQ0_DET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	SEC0_DET	0	LIGHT_D...	TEMP_DET	VOLT_DET	CONFIG_...	CLOCK_D...	OSC_RDY	LDO_RDY	TIMER1_...	TIMER0_...	WAKEUP_...	POR_DET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-20 —	Reserved
19 IRQ3_DET	Interrupt 3 Detect Enables the corresponding interrupt in Status A (STATUSA). **NOTE** See the chip-specific VBAT information section for IRQn_DET connections. 0b - Disable 1b - Enable
18 IRQ2_DET	Interrupt 2 Detect Enables the corresponding interrupt in Status A (STATUSA). 0b - Disable 1b - Enable
17	Interrupt 1 Detect

Table continues on the next page...

Table continued from the previous page...

Field	Function
IRQ1_DET	Enables the corresponding interrupt in Status A (STATUSA). 0b - Disable 1b - Enable
16 IRQ0_DET	Interrupt 0 Detect Enables the corresponding interrupt in Status A (STATUSA). 0b - Disable 1b - Enable
15 —	Reserved
14 —	Reserved
13 —	Reserved
12 SEC0_DET	Input 0 Detect Enables the corresponding interrupt in status register. 0b - Disable 1b - Enable
11 —	Reserved
10 LIGHT_DET	Light Detect Enables the corresponding interrupt in status register. 0b - Disable 1b - Enable
9 TEMP_DET	Temperature Detect Enables the corresponding interrupt in status register. 0b - Interrupt disabled 1b - Interrupt enabled
8 VOLT_DET	Voltage Detect Enables the corresponding interrupt in status register. 0b - Disable 1b - Enable

Table continues on the next page...

Table continued from the previous page...

Field	Function
7 CONFIG_DET	Configuration Detect Enables the corresponding interrupt in Status A (STATUSA). 0b - Disable 1b - Enable
6 CLOCK_DET	Clock Detect Enables the corresponding interrupt in status register. 0b - Disable 1b - Enable
5 OSC_RDY	OSC32k Ready Enables the corresponding interrupt in Status A (STATUSA). 0b - Disable 1b - Enable
4 LDO_RDY	LDO Ready Enables the corresponding interrupt in Status A (STATUSA). 0b - Disable 1b - Enable
3 TIMER1_FLAG	Bandgap Timer 2 Enables the corresponding interrupt in Status A (STATUSA). 0b - Disable 1b - Enable
2 TIMER0_FLAG	Bandgap Timer 0 Enables the corresponding interrupt in Status A (STATUSA). 0b - Disable 1b - Enable
1 WAKEUP_FLAG	Wakeup Pin Flag Enables the corresponding interrupt in Status A (STATUSA). 0b - Disable 1b - Enable
0 POR_DET	POR Detect Enables the corresponding interrupt in Status A (STATUSA). 0b - Disable

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - Enable

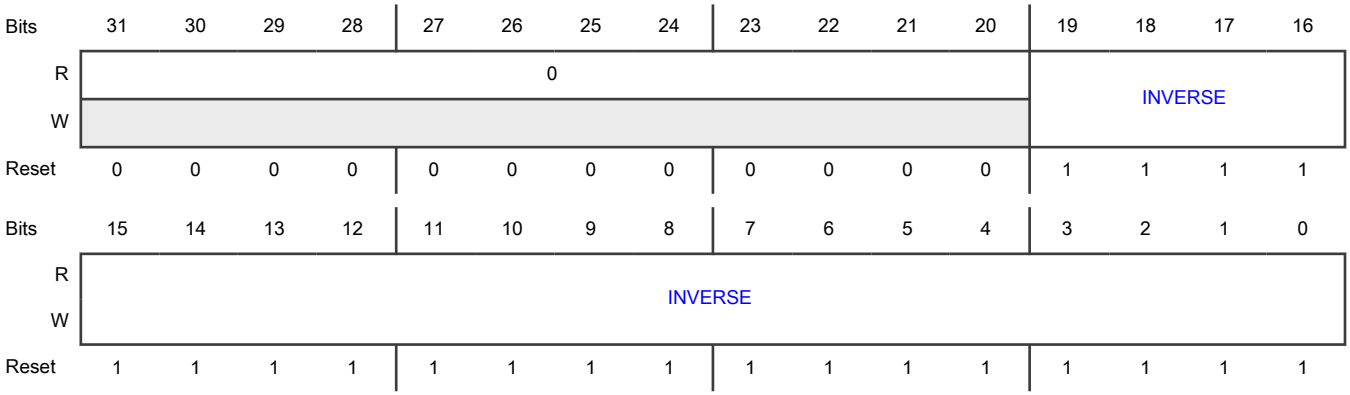
12.7.1.6 Interrupt Enable B (IRQENB)

Offset

Register	Offset
IRQENB	1Ch

Function
Contains the inverted interrupt enables for VBAT.

Diagram



Fields

Field	Function
31-20 —	Reserved
19-0 INVERSE	Inverse Value Contains the inverted contents of Interrupt Enable A (IRQENA). When configuring the IRQ enables, both enable A and enable B registers need to be programmed to the appropriate values.

12.7.1.7 Wake-up Enable A (WAKENA)

Offset

Register	Offset
WAKENA	20h

Function

Contains all wake-up enables for VBAT. When configuring the wake enables, both enable A and enable B registers need to be programmed to the appropriate values.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0												IRQ3_DET	IRQ2_DET	IRQ1_DET	IRQ0_DET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	SEC0_DET	0	LIGHT_D...	TEMP_DET	VOLT_DET	CONFIG_...	CLOCK_D...	OSC_RDY	LDO_RDY	TIMER1_...	TIMER0_...	WAKEUP_...	POR_DET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-20 —	Reserved
19 IRQ3_DET	Interrupt 3 Detect Enables the corresponding wake-up in Status A (STATUSA). **NOTE** See the chip-specific VBAT information section for IRQn_DET connections. 0b - Disable 1b - Enable
18 IRQ2_DET	Interrupt 2 Detect Enables the corresponding wake-up in Status A (STATUSA). 0b - Disable 1b - Enable
17 IRQ1_DET	Interrupt 1 Detect Enables the corresponding wake-up in Status A (STATUSA). 0b - Disable 1b - Enable
16 IRQ0_DET	Interrupt 0 Detect Enables the corresponding wake-up in Status A (STATUSA).

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Disable 1b - Enable
15 —	Reserved
14 —	Reserved
13 —	Reserved
12 SEC0_DET	Input 0 Detect Enables the corresponding wakeup in Status Register. 0b - Disabled 1b - Enabled
11 —	Reserved
10 LIGHT_DET	Light Detect Enables the corresponding wake-up in status register. 0b - Disable 1b - Enable
9 TEMP_DET	Temperature Detect Enables the corresponding wake-up in status register. 0b - Disable 1b - Enable
8 VOLT_DET	Voltage Detect Enables the corresponding wake-up in status register. 0b - Disable 1b - Enable
7 CONFIG_DET	Configuration Detect Enables the corresponding wake-up in Status A (STATUSA). 0b - Disable 1b - Enable

Table continues on the next page...

Table continued from the previous page...

Field	Function
6 CLOCK_DET	Clock Detect Enables the corresponding wake-up in status register. 0b - Disable 1b - Enable
5 OSC_RDY	OSC32K Ready Enables the corresponding wake-up in Status A (STATUSA). 0b - Disable 1b - Enable
4 LDO_RDY	LDO Ready Enables the corresponding wake-up in Status A (STATUSA). 0b - Disable 1b - Enable
3 TIMER1_FLAG	Bandgap Timer 2 Enables the corresponding wake-up in Status A (STATUSA). 0b - Disable 1b - Enable
2 TIMER0_FLAG	Bandgap Timer 0 Enables the corresponding wake-up in Status A (STATUSA). 0b - Disable 1b - Enable
1 WAKEUP_FLAG	Wake-up Pin Flag Enables the corresponding wake-up in Status A (STATUSA). 0b - Disable 1b - Enable
0 POR_DET	POR Detect Enables the corresponding wake-up in Status A (STATUSA). 0b - Disable 1b - Enable

12.7.1.8 Wake-up Enable B (WAKENB)

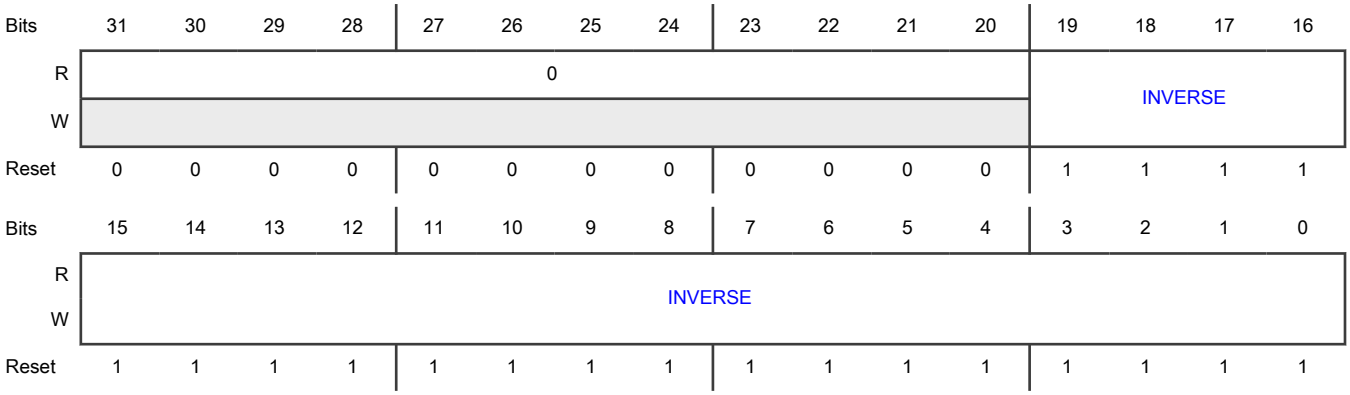
Offset

Register	Offset
WAKENB	24h

Function

Contains the inverted wake-up enables for VBAT. When configuring the wake enables, both enable A and enable B registers need to be programmed to the appropriate values.

Diagram



Fields

Field	Function
31-20 —	Reserved
19-0 INVERSE	Inverse Value Contains the inverted contents of Wake-up Enable A (WAKENA).

12.7.1.9 Tamper Enable A (TAMPERA)

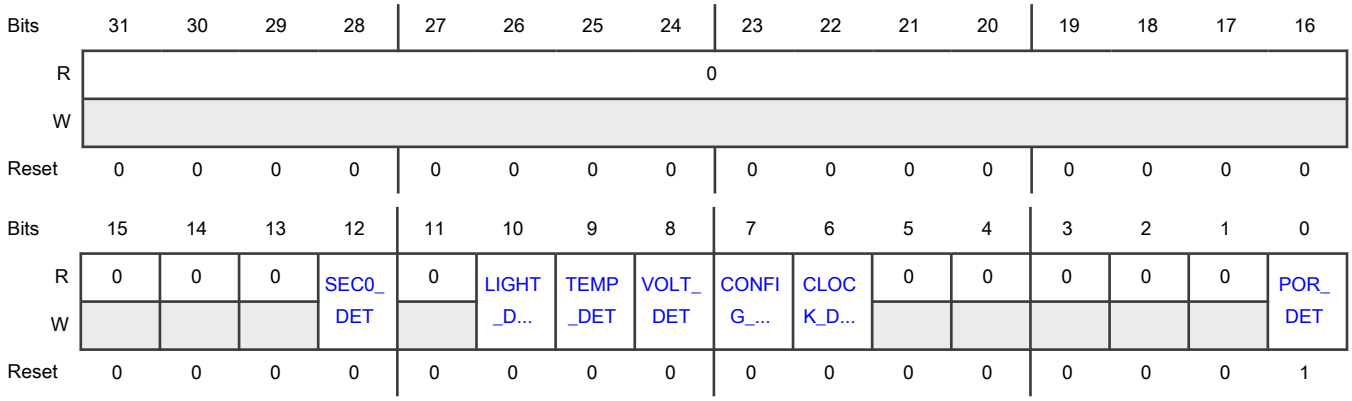
Offset

Register	Offset
TAMPERA	28h

Function

Contains all tamper enables for VBAT.

Diagram



Fields

Field	Function
31-16 —	Reserved
15 —	Reserved
14 —	Reserved
13 —	Reserved
12 SEC0_DET	Input 0 Detect Enables the corresponding tamper in Status Register. 0b - Tamper disabled 1b - Tamper enabled
11 —	Reserved
10 LIGHT_DET	Light Detect Enables the corresponding tamper in Status Register. 0b - Tamper disabled 1b - Tamper enabled
9 TEMP_DET	Temperature Detect Enables the corresponding tamper in Status Register.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Tamper disabled 1b - Tamper enabled
8 VOLT_DET	Voltage Detect Enables the corresponding tamper in Status Register. 0b - Tamper disabled 1b - Tamper enabled
7 CONFIG_DET	Configuration Detect Enables the corresponding interrupt in Status Register. 0b - Tamper disabled 1b - Tamper enabled
6 CLOCK_DET	Clock Detect Enables the corresponding tamper in Status Register. 0b - Tamper disabled 1b - Tamper enabled
5 —	Reserved
4 —	Reserved
3 —	Reserved
2 —	Reserved
1 —	Reserved
0 POR_DET	POR Detect Enables the corresponding tamper in Status Register. 0b - Tamper disabled 1b - Tamper enabled

12.7.1.10 Tamper Enable B (TAMPERB)

Offset

Register	Offset
TAMPERB	2Ch

Function
Contains the inverted tamper enables for VBAT.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INVERSE															
W																
Reset	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

Fields

Field	Function
31-16 —	Reserved
15-0 INVERSE	Inverse value Must be programmed with the inverted contents of IRQENA.

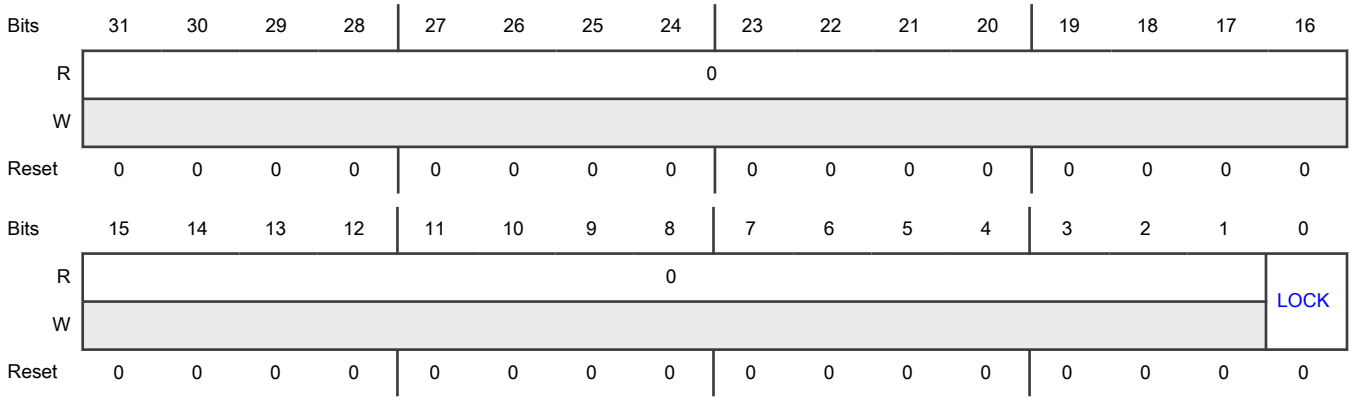
12.7.1.11 Lock A (LOCKA)

Offset

Register	Offset
LOCKA	30h

Function
Contains the lock bits. Writes to this register are blocked when lock register is set.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock When set, blocks writes to the interrupt, wakeup, and tamper enable registers, and asserts a configuration error if any of the tamper/wakeup/interrupt enable A registers are not equal to their corresponding inverted B registers. 0b - Disables lock 1b - Enables lock. Cleared by VBAT POR.

12.7.1.12 Lock B (LOCKB)

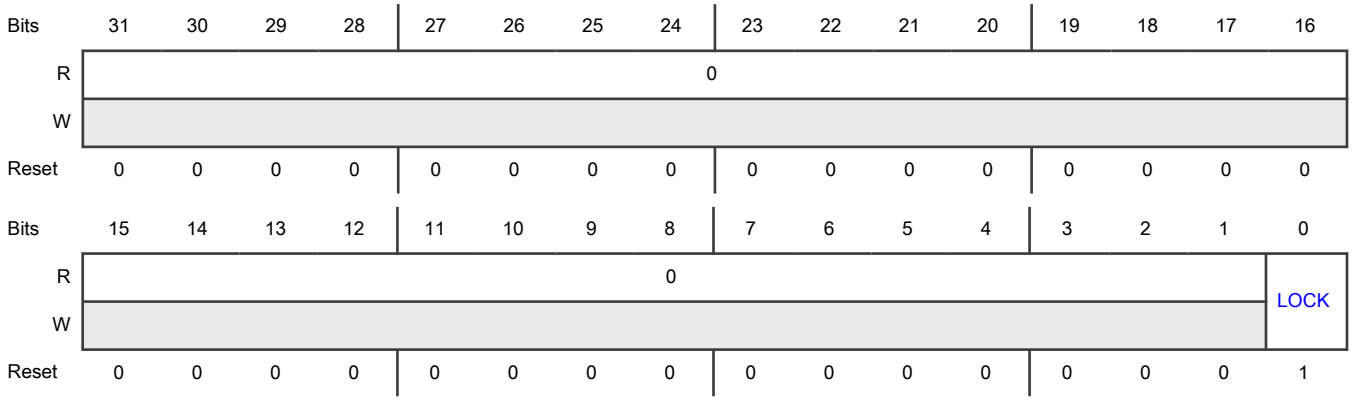
Offset

Register	Offset
LOCKB	34h

Function

Contains the inverted lock bits. Writes to this register are blocked when the lock register is set.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock When clear, blocks writes to the interrupt and tamper enable registers and asserts a configuration error if any of the tamper/interrupt enable A registers are not equal to inverted tamper/interrupt B registers. 0b - Enables lock 1b - Disables lock

12.7.1.13 Wake-up Configuration (WAKECFG)

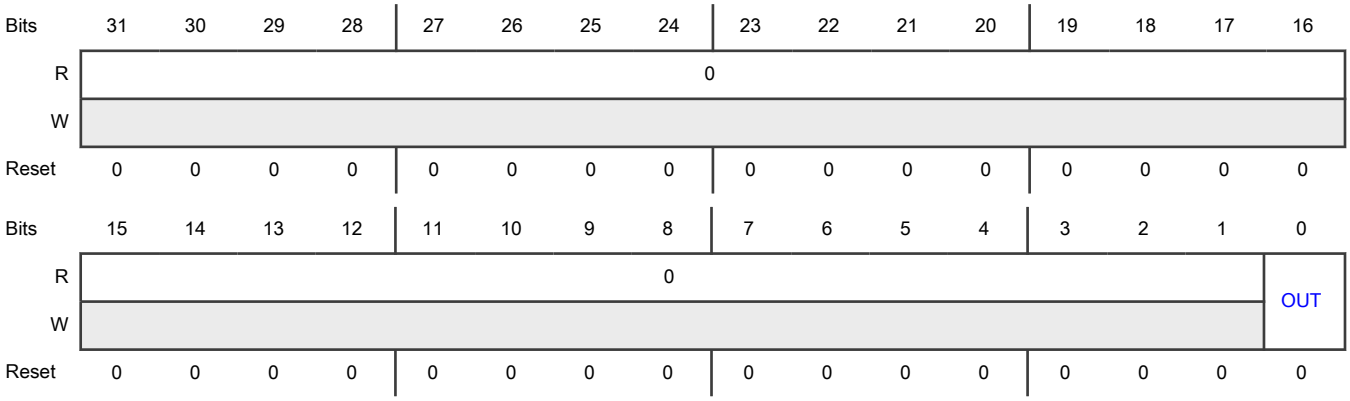
Offset

Register	Offset
WAKECFG	38h

Function

Configures the default state of the WAKEUP_b signal when no enabled wake-up source is asserted.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 OUT	Output Specifies the output state of WAKEUP_b signal. When the value of this field is 1, WAKEUP_b output state is logic one, unless an enabled wakeup source asserts. 0b - Logic zero (asserted) 1b - Logic one

12.7.1.14 Oscillator Control A (OSCCTLA)

Offset

Register	Offset
OSCCTLA	100h

Function

Contains the oscillator control fields. When configuring the oscillator, both control A and control B registers need to be programmed to the appropriate values. Writes to this register are blocked when OSCLCKA[LOCK] is 1.

Some fields in this register uses the following equation to calculate the load capacitance:

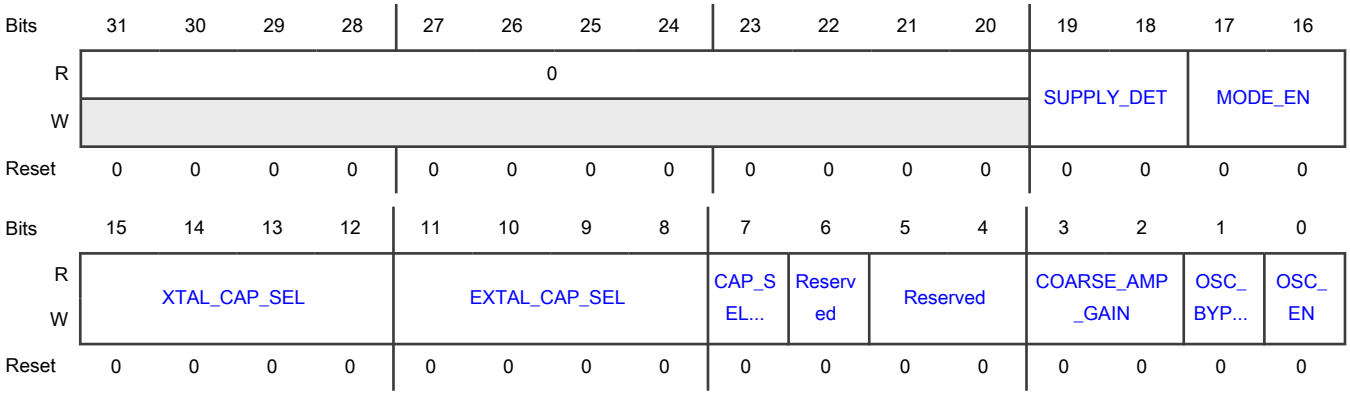
$$C_L = \frac{C_{extal} \times C_{xtal}}{C_{extal} + C_{xtal}} + C_{shunt}$$

Equation 1. Equation for load capacitance

where:

- C_{extal} is the selected capacitance on the EXTAL pin.
- C_{xtal} is the selected capacitance on the XTAL pin.
- C_{shunt} is the internal shunt capacitance. You can determine this capacitance value from the chip's data sheet.

Diagram



Fields

Field	Function
31-20 —	Reserved
19-18 SUPPLY_DET	Supply Detector Trim Specifies supply detector for low power mode. 00b - VBAT supply is less than 3V 01b - VBAT supply is greater than 3V
17-16 MODE_EN	Mode Enable Configures Crystal Oscillator mode. 00b - Normal mode 01b - Startup mode 11b - Low power mode
15-12 XTAL_CAP_SEL	Crystal Load Capacitance Selection Selects the internal capacitance for the XTAL pin from the capacitor bank. Calculate the final load capacitance as shown in Equation 1. You must write 0000b to this field in low power mode. 0000b - 0 pF 0001b - 2 pF 0010b - 4 pF

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0011b - 6 pF 0100b - 8 pF 0101b - 10 pF 0110b - 12 pF 0111b - 14 pF 1000b - 16 pF 1001b - 18 pF 1010b - 20 pF 1011b - 22 pF 1100b - 24 pF 1101b - 26 pF 1110b - 28 pF 1111b - 30 pF
11-8 EXTAL_CAP_SEL	Crystal Load Capacitance Selection Selects the internal capacitance for the EXTAL pin from the capacitor bank. Calculate the final load capacitance as shown in Equation 1. The configuration EXTAL_CAP_SEL=0000 and CAP_SEL_EN=1 is required in low power mode and is not supported in other modes 0000b - 0 pF 0001b - 2 pF 0010b - 4 pF 0011b - 6 pF 0100b - 8 pF 0101b - 10 pF 0110b - 12 pF 0111b - 14 pF 1000b - 16 pF 1001b - 18 pF 1010b - 20 pF 1011b - 22 pF 1100b - 24 pF 1101b - 26 pF 1110b - 28 pF 1111b - 30 pF

Table continues on the next page...

Table continued from the previous page...

Field	Function
7 CAP_SEL_EN	Crystal Load Capacitance Selection Enable Enables the internal capacitor banks on the EXTAL and XTAL pins that vary the load capacitance on these pins. You must write 1 to this field and OSCCTLA[OSC_EN] simultaneously. **NOTE** The configuration EXTAL_CAP_SEL=0000 and CAP_SEL_EN=1 is required in low power mode and is not supported in other modes 0b - Disable 1b - Enable
6 —	Reserved
5-4 —	Reserved
3-2 COARSE_AMP_GAIN	Amplifier gain adjustment bits to allow the use of a wide range of external crystal ESR values See the device datasheet for the ranges supported by this device Tunes the internal transconductance (g_m) by increasing the current. 00b - ESR Range 0 01b - ESR Range 1 10b - ESR Range 2 11b - ESR Range 3
1 OSC_BYP_EN	Crystal Oscillator Bypass Enable Bypasses the crystal oscillator. The oscillator outputs the clock, but this clock is the same as the clock provided on EXTAL pin. You must write 1 to both OSCCTLA[OSC_EN] and this field to enable the Bypass mode of the oscillator. To exit Bypass mode, you must write 0 to both OSCCTLA[OSC_EN] and this field. Do not move the oscillator from Bypass mode directly to Normal mode. 0b - Does not bypass 1b - Bypass
0 OSC_EN	Crystal Oscillator Enable Enables the crystal oscillator. The oscillator starts giving the clock output but the clock is not available to the chip until STATUSA[OSC_RDY] is asserted. 0b - Disable 1b - Enable

12.7.1.15 Oscillator Control B (OSCCTLB)

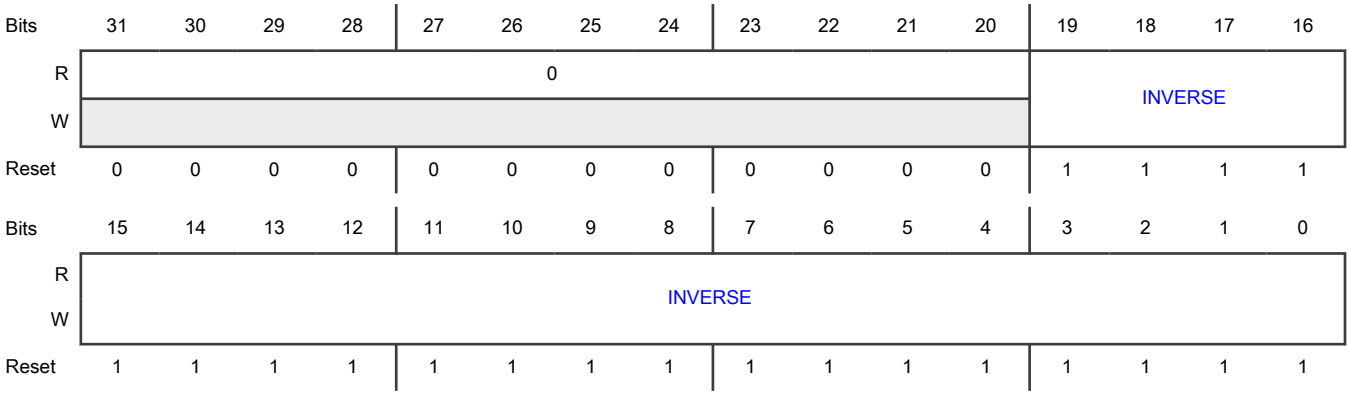
Offset

Register	Offset
OSCCTLB	104h

Function

Contains the inverted oscillator control field. Configure control A and control B registers with the appropriate values when configuring the oscillator. Writes to this register are blocked when OSCLCKB[LOCK] is 1.

Diagram



Fields

Field	Function
31-20 —	Reserved
19-0 INVERSE	Inverse Value Contains the inverted contents of Oscillator Control A (OSCCTLA).

12.7.1.16 Oscillator Configuration A (OSCCFGA)

Offset

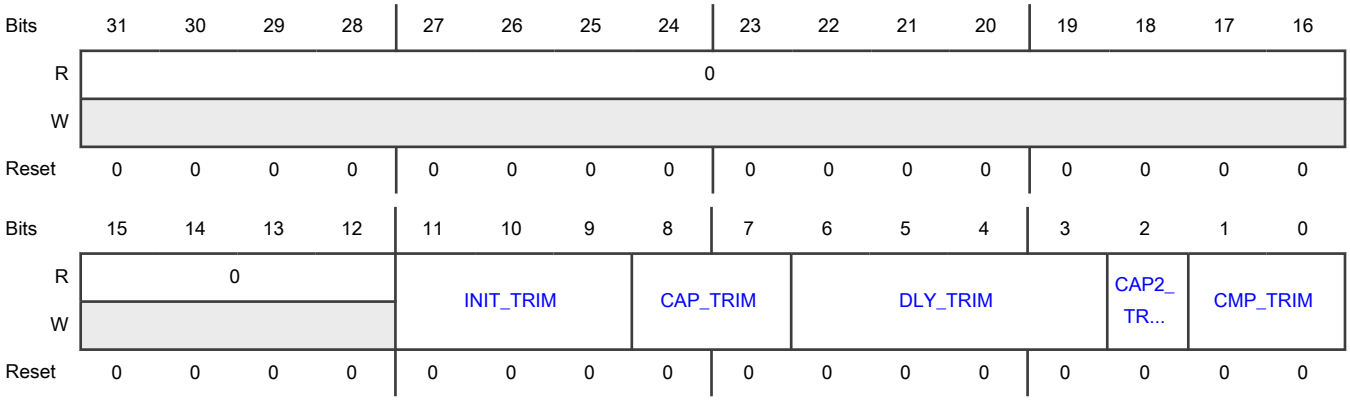
Register	Offset
OSCCFGA	108h

Function

Contains the oscillator configuration fields. When configuring the oscillator, both configuration A and configuration B registers need to be programmed to the appropriate values. Writes to this register are blocked when OSCLCKA[LOCK] is 1.

Reset values are loaded out of IFR.

Diagram



Fields

Field	Function
31-12 —	Reserved
11-9 INIT_TRIM	Initialization Trim Configures the start-up time of the oscillator. 000b - 8 s 001b - 4 s 010b - 2 s 011b - 1 s 100b - 0.5 s 101b - 0.25 s 110b - 0.125 s 111b - 0.5 ms
8-7 CAP_TRIM	Capacitor Trim Cap_trim changes the position of charge injection and removal for low power mode. 00b - Default (when CAP2_TRIM = 0 and CAP_TRIM[1:0] = 00) 01b - -1us (when CAP2_TRIM = 0 and CAP_TRIM[1:0] = 01) 10b - -2us (when CAP2_TRIM = 0 and CAP_TRIM[1:0] = 10) or or +3.5us (when CAP2_TRIM = 1 and CAP_TRIM[1:0] = 10) 11b - -2.5us (when CAP2_TRIM = 0 and CAP_TRIM[1:0] = 11) or +1us (when CAP2_TRIM = 1 and CAP_TRIM[1:0] = 11)
6-3 DLY_TRIM	Delay Trim Changes the p current and n current used for biasing the amplifier, applicable for low power and normal mode.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0000b - P current 9(nA) and N Current 6(nA) 0001b - P current 13(nA) and N Current 6(nA) 0011b - P current 4(nA) and N Current 6(nA) 0100b - P current 9(nA) and N Current 4(nA) 0101b - P current 13(nA) and N Current 4(nA) 0111b - P current 4(nA) and N Current 4(nA) 1000b - P current 9(nA) and N Current 2(nA) 1001b - P current 13(nA) and N Current 2(nA) 1011b - P current 4(nA) and N Current 2(nA)
2 CAP2_TRIM	CAP2_TRIM Affects the function of CAP_TRIM, see CAP_TRIM for details.
1-0 CMP_TRIM	Comparator Trim CMP_TRIM[1:0] changes the internal low power supply for low power and normal mode. 00b - 760 mV 01b - 770 mV 11b - 740 mV

12.7.1.17 Oscillator Configuration B (OSCCFGB)

Offset

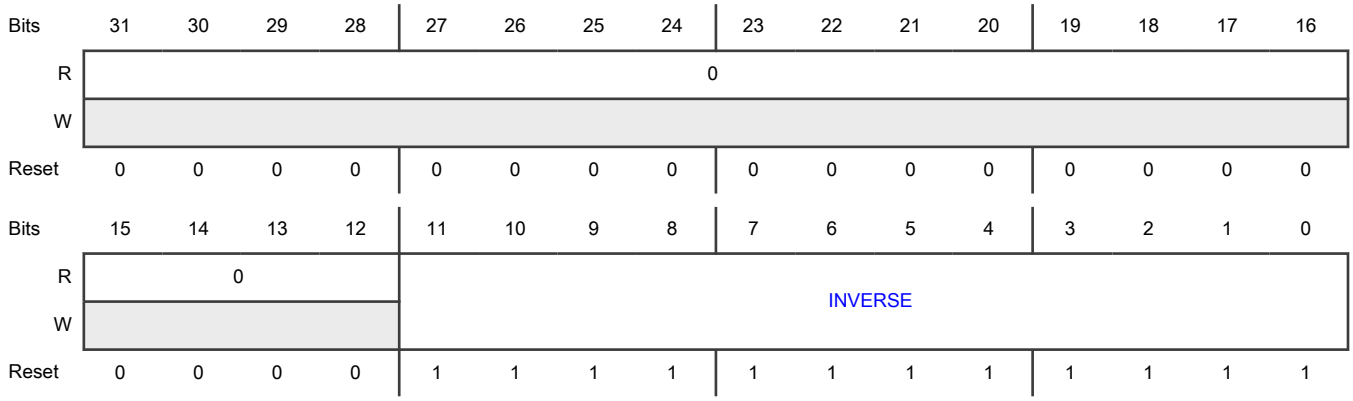
Register	Offset
OSCCFGB	10Ch

Function

Contains the inverted oscillator control field. Configure configuration A and configuration B registers with the appropriate values when configuring the oscillator. Writes to this register are blocked when OSCLCKB[LOCK] is 1.

Reset values are loaded out of IFR.

Diagram



Fields

Field	Function
31-12 —	Reserved
11-0 INVERSE	Inverse Value Contains the inverted contents of Oscillator Configuration A (OSCCFGA).

12.7.1.18 Oscillator Lock A (OSCLCKA)

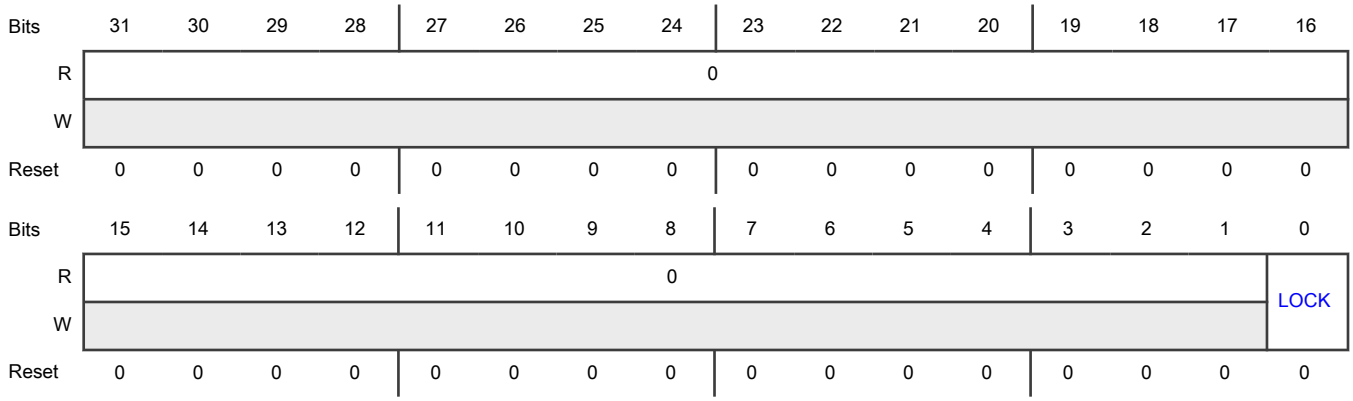
Offset

Register	Offset
OSCLCKA	118h

Function

Allows you to block any write to the oscillator registers.When configuring the oscillator, both lock A and lock B registers need to be programmed to the appropriate values. Writes to this register are blocked when OSCLCKA[LOCK] is 1.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock Blocks any write to the oscillator registers. 0b - Do not block 1b - Block

12.7.1.19 Oscillator Lock B (OSCLCKB)

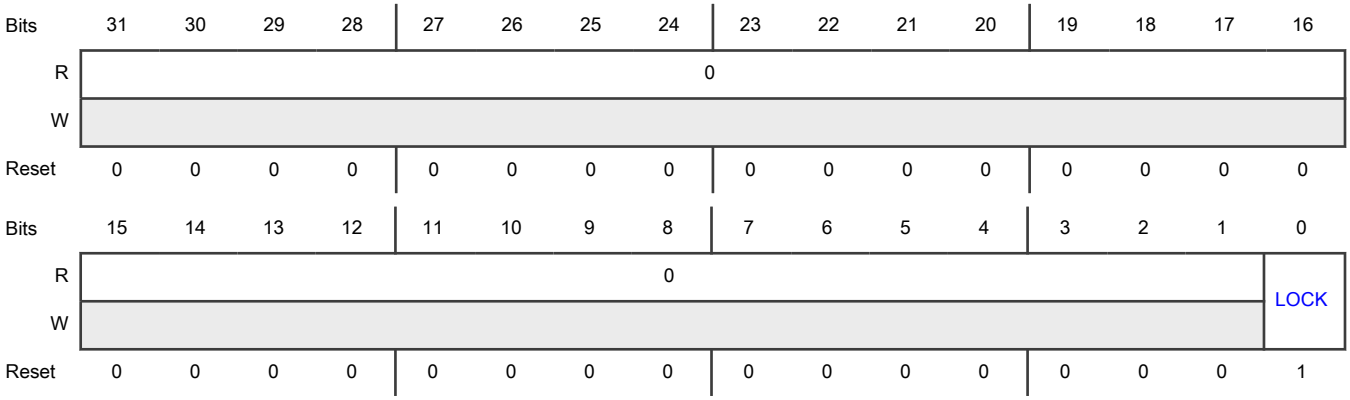
Offset

Register	Offset
OSCLCKB	11Ch

Function

Contains the inverted oscillator lock field. Configure lock A and lock B registers with the appropriate values when configuring the oscillator. Writes to this register are blocked when OSCLCKB[LOCK] is 1.

Diagram



Fields

Field	Function
31-1 —	Reserved
0	Lock

Table continues on the next page...

Table continued from the previous page...

Field	Function
LOCK	Blocks any write to the oscillator registers and asserts a configuration error. The lock is enabled if any of the oscillator A registers are not equal to inverted oscillator B registers. 0b - Block 1b - Do not block

12.7.1.20 Oscillator Clock Enable (OSCCLKE)

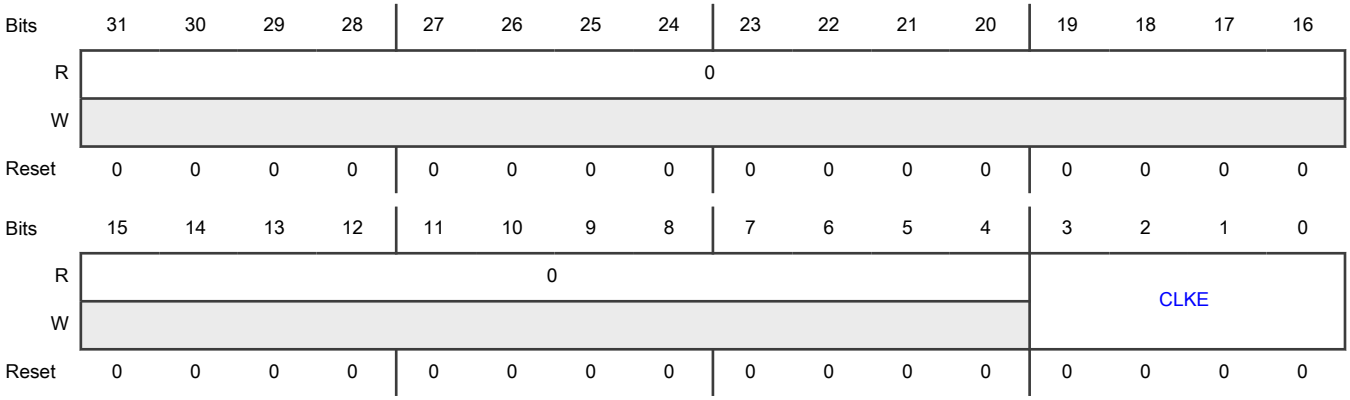
Offset

Register	Offset
OSCCLKE	120h

Function

Contains clock gating register field to gate the OSC32k clock to other modules.

Diagram



Fields

Field	Function
31-4 —	Reserved
3-0 CLKE	Clock Enable Enables the corresponding OSC32 kHz output clock to other modules when you write 1 to this field. See the chip-specific VBAT information section for more information.

12.7.1.21 FRO16K Control A (FROCTLA)

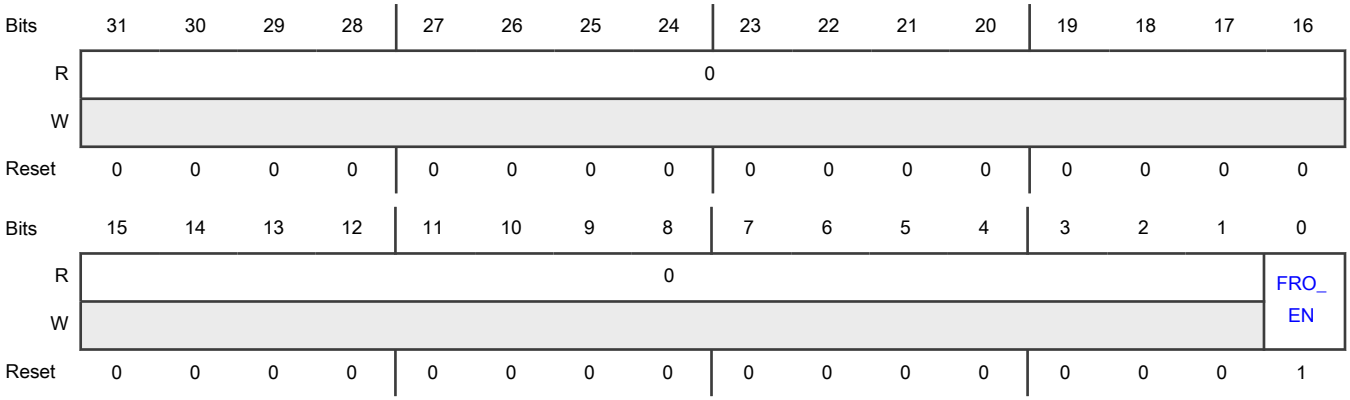
Offset

Register	Offset
FROCTLA	200h

Function

Controls FRO16K. When configuring the FRO16K, both control A and control B registers need to be programmed to the appropriate values. Writes to this register are blocked when FROLCKA[LOCK] is 1.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 FRO_EN	FRO16K Enable 0b - Disable 1b - Enable

12.7.1.22 FRO16K Control B (FROCTLB)

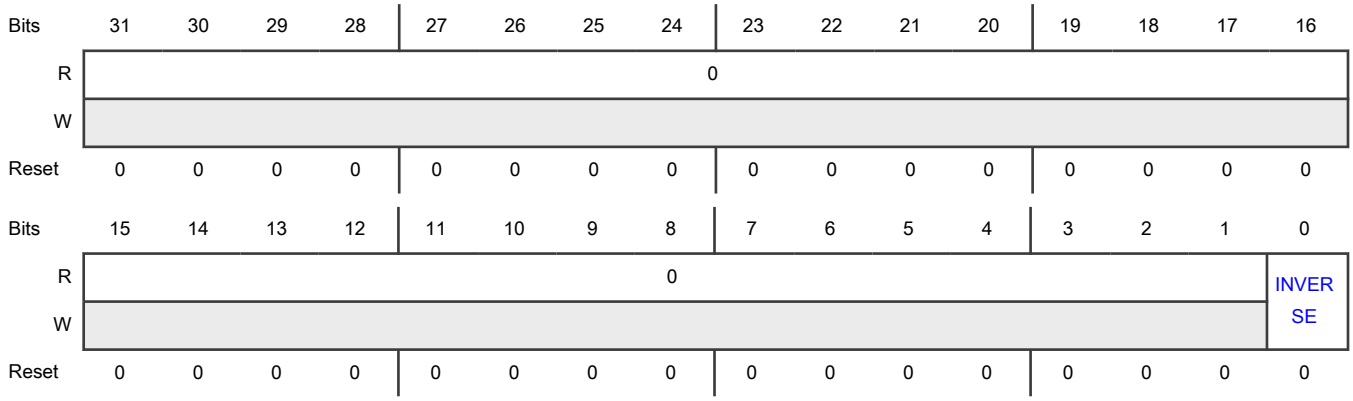
Offset

Register	Offset
FROCTLB	204h

Function

Contains the inverted FRO16K control field. Configure control A and control B registers with the appropriate values when configuring FRO16K. Writes to this register are blocked when FROLCKB[LOCK] is 1.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INVERSE	Inverse Value Contains the inverted contents of FRO16K Control A (FROCTLA).

12.7.1.23 FRO16K Lock A (FROLCKA)

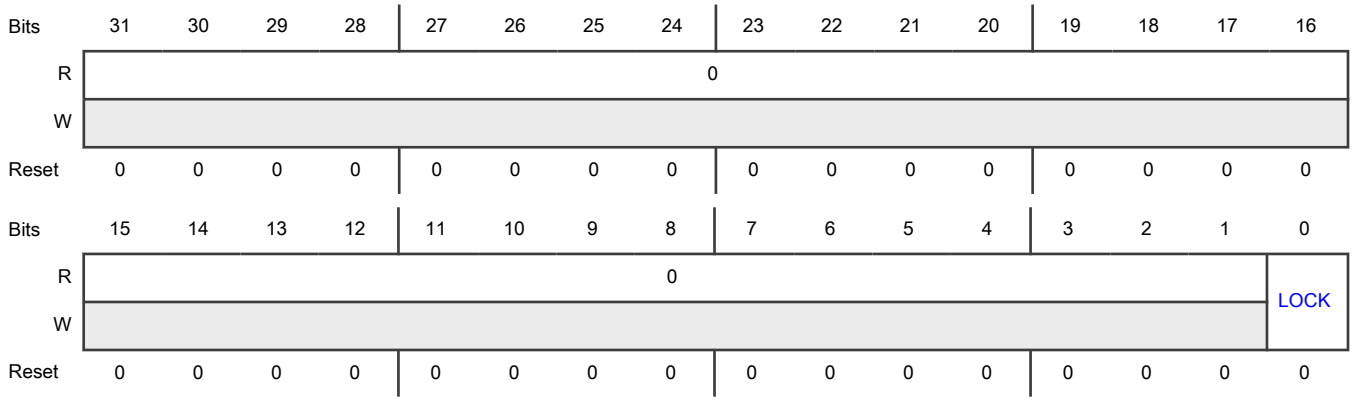
Offset

Register	Offset
FROLCKA	218h

Function

Contains the lock field. When configuring the FRO16K, both lock A and lock B registers need to be programmed to the appropriate values. Writes to this register are blocked when FROLCKA[LOCK] is 1.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock Blocks any write to the FRO16K registers when you write 1 to this field. VBAT POR writes 0 to this field. 0b - Do not block 1b - Block

12.7.1.24 FRO16K Lock B (FROLCKB)

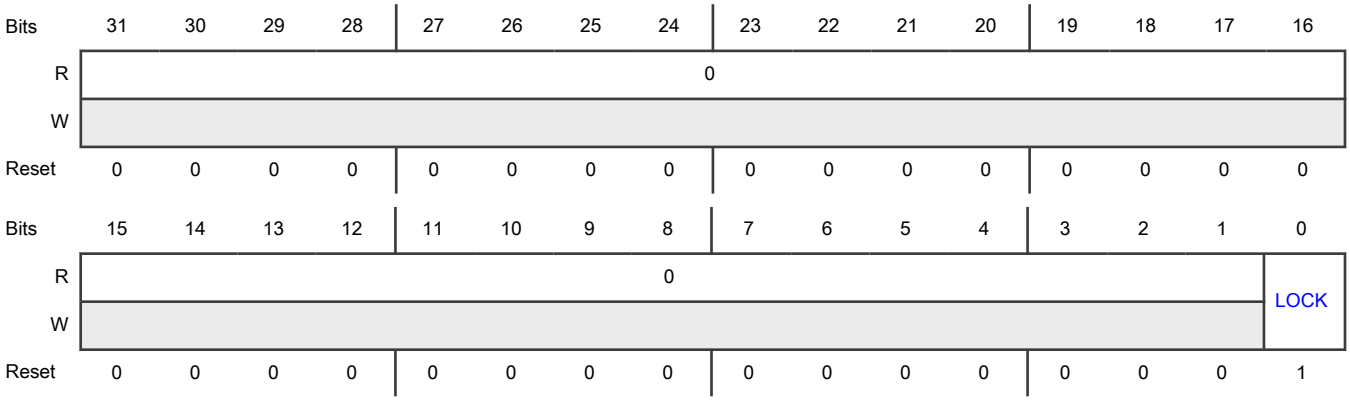
Offset

Register	Offset
FROLCKB	21Ch

Function

Contains the inverted FRO16K lock field. Configure lock A and lock B registers with the appropriate values when configuring FRO16K. Writes to this register are blocked when FROLCKB[LOCK] is 1.

Diagram



Fields

Field	Function
31-1 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
0 LOCK	Lock Blocks any write to the FRO16K registers and asserts a configuration error when you write 0 to this field. The lock is enabled if any of the FRO16K A registers are not equal to inverted FRO16K B registers. 0b - Block 1b - Do not block

12.7.1.25 FRO16K Clock Enable (FROCLKE)

Offset

Register	Offset
FROCLKE	220h

Function

Contains clock gating register field to gate the FRO16K clock to other modules.

NOTE

FROCLKE cannot be locked (not affected by FRO16K Lock A (FROLCKA)).

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0												CLKE			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-4 —	Reserved
3-0 CLKE	Clock Enable Enables the corresponding FRO16 kHz output clock to other modules when you write 1 to the corresponding bit in this field. See the chip-specific VBAT information section for more information.

12.7.1.26 LDO_RAM Control A (LDOCTLA)

Offset

Register	Offset
LDOCTLA	300h

Function

Contains the LDO_RAM control field. When configuring the LDO_RAM, both control A and control B registers need to be programmed to the appropriate values. Writes to this register are blocked when LDOLCKA[LOCK] is 1.

NOTE

The FRO16K must be enabled before enabling the SRAM LDO or the bandgap

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0													REFRESH...	LDO_EN	BG_EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-3 —	Reserved
2 REFRESH_EN	Refresh Enable Enables the Bandgap Low-Power Refresh mode. The refresh mode must also be enabled for lowest power consumption. 0b - Refresh mode is disabled 1b - Refresh mode is enabled for low power operation
1 LDO_EN	LDO Enable Enables the backup SRAM regulator. LDOCTLA[BG_EN] must enable the bandgap when the backup SRAM regulator is enabled. After you enable the bandgap and backup SRAM regulator, you must wait until STATUSA[LDO_RDY] becomes 1 before the SRAM can enter a low-power state where the regulator supplies the array contents.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Disable 1b - Enable
0 BG_EN	Bandgap Enable Enables the LDO_RAM bandgap. 0b - Disable 1b - Enable

12.7.1.27 LDO_RAM Control B (LDOCTLB)

Offset

Register	Offset
LDOCTLB	304h

Function

Contains the inverted LDO_RAM control field. Configure control A and control B registers with the appropriate values when configuring the LDO_RAM. Writes to this register are blocked when LDOLCKB[LOCK] is 1.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0												INVERSE			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Fields

Field	Function
31-3 —	Reserved
2-0 INVERSE	Inverse Value Contains the inverted contents of LDO_RAM Control A (LDOCTLA).

12.7.1.28 LDO_RAM Lock A (LDOLCKA)

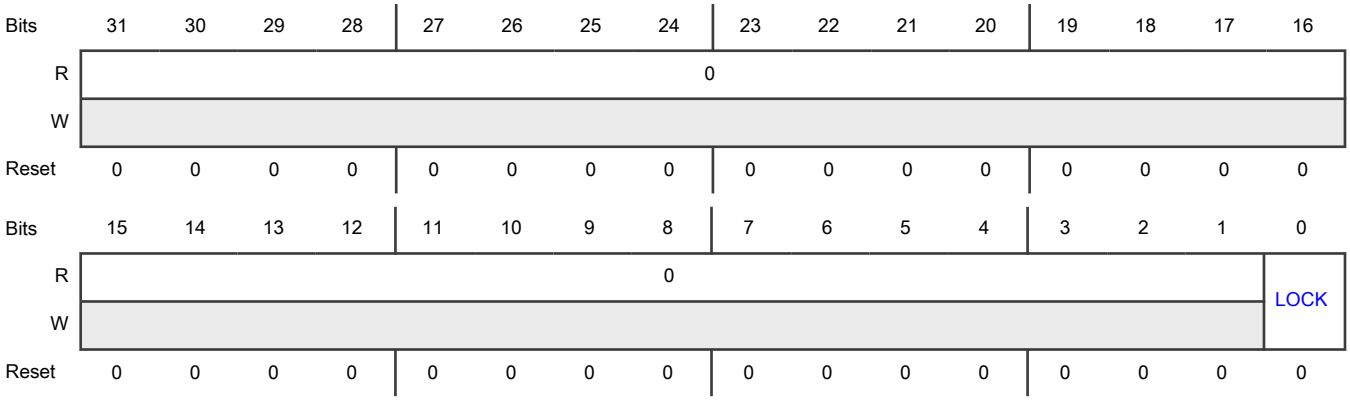
Offset

Register	Offset
LDOLCKA	318h

Function

Contains the lock field. When configuring the LDO_RAM, both lock A and lock B registers need to be programmed to the appropriate values. Writes to this register are blocked when the LDO_RAM lock register is 1.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock Blocks any write to the LDO_RAM registers. VBAT POR writes 0 to this field. 0b - Do not block 1b - Block

12.7.1.29 LDO_RAM Lock B (LDOLCKB)

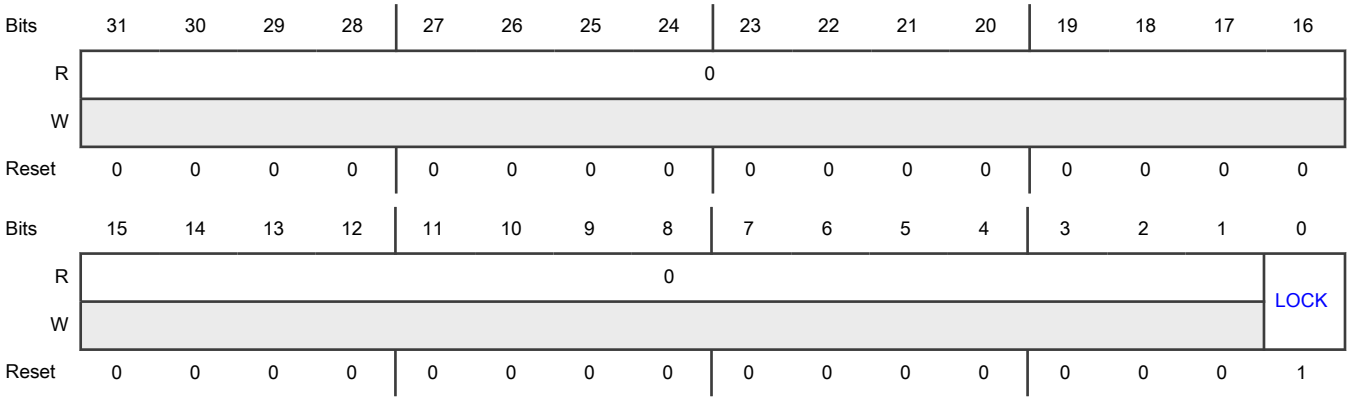
Offset

Register	Offset
LDOLCKB	31Ch

Function

Contains the inverted LDO_RAM lock field. Configure lock A and lock B registers with the appropriate values when configuring the LDO_RAM. Writes to this register are blocked when LDOLCKB[LOCK] is 1.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock Blocks any write to the LDO_RAM registers and asserts a configuration error. The lock is enabled if any of the LDO_RAM A registers are not equal to inverted LDO_RAM B registers. 0b - Block 1b - Do not block

12.7.1.30 RAM Control (LDORAMC)

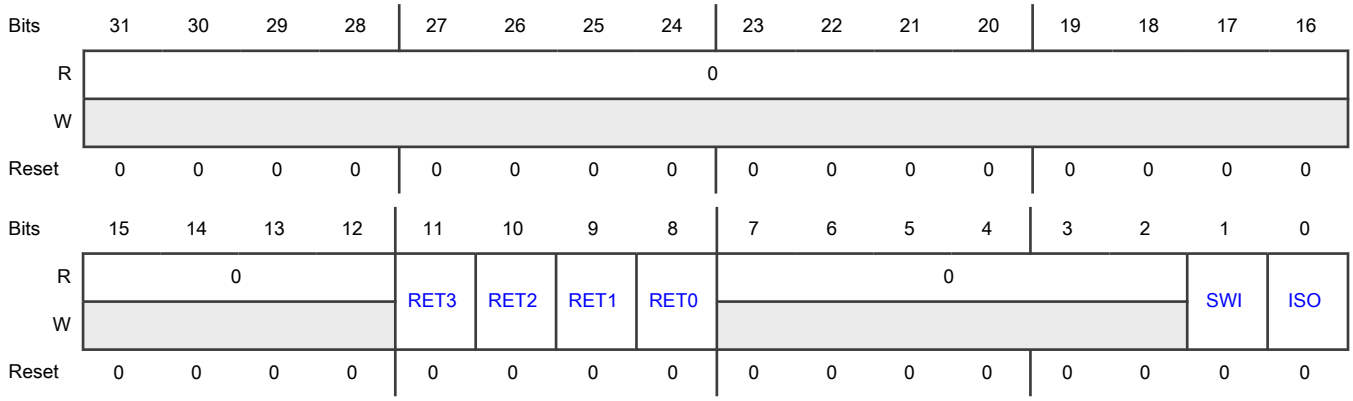
Offset

Register	Offset
LDORAMC	320h

Function

Contains software overrides for the SRAM isolation and backup switch. Always isolate the SRAM array before asserting or negating the switch control.

Diagram



Fields

Field	Function
31-12 —	Reserved
11-8 RETn	Retention Configures retention of each SRAM array in low power modes. See the VBAT chip-specific section for more details on these SRAM arrays. When LDO_RAM is enabled, at least one SRAM array must be retained. When LDO_RAM is disabled, this field specifies whether VDD_CORE continues to provide power to the SRAM array. 0b - Corresponding SRAM array is retained in low-power modes 1b - Corresponding SRAM array is not retained in low-power modes
7-2 —	Reserved
1 SWI	Switch SRAM Specifies how the SRAM array is powered. Change this field when LDORAMC[ISO] = 1. 0b - Supply follows the chip power modes 1b - LDO_RAM powers the array
0 ISO	Isolate SRAM Specifies how the SRAM array is isolated. 0b - State follows the chip power modes 1b - Isolates SRAM and places it in Low-Power Retention mode

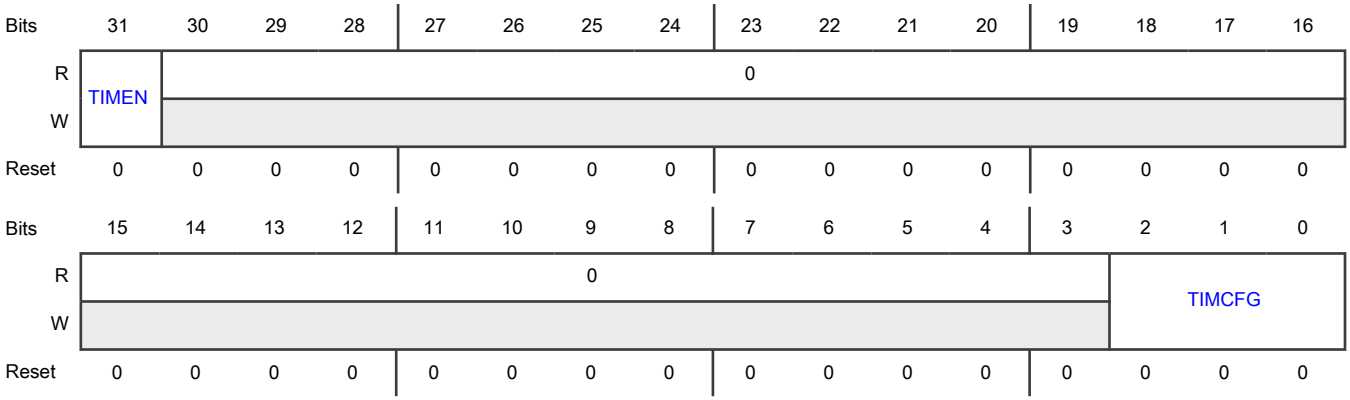
12.7.1.31 Bandgap Timer 0 (LDOTIMER0)

Offset

Register	Offset
LDOTIMER0	330h

Function
Controls one software-configurable timer when the bandgap is enabled and are clocked by FRO16K.

Diagram



Fields

Field	Function
31 TIMEN	Bandgap Timeout Period Enable 0b - Disable 1b - Enable
30-3 —	Reserved
2-0 TIMCFG	Timeout Configuration Configures the bandgap timeout 0 period. Changed when the timer is disabled. 000b - 1 s 001b - 500 ms 010b - 250 ms 011b - 125 ms 100b - 62.5 ms 101b - 31.25 ms 110b - 15.625 ms 111b - 7.8125 ms

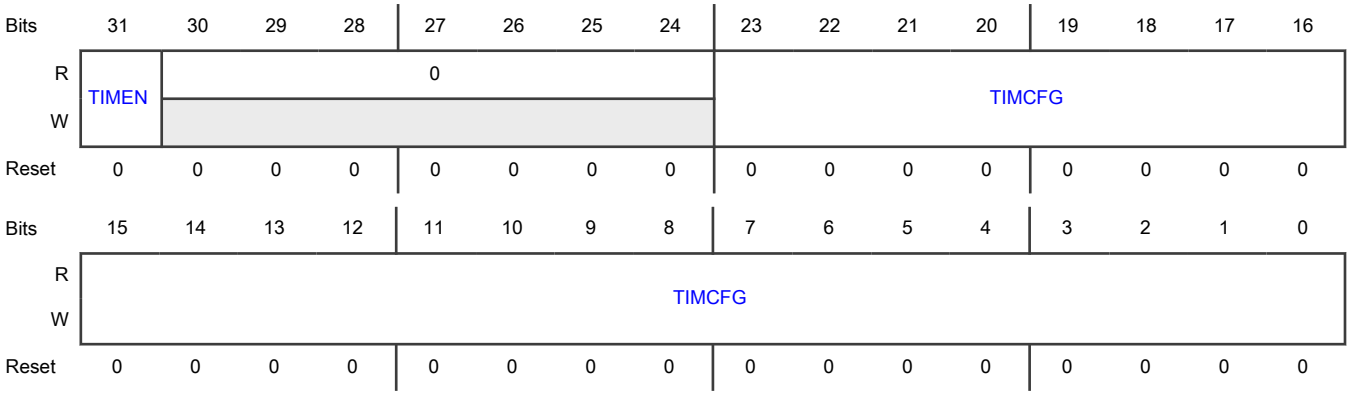
12.7.1.32 Bandgap Timer 1 (LDOTIMER1)

Offset

Register	Offset
LDOTIMER1	338h

Function
Controls the other software-configurable timer when the bandgap is enabled and are clocked by the FRO16K.

Diagram



Fields

Field	Function
31 TIMEN	Bandgap Timeout Period Enable 0b - Disable 1b - Enable
30-24 —	Reserved
23-0 TIMCFG	Timeout Configuration Configures the bandgap timeout 1 period. Configures timeout in number of seconds, ranging from 1 to 16,777,216 s. You can change this field when the timer is disabled.

12.7.1.33 CLKMON Control A (MONCTLA)

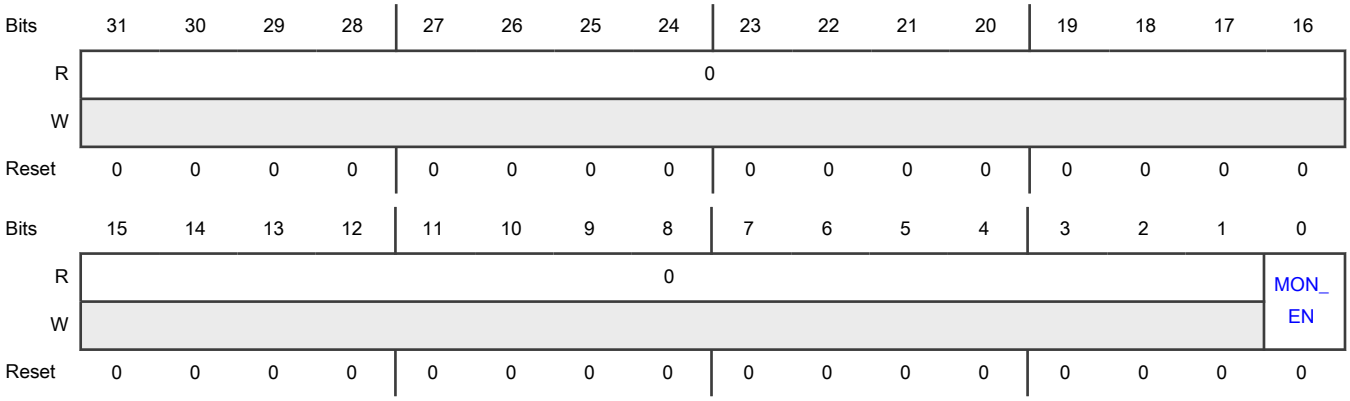
Offset

Register	Offset
MONCTLA	400h

Function

Contains the CLKMON control bits. Writes to this register are blocked when the CLKMON lock register is set.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 MON_EN	CLKMON Enable Enables the CLKMON when set to '1'. 0b - CLKMON is disabled 1b - CLKMON is enabled

12.7.1.34 CLKMON Control B (MONCTLB)

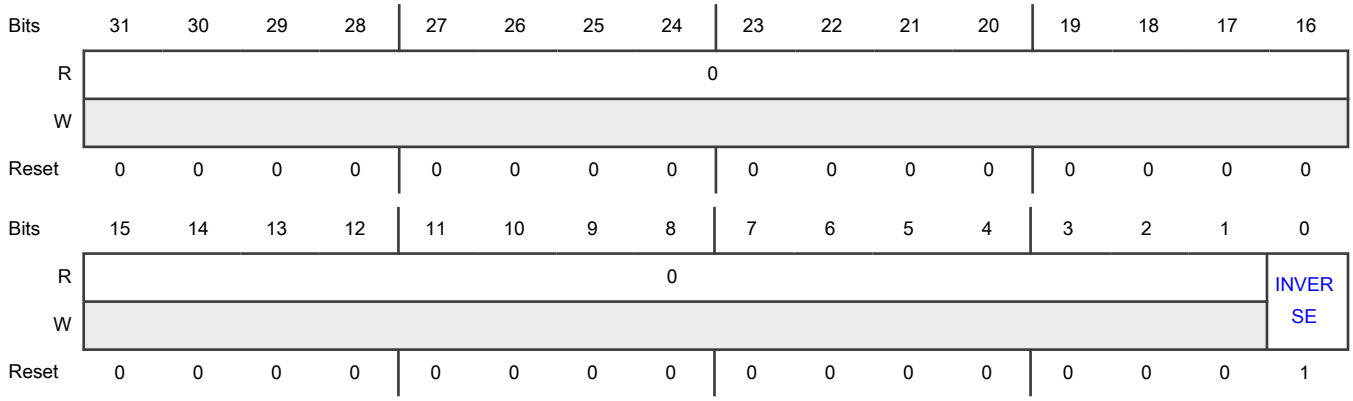
Offset

Register	Offset
MONCTLB	404h

Function

Contains the inverted CLKMON control bits. When configuring the CLKMON, both control A and control B registers need to be programmed to the appropriate values. Writes to this register are blocked when the CLKMON lock register is set.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INVERSE	Inverse value Must be programmed with the inverted contents of MONCTLA.

12.7.1.35 CLKMON Configuration A (MONCFGA)

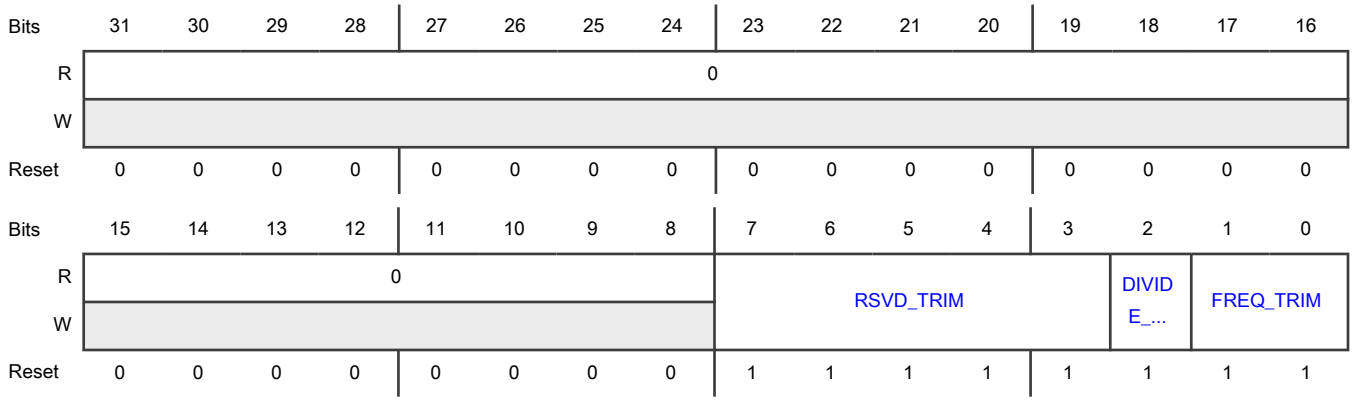
Offset

Register	Offset
MONCFGA	408h

Function

Contains the CLKMON configuration bits. Writes to this register are blocked when the CLKMON lock register is set.

Diagram



Fields

Field	Function
31-8 —	Reserved
7-3 RSVD_TRIM	Reserved Trim Clock monitor reserved trim.
2 DIVIDE_TRIM	Divide Trim Clock monitor divide trim. 0b - Clock monitor operates at 1 kHz 1b - Clock monitor operates at 64 Hz
1-0 FREQ_TRIM	Frequency Trim Clock monitor frequency trim. 00b - Clock monitor asserts 2 cycle after expected edge 01b - Clock monitor asserts 4 cycles after expected edge 10b - Clock monitor asserts 6 cycles after expected edge 11b - Clock monitor asserts 8 cycles after expected edge

12.7.1.36 CLKMON Configuration B (MONCFGB)

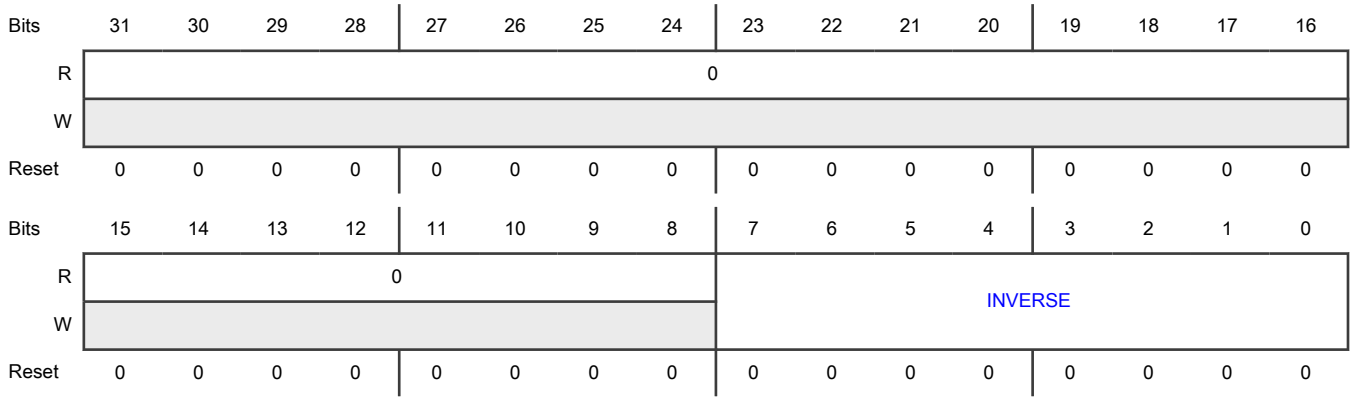
Offset

Register	Offset
MONCFGB	40Ch

Function

Contains the inverted CLKMON configuration bits. When configuring the CLKMON, both configuration A and configuration B registers need to be programmed to the appropriate values. Writes to this register are blocked when the CLKMON lock register is set.

Diagram



Fields

Field	Function
31-8 —	Reserved
7-0 INVERSE	Inverse value Must be programmed with the inverted contents of MONCFGA.

12.7.1.37 CLKMON Test A (MONTSTA)

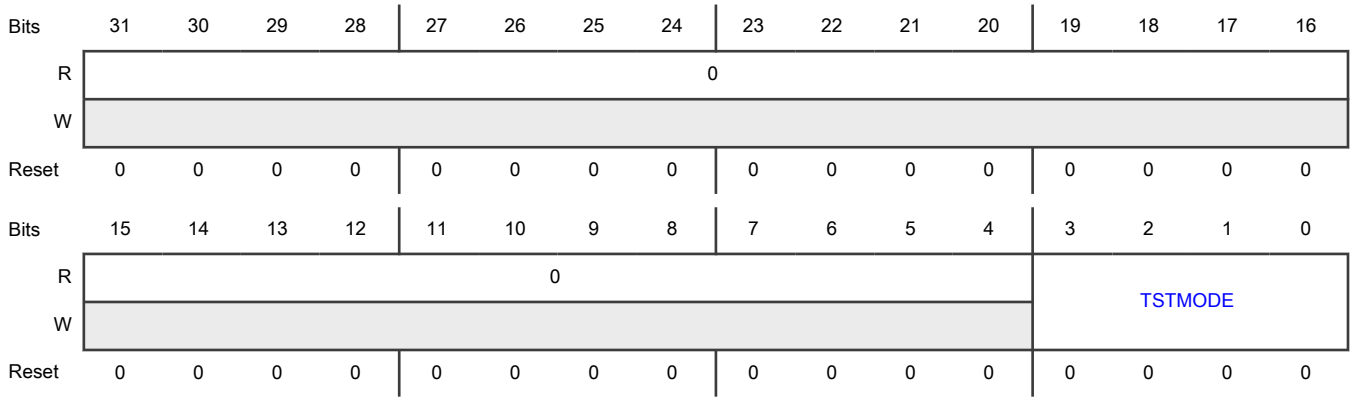
Offset

Register	Offset
MONTSTA	410h

Function

Contains the CLKMON test bits. Writes to this register are blocked when the CLKMON lock register is set.

Diagram



Fields

Field	Function
31-4 —	Reserved
3-0 TSTMODE	Test Mode Configure the CLKMON test mode.

12.7.1.38 CLKMON Test B (MONTSTB)

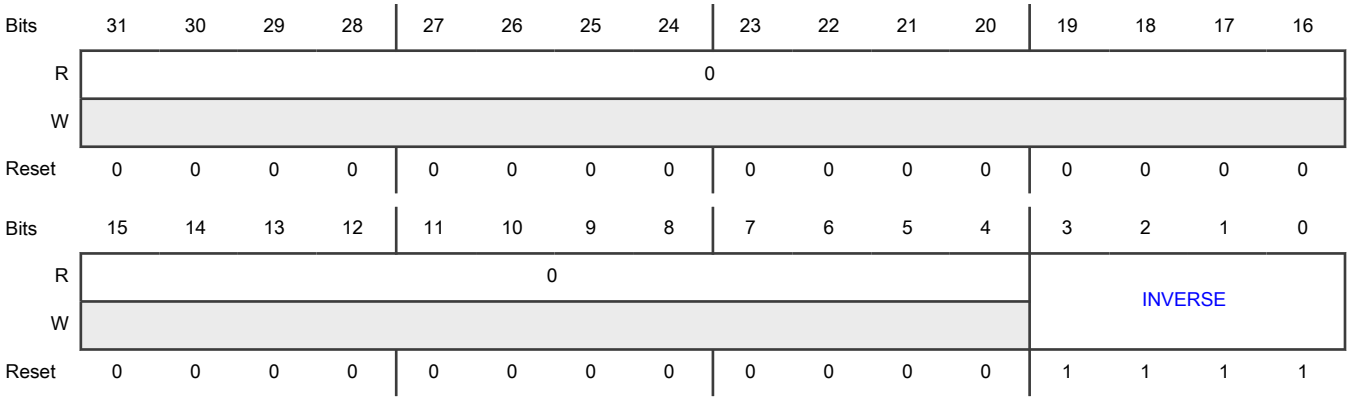
Offset

Register	Offset
MONTSTB	414h

Function

Contains the inverted CLKMON test bits. When configuring the CLKMON, both test A and test B registers need to be programmed to the appropriate values. Writes to this register are blocked when the CLKMON lock register is set.

Diagram



Fields

Field	Function
31-4 —	Reserved
3-0 INVERSE	Inverse value Must be programmed with the inverted contents of MONTSTA.

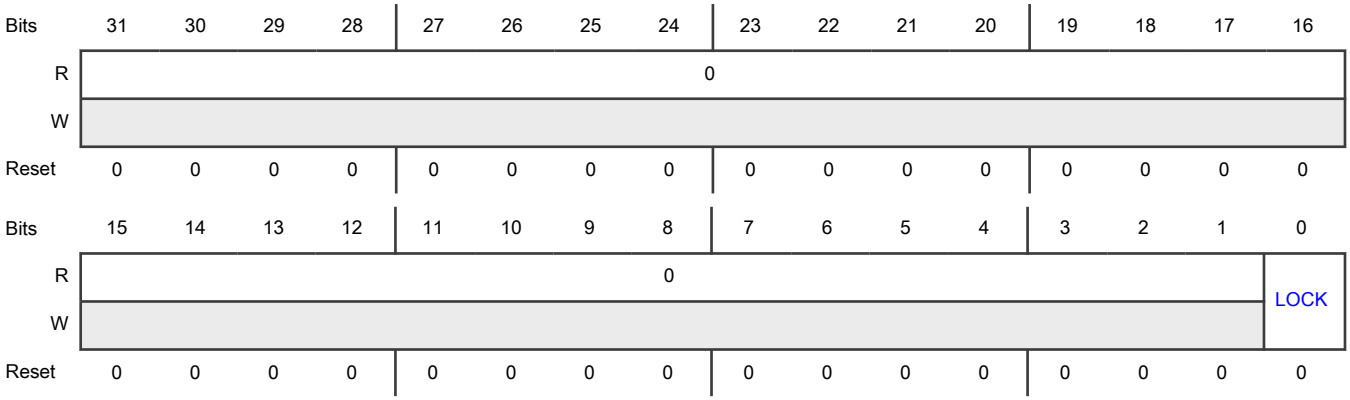
12.7.1.39 CLKMON Lock A (MONLCKA)

Offset

Register	Offset
MONLCKA	418h

Function
Contains the lock bits. Writes to this register are blocked when the CLKMON lock register is set.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock When set, blocks writes to the CLKMON registers. 0b - Lock is disabled 1b - Lock is enabled

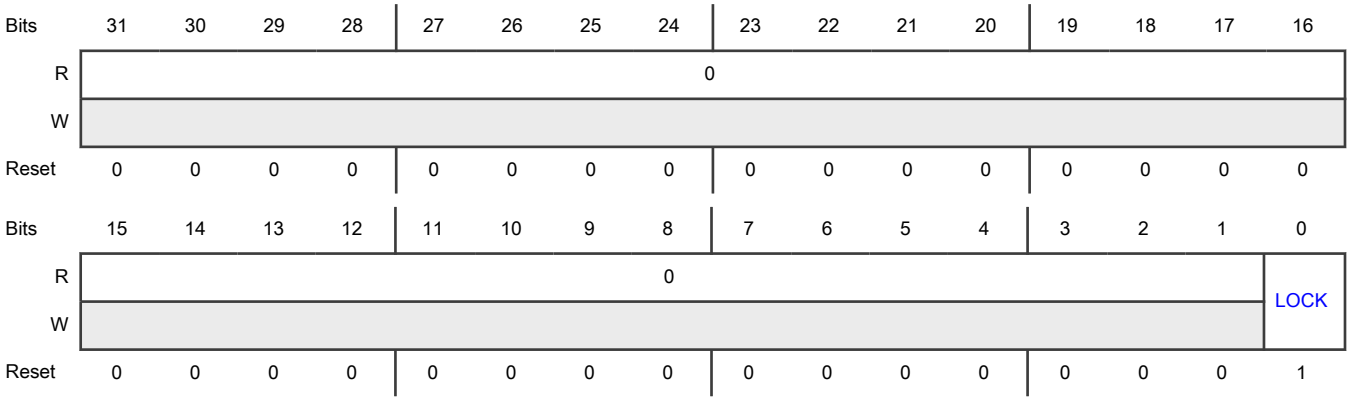
12.7.1.40 CLKMON Lock B (MONLCKB)

Offset

Register	Offset
MONLCKB	41Ch

Function
Contains the inverted CLKMON lock bits. When configuring the CLKMON, both lock A and lock B registers need to be programmed to the appropriate values. Writes to this register are blocked when the CLKMON lock register is set.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock When clear, blocks writes to the CLKMON registers and asserts a configuration error if any of the CLKMON A registers are not equal to inverted CLKMON B registers. 0b - Lock is enabled 1b - Lock is disabled

12.7.1.41 TAMPER Control A (TAMCTLA)

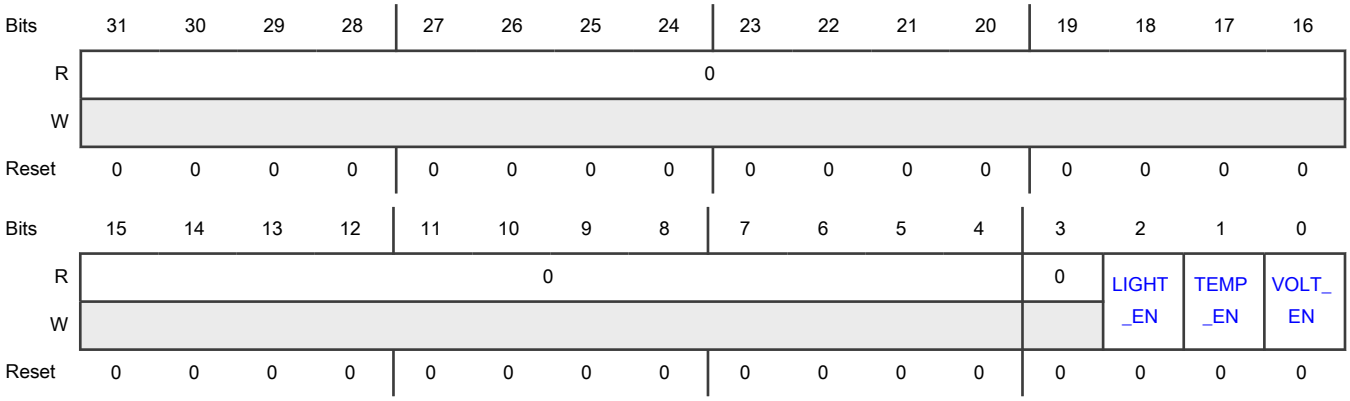
Offset

Register	Offset
TAMCTLA	500h

Function

Contains the non-inverted TAMPER control bits. When configuring the TAMPER, both control A and control B registers need to be programmed to the appropriate values. Writes to this register are blocked when the TAMPER lock register is set.

Diagram



Fields

Field	Function
31-4 —	Reserved
3 —	Reserved
2 LIGHT_EN	Light Detect Enable Enables the Light detect. 0b - Light detect is disabled 1b - Light detect is enabled
1 TEMP_EN	Temperature Detect Enable Enables the temperature detect. 0b - Temperature detect is disabled 1b - Temperature detect is enabled
0 VOLT_EN	Voltage Detect Enable Enables the Voltage detect. 0b - Voltage detect is disabled 1b - Voltage detect is enabled

12.7.1.42 TAMPER Control B (TAMCTLB)

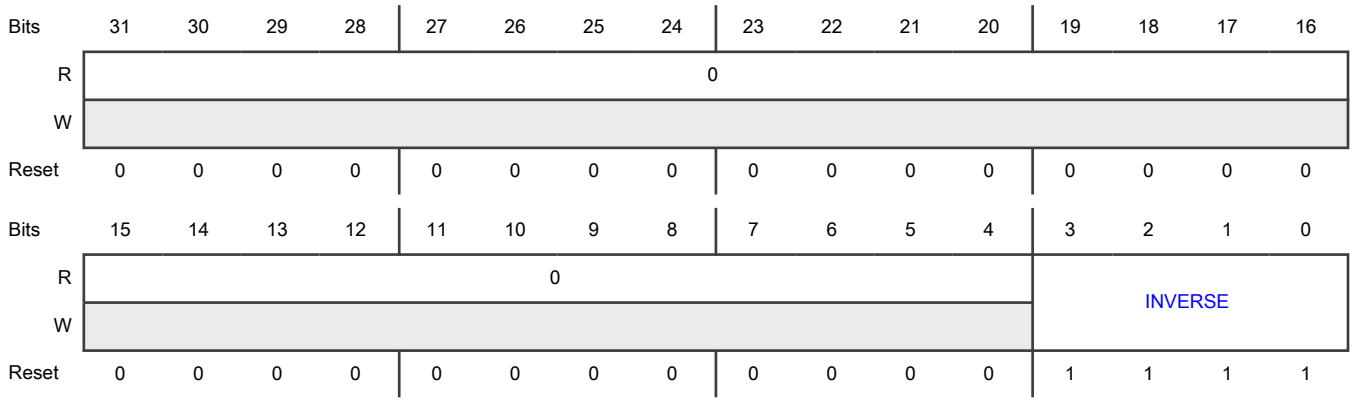
Offset

Register	Offset
TAMCTLB	504h

Function

Contains the inverted TAMPER control bits. When configuring the TAMPER, both control A and control B registers need to be programmed to the appropriate values. Writes to this register are blocked when the TAMPER lock register is set.

Diagram



Fields

Field	Function
31-4 —	Reserved
3-0 INVERSE	Inverse value Must be programmed with the inverted contents of TAMCTLA.

12.7.1.43 TAMPER Lock A (TAMLCKA)

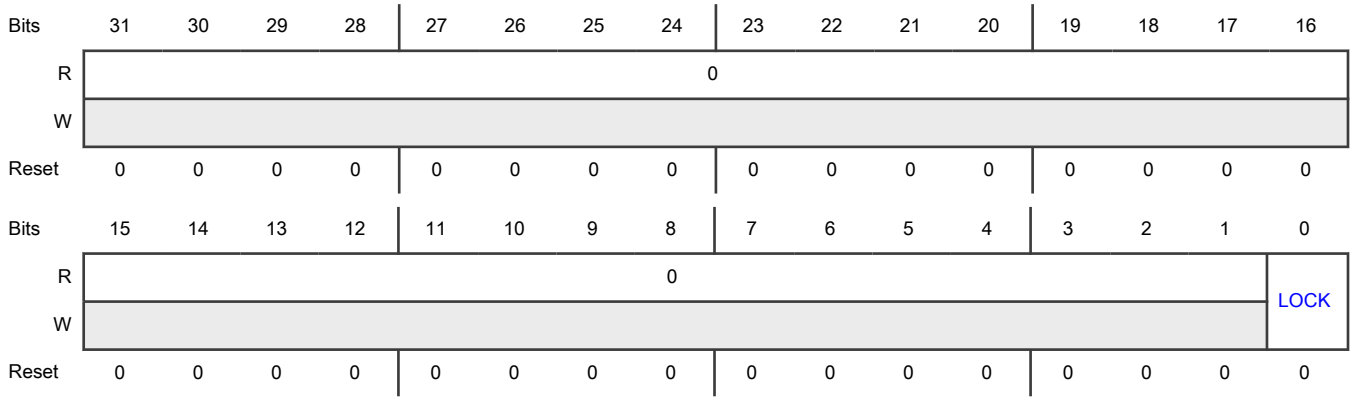
Offset

Register	Offset
TAMLCKA	518h

Function

Contains the non-inverted lock bits. When configuring the TAMPER, both lock A and lock B registers need to be programmed to the appropriate values. Writes to this register are blocked when the TAMPER lock register is set.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock When set, blocks writes to the TAMPER registers and asserts a configuration error if any of the TAMPER A registers are not equal to inverted TAMPER B registers. 0b - Lock is disabled 1b - Lock is enabled

12.7.1.44 TAMPER Lock B (TAMLCKB)

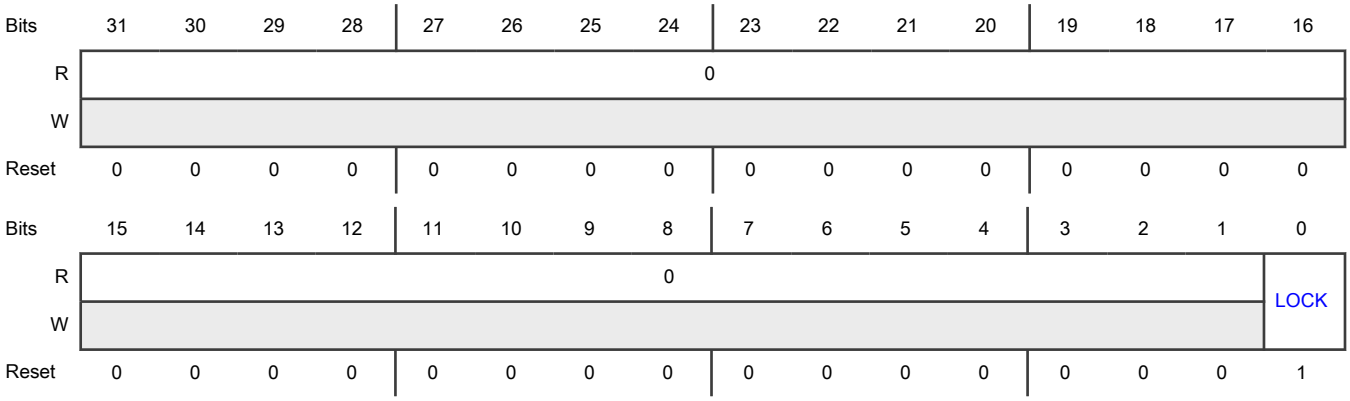
Offset

Register	Offset
TAMLCKB	51Ch

Function

Contains the inverted TAMPER lock bits. When configuring the TAMPER, both lock A and lock B registers need to be programmed to the appropriate values. Writes to this register are blocked when the TAMPER lock register is set.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock When clear, blocks writes to the TAMPER registers and asserts a configuration error if any of the TAMPER A registers are not equal to inverted TAMPER B registers. 0b - Lock is enabled 1b - Lock is disabled

12.7.1.45 Switch Control A (SWICTLA)

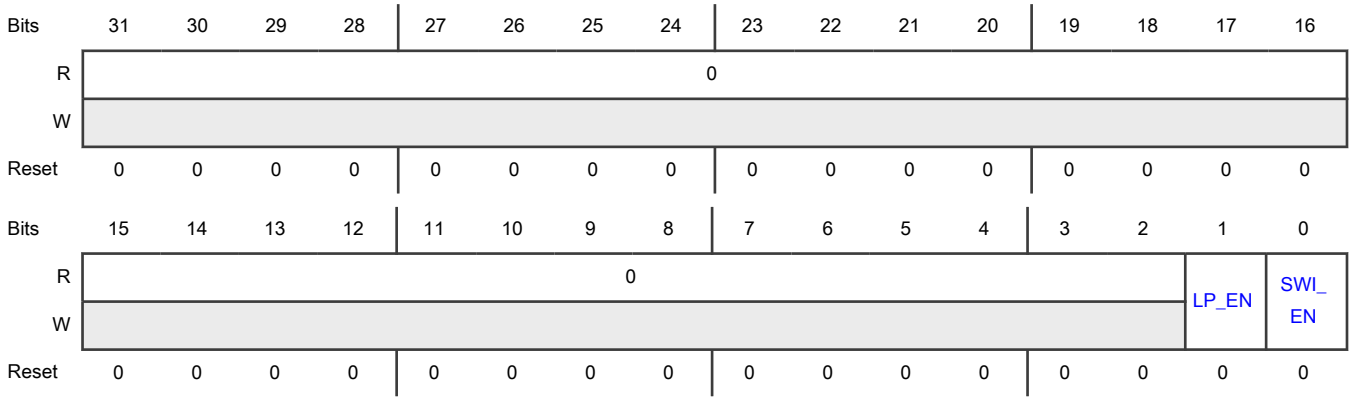
Offset

Register	Offset
SWICTLA	600h

Function

Contains the switch control field. When configuring the switch, both control A and control B registers need to be programmed to the appropriate values. Writes to this register are blocked when Switch Lock A (SWILCKA) is 1. Both VBAT Cold Reset and VDD_SYS Cold Reset reset this register.

Diagram



Fields

Field	Function
31-2 —	Reserved
1 LP_EN	Low Power Enable Configures the VBAT internal switch in low-power modes. 0b - VDD_BAT always supplies VBAT modules in low-power modes 1b - VDD_SYS always supplies VBAT modules if SWI_EN is also 1
0 SWI_EN	Switch Enable Specifies the supply for VBAT modules. 0b - VDD_BAT 1b - VDD_SYS

12.7.1.46 Switch Control B (SWICTLB)

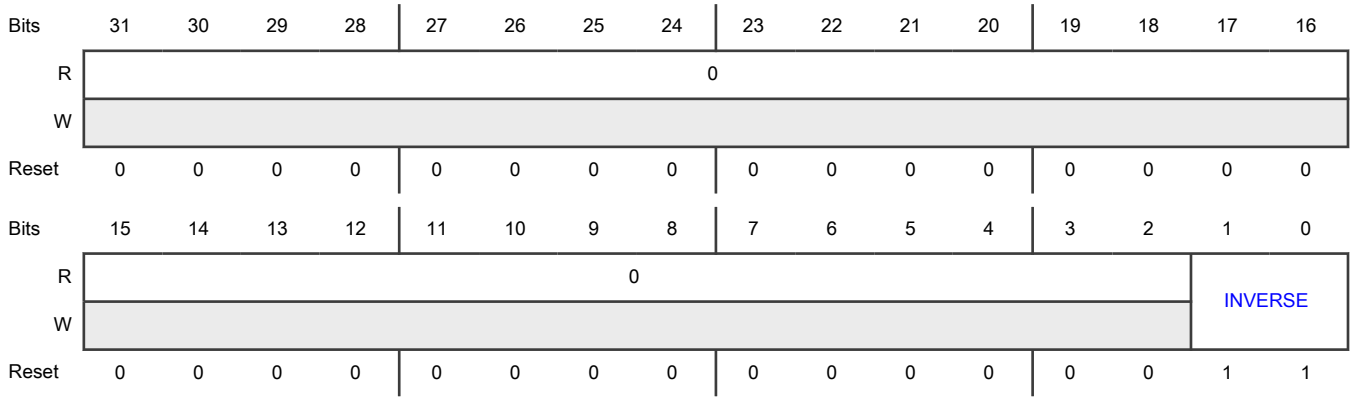
Offset

Register	Offset
SWICTLB	604h

Function

Contains the inverted switch control field. Configure control A and control B registers with the appropriate values when configuring the switch. Writes to this register are blocked when Switch Lock B (SWILCKB) is 1. Both VBAT Cold Reset and VDD_SYS Cold Reset reset this register.

Diagram



Fields

Field	Function
31-2 —	Reserved
1-0 INVERSE	Inverse Value Contains the inverted contents of Switch Control A (SWICTLA).

12.7.1.47 Switch Lock A (SWILCKA)

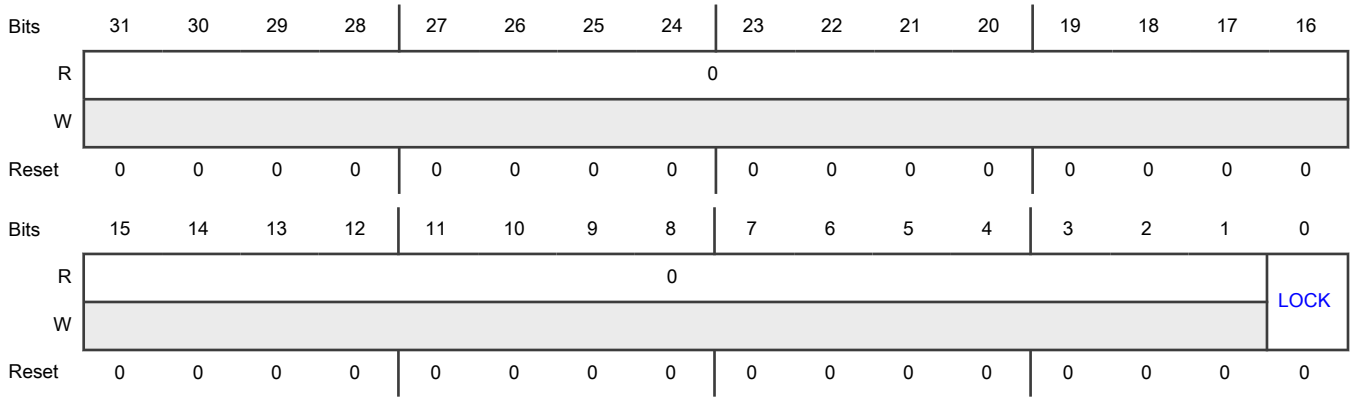
Offset

Register	Offset
SWILCKA	618h

Function

Contains the lock field. When configuring the switch, both lock A and lock B registers need to be programmed to the appropriate values. Writes to this register are blocked when the fields in SWITCH lock registers are 1. Both VBAT Cold Reset and VDD_SYS Cold Reset reset this register.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock Blocks any write to the switch registers. 0b - Do not block 1b - Block

12.7.1.48 Switch Lock B (SWILCKB)

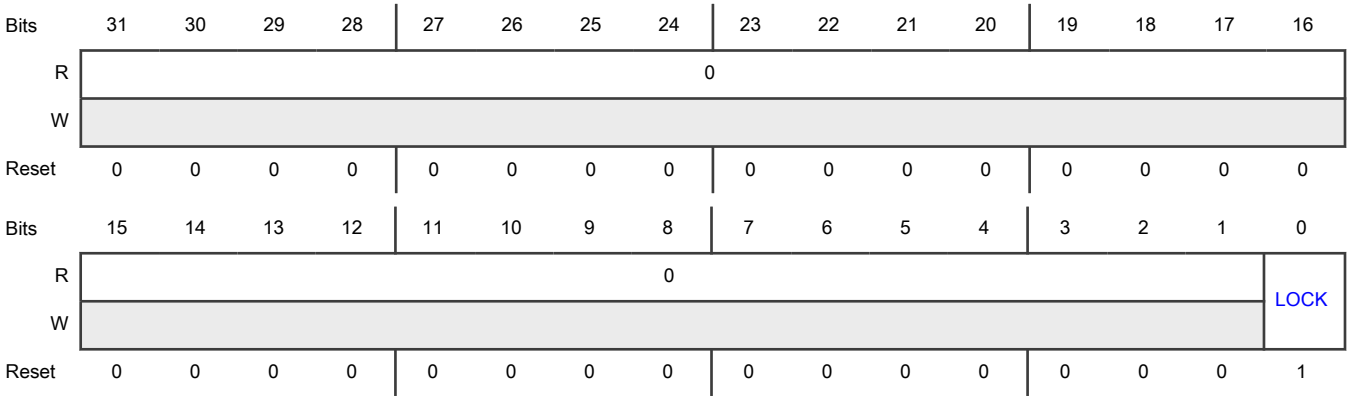
Offset

Register	Offset
SWILCKB	61Ch

Function

Contains the inverted switch lock field. Configure control A and control B registers with the appropriate values when configuring the switch. Writes to this register are blocked when the fields in switch lock registers are 1. Both VBAT Cold Reset and VDD_SYS Cold Reset reset this register.

Diagram



Fields

Field	Function
31-1 —	Reserved
0	Lock

Table continues on the next page...

Table continued from the previous page...

Field	Function
LOCK	Blocks any write to the switch registers and asserts a configuration error. The lock is enabled if any of the switch A registers are not equal to inverted switch B registers. 0b - Block 1b - Do not block

12.7.1.49 Wakeup 0 Register A (WAKEUP0A - WAKEUP1A)

Offset

Register	Offset
WAKEUP0A	700h
WAKEUP1A	708h

Function

Contains software writable bits. Writes to this register are blocked when WAKEUP lock register is set.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	REG															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	REG															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 REG	Register Software writable value.

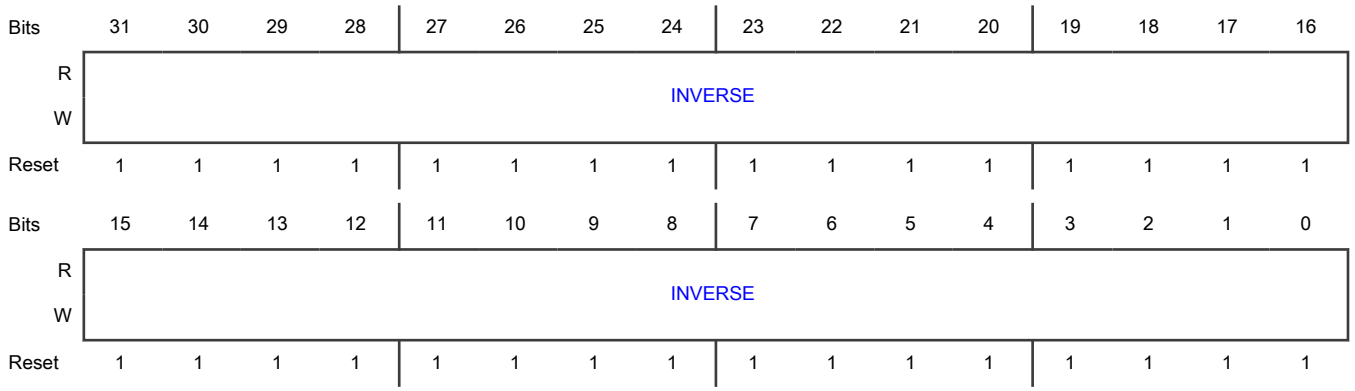
12.7.1.50 Wakeup 0 Register B (WAKEUP0B - WAKEUP1B)

Offset

Register	Offset
WAKEUP0B	704h
WAKEUP1B	70Ch

Function
Contains the inverted Wakeup 0 software writable bits. When configuring the wakeup registers, both wakeup A and wakeup B registers need to be programmed to the appropriate values. Writes to this register are blocked when WAKEUP lock register is set.

Diagram



Fields

Field	Function
31-0 INVERSE	Inverse value Must be programmed with the inverted contents of WAKEUP0A.

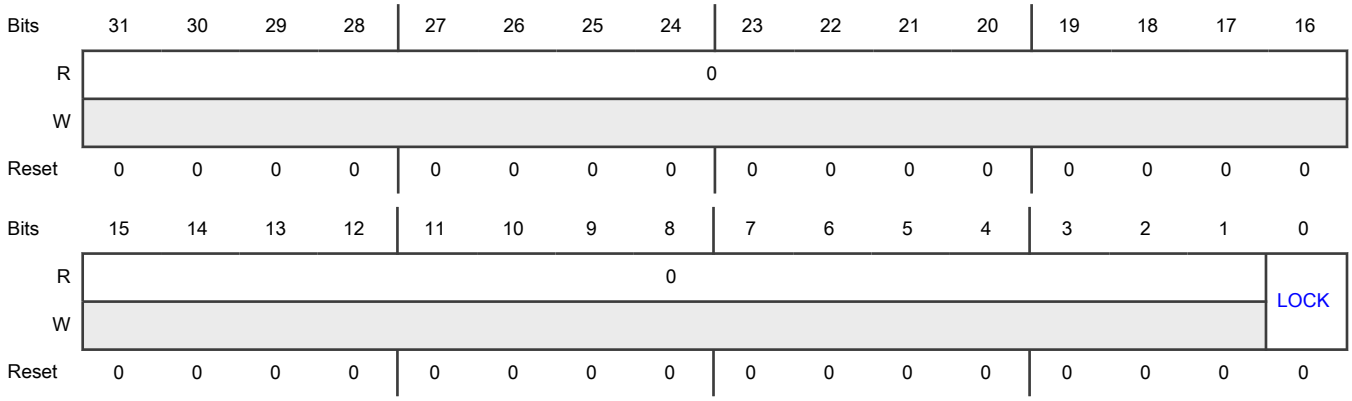
12.7.1.51 Wakeup Lock A (WAKLCKA)

Offset

Register	Offset
WAKLCKA	7F8h

Function
Contains the lock bits. Writes to this register are blocked when Wakeup lock registers are set.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock When set, blocks writes to the Wakeup registers. 0b - Lock is disabled 1b - Lock is enabled

12.7.1.52 Wakeup Lock B (WAKLCKB)

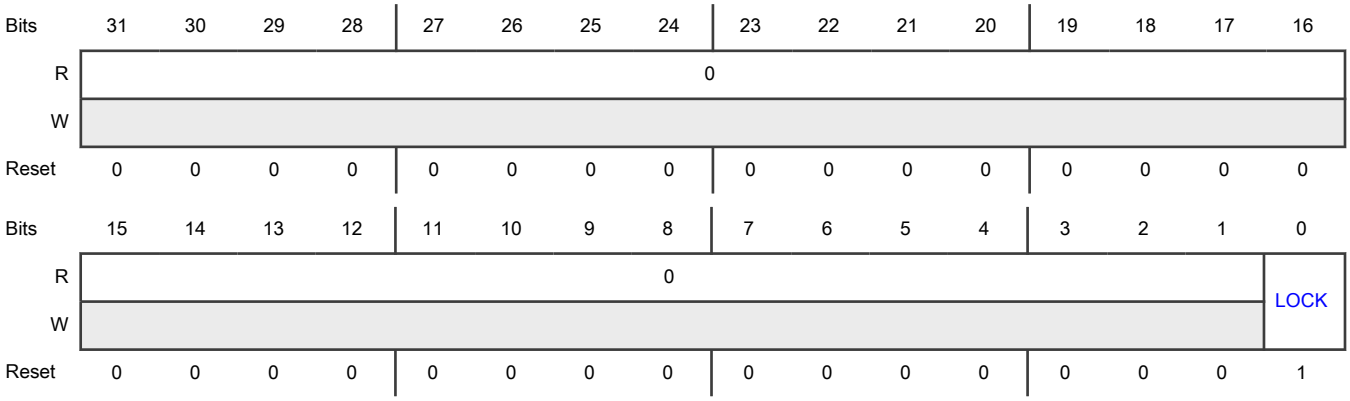
Offset

Register	Offset
WAKLCKB	7FCh

Function

Contains the inverted wakeup lock bits. When configuring the wakeup registers, both lock A and lock B registers need to be programmed to the appropriate values. Writes to this register are blocked when wakeup lock registers are set.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 LOCK	Lock When clear, blocks writes to the wakeup registers and asserts a configuration error if any of the wakeup A registers are not equal to inverted wakeup B registers. 0b - Lock is enabled 1b - Lock is disabled

Chapter 13
EdgeLock Secure Subsystem (ELS)

13.1 Chip-specific ELS information

Table 245. Reference links to related information[1]

Topic	Related module	Reference
Full description	ELS	ELS
System memory map		See the section "System memory map"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

NOTE

The reset value of STATUS, CMDCFG0, and VERSION registers may change depending on the device's boot settings.

NOTE

The reset value of STATUS, and VERSION registers may change on sync reset.

NOTE

The EdgeLock Secure Subsystem (ELS) is also known as EdgeLock Secure Enclave, Core Profile (ELE). This document uses the ELS name, but other materials might refer to this module as EdgeLock Secure Enclave, Core Profile or ELE.

13.1.1 Module instances

This device has one instance of the ELS module with four aliased base addresses.

13.1.2 Security considerations

The ELS module implements an Access control and access rights feature. The ELS uses the input secure/non-secure and privileged/non-privileged to determine the keys and commands the current master can use. The attributes from the current master are also propagated to the system bus if a ELS command generates any memory accesses on the system bus.

The ELS module is instantiated to use four module slots—ELS, ELS_alias1, ELS_alias2, and ELS_alias3. At the Secure AHB controller, each ELS slot can be configured for a different secure/non-secure and privileged/non-privileged access. This allows the ELS module to be used by any access level using the appropriate slot, while the Secure AHB controller's MISC_CTRL_REG[DISABLE_STRICT_MODE] option is configured for strict mode. The ELS module's per-key access levels are then used to restrict access to keys and commands and also determine the level used for ELS system bus accesses.

13.1.3 Crypto subsystem

The crypto subsystem provides a strong hardware isolation of the root key material and supports all the key related operations on the device, including key generation, key derivation, key (de)obfuscation, key wrapping/unwrapping, and the cryptographic acceleration (symmetric and asymmetric key crypto, hash function).

AES operations are protected against side-channel attacks, and ECDSA P-256 operations are protected against side-channel and low-cost fault attacks.

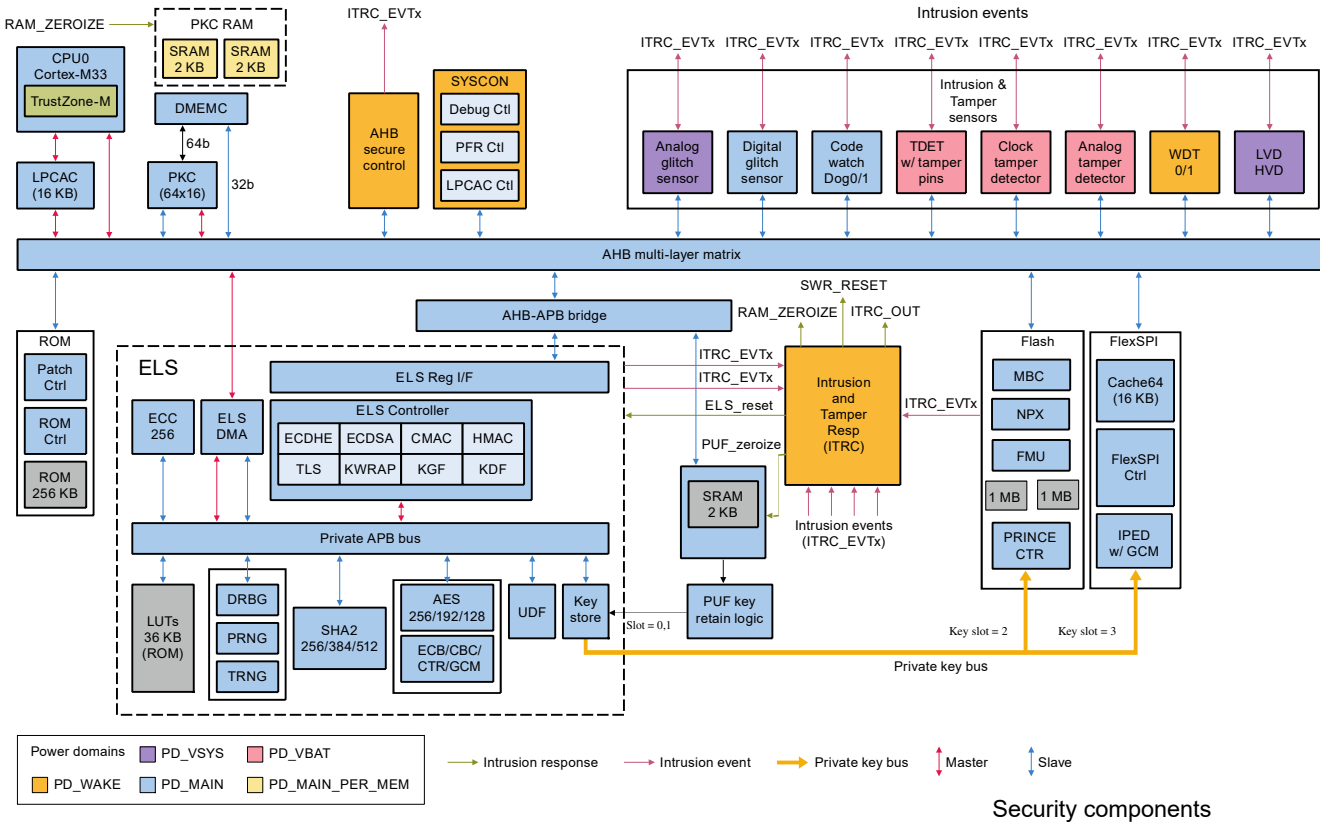


Figure 66. ELS subsystem

13.1.4 ELS private key bus usage

ELS has private key buses that are used for both key input and output. The table below describes how private key buses are connected on this device.

Table 246. ELS private key bus usage

ELS key slot	Input/output	Key	Comments
0, 1	Input	NXP_DIE_MK_SK	The Boot ROM executes KEYIN command on each start-up of the device, to transfer the 256-bit Device Unique Key (DUK) called NXP_DIE_MK_SK from PUF into ELS key slots 0 & 1. See Key management for more detailed description."
2	output	NXP_DIE_MEM_ENC_SK	128-bit PRINCE encryption key for internal flash bus encryption engine (NPX).
3	output	NXP_DIE_EXT_MEM_ENC_SK	128-bit PRINCE encryption key for external flash bus encryption engine (IPED).

13.1.5 Asset Protection Port

The ELS asset protection is fully under control of Boot ROM. The assets are always protected by setting port value to 'b10 when ROM is leaving to user context. The Boot ROM also locks access to asset protection port, so that it cannot be changed outside of the ROM context.

13.2 Terminology

13.2.1 Notation

Table 247. Notation

Notation	Meaning	Explanation
||	String concatenation	In the notation A = B || C If B = "abc", and C = "xyz" then A = "abcxyz"

13.2.2 Acronyms

Acronym	Explanation
ECDSA	Elliptic Curve Digital Signature Algorithm: a variant of the Digital Signature Algorithm (DSA) which uses elliptic curve cryptography.
HW	Hardware
IV	Initialization Vector
KDF	Key Derivation Function
REVF	Reverse Fetch
RNG	Random Number Generator
SFR	Special Function Register
SHA	Secure Hash Algorithm
SW	Software

13.2.3 Terms

NOTE

The following terms will only have the given meaning when used in this document within single quotes (that is, crypto command)

Term	Meaning
Key slot	A 128-bit register, used to store crypto data or key info
Bus access	A bus read or write
Crypto computation	A crypto computation refers to an AES, CMAC, DES, ECDSA, or SHA computation
PCLK	Is the clock for the AMBA APB bus interface on which ELS is a slave

13.3 Overview

The EdgeLock® Secure Subsystem (ELS) S50 module is a security subsystem supporting a wide range of cryptographic algorithms (listed in below table) and providing strong key isolation from the rest of the system. ELS is the main building block of an SOC's immutable Root of Trust. It is used as part of the trust anchor during secure boot, secure debug access, life-cycle management, and trust provisioning. AES operations of ELS are configured to provide resistance against side-channel attacks. ECC P-256 operations provide protection against side-channel attacks as well as low-cost fault attacks, such as non-localized electromagnetic fault injection (EMFI), power supply, and clock glitch attacks.

The table below lists (by command) the standards for cryptographic algorithms that are supported by ELS.

Table 248. Cryptographic standards

Command	Standard
CIPHER	FIPS-197 (AES) NIST SP 800-38A Block Cipher Modes
AUTH_CIPHER	FIPS-197 (AES) NIST SP 800-38D Galois/Counter Mode (GCM)
ECSIGN	FIPS PUB 186-4. Digital Signature Standard (DSS)
ECVFY	FIPS PUB 186-4. Digital Signature Standard (DSS)
ECKXCH	Internet Engineering Task Force (IETF) RFC8418: Use of the Elliptic Curve Diffie-Hellman Key Agreement Algorithm with X25519 and X448 in the Cryptographic Message Syntax (CMS).
KEYIN(RFC), KEYOUT(RFC)	Network Working Group: RFC3394: AES Key Wrap Algorithm
CKDF	NIST SP 800-108 Key Derivation using pseudorandom functions CMAC based Two-Step Key Derivation as described in NIST SP 800-56C "Recommendation for Key-Derivation Methods in Key-Establishment Schemes"
HKDF	Internet Engineering Task Force (IETF): RFC5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF) hash based option in NIST SP 800-56C "Recommendation for Key-Derivation Methods in Key-Establishment Schemes"
TLS_INIT	Network Working Group: RFC5246: The Transport Layer Security (TLS) Protocol Version 1.2
HASH	FIPS 180-4: Secure Hash Standard (SHS) SHA2-224, 256, 384, 512
HMAC	FIPS 198-1: The Keyed-Hash Message Authentication Code (HMAC)
CMAC	NIST SP 800-38B Block Cipher Modes: CMAC Mode
RND_REQ	NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators

13.3.1 Block diagram

The figure below shows the ELS placed in an SoC system.

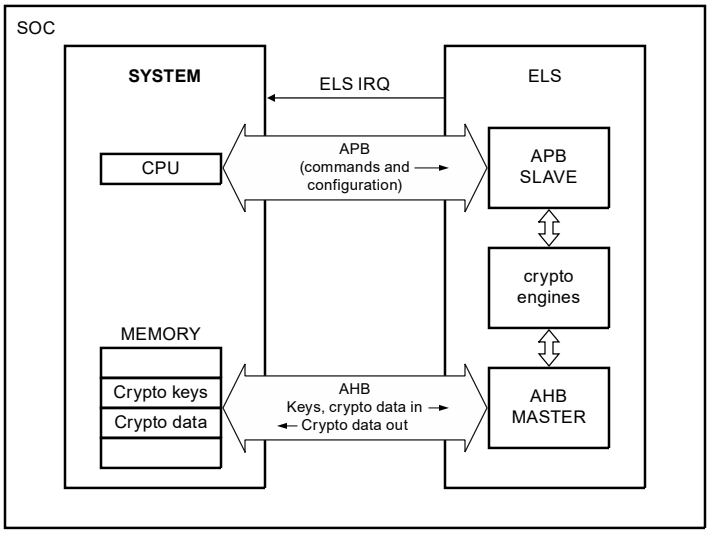


Figure 67. ELS in SOC

The architecture of the ELS is based on a microcode state machine, the "ELS controller" that operates the internal submodules within ELS, to implement complex crypto commands that can combine more than one type of crypto computation (for example hashing plus elliptic curve digital signature).

The internal submodules connect to the controller as peripherals on a private APB bus. As well as being a master of the private APB bus, the controller acts as a slave on the host APB bus, to allow the host to drive ELS.

ELS supports hardware controlled transfer of cryptographic info to and from the host via the DMA Control block. This acts as an AHB master to access host memory. It acts as an APB master on the private bus to access registers in the internal submodules. The DMA control block is also an APB Slave to allow the ELS controller to transfer configuration info to the DMA Control.

NOTE

See chip-specific section to view ELS subsystem diagram.

13.3.2 Features

Following are the features of ELS:

- Crypto Acceleration
  - AES-128/192/256 ECB, CBC, CTR, GCM. AES-192 is available only for external key.
  - SHA-224/256/384/512
  - HMAC SHA-256
  - ECC P-256, ECDSA sign and verify, ECC key generation, ECDHE
  - DRBG + TRNG
  - RFC3394 key wrapping
  - AES-CMAC based KDF
  - HMAC Based KDF
  - TLS 1.2 KDF
  - PRNG SFR Access
- Strong key isolation

  - All keys stored in ELS are hidden from SW
  - Explicit and implicit key access control
  - Temporal and spatial key isolation
- SoC interface
  - DMA access (data)
  - Register APB interface (commands)
- Protection against side-channel attacks and low-cost glitch attacks (EMFI, power supply/clock glitching)

13.4 Functional description

This section describes the user interface to ELS hardware and software.

The section contains:

- Operations: Descriptions of the crypto operations that are supported by ELS.
- Other operational details: Additional Information required by ELS users, such as busy handling, error handling, and a list of limitations.

13.4.1 ELS operation flow chart

The below figure shows operation of ELS embedded in a system.

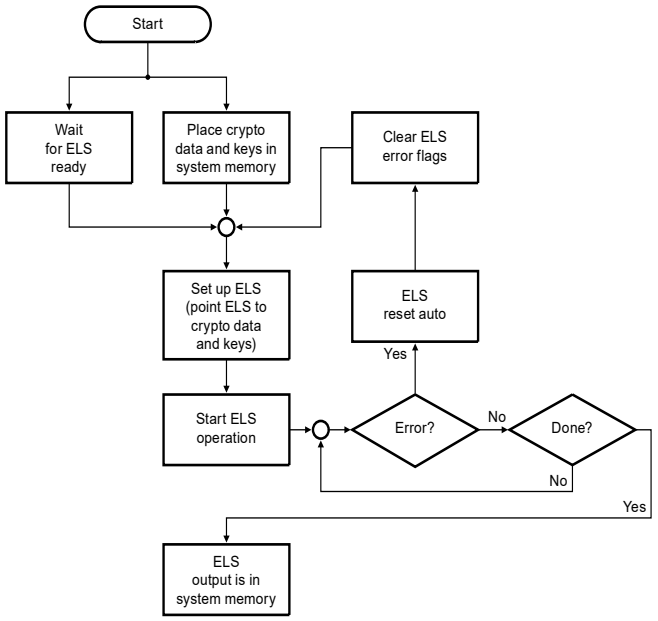


Figure 68. ELS operation flowchart

13.4.2 Commands overview

The primary user interface to ELS is via the ELS commands. These are listed in the table below.

Table 249. List of commands

Command	COMMAND ID (decimal)	Description
CIPHER	0	Runs a block cipher within one of the cipher modes, ECB, CBC, CTR as described in NIST SP 800-38a "BLOCK CIPHER MODES". The input is a message consisting of 1 or more "blocks" of plaintext or ciphertext data in system memory, and the output is an encrypted or decrypted version of the input. In addition a "cipher mode" can be specified: This means additional processing of either or both the output or the input blocks to provide additional security properties compared to the base algorithm.
AUTH_CIPHER	1	Runs an authenticated block cipher as described in NIST SP 800-38d_topic_is_GCM_BLOCK_CIPHER_MODE. In the ELS implementation the underlying encryption algorithm is the AES standard. Input is a message consisting of 1 or more "blocks" of plaintext or ciphertext data in system memory. Outputs are: 1. an encrypted or decrypted version ciphertext message 2. a message authentication code (MAC)
ECSIGN	4	Runs an ECDSA P-256 sign operation.
ECVFY	5	Runs an ECDSA P-256 signature verify operation.
ECKXCH	6	Runs a ECDSA P-256 diffie-hellman key exchange calculation to create a shared secret.
KEYGEN	8	Creates an asymmetric key pair: The 256 bit private key is generated from DRBG, or supplied from the key material of another internal key. The 256 bit public key is generated from the private key using ECDSA P256 key generation algorithm.
KEYIN	9	Loads a key from external storage to an ELS internal key register. The key: • may be loaded via a dedicated hardware interface from a PUF that is external to ELS • can be unpacked from an RFC3394 wrapped key stored in system memory • can be a public key, which is imported from a data structure in system memory that contains the public key
KEYOUT	10	"Wraps" (encrypts) an ELS internal key using the RFC3394 algorithm and stores the encrypted key in system memory.

Table continues on the next page...

Table 249. List of commands (continued)

Command	COMMAND ID (decimal)	Description
KDELETE	11	Deletes a key from ELS internal keystore. The purpose is to free up keystore memory.
CKDF	16	Derives (creates) a new key from an ELS master key and stores the new key in ELS internal keystore. CKDF uses the NIST SP 800-108 algorithm to create the new Key. Specifically NIST SP 800-108 with KDF (Key Derivation Function) as counter mode, and PRF (pseudo random) as AES-CMAC (NIST SP 800-38B).
HKDF	17	Derives (creates) a new key from an ELS master key and stores the new key in ELS internal keystore. HKDF uses either the RFC5869 algorithm or the hash based option in the NIST SP 800-56C algorithm to create the new key.
TLS_INIT	18	Creates session keys as defined by the TLS 1.2 standard. To do this TLS_INIT supports 2 phases, which must each be run as a separate call to TLS_INIT: Create the TLS master Secret Key from the TLS Pre master secret key Create the TLS session keys from the TLS master secret key
HASH	20	Hashes a message using the NIST FIPS 180-4 (SHA2) standard.
HMAC	21	Creates an authenticated message hash as defined by NIST FIPS 198-1
CMAC	22	Creates an authenticated message hash as defined by NIST SP 800-38b: "Recommendation for Block Cipher Operation:The CMAC Mode for Authentication".
RND_REQ	24	Creates either a block of Pseudo Random Data as defined by NIST SP 800-90Ar1 or a block of raw random data that is output by the ELS TRNG.
DRBG_TEST	25	This command is provided to support FIPS CAVP testing. It has the following modes. • 0 = DRBG_instantiate: Instantiates the DRBG using 256 bits of entropy fetched from system memory • 1 = DRBG_extract: Extracts random data from the DRBG after it has been instantiated. • 2 = AES CTR test: Allows AES counter mode of the DRBG to be used to encrypt data provided by the system, • 3 = AES ECB test: Allows the AES ECB mode of the DRBG to be used to encrypt data provided by the system.
DTRNG_CFG_LOAD	28	Loads a prepared TRNG configuration to properly configure the ELS internal TRNG.

Starting a command

Commands are started by setting register bit CTRL[ELS_START], while a valid command is selected. The recommended procedure is described below.

1. Set up the command parameters: Refer to the parameter list for each command in Commands overview to see the list of registers that need to be configured for each of the ELS commands.

   NOTE

   All ELS command parameters retain their values between commands and so must be explicitly set up before each new command is run.

2. Select the command by writing CTRL[ELS_CMD].
3. Start the command by setting CTRL[ELS_START].

   NOTE

   Steps ELS_CMD and ELS_START can be set simultaneously by a single write to the CTRL register.

Starting a command with ID that is not supported is not permitted and will trigger an ELS OPN (operation) error.

Command specific parameters: CMDCFG0 register and its bit assignments

Some parameters are specific to one command or to a small number of commands. To save resources, those parameters are grouped in a single register, CMDCFG0. The fields of CMDCFG0 have different meaning depending on what command is running. This approach is safe because only one ELS command can be run at a time. Refer to the detailed description for each command in the following section for the encodings for each of the parameters.

Detailed information about individual ELS commands is provided in the subsequent sections.

13.4.2.1 CIPHER command

Introduction

CIPHER encrypts or decrypts a text message. Cipher supports the following block cipher modes of operation as described in NIST SP 800-38a "BLOCK CIPHER MODES"

1. ECB - electronic code book
2. CBC - cipher block chaining
3. CTR - counter mode

CIPHER supports Partial processing.

Keys and data

CIPHER requires a key. The key can be provided by either (1) a keystore key or (2) a plaintext key held in system memory. Parameter EXTKEY selects between these 2 options:

1. EXTKEY = 0: use a Keystore Key
   a. The Key to be used is specified by keystore index KIDX0.
   b. The command uses the key size that is specified in the key properties for the specified key. The supported key sizes are 128 bits and 256 bits.
2. EXTKEY = 1: use an External Key
   a. The DMA_SRC2 parameter points to the start address in system memory of the key.

b. The DMA_SRC2_LEN parameter indicates the length in bytes of the key. Supported key lengths are 16 (128 bits), 24 (192 bits), 32 (256 bits).

The input message must be located in system memory. The start address and size in bytes of the message are defined by registers DMA_SRC0 and DMA_SRC0_LEN respectively.

The result will be stored in the system memory range specified by DMA_RES0 (start address) and DMA_RES0_LEN (size in bytes).

CIPHER MODES

1. ECB MODE

   No additional switches are needed for ECB. Refer to the partial programming section below.

2. CBC MODE

   CBC requires a 128 bit initialization vector, and that is fetched from the external memory location referenced by parameter DMA_SRC1.

3. CTR MODE

   CTR requires a 128 bit initial counter value, and that is fetched from the external memory location referenced by parameter DMA_SRC1.

CIPHER example sequence

1. Data preparation
   a. An AES key of the required size should exist in keystore, or a plaintext AES key should exist in system memory.
   b. The input message should exist in system memory.
   c. For CBC mode, the initialization vector IV should exist in system memory, and for CTR mode, the initial counter value should exist in system memory.
   d. Free space to hold the output message, should exist in system memory.
2. Set the command parameters and start the command.
   a. The keystore index of a keystore key, or start address in system memory of a plaintext external key, and set EXTKEY = 1.
   b. The start addresses in system memory of the input message.
   c. The start address in system memory of the output message.
   d. If CBC mode, the start address in system memory of the initialization vector IV.
   e. If CTR mode, the start address in system memory of the initial counter value.

   NOTE

   See Partial processing below for descriptions of additional parameters required for partial mode.

3. After completion of the command, the output data will be available in system memory.

CIPHER parameters

Table 250. CIPHER parameters

Parameter	Register	Description
SRC0_ADDR	DMA_SRC0	Start address in system memory of the complete message or "message chunk" (partial mode) to be processed. This parameter is required for all CIPHER block modes.
SRC0_LEN	DMA_SRC0_LEN	Size in bytes of the message chunk to be processed The size must be an integer number of AES blocks, that is, a multiple of 16 bytes. Its size is limited by the size of SRC0_LEN and must be between 16 bytes and (2^32)-16 bytes • Specifying a size less than 16 bytes will trigger an ELS OPN error. • For sizes 16 or greater, ELS will round down the size to a multiple of 16 bytes before use.
SRC1_ADDR	DMA_SRC1	State information: This is used when the selected cipher mode includes the use of state information (CBC or CTR mode). For either mode (CBC or CTR), the state information is always 16 bytes (128 bits) in length. Therefore, there is no parameter to specify the state information length. For the first block of any message, this is used to input the initialization vector (CBC mode) or initial counter value (CTR mode). When a message is being processed in partial mode (CPHSOE = 1), and state input/output is required, this should point to the state information that was output with the previously processed chunk of the message. At the end of processing the chunk, this pointer will be used to output the state information value at the end of the command. Partial mode means the message is split into chunks with each chunk being processed by a separate call to CIPHER.
SRC2_ADDR	DMA_SRC2	If EXTKEY = 1 start address in system memory of external key, else ignored. **NOTE** This address must be 32-bit word aligned. Unaligned address will trigger an ELS OPN error.
SRC2_LEN	DMA_SRC2_LEN	If EXTKEY = 1 length in bytes of external key. **NOTE** The length must be one of 16, 24, 32. Any other value will trigger an ELS OPN error.
RES0_ADDR	DMA_RES0	Start address in system memory where the output message will be stored.
KIDX0	KIDX0	Index (in ELS keystore) of the key used to encrypt or decrypt the message. Only used if EXTKEY = 0.
DCRPT	CMDCFG0[1]	Encrypt/decrypt: 0 = encrypt, 1 = decrypt
CPHRMDE	CMDCFG0[3:2]	Block cipher mode per NIST.SP.800-38a "Block Cipher Modes". The following are supported: 0 = ECB Electronic Code Book 1 = CBC Cipher Block Chaining 2 = CTR Counter Mode

Table continues on the next page...

Table 250. CIPHER parameters (continued)

Parameter	Register	Description
CPHSOE	CMDCFG0[4]	Cipher State Out enable: This switch supports partial processing: Each call to cipher command processes a subsection ("chunk") of the full message. 0 = Cipher State out disable: Do not output Cipher state information to system memory at the end of the cipher command. 1 = Cipher State out enable: Output cipher state information to system memory at the end of the cipher command.
EXTKEY	CMDCFG0[13]	0 = use internal key from ELS keystore 1 = use external (plaintext) key from system memory

Data structures

ELS handles endianness differently for different data objects and different commands. For a description, see Endianness.

Table 251. CIPHER data structures

Object	Description
Key	If a keystore key is used the key size will be either 128 or 256 bits. See Keystore keys for information about how the keystore is organized. If an external key is used, the key is located in system memory and can be 128, 192, or 256 bits, where the SRC2_LEN parameter determines the size of the key.
Initial value IV (CBC mode)	The Initial value IV is located in system memory. Its size is fixed at 128 bits
Initial counter value (CTR mode)	The Initial Counter value is located in system memory. Its size is fixed at 128 bits.
Input message	The Input message is located in system memory. See SRC0_LEN description in CIPHER parameters for a description of possible sizes.
Output message	The output message will be the same size as the input message. It is the same size of the input length ELS uses which isn't necessarily the same as SRC0_LEN (if a non-block aligned length is provided).
Partial processing state	The state of the calculation may be either read from or written to system memory, for use cases where partial message processing is enabled. In all cases the state is a fixed size 128 bits, that is, 1 AES block.

Partial processing

Cipher command supports partial processing. See section Partial processing

Table 252. Cipher partial processing

Mode	Description
ECB	No additional switches are needed for ECB. If an ECB input message is split into chunks for partial processing, each chunk can be resumed simply by setting the message input, output pointers to the end of the previous chunk.
CBC ENCRYPT/CBC DECRYPT	CPHSOE is not used for CBC mode, for the reasons given below. ELS ignores CPHSOE if it is set in CBC mode. • In this mode, the "state in" for the first block of a chunk is the last ciphertext **OUT or IN** of the previous chunk: Or, in the case of the first block of the first chunk, the "state in" will be the IV. In either case, the "state in" is available in system memory, and will be loaded from system memory in both cases. • Since "state out" is last ciphertext OUT or IN, no "state out" enable is needed, and there is no explicit "state out" pointer. • The "state in" pointer is DMA_SRC1. For the first message chunk this should point to the initialization vector. For all other chunks, this should point to the last ciphertext OUT or IN of the previous chunk.
CTR	State input: for all chunks of the message, state is loaded from system memory at the start of the command before ELS starts to process the message chunk. 1. For the 1st chunk the state loaded is the initial counter value. 2. For subsequent chunks the state loaded is the current counter value. State output: is controlled by switch CPHSOE • CPHSOE = 0: At the end of processing the chunk **do not** output the next state to system memory. **NOTE** CPHSOE should be set to 0 for the last chunk of a partial message: state output is not required in that case. • CPHSOE = 1: At the end of processing the chunk **do** output the next state to system memory. **NOTE** CPHSOE must be set to 1 for all except the last chunk of a partial message. ELS calculates the next state by taking the 128-bit input state and incrementing it by 1.

13.4.2.2 AUTH_CIPHER command

Introduction

AUTH_CIPHER does an authenticated encrypt or decrypt of a text message as described in NIST standard SP.800-38d "Recommendations for Block Cipher Modes of operation: Galois/Counter Mode (GCM) and GMAC"

For implementation reasons, AUTH_CIPHER splits the full authenticated cipher operation into 4 stages. Each stage requires a separate call to AUTH_CIPHER. This means that to complete a full authenticated cipher operation requires 4 calls to AUTH_CIPHER.

A summary of the sequence is given below.

1. Prepare the input data in system memory: Input data consists of the Initial Value (IV), the Auxiliary Authentication Data (AAD), and the input text message.

2. Set command parameters that are common to all stages.
3. INITIALIZE stage: Process the initialization vector IV to create the starting counter state J0, by calling AUTH_CIPHER with ACPMOD = INITIALIZE.
4. Process AAD stage: Process the AAD to create the starting tag value by calling AUTH_CIPHER with ACPMOD=AADPROC.
5. Process Message stage: Process the message, outputting the processed text, and updating the tag, by calling AUTH_CIPHER with ACPMOD=MSGPROC.
6. FINALIZE stage: Perform additional processing to create the final tag, by calling AUTH_CIPHER with ACPMODE=FINALIZE.

NOTE

Each of the stages above prepares data for the following stage. The data is output to system memory as Authenticated Cipher state.

If there is a requirement to run other ELS commands between the stages, then the Authenticated Cipher State in system memory must be preserved until the next stage of authenticated cipher is run. In addition, the key must be preserved between stages. The user is responsible for preserving these.

AUTH_CIPHER supports partial message processing as described in section Partial processing.

AUTH_CIPHER example sequence

1. Data preparation
   a. An AES key of the required size should exist in keystore, or a plaintext AES key should exist in system memory.

   NOTE

   The Key must be present during each of the 4 stages (INITIALIZE stage - FINALIZE stage) described below.

   b. The initialization value IV should be padded as described in the standard and placed in system memory. See Padding of IV below.
   c. The auxiliary authentication data AAD should be padded as described in the standard and place it in system memory.
   d. Pad the last block of the input message with binary zeros, to block size. This is required by the hardware implementation.
   e. The data LEN(A) || LEN(C) should be prepared as described in the standard and placed in system memory.
   f. A fixed space in system memory should be allocated to hold ACPSTATE. See Data structures.
2. Set common parameters that are unchanged during all 4 stages.
   a. If using a keystore key, set EXTKEY = 0, and set KIDX0 to the index of the key.

      If using a plaintext key from system memory, set EXTKEY = 1, set SRC2_ADDR to the start address in system memory of the plaintext key, and set SRC2_LEN to the length in bytes of the key.
   b. Set DCRPT = 0 for encryption, DCRPT = 1 for decryption.
   c. Set SRC1_ADDR to the start address in system memory of ACPSTATE.
3. INITIALIZE STAGE: Initialize the counter, process the initialization vector IV to create the starting counter state J0:
   a. Set the command parameters and start the command.
      i. Set switch ACPMOD = INITIALIZE, ACPSIE = 0.
      ii. Set SRC0_ADDR to the start address in system memory of the padded Initialization vector IV, and SRC0_LEN to the length in bytes of the padded IV.
   b. After completion of the command, the output message will be available in system memory. ELS will hold the initial counter value J0 internally for the next stage.

4. Process AAD Stage: Process the AAD to create the starting tag.
    a. Set the command parameters and start the command.
        i. Set switch ACPMOD=AADPROC.
        ii. Set SRC0_ADDR to the start address in system memory of the padded Auxiliary Authentication data AAD, and SRC0_LEN to the length in bytes of the padded AAD.
    b. After completion of the command, the output message will be available in system memory. ELS will hold the tag value after AAD processing internally for the next stage.
5. Process Message Stage: Process the message, output the processed text, and update the tag.
    a. Set the command parameters and start the command.
        i. Set switch ACPMOD=MSGPROC.
        ii. Set SRC0_ADDR to the start address in system memory of the padded message,

            Set SRC0_LEN to the length in bytes of the padded input message. **Note:** SRC0_LEN also determines the length of the output message
        iii. Set RES0_ADDR to the start address in system memory where the output (processed) message will be written.
        iv. Set MSGENDW to indicate the size in bytes of the data component of the last block, that is, before it was padded by the user (a partial last block is supported by the standard).
    b. After completion of the command, the output message will be available in system memory. ELS will update the tag value and hold that internally for the next stage.

NOTE

ELS will pad the last output block with binary zeros up to block size.

6. FINALIZE stage: Finalize the Tag, perform additional processing to create the final tag. AUTH_CIPHER
    a. Set the command parameters and start the command.
        i. Set switch ACPMOD=FINALIZE.
        ii. Set SRC0_ADDR to the start address in system memory of the field LEN(A) || LEN(C). SRC0_LEN does not need to be set because LEN(A) || LEN(C) is 128 bits long in total.
        iii. Set RES0_ADDR to the start address in system memory where the final TAG will be written.
    b. After completion of the command, the output message will be available in system memory. The authenticated cipher is now complete.

NOTE

See Partial processing below for descriptions of additional parameters required for partial mode.

AUTH_CIPHER parameters

Table 253. AUTH_CIPHER parameters

Parameter	Register	Description
ACPMOD[1:0]	CMDCFG0[3:2]	Auth Cipher **stage**, determines the input and output 0: INITIALIZE: (IV -> J0 (stored internal)) 1: AADPROC: process Auxiliary Authenticated Data

Table continues on the next page...

Table 253. AUTH_CIPHER parameters (continued)

Parameter	Register	Description
		2: MSGPROC: Encrypt or Decrypt a message chunk 3: FINALIZE: Create final tag, using intermediate tag, LEN(A) || LEN(C), and J0 as inputs
SRC0_ADDR	DMA_SRC0	The system memory address of the data to process ACPMOD = INITIALIZE: The padded IV ACPMOD = AADPROC: Auxiliary Authentication Data (AAD) ACPMOD = MSGPROC: message chunk. **note:** message chunk must be zero padded to a multiple of 128 bits ACPMOD = FINALIZE: "LEN(A) || LEN(C)" : refer to NIST.SP.800-38d "GCM block cipher mode"
SRC0_LEN	DMA_SRC0_LEN	ACPMOD = INITIALIZE: The length in bytes of the padded IV ACPMOD = AADPROC: Length in bytes of the padded Auxiliary Authenticated Data ACPMOD = MSGPROC: The length in bytes of the complete (padded) plaintext chunk ACPMOD = FINALIZE: Ignored: supplied by ELS because length of Len(A) || Len(C] field is fixed and known
SRC1_ADDR	DMA_SRC1	Start address in system memory of ACPSTATE. ACPSTATE = authenticated cipher state, required to support partial message processing Address to Input ACPSTATE : IGNORED if ACPMOD=Initialize and ACPSIE = 0 Address to which ELS will output ACPSTATE
SRC2_ADDR	DMA_SRC2	If EXTKEY = 1 start address in system memory of external key **NOTE** This address must be 32-bit word aligned. Unaligned address will trigger an ELS OPN error.
SRC2_LEN	DMA_SRC2_LEN	If EXTKEY = 1 length in bytes of external key **NOTE** The length must be one of 16, 24, 32. any other value will trigger an ELS OPN error.
RES0_ADDR	DMA_RES0	If (ACPMOD = MSGPROC): Start address in system memory of the output message else if (ACPMOD = FINALIZE): start address in system memory of the TAG else ignored.
KIDX0	KIDX0	Index (in ELS keystore) of the key used to encrypt or decrypt the message
DCRPT	CMDCFG0[1]	0: encrypt, 1=decrypt
ACPSIE	CMDCFG0[5]	Authenticated Cipher State In Enable:

Table continues on the next page...

Table 253. AUTH_CIPHER parameters (continued)

Parameter	Register	Description
		1. INITIALIZE 0=state is not input at start of INITIALIZE (use when processing the full IV in a single command or processing the first chunk of IV). 1=state is input at start of INITIALIZE (use when processing the IV in chunks, for the second and subsequent chunks of the IV). 2. ADDPROC, MSGPROC, FINALIZE: **APCSIE is ignored. ELS reads ACPSTATE from system memory in all cases.**
MSGENDW [3:0]	CMDCFG0[9:6]	ACPMOD=MSGPROC: Byte width of the last message block. • Assumption: The block will be padded with zeros by software. • ELS command must create the pre-ciphertext by xoring the encrypted counter with the padded plaintext, that is pad the last block with zeroes up to a size of one aes block. • Calculate the padding width = 128 bit - MSGENDW*8, then zero the padding width msbits of the pre-ciphertext to obtain the ciphertext.
LASTINIT	CMDCFG0[10]	LASTINIT handles the following special case from the standard: Section 7.1 "Algorithm for the Authenticated Encryption Function", paragraph "Steps:", excerpt **Define a block, J0, as follows:** **If len(IV) = 96, then let J0 = IV || 031 ||1.** **If len(IV) ≠ 96, then let s = 128 {len(IV)/128}-len(IV), and let J0 = GHASHH(IV || 0s+64 || [len(IV)]64).** Here, '{len(IV)/128}' represents the least integer that is not less than the real number len[IV]/128. In words: Case1: IVs with unpadded size of 96 bits, do not get GHASHED, Case2: IVs with all other unpadded sizes apart from 96 do get GHASHED USER should set LASTINIT as follows to differentiate between case1 and case2 above: If chunk length != 16 bytes (128 bits): chunk cannot possibly represent an IV with unpadded size 96 bits: That is case 2: User does not care about value of LASTINIT, because ELS will ignore it If chunk length = 16 bytes (128 bits) , and chunk contains the entire padded IV, then that is case 1 , i.e. chunk is an entire IV with unpadded length=96 bits, User indicates that by setting LASTINIT to 1 If chunk length = 16 bytes (128 bits) but chunk is only part of an IV, then that cannot be case1, must be case2, because the unpadded size of the full IV must be > 96bits: User indicates that by clearing LASTINIT to 0, ELS interprets LASTINIT as follows: IF (length of IV chunk != 16 bytes (128 bits)): Ignore LASTINIT, and GHASH the chunk ELSE IF ((length of IV chunk = 16 bytes (128 bits)) AND (LASTINIT = 1)) DO NOT GHASH the chunk

Table continues on the next page...

Table 253. AUTH_CIPHER parameters (continued)

Parameter	Register	Description
		ELSE GHASH the chunk **Note:** The final ELSE above handles the case where the IV chunk size is 16 bytes, but the IV chunk is not a complete IV, i.e. is only part of an IV
EXTKEY	CMDCFG0[13]	0=use internal key from ELS keystore 1=use external (plaintext) key from system memory

Data structures

ELS handles endianness differently for different data objects and different commands. For a description, refer to Endianness

Table 254. AUTH_CIPHER data structures

Object	Description
Key	If a keystore key is used the key size will be either 128 or 256 bits. See Keystore keys for information about the KSIZE parameter of a keystore key If an external key is used, the key is located in system memory and can be 128, 192, or 256 bits. AUTH_CIPHER uses the SRC2_LEN parameter to determine the size of the key.
Padded IV	The Padded IV is located in system memory. Its length is limited by the size of SRC0_LEN and must be between 16 bytes and (2^32)-16 bytes. Its size must be a multiple of 128 bits.
Padded AAD	The Padded AAD is located in system memory. Its length is limited by the size of SRC0_LEN and must be between 16 bytes and (2^32)-16 bytes. Its size must be a multiple of 128 bits.
Padded input message	The Padded input message is located in system memory. Its length is limited by the size of SRC0_LEN and must be between 16 bytes and (2^32)-16 bytes. Its size must be a multiple of 128 bits.
LEN(A) || LEN(C)	This should be located in system memory. It is a fixed size, 128 bits
ACPSTATE	If ACPMOD != FINALIZE, a block of system memory must be freed for it. The size of ACPSTATE is 512 bits (16 32-bit word)

Padding of IV

The standard describes the padding of IV as follows:

1. if len(IV) = 96 : Padded IV = IV || 0^31 ||1

   in words: pad IV with 31 binary 0's followed by a binary 1
2. if len(IV) != 96

   padded IV = (IV || 0^s+64 || [len(IV)]64, where s = (128 * round_up (len(IV)/128) - len(IV),

   in words:

   a. Find the length (bit width) of IV, and calculate how many bits that is short of being a multiple of 128. The answer is "s"

   b. Pad IV by (s+64) binary zeros, then pad the result of that by a 64 bit representation of the length of IV

Partial processing

AUTH_CIPHER command supports partial processing. See section Partial processing.

1. **Definition of a sub-operation**. The term sub-operation is used in the descriptions below: **a "sub-operation", is any call to AUTH_CIPHER**

   NOTE

   This applies to sub-operations within a stage, (for example, a message processing sub-operation followed by another message processing sub-operation).

   NOTE

   It also applies to sub-operations on the boundary of 2 stages (for example, a message processing stage sub-operation followed by a FINALIZE stage sub-operation).

2. ACPSIE
   a. ACPMODE = INITIALIZE

      ACPSIE = 0: State is not input at start of INIITIALIZE. Use to process the full IV in a single command (full processing), or, when processing the IV in chunks (partial processing) to process the first chunk of IV).

      ACPSIE = 1: State is input at start of command. Use when processing the IV in chunks (partial processing), for the second and subsequent chunks of the IV.

   b. ACPMOD = AADPROC, MSGPROC, FINALIZE. ACPSIE is ignored.

   NOTE

   AUTH_CIPHER can be completed without any partial processing. In this case, 4 calls to AUTH_CIPHER are required, one for each stage, and ACPSIE must be set to 0 for ACPMOD=INITIALIZE stage.

Limitations

- Tag comparison is not supported. AUTH_CIPHER only generates the authentication tag, does not compare it. The user is responsible for comparing the tag.
- Tag generation supports only a 128-bit wide tag.

13.4.2.3 ECSIGN command

Introduction

The ECSIGN command generates an ECC P-256 digital signature. To do that it calculates the signature over a challenge that has been provided by the caller, and using an ECC signing key, from ELS keystore.

ECSIGN consists of the following sub-operations:

1. Read the challenge (text message) from system memory, and hash it if required (the challenge may be pre-hashed, in which case hashing is not required).
2. Use the ECC signing key to compute the signature over the hash of the challenge.
3. Write the signature to system memory.

ECSIGN example sequence: not signing Run Time Fingerprint (RTF)

1. User prepares the input Message.

   If (ECHASHCHL = 0) (do not "hash the challenge"), the input message should be hashed, and the resulting digest placed in system memory.

if (ECHASHCHL = 1) ("hash the challenge)", the **padded** message must be placed in system memory. Standard SHA2-256 padding is required.

2. User sets the command parameters (SIGNRTF must = 0) and starts ECSIGN.
3. ECSIGN completes. The signature is output to system memory.

ECSIGN example sequence: signing RTF

NOTE

See Run Time Fingerprint (RTF)

1. User prepares input Message:
   a. The unhashed and padded input Message must be placed in system memory.

      Padding for RTF: the message will actually be signed is the concatenation of RTF, Hardware Attestation Data (HAD) and the input message: **concatenation = RTF || HAD || Input Message**

      To allow for the above, padding for a message of size RTF || HAD || Input Message should be added to the input message.
2. User prepares run time fingerprint (RTF): only required if signing RTF SIGNRTF = 1.
3. Preparation of HAD.

   Hardware attestation data is a block of data in system memory that reflects the state of the system hardware. See Data structures section below. Refer chip-specific section to get details on hardware attestation data.
4. User sets the command parameters (SIGNRTF must = 1) and starts ECSIGN.
5. ECSIGN completes. The signature is output to system memory.

ECSIGN parameters

Table 255. ECSIGN parameters

Parameter	Register	Description
SRC0_ADDR	DMA_SRC0	Start address of the Digest/Challenge. If ECHASHCHL = 0, Start address in system memory of the digest. The challenge has been pre-hashed, and the digest is the hash of the challenge. If ECHASHCHL = 1, Start address in system memory of the unhashed challenge. In this case, the command will hash the challenge, and use the resulting digest in the ecc sign operation.
SRC0_LEN	DMA_SRC0_LEN	ECHASHCHL = 0: ignored. A digest has fixed length, no need to specify the length. ECHASHCHL = 1: Length in bytes of the unhashed challenge.
RES0_ADDR	DMA_RES0	Start address in system memory where the signature will be stored.
KIDX0	KIDX0	Index in ELS keystore of the key used to sign the message.

Table continues on the next page...

Table 255. ECSIGN parameters (continued)

Parameter	Register	Description
SIGNRTF	CMDCFG0[1]	0=RTF will not be incorporated in the message to be signed (the challenge). 1=RTF will be incorporated into the message to be signed (the challenge). The algorithm used to incorporate RTF is given by the expression below: signature = ECDSA P-256 (signing Key, SHA256(RTF || HAD || message_to_be_signed_with_padding_adjusted_for_sizeof_RTF_plus_HAD)) **NOTE** If SIGNRTF = 1, then at least one of URTF or UECSG must be set on the signing key. This is also described in Key usage rules.
ECHASHCHL	CMDCFG0[0]	Indicates whether or not the challenge has been pre-hashed. If the challenge has not been hashed, the command will Hash the challenge to create a digest and use the digest in the ecc sign operation. 0=challenge is pre-hashed (a digest). The command will use that directly in the ecc sign operation. 1=challenge is not pre-hashed. The command will hash the challenge, and use the resulting digest in the ecc sign operation.
REVF	CMDCFG0[4]	0=The digest and the signature (but not the challenge, see below) are accessed (read/written) with the assumption that they are in 32-bit word little endian format. 1=The digest and the signature (but not the challenge, see below) are accessed (read/written) with the assumption that they are in 32-bit word big endian format. **NOTE** REVF is ignored when reading the challenge. The challenge is always read with the assumption that it is in 32-bit word big endian format.

Table continues on the next page...

Table 255. ECSIGN parameters (continued)

Parameter	Register	Description
		NOTE When the signature R || S, described in Data structures below, is written: 1. R, S, are treated as separate data structures. 2. Each is formatted according to REVF. 3. R is written to the lower part of the system memory address range for R || S and S is written to the higher part. **NOTE** When REVF = 1, the output signature start address RES0_ADDR must be 32-bit word aligned.

Data structures

ELS handles endianness differently for different data objects and different commands. For a description, see Endianness

Table 256. ECSIGN data structures

Object	Description
Key	A 256-bit keystore key must be used. See Keystore keys for information about the KSIZE parameter of a keystore key.
Input message: "the challenge"	If ECSIGN is configured to get the challenge from system memory, the challenge must be placed there by the user. Otherwise no action is required by the user because ELS already holds the challenge internally. If the challenge is to be hashed, its size must be an integer number of SHA-256 blocks. A SHA2-256 block is 512 bits. If the challenge has already been hashed, its size should be 512 bits.
Run Time Fingerprint (RTF)	This is a 256-bit ELS internal register that contains the result of one or more hashes of system memory. When command switch SIGNRTF = 1, RTF is concatenated with the input message, and the hardware attestation data (HAD), and the result is signed. The concatenation formula is **concatenation = RTF || HAD || Input Message**.
Hardware attestation data (HAD)	This is a block of system memory that should contain data that represents the hardware "state" of the system. When command switch SIGNRTF = 1, HAD is concatenated with the input message and RTF, and the result is signed. The concatenation formula is **concatenation = RTF || HAD || Input Message**.

Table continues on the next page...

Table 256. ECSIGN data structures (continued)

Object	Description
Signature	The signature is written to system memory by ECSIGN. Its size is 512 bits, it consists of the concatenation of two 256-bit components, R and S. The concatenation is represented as R || S. If R || S is located in system memory, R should be located at lower addresses and S should be located at higher addresses.

Endianness requirements

See Endianness requirements by data structure.

ECSIGN command operation

The start address and size in bytes of the challenge are defined by registers DMA_SRC0 and DMA_SRC0_LEN respectively.

The private key must be an ECC signing key that has been created via the KEYGEN command. See KEYGEN command.

The output of the command is stored in system memory with the start address specified by DMA_RES0.

Background: Use of digital signature in an authentication scenario

The ECSIGN command generates a digital signature. Typically that would be required as part of a "crypto authentication" scenario where ELS is embedded in a "peripheral" that must authenticate itself to a "host". To do that, the peripheral must hold the private key of an asymmetric key pair, plus a certificate containing the public key of the pair.

The sequence is as follows:

1. The peripheral sends its certificate to the host.

   The host then verifies and unpacks the certificate to get the peripheral's public key. In doing so the host confirms that the peripheral's public key is authentic.
2. The host generates a random number "the challenge" and sends it to the peripheral as plaintext.
3. Optionally, the peripheral creates a random number "the nonce", and concatenates it to the challenge.
4. The peripheral signs the challenge using a standard digital signature algorithm. This step requires the private key. The result is the "response", which the peripheral then returns to the host. If the challenge included a nonce, the peripheral must return both the nonce and the response to the host.
5. The host verifies the response using a standard verification algorithm. If the result is ok, the peripheral has proved it is authentic.

The role of ELS in the above sequence is:

1. Generate the nonce. ELS does this via the RND_REQ command. This is optional.
2. Sign the challenge. ELS does this via the ECSIGN command. ECSIGN performs the operation {r, s} = signature(hash(c)), where:
   a. {r, s} is the signature that is output from ECSIGN.
   b. c is the challenge: a random number that has been received from the host and is located in the memory of the peripheral.
   c. hash is a SHA256 hash.
   d. signature is an ECDSA signature using the NIST P-256 curve.

13.4.2.4 ECVFY command

Introduction

ECVFY command verifies an ECC P-256 digital signature. To do that it needs access to:

- Challenge that was signed to create the signature
- Public key corresponding to the private key that was used to create the signature (the public key can be located in either keystore or system memory)
- Signature

ECVFY example sequence

1. User prepares input data in system memory:
    a. If public key from system memory: concatenation of the digital signature R || S and the public key "QX || QY" as described in Table 257.

       If public key from keystore, the signature R || S as described in Table 257.

    b. Stores either the challenge or the hash of the challenge in system memory.
2. User sets the parameters and starts ECVFY.
3. ECVFY completes. The command has following outputs:
    a. A pass/fail result in status bits VFY_STATUS.
    b. A value that should be same as the R component of the original signature R, S is written to system memory. This is to allow the user to do an external comparison between the input value of R, and the value of R that was re-calculated by the ECVFY command.

ECVFY parameters

Table 257. ECVFY parameters

Parameter	Register	Description
SRC0_ADDR	DMA_SRC0	If ECHASHCHL = 0, Start address in system memory of the digest. The challenge has been pre-hashed, and the digest is the hash of the challenge. If ECHASHCHL = 1, Start address in system memory of the unhashed challenge. In this case the command will hash the challenge, and use the resulting digest in the ECSIGN operation.
SRC0_LEN	DMA_SRC0_LEN	ECHASHCHL = 0: ignored. A digest has fixed length, no need to specify the length. ECHASHCHL = 1: Length in bytes of the unhashed challenge. **NOTE** The unhashed challenge must be padded per SHA-256.
SRC1_ADDR	DMA_SRC1	EXTKEY = 0: start address in system memory of the signature only as described in Table 258.

Table continues on the next page...

Table 257. ECVFY parameters (continued)

Parameter	Register	Description
		EXTKEY = 1: start address in system memory of a structure that contains the concatenation of Signature and Public Key as described in Table 258.
RES0_ADDR	DMA_RES0	Re-calculated value of R. This allows the user to do an external comparison between the R that is input as part of the Signature/Public key concatenation, and the value that is calculated. If the values match, the verify result is "passed". RES0_ADDR is the start address in system memory where the calculated value of R is stored.
KIDX2	KIDX2	EXTKEY = 0: Index in ELS keystore of the input public key. EXTKEY = 1: ignored.
VFY_STATUS	STATUS	Result of the ECVFY command: 2'h0 = verify not run, 2'h1 = ECVFY failed, 2'h2 = ECVFY passed, 2'h3 = Invalid, error
ECHASHCHL	CMDCFG0[0]	Indicates whether or not the challenge has been pre-hashed. If the challenge has not been hashed, the command will Hash the challenge to create a digest and use the digest in the ECC verify operation. 0 = challenge is pre-hashed (a digest). The command will use that directly in the ECC sign operation. 1 = challenge is not pre-hashed. The command will hash the challenge, and use the resulting digest in the ECC verify operation.
EXTKEY	CMDCFG0[13]	Selects whether the public key input to command is taken from system memory or keystore. 0 = Public key is taken from keystore key referenced by KIDX2 (UKPUK property required). 1 = Public key is taken from system memory, referenced by SRC1_ ADDR.
REVF	CMDCFG0[4]	0 = The digest and the signature (but not the challenge, see below) are accessed (read/written) with the assumption that they are in 32-bit word little endian format. 1 = The digest and the signature (but not the challenge, see below) are accessed (read/written) with the assumption that they are in 32-bit word big endian format. **NOTE** REVF is ignored when reading the challenge. The challenge is always read with the assumption that it is in 32-bit word big endian format.

Table continues on the next page...

Table 257. ECVFY parameters (continued)

Parameter	Register	Description
		NOTE When the concatenation of signature and public key R || S || QX || QY or the signature, only R || S (all described in data structures below) is read. • R, S, QX, QY are treated as separate data structures. • Each is formatted according to REVF. • R is written to the lowest part of the system memory address range for R || S || QX || QY, S is written to the next lowest part, and so on. **NOTE** When REVF = 1, the input signature start address SRC1_ADDR must be 32-bit word aligned.

Data structures

ELS handles endianness differently for different data objects and different commands. For a description, see Endianness.

Table 258. ECVFY data structures

Object	Description
Input message: "the challenge"	If ECVFY is configured to get the challenge from system memory, the challenge must be placed there by the user. Otherwise no action is required by the user because ELS already holds the challenge internally. If the challenge is to be hashed, its size must be an integer number of SHA-256 blocks. A SHA2-256 block is 512 bits. If the challenge has already been hashed, its size should be 512 bits.
Concatenation of signature and public key or signature only	EXTKEY = 0: the signature R || S must be placed in system memory by the user. If the signature is represented as a concatenation of fields R || S, then R should be stored at the lowest address in system memory and S should be stored at the highest address. EXTKEY = 1: the signature R || S and the public key QX || QY must be concatenated and placed in system memory by the user. The fields R, S, QX, QY each have size 256 bits (total size of R || S || QX || QY is 1024 bits). If the concatenation is represented as a concatenation of fields R || S || QX || QY, then R should be stored at the lowest address in system memory and QY should be stored at the highest address. ENDIANNESS: Refer to the definition of endianness in section Endianness. Within each field R, S, QX, QY:

Table continues on the next page...

Table 258. ECVFY data structures (continued)

Object	Description
	• Register field BYTE_ORDER controls the byte endianness when the field is read/written to system memory. • Command parameter REVF controls the 32-bit word endianness when the field is read/written to system memory.
Output: the re-calculated value R.	As stated above, R has size 256 bits.

Endianness requirements

Refer to Endianness requirements by data structure.

13.4.2.5 ECKXCH command

Introduction

ECKXCH runs an ECC P-256 Diffie-Hellman key exchange to create a shared secret. To do that it needs access to:

- A private key
- The public key of a third party (the public key can be located in either keystore or system memory)

ECKXCH example sequence

1. User prepares the input data.
   a. If public key from system memory: public key "QX || QY" as described in Data structures must be placed in system memory.

      If public key from keystore, the public key must have been created in keystore by previous ELS operations (see KEYIN).

   b. The private key is created in keystore (see KEYGEN command).
2. User sets the command parameters, and starts ECKXCH.
3. ECKXCH completes. The command outputs the shared secret to ELS keystore.

NOTE

After command completion, the input private key remains in Keystore.

There is a security requirement that the input private key should be used only once. It is the user's responsibility to enforce that. KDELETE command can be used to delete the key.

ECKXCH parameters

Table 259. ECKXCH parameters

Parameter	Register	Description
SRC1_ADDR	DMA_SRC1	EXTKEY = 0: ignored. EXTKEY = 1: start address in system memory of the Public Key of the 3rd party.

Table continues on the next page...

Table 259. ECKXCH parameters (continued)

Parameter	Register	Description
KIDX0	KIDX0	Index in ELS keystore of the input Private key.
KIDX1	KIDX1	Index in ELS keystore of the output shared secret.
KIDX2	KIDX2	EXTKEY = 0: Index in ELS keystore of the input public key. EXTKEY = 1: ignored.
KPROPIN	KPROPIN	Requested properties of the shared secret that is output to ELS keystore. **NOTE** The bit positions of KPROPIN map to the list of key properties that is defined in Keystore key properties and status. All properties may be set, with the following exceptions: • For setting of usage properties, refer to Key creation rules: Setting the usage control properties • ECKXCH creates a fixed size (256 bits) key: Fields KSIZE, KBASE, KACT are set automatically by ELS, not copied from KPROPIN.
REVF	CMDCFG0[4]	0=data structures are accessed (read/written) with the assumption that they are in 32-bit word little endian format. 1=data structures are accessed (read/written) with the assumption that they are in 32-bit word big endian format. **NOTE** If the public key is located in system memory, when "QX || QY" is read: 1. QX, QY are treated as separate data structures. 2. Each is formatted according to REVF. 3. QX is written to the lower part of the system memory address range for QX || QY, and QY is written to the higher part. **NOTE** When REVF = 1, the input public key start address SRC1_ADDR must be 32-bit word aligned.
EXTKEY	CMDCFG0[13]	Selects whether the public key input to command is taken from system memory or keystore. 0=Public key is taken from keystore key referenced by KIDX2 (UKPUK property required). 1=Public key is taken from system memory, referenced by SRC1_ ADDR.

Key properties that are set automatically by ECKXCH

Properties KSIZE, KBASE, and KACT are set automatically by the command. All other properties are set from KPROPIN. See Keys for more information.

Data structures

For a description of how ELS handles endianness, refer to Endianness

Table 260. ECKXCH data structures

Object	Description
Input private key	A 256-bit keystore key must be used. Refer to Keystore keys for information about the KSIZE parameter of a keystore key.
Input public key	EXTKEY = 0: the public key must have been created in keystore by previous ELS operations (see KEYIN). EXTKEY = 1: the input public key placed in system memory by the user. The public key is represented as the concatenation of 256 bit fields "QX || QY". QX should be stored at the lowest address in system memory and QY should be stored at the highest address. ENDIANNESS: Refer to the definition of endianness in section Endianness Within each field QX, QY: 1. Register field BYTE_ORDER controls the byte endianness when the field is read/written to system memory. 2. Command parameter REVF controls the 32-bit word endianness when the field is read/written to system memory.
Output shared secret	ECKXCH will output a 256-bit key to keystore. The size property of the key will be set automatically by the command.

13.4.2.6 KEYGEN command

Introduction

KEYGEN creates a public and private key pair for use in ECC operations. The private key is written to keystore, and the public key is output to system memory.

Optionally, KEYGEN can also sign the public part of the generated key with a Key Signing key. The purpose of signing with a Key Signing key is to allow authentication that the Public key was actually generated by an trustworthy source. In this case KEYGEN uses the NIST P-256 curve to generate the key pair.

KEYGEN example sequence

1. User prepares the input data.
    a. If the Key Material for the new key will be taken from an existing keystore key, the user must first have created that source key.
2. User sets the command parameters and starts KEYGEN. KEYGEN has switches to select what type of key is created and what the source of the key material is.
3. KEYGEN completes. The command outputs the new key to keystore.

KEYGEN parameters

Table 261. KEYGEN parameters

Parameter	Register	Description
SRC0_ADDR	DMA_SRC0	KGSIGN_RND = 0: ignored

Table continues on the next page...

Table 261. KEYGEN parameters (continued)

Parameter	Register	Description
		KGSIGN_RND = 1: start address in system memory of the "ClientHello.random + ServerHello.random" part of the Server Key Exchange structure. **NOTE** Parameter SRC0_LEN is not required, because the structure is a fixed size, 512 bits.
RES0_ADDR	DMA_RES0	Start address in system memory of the output public key. If REVF = 1 is enabled, RES0_ADDR must be 32 bits aligned (that is, must be a multiple of 4). If this condition is not met, an ELS OPN error will be triggered. **NOTE** Ignored if no public key is to be output. Refer to command parameter SKIP_PBK below.
KIDX0	KIDX0	Keystore index of the private key that is created by KEYGEN, **AND** If (KGSRC=internal) **AND** (KGTYPEDH = 0): keystore index of the key that provides the source Key Material. **NOTE** In this case (creating a signing key with Key Material taken from keystore) the source Key Material will be overwritten by the output private key.
KIDX1	KIDX1	If (KSGIGN = 1) Keystore Index of the Key Signing Key Else: ignored
KGSRC	CMDCFG0[2]	Source of the Key Material 1. if(KGTYPEDH = 0) (output private key is a signing key) a. if (KGSRC = 0) Key Material source will be an Internal (Keystore) Key): this creates a deterministic output private key. b. If (KGSRC = 1) Key Material source will be random number from ELS DRBG: This creates a fully random output private key. 2. else if (KGTYPEDH = 1) (output private key is a Diffie Helman key for use in ECKXCH) KGSRC is ignored, Key Material source is always a random number from ELS DRBG.
KGTYPEDH	CMDCFG0[1]	0 = output private key can be used as input for ECSIGN. 1 = output private key can be used as input for ECKXCH. **NOTE** When a key is created for use by ECKXCH, the value of the key is always set to an odd number.
KGSIGN	CMDCFG0[0]	0 = do not sign the output public key. 1 = sign the output public key. Signature will be concatenated to the output public key.

Table continues on the next page...

Table 261. KEYGEN parameters (continued)

Parameter	Register	Description
		NOTE If KGSIGN = 1, then either UECSG or UKSK must be set on the signing key. This is also described in Key usage rules.
SKIP_PBK	CMDCFG0[3]	0 = generate and output, a public key. 1 = skip public key generation and don't output a public key. **NOTE** If KGTYPEDH = 1, then SKIP_PBK is ignored, public key is always output when KGTYPEDH = 1. If KGSIGN = 1 and SKIP_PBK = 1, KGSIGN is ignored.
KPROPIN	KPROPIN	Requested properties of the output private key. **NOTE** The bit positions of KPROPIN map to the list of key properties that is defined in Keystore key properties and status. The only fields that are copied to the output private key are: • UPPROT • UTECDH: copied only when KGTYPEDH = 1, otherwise ignored. All other fields of KPROPIN are ignored. Refer to Key properties that are set automatically by KEYGEN.
REVF	CMDCFG0[4]	0 = data structures are accessed (read/written) with the assumption that they are in 32-bit word little endian format. 1 = data structures are accessed (read/written) with the assumption that they are in 32-bit word big endian format. If REVF = 1 is enabled, RES0_ADDR must be 32 bits aligned (that is, must be a multiple of 4). If this condition is not met, an ELS OPN error will be triggered. **NOTE** When the concatenation of signature and public key QX || QY || R || S (described in "KEYGEN data structures-output public key" below) is read: 1. QX, QY, R, S are treated as separate data structures. 2. Each of QX, QY, R, S, is processed by ELS according to the value of REVF. 3. QX is written to the lowest part of the system memory address range for QX || QY || R || S, QY is written to the next lowest part, R to the next lowest part, and S to the top part. **NOTE** REVF is ignored when reading the challenge, or digest. These are always read with the assumption that they are in 32-bit word big endian format.

Table continues on the next page...

Table 261. KEYGEN parameters (continued)

Parameter	Register	Description
		NOTE When REVF = 1, the output public key start address RES0_ADDR must be 32-bit word aligned.
KGSIGN_RND	CMDCFG0[5]	See Signing the generated key. 0=Do not include user provided client and server random data in the structure that is hashed as part of public key signing. 1=Include user provided client and server random data in the structure that is hashed as part of public key signing.

Key properties that are set automatically by KEYGEN

UPPROT is copied from KPROPIN, it is not set automatically by the command. UTECDH is also copied from KPROPIN. All other fields are controlled by the command; the corresponding fields of KPROPIN are ignored.

Table 262. Key properties that are set automatically by KEYGEN

Key property	How that is set on the output private key
URTF UKSK	If (KGSRC = 0 (internal)) AND (KGTYPEDH = 0 (signing key requested)) then URTF and UKSK will be copied from the input key that is the source of the Key Material. Else, URTF and UKSK are cleared to 0.
UECSG	if (KGTYPEDH = 0 (signing Key requested)), set to 1, else cleared to 0.
UECDH	if (KGTYPEDH = 1 (DH key requested)), set to 1, else cleared to 0.
WRPOK	Set to 1 in all cases.

Data structures

For a description of how ELS handles endianness, refer to Endianness.

Table 263. KEYGEN data structures

Object	Description
Input Key Material	If the Key Material is being supplied from deterministic Key Material, a 256-bit keystore key must be used. See Keystore keys for information about the KSIZE parameter of a keystore key. If the Key Material is being supplied from random Key Material, no preparation for that is needed.
Output private key	KEYGEN will output a 256-bit key to keystore.
Output public key	Optionally KEYGEN will output a 512 bit public key {QX, QY} to system memory. Refer to command parameter SKIP_PBK above.

Table continues on the next page...

Table 263. KEYGEN data structures (continued)

Object	Description
	In addition, if signing of the public key is requested (KGSIGN = 1), KEYGEN will append a 512 bit signature {R, S} to the public key in system memory. If the concatenation is represented as a concatenation of fields QX || QY || R || S, then QX will be written to the lowest address in system memory and S will be written to the highest address. ENDIANNESS: See the definition of endianness in section Endianness. Within each field QX, QY, R, S: 1. Register field BYTE_ORDER controls the byte endianness when the field is read/written to system memory. 2. Command parameter REVF controls the 32-bit word endianness when the field is read/written to system memory.

Using KEYGEN for provisioning

KEYGEN is intended to be used as part of a "trust provisioning" scenario where ELS is embedded in a "peripheral" that is being provisioned (provisioned means loaded with a public and private key pair, that will later be used to authenticate the peripheral.), However, the assumption is that the provisioning is taking place in an insecure environment. The environment is not trusted to provide the key pair, therefore ELS must generate the key pair itself, and obfuscate the private key part before storing it in the system memory of the peripheral.

It is also possible that an "insecure environment" mentioned above could generate a fake public key and store it in ELS. The optional key signing step of KEYGEN that happens when KGSIGN = 1, is provided to protect against that case. It uses a Root Key to create a signature for the Public Key to "prove" that the public key was actually created by ELS.

NOTE

The trust provisioning scenario normally includes embedding the public key that was mentioned above, in a certificate. That part of the scenario is not supported by ELS.

Signing the generated key

If the command switch KGSIGN is set, KEYGEN will sign the public part of the created key with a Key Signing Key

A Key signing key is a key with either UKSK = 1 or UECSG = 1. Refer to Key usage rules.

How the public key is signed:

1. When the ELS signs a generated Public Key (KEYGEN parameter KGSIGN = 1), the public key is hashed in a data structure as per RFC8422 (https://tools.ietf.org/html/rfc8422). This consists of adding constant data to the Public key to denote the curve type and named curve as detailed below:

```
/* RFC8422 Data Structure */
/* struct { */
/* uint8 curve_type = 0x03; //named_curve (3) */
/* uint16 NamedCurve = 0x0017; //secp256r1 (23) */
/* ECPoint public; */
/* } ServerECDHParams; */
/* uint8 PubKeyLength = 0x41 // 65 total bytes for Pub Key */
/* uint ECpointformat = 0x04 // Uncompressed */
/* Qx Big Endian */
/* Qy Big Endian */
```

2. Additionally, if the KEYGEN config switch KGSIGN_RND is set (KEYGEN parameter KGSIGN_RND = 1) then the Public Key Data structure is prepended with data provided from system memory.

   This is to support the TLS.12 ServerKeyExhange protocol as described in https://tools.ietf.org/html/rfc5246.

   "TLS1.2 Key Signing Structure:

   ServerKeyExchange.signed_params.sha_hash SHA(ClientHello.random + ServerHello.random + ServerKeyExchange.params);

   The data provided by the system represents the "ClientHello.random + ServerHello.random" part of the Server Key Exchange structure.

   This is read by the ELS as 512 bits of data pointed to by DMA_SRC0.

   The length of this data field is fixed at 512 bits and therefore DMA_SRC0_LEN is not required, and its contents are ignored.

   The data structure which is hashed by the ELS to create the Public Key Digest for signing is shown in Table 264 below for clarity.

   The case shown in the table is REVF = 1, BYTE_ORDER = 1 : for other combinations of REVF, BYTE_ORDER, see Public key signing: Data Structures for other combinations of REVF, BYTE_ORDER below

   The initial 512 bits are only included in the Hash Digest if requested (KGSIGN_RND = 1).

   NOTE

   The data structure constants , Public key, and SHA padding are provided internally by ELS.

   The resulting signature will be written to system memory.

Table 264. Public Key digest for Signing: Data structure that is hashed: KGSIGN_RND = 1, REVF = 1, BYTE_ORDER = 1

Word Index	Byte[3]	Byte[2]	Byte[1]	Byte[0]	Description
0	CLIENT_RND[31]	CLIENT_RND[30]	CLIENT_RND[29]	CLIENT_RND[28]	Client Random Data
4	CLIENT_RND[27]	CLIENT_RND[26]	CLIENT_RND[25]	CLIENT_RND[24]	
8	CLIENT_RND[23]	CLIENT_RND[22]	CLIENT_RND[21]	CLIENT_RND[20]	
C	CLIENT_RND[19]	CLIENT_RND[18]	CLIENT_RND[17]	CLIENT_RND[16]	
10	CLIENT_RND[15]	CLIENT_RND[14]	CLIENT_RND[13]	CLIENT_RND[12]	
14	CLIENT_RND[11]	CLIENT_RND[10]	CLIENT_RND[9]	CLIENT_RND[8]	
18	CLIENT_RND[7]	CLIENT_RND[6]	CLIENT_RND[5]	CLIENT_RND[4]	
1C	CLIENT_RND[3]	CLIENT_RND[2]	CLIENT_RND[1]	CLIENT_RND[0]	
20	SERVER_RND[31]	SERVER_RND[30]	SERVER_RND[29]	SERVER_RND[28]	Server Random Data
24	SERVER_RND[27]	SERVER_RND[26]	SERVER_RND[25]	SERVER_RND[24]	
28	SERVER_RND[23]	SERVER_RND[22]	SERVER_RND[21]	SERVER_RND[20]	
2C	SERVER_RND[19]	SERVER_RND[18]	SERVER_RND[17]	SERVER_RND[16]	

Table continues on the next page...

Table 264. Public Key digest for Signing: Data structure that is hashed: KGSIGN_RND = 1, REVF = 1, BYTE_ORDER = 1 (continued)

Word Index	Byte[3]	Byte[2]	Byte[1]	Byte[0]	Description
30	SERVER_RND[15]	SERVER_RND[14]	SERVER_RND[13]	SERVER_RND[12]	
34	SERVER_RND[11]	SERVER_RND[10]	SERVER_RND[9]	SERVER_RND[8]	
38	SERVER_RND[7]	SERVER_RND[6]	SERVER_RND[5]	SERVER_RND[4]	
3C	SERVER_RND[3]	SERVER_RND[2]	SERVER_RND[1]	SERVER_RND[0]	
40	0x3	0x0	0x17	0x41	RFC8442 3 Byte Prefix + 1 Byte of Public Key size Public Key
44	0x4	Qx[31]	Qx[30]	Qx[29]	Public Key X Co-Ordinate
48	Qx[28]	Qx[27]	Qx[26]	Qx[25]	
4C	Qx[24]	Qx[23]	Qx[22]	Qx[21]	
50	Qx[20]	Qx[19]	Qx[18]	Qx[17]	
54	Qx[16]	Qx[15]	Qx[14]	Qx[13]	
58	Qx[12]	Qx[11]	Qx[10]	Qx[9]	
5C	Qx[8]	Qx[7]	Qx[6]	Qx[5]	
60	Qx[4]	Qx[3]	Qx[2]	Qx[1]	
64	Qx[0]	Qy[31]	Qy[30]	Qy[29]	Public Key Y Co-ordinate
68	Qy[28]	Qy[27]	Qy[26]	Qy[25]	
6C	Qy[24]	Qy[23]	Qy[22]	Qy[21]	
70	Qy[20]	Qy[19]	Qy[18]	Qy[17]	
74	Qy[16]	Qy[15]	Qy[14]	Qy[13]	
78	Qy[12]	Qy[11]	Qy[10]	Qy[9]	
7C	Qy[8]	Qy[7]	Qy[6]	Qy[5]	
80	Qy[4]	Qy[3]	Qy[2]	Qy[1]	
84	Qy[0]	0x80	0x0	0	Initial Padding Byte

Table continues on the next page...

Table 264. Public Key digest for Signing: Data structure that is hashed: KGSIGN_RND = 1, REVF = 1, BYTE_ORDER = 1 (continued)

Word Index	Byte[3]	Byte[2]	Byte[1]	Byte[0]	Description
88	0	0	0	0	
8C	0	0	0	0	
90	0	0	0	0	
94	0	0	0	0	
98	0	0	0	0	
9C	0	0	0	0	
A0	0	0	0	0	
A4	0	0	0	0	
A8	0	0	0	0	
AC	0	0	0	0	
B0	0	0	0	0	
B4	0	0	0	0	
B8	0	0	0	0	
BC	0	0	4	0x28	

Public key signing: Data Structures for other combinations of REVF, BYTE_ORDER

Table 265. Public key signing: Data structure for KGSIGN_RND = 1, REVF = 1, BYTE_ORDER = 0

Word index	Byte[3]	Byte[2]	Byte[1]	Byte[0]	Description
0	CLIENT_RND[28]	CLIENT_RND[29]	CLIENT_RND[30]	CLIENT_RND[31]	Client Random Data
etcetera					
1C	CLIENT_RND[0]	CLIENT_RND[1]	CLIENT_RND[2]	CLIENT_RND[3]	
20	SERVER_RND[28]	SERVER_RND[30]	SERVER_RND[29]	SERVER_RND[28]	Server Random Data
etcetera					
3C	SERVER_RND[0]	SERVER_RND[1]	SERVER_RND[2]	SERVER_RND[3]	

Table continues on the next page...

Table 265. Public key signing: Data structure for KGSIGN_RND = 1, REVF = 1, BYTE_ORDER = 0 (continued)

Word index	Byte[3]	Byte[2]	Byte[1]	Byte[0]	Description
40	0x41	0x17	0x0	0x3	RFC8442 3 Byte Prefix + 1 Byte of Public Key size Public Key
44	0x4	Qx[28]	Qx[29]	Qx[30]	Public Key X Co-ordinate
etcetera					
60	Qx[7]	Qy[0]	Qy[1]	Qy[2]	
64	Qx[3]	Qy[31]	Qy[30]	Qy[29]	Public Key Y Co-ordinate
68	Qy[28]	Qy[27]	Qy[26]	Qy[25]	
etcetera					
80	Qy[4]	Qy[3]	Qy[2]	Qy[1]	
84	Qy[0]	0	0	0	Initial Padding Byte
88	0	0	0	0x80	
8C	0	0	0	0	
etcetera					
BC	0x40	0x4	0	0	

Table 266. Public key signing: Data structure for KGSIGN_RND = 1, REVF = 0, BYTE_ORDER = 1

Word Index	Byte[3]	Byte[2]	Byte[1]	Byte[0]	Description
0	CLIENT_RND[31]	CLIENT_RND[30]	CLIENT_RND[29]	CLIENT_RND[28]	Client Random Data
etcetera					
1C	CLIENT_RND[3]	CLIENT_RND[2]	CLIENT_RND[1]	CLIENT_RND[0]	
20	SERVER_RND[31]	SERVER_RND[30]	SERVER_RND[29]	SERVER_RND[28]	Server Random Data
etcetera					
3C	SERVER_RND[3]	SERVER_RND[2]	SERVER_RND[1]	SERVER_RND[0]	

Table continues on the next page...

Table 266. Public key signing: Data structure for KGSIGN_RND = 1, REVF = 0, BYTE_ORDER = 1 (continued)

Word Index	Byte[3]	Byte[2]	Byte[1]	Byte[0]	Description
40	0x3	0x0	0x17	0x41	RFC8442 3 Byte Prefix + 1 Byte of Public Key size Public Key
44	0x4	Qx[3]	Qx[2]	Qx[1]	Public Key X Co-Ordinate
48	Qx[0]	Qx[7]	Qx[6]	Qx[5]	
etcetera					
60	Qx[24]	Qx[31]	Qx[30]	Qx[29]	
64	Qx[28]	Qy[3]	Qy[2]	Qy[1]	Public Key Y Co-Ordinate
68	Qy[0]	Qy[7]	Qy[6]	Qy[5]	
etcetera					
80	Qy[24]	Qy[31]	Qy[30]	Qy[29]	
84	Qy[28]	0x80	0	0	Initial Padding Byte
88	0	0	0	0	
etcetera					
BC	0x0	0x0	0x4	0x28	

Table 267. Public key signing: Data structure for KGSIGN_RND = 1, REVF = 0, BYTE_ORDER = 0

Word Index	Byte[3]	Byte[2]	Byte[1]	Byte[0]	Description
0	CLIENT_RND[28]	CLIENT_RND[29]	CLIENT_RND[30]	CLIENT_RND[31]	Client Random Data
etcetera					
1C	CLIENT_RND[0]	CLIENT_RND[1]	CLIENT_RND[2]	CLIENT_RND[3]	
20	SERVER_RND[28]	SERVER_RND[29]	SERVER_RND[30]	SERVER_RND[31]	Server Random Data
etcetera					
3C	SERVER_RND[0]	SERVER_RND[1]	SERVER_RND[2]	SERVER_RND[3]	

Table continues on the next page...

Table 267. Public key signing: Data structure for KGSIGN_RND = 1, REVF = 0, BYTE_ORDER = 0 (continued)

Word Index	Byte[3]	Byte[2]	Byte[1]	Byte[0]	Description
40	0x41	0x17	0x0	0x3	RFC8442 3 Byte Prefix + 1 Byte of Public Key size Public Key
44	0x4	Qx[0]	Qx[1]	Qx[2]	Public Key X Co-Ordinate
48	Qx[3]	Qx[4]	Qx[5]	Qx[6]	Public Key X Co-Ordinate
etcetera					
60	Qx[27]	Qx[28]	Qx[29]	Qx[30]	
64	Qx[31]	Qy[0]	Qy[1]	Qy[2]	Public Key Y Co-Ordinate
68	Qy[3]	Qy[4]	Qy[5]	Qy[6]	
etcetera					
80	Qy[27]	Qy[28]	Qy[29]	Qy[30]	
84	Qy[31]	0	0	0	Initial Padding Byte
88	0	0	0	0x80	
etcetera					
BC	0x40	0x4	0	0	

13.4.2.7 KEYIN command

Introduction

KEYIN transfers a key to keystore. The transfer may include converting the input key from a different format before writing it to key store. The input key is one of the following:

1. The device unique key (DUK), a master key which is transferred from PUF via a dedicated hardware interface.
2. An encrypted (wrapped) key in system memory. KEYIN unwraps these keys before writing them to keystore. ELS key unwrapping uses the algorithm defined in the RFC3394 standard.
3. A (plain text) public key in system memory.

KEYIN example sequences

- PUF transfer: "KEYIN(PUF)"
    1. Prepare the input data: The PUF must have been properly initialized. PUF initialization is the responsibility of the system, not of ELS.
    2. Set the command parameters and start the command. The transfer format should be PUF.

NOTE

PUF transfer automatically creates a 256-bit key in keystore slots 0 and 1. The output key index KIDX0 is ignored for PUF in transfers.

3. After completion of the command, the DUK will be available in slot 0 and slot 1 in keystore.

- RFC3394 transfer (unwrap): "KEYIN(RFC)"
  1. Data preparation
     a. The correct key wrapping key (or unwrapping only key) must have been created in keystore.

     NOTE

     The key wrapping keys can be used to both wrap and unwrap.

     b. The wrapped key must exist in system memory.
  2. Set the command parameters and start the command. The transfer format should be set to RFC3394, the location of the wrapped key in system memory should be defined, and the destination slot in keystore should be defined.
  3. After completion of the command, the source key material (key value) is unwrapped from system memory and is output to the specified destination slot or slots in keystore.

     The properties of the source key are also unwrapped from the source key and stored in the key status register (KEY_STATUS0 - KEY_STATUSn) that corresponds to the KEY_STATUSn slot of the output key.

     Supported key sizes are 128 bits, and 256 bits. ELS determines the size of the wrapped key from the size of the wrapped key data in system memory.
- Public Key Import: "KEYIN(PBK)": In this operation, ELS first ECC verifies the incoming public key against a signature supplied by the user. If the signature is correct, ELS imports the public key to keystore.
  1. Data preparation
     a. The incoming public key, together with header and padding information must be placed in a defined data structure in system memory.
     b. A signature must have been created independently of ELS by signing the entire data structure mentioned above, using the private part of a root key pair. The signature must be placed in a separate location in system memory.
     c. The public key part of the root key pair must have been placed in keystore. This is achieved separately, via the ELS KEYIN(RFC3394) command.
  2. Set the command parameters and start the command.

     Required parameters are:

     a. Transfer format (= PBK)
     b. Start address in system memory of the data structure that includes the incoming public key and length in bytes of the data structure
     c. Start address in system memory of the signature
     d. Offset in bytes from the base of the data structure to the base address of the incoming public key
     e. Keystore slot that contains the public key part of the root key pair
     f. Keystore slot that will receive the incoming public key,

     The import procedure consists of the following:

     a. Import the data structure (public key plus header information).
     b. Import the signature (incoming public key signed with the private part of the root key pair).
     c. ECC verify the signature against the data structure, using the public key part of the root key pair. This proves that the data structure is authentic, and extraction of the incoming public key can proceed.

Failure of this check triggers an ELS OPN error.

d. Import the incoming public key (Key has to be re-imported due to operational limitation of ELS). Perform a proprietary integrity check to ensure that the incoming public key has not been modified since the import for ECC verify above. Failure of this check triggers an ELS FLT error.

e. If no errors, write the incoming public key to keystore.

3. When the command has completed, the incoming public key will have been imported from system memory to keystore.

   The new keystore key will have size 512 bits, and usage property UKPUK will be set.

KEYIN parameters

Table 268. KEYIN parameters

Parameter	Register	Description
KFMT	CMDCFG0[7:6]	Format of the input key. This also determines how the input key is fetched. 1=RFC3394 2=PUF: In this case input key will be fetched over the ELS PUF IN hardware interface. 3=PBK: ECC public key import
SRC0_ADDR	DMA_SRC0	• KFMT=PUF: ignored • KFMT=RFC3394: Start address in system memory where the encrypted key will be fetched from. • KFMT=PBK: Start address in system memory of the data structure that contains the incoming public key.
SRC0_LEN	DMA_SRC0_LEN	• KFMT=PUF: ignored • KFMT=RFC3394: Size in bytes of the RFC3394 blob containing the input key. • KFMT=PBK: Length in bytes of the data structure that contains the incoming public key.
SRC1_ADDR	DMA_SRC1	• KFMT=PUF, RFC3394: ignored • KFMT=PBK: Start address in system memory of a structure that contains the concatenation of Signature, R||S.
SRC2_ADDR	DMA_SRC2	• KFMT=PUF, RFC3394: ignored • KFMT=PBK: Offset Index in bytes to the start of the Public Key Data within the input data structure. ELS adds this index to SRC0_ADDR to determine the base address of the public key in system memory. This allows flexibility for different public key certificate structures.
RES0_ADDR	DMA_RES0	• KFMT = PBK: Re-Calculated value of R. This allows the user to do an optional external comparison between the R field that is part of the input Signature, and the R that is calculated by the ECC VFY step during public key import. The external comparison by user is independent of and complementary to the internal comparison that is always done by ELS. If the two values match, the user can consider the external comparison to have passed.

Table continues on the next page...

Table 268. KEYIN parameters (continued)

Parameter	Register	Description
		RES0_ADDR is the start address in system memory where the calculated value of R is stored. • KFMT = PUF, RFC3994: Ignored. This parameter is not used for PUF and RFC3394.
KIDX0	KIDX0	• KFMT=PUF: ignored. • KFMT=RFC3394: Keystore index of the unwrap key. • KFMT=PBK: ignored.
KIDX1	KIDX1	• KFMT=PUF: ignored. The unpacked key is automatically stored in key slots 0, 1. • KFMT=RFC3394: Keystore index of the key that is unwrapped. • KFMT=PBK: Keystore index of the key that is created.
KIDX2	KIDX2	• KFMT=PUF: ignored. • KFMT=RFC3394: ignored. • KFMT=PBK: Keystore index of the root PBK that will be used to verify the signature.
REVF	CMDCFG0[4]	• KFMT=PUF: ignored. • KFMT=RFC3394: ignored. • KFMT=PBK: REVF controls how ELS loads the embedded public key, during the "embedded public key import" stage of KEYIN(PBK). To understand the effect of REVF, refer to the example certificate endianness cases in the sections after Notation used for PBK (REVF, BYTE_ORDER) endianness cases. REVF also controls how ELS loads the Signature. **NOTE** ELS reads the public key twice during KEYIN(PBK). The first time is part of an ECC verify of the certificate. This read is not influenced by the value of REVF, ELS assumes that organization of the certificate in system memory, is as shown in: Table 272, Table 273, Table 274, and Table 275. **NOTE** When REVF = 1, the signature start address SRC1_ADDR must be 32-bit word aligned. When REVF = 1, the Output 'R' Start address (RES0_ADDR) must be 32-bit word aligned.
KPROPIN	KPROPIN	• KFMT=PUF, KFMT=RFC3394: ignored. • KFMT=PBK: Requested properties of the imported key. All properties except UPPROT are set automatically by the command. This means that ELS ignores all fields of KPROPIN, except for UPPROT.

Table continues on the next page...

Table 268. KEYIN parameters (continued)

Parameter	Register	Description
		NOTE If the requested privilege level UPPROT for the imported key is higher than that of the root public key, an ELS OPN error will be triggered. **NOTE** The bit positions of KPROPIN map to the list of key properties that is defined in Keystore key properties and status.

KEYIN data structures

For a description of how ELS handles endianness, refer to Endianness.

Table 269. KEYIN data structures

Format	Object	Description
PUF	Input packed key	Packed key is downloaded automatically from PUF. No action required by user.
	Output key	A 256 keystore key created at slots 0 and 1.
RFC3394	Input packed key: Also known as wrapped key or blob.	Wrapped version of the key must be placed in system memory. Size of wrapped version of 128 bit key plus its properties is 256 bits. Size of wrapped version of 256 bit key plus its properties is 384 bits. Size of wrapped version of 512 bit key plus its properties is 640 bits. This covers the case when a public key is imported via KEYIN(RFC) instead of via KEYIN(PBK).

Table continues on the next page...

Table 269. KEYIN data structures (continued)

Format	Object	Description
		NOTE There are cases where the user has created the wrapped (packed) key and not ELS. To cover these cases, the user needs to know the composition of the blob. The procedure by which the user would create the blob is described below. 1. Prepare the input data to the wrapping process. a. 128-bit key: {64'ha6a6a6a6a6a6a6a6, 32'hKEY_PROPS, 32'h0, plain_key[127:0]} (256 bits) and that would have output the encrypted string ENCRYPTED_KEY_AND_PROPS[255:0]. ENCRYPTED_KEY_AND_PROPS should be stored 32-bit word little endian, byte little endian in system memory. b. 256-bit key: {64'ha6a6a6a6a6a6a6a6, 32'hKEY_PROPS, 32'h0, plain_key[255:0]} (384 bits). c. 512-bit key (root public key imported via KEYIN(RFC): {64'ha6a6a6a6a6a6a6a6, 32'hKEY_PROPS, 32'h0, plain_key[511:0]} (640 bits). 2. Encrypt the input data using the RFC3394 algorithm and organizing the output as 32-bit word little endian, byte little endian in system memory.
	Wrapping key	The wrapping key must exist in keystore, and its size may be either 128 bits or 256 bits.
	Output key	KEYIN will output a 128, 256, or 512 bits key to keystore. The size property of the output key will be set automatically by the command. User is responsible to specify a free area of keystore to receive the output key.
KEYIN (PBK)	Certificate	• Padding The certificate must be padded in accordance with the SHA2-256 standard. • Size The size of the padded certificate must be an integer number of SHA2-256 blocks. The size of the padded certificate must be a minimum of 2 SHA2-256blocks. • Compression ELS does not support compressed certificates; it only supports uncompressed certificates.

Table continues on the next page...

Table 269. KEYIN data structures (continued)

Format	Object	Description
		NOTE ELS does not explicitly check the above requirements, but ELS behavior is not guaranteed if the requirements are not met. For cases that do not meet the requirements, KEYIN(PBK) command may trigger an ELS error.
	The output key	KEYIN (PBK) will output a 512 bit key to keystore. The size property of the output key will be set automatically by the command. User is responsible to specify a free area of keystore to receive the output key.

Key Property setting by KEYIN

Table 270. Key Property setting by KEYIN

Property Type	Usage control	Access control
FORMAT (KFMT) PUF	Set automatically by ELS • DUK = 1 • CKDF = 1	ELS automatically sets the access control property UPPROT to secure, privileged. ELS automatically sets the access control property WRPOK to 0. **NOTE** KEYIN(KFMT = PUF) can be run by a user with any access control setting, but KEYIN PUF always creates a KEY with UPPROT set to secure, privileged.
FORMAT (KFMT) RFC3394	Extracted from the input ELS automatically extracts these from the input (encrypted) key and copies them to the output (decrypted) key.	Extracted from the input ELS automatically extracts these from the input (encrypted) key and copies them to the output (decrypted) key.
FORMAT (KFMT) PBK	Set automatically by ELS • UKPUK = 1	

KEYIN(RFC) data organization of an ELS key blob

An ELS key blob can be unwrapped with KEYIN to import a key to ELS keystore. The blob will have been previously created either by using ELS KEYOUT to export the key from keystore or prepared independently of ELS by a 3rd party, using a wrapping key that must be shared with ELS in order to allow ELS to unwrap the key.

The table below shows, the expected organization of an ELS RFC3394 key blob in system memory, and how the imported key material is stored in ELS keystore.

The example used is a 512 bit root public key {Qy[255:0],Qx[255:0]}

Table 271. KEYIN(RFC) Data Organization of ELS key blob

Address	Value	Keyslot	Comment
0x0	32'ha6a6a6a6	N/A	RFC "initial value"
0x4	32'ha6a6a6a6	N/A	RFC "initial value"
0x8	key properties[31:0]	N/A	ELS key properties of the incoming key
0xc	32'h0	N/A	
0x10	Qy[255:224]	Base	Encrypted representation of Qy[255:224]
0x14	Qy[223:192]	Base	
etcetera			
0x20	Qy[127:96]	Base + 1	
etcetera			
0x30	Qx[255:224]	Base + 2	
etcetera			
0x40	Qx[127:96]	Base + 3	
etcetera			
0x4c	Qx[31:0]	Base + 3	

Expected organization of the root public key in keystore

For KEYIN(PBK), given the root public key = {Qy[255:0],Qx[255:]} and "kidx" is the keybase keystore index at which the root public key is stored, then the 4 kstore slots should be organized like this: kidx -> Qx[127:0] kidx+1 -> Qx[255:128] kidx+2 -> Qy[127:0] kidx+3 -> Qy[255:128]

The root public key will be installed in ELS via KEYIN(RFC). The root public key will have been wrapped by a third party, not by ELS. The third party is responsible for ensuring that after KEYIN(RFC) by ELS, the key organization in ELS keystore is as described above. See KEYIN(RFC) data organization of an ELS key blob.

Usage restriction: KEYIN(KFMT=PUF) may be run only once

KEYIN(KFMT=PUF) may only be run once. Before repeating this command case, ELS must be reset (either synchronous or asynchronous). An attempt to run the command case without resetting ELS will trigger an ELS OPN error.

Notation used for PBK (REVF, BYTE_ORDER) endianness cases

This section defines that notation that is used in the following sections which contain an example certificate stored with different endianness cases.

A certificate is represented as a sequence of items as follows:

certificate = header, Qx, Qy, byte_order_padding (only when byte_order = 0), sha2 padding

The items are described below:

1. The certificate header, represented as a sequence of bytes as follows: header = client_rnd31, client_rnd30 … client_rnd1, client_rnd0, server_rnd31, server_rnd_30 … server_rnd1, server_rnd0, 0x03, 0x00, 0x17, 0x41, 0x04
2. Qx: the X coordinate of a public key: a 256 bit number represented as a sequence of bytes as follows: Qx = {Qx31, Qx30, ... Qx1, Qx0}: Consistent with C
3. Qy: the Y coordinate of a public key: a 256 bit number represented as a sequence of bytes as follows: Qy = {Qy31, Qy30, ... Qy1, Qy0}: Consistent with ELS endianness definition, Qy31 is the most significant byte.
4. byte_order padding: this is needed in byte_order = 0 cases.

   ELS can only reverse the byte order of SHA2 padding when the size of the padding is a multiple of 4 bytes. This is an ELS implementation limitation. Because of the limitation, in these cases the message must be padded to a multiple of 4 bytes, with non-sha2 padding: This is referred to as "byte_order padding". It appears in the sections below as a sequence of 3, 0x00 bytes that is inserted immediately after the last byte of Qy.

   NOTE

   In the examples that follow, the presence/absence of byte_order padding causes the "message size" field of the sha2 padding to be 0x428 for byte_order = 1 and 0x440 for byte_order = 0.

5. sha2 padding: represented as a sequence of bytes as follows:

   byte_order = 0, Padding = 0x80, 0x00, … 0x00, 0x04, 0x40

   byte_order = 1, Padding = 0x80, 0x00, … 0x00, 0x04, 0x28

Certificate structure for PBK import (REVF = 1, BYTE_ORDER = 1)

The table below shows how the embedded public key should be organized in system memory for use with KEYIN(PBK) with REVF = 1, and BYTE_ORDER = 1.

Table 272. Expected certificate structure: REVF = 1, BYTE_ORDER = 1

System memory byte_address	Contents byte_address+0	Contents byte_address+1	Contents byte_address+2	Contents byte_address+3	Description
0x00	client_rnd31	client_rnd30	client_rnd29	client_rnd28	and so on until...
0x1c	client_rnd3	client_rnd2	client_rnd1	client_rnd0	
0x20	server_rnd31	server_rnd30	server_rnd29	server_rnd28	and so on until...
0x3c	server_rnd3	server_rnd2	server_rnd1	server_rnd0	
0x40	0x03	0x00	0x17	0x41	
0x44	0x04	**Qx31**	Qx30	Qx29	and so on until...

Table continues on the next page...

Table 272. Expected certificate structure: REVF = 1, BYTE_ORDER = 1 (continued)

System memory byte_address	Contents byte_address+0	Contents byte_address+1	Contents byte_address+2	Contents byte_address+3	Description
0x48	Qx28	Qx27	Qx26	Qx25	
0x60	Qx4	Qx3	Qx2	Qx1	
0x64	Qx0	**Qy31**	Qy30	Qy29	and so on until...
0x80	Qy4	Qy3	Qy2	Qy1	
0x84	Qy0	0x80	0x00	0x00	
0x88	0x00	0x00	0x00	0x00	and so on.
0xBC	0x00	0x00	0x04	0x28	

Certificate structure for PBK import (REVF = 1, BYTE_ORDER = 0)

The table below shows how the embedded public key should be organized in system memory for use with KEYIN(PBK) with REVF = 1, and BYTE_ORDER = 0.

Table 273. Expected certificate structure: REVF = 1, BYTE_ORDER = 0

System memory byte_address	Contents byte_address+0	Contents byte_address+1	Contents byte_address+2	Contents byte_address+3	Description
0x00	client_rnd28	client_rnd29	client_rnd30	client_rnd31	and so on until...
0x1c	client_rnd0	client_rnd1	client_rnd2	client_rnd3	
0x20	server_rnd28	server_rnd29	server_rnd30	server_rnd31	and so on until...
0x3c	server_rnd0	server_rnd1	server_rnd2	server_rnd3	
0x40	0x41	0x17	0x00	0x03	
0x44	0x04	**Qx28**	Qx29	Qx30	
0x48	Qx31	Qx24	Qx25	Qx26	and so on until...
0x60	Qx7	Qx0	Qx1	Qx2	
0x64	Qx3	**Qy28**	Qy29	Qy30	and so on until..
0x80	Qy7	Qy0	Qy1	Qy2	
0x84	Qy3	0x00	0x00	0x00	byte_order padding
0x88	0x00	0x00	0x00	0x80	and so on.
0xBC	0x40	0x04	0x00	0x00	

Certificate structure for PBK import (REVF = 0, BYTE_ORDER = 1)

The table below shows how the embedded public key should be organized in system memory for use with KEYIN(PBK) with REVF = 0, and BYTE_ORDER = 1.

Table 274. Expected certificate structure: REVF = 0, BYTE_ORDER = 1

System memory byte_address	Contents byte_address+0	Contents byte_address+1	Contents byte_address+2	Contents byte_address+3	Description
0x00	client_rnd31	client_rnd30	client_rnd29	client_rnd28	and so on until...
0x1c	client_rnd3	client_rnd2	client_rnd1	client_rnd0	
0x20	server_rnd31	server_rnd30	server_rnd29	server_rnd28	and so on until...
0x3c	server_rnd3	server_rnd2	server_rnd1	server_rnd0	
0x40	0x03	0x00	0x17	0x41	
0x44	0x04	**Qx3**	Qx2	Qx1	
0x48	Qx0	Qx7	Qx6	Qx5	and so on until...
0x60	Qx24	Qx31	Qx30	Qx29	
0x64	Qx28	**Qy3**	Qyy2	Qy1	and so on until...
0x80	Qy24	Qy31	Qy30	Qy29	
0x84	Qy28	0x80	0x00	0x00	
0x88	0x00	0x00	0x00	0x00	and so on until...
0xBC	0x00	0x00	0x04	0x28	

Certificate structure for PBK key import (REVF = 0, BYTE_ORDER = 0)

The table below shows how the embedded public key should be organized in system memory for use with KEYIN(PBK) with REVF = 0, and BYTE_ORDER = 0.

Table 275. Expected certificate structure: REVF = 0, BYTE_ORDER = 0

System memory byte_address	Contents byte_address+0	Contents byte_address+1	Contents byte_address+2	Contents byte_address+3	Description
0x00	client_rnd28	client_rnd29	client_rnd30	client_rnd31	and so on until...
0x1c	client_rnd0	client_rnd1	client_rnd2	client_rnd3	
0x20	server_rnd28	server_rnd29	server_rnd30	server_rnd31	and so on until...
0x3c	server_rnd0	server_rnd1	server_rnd2	server_rnd3	

Table continues on the next page...

Table 275. Expected certificate structure: REVF = 0, BYTE_ORDER = 0 (continued)

System memory byte_address	Contents byte_address+0	Contents byte_address+1	Contents byte_address+2	Contents byte_address+3	Description
0x40	0x41	0x17	0x00	0x03	
0x44	0x04	**Qx0**	Qx1	Qx2	and so on until...
0x48	Qx3	Qx4	Qx5	Qx6	
0x60	Qx27	Qx28	Qx29	Qx30	
0x64	Qx31	**Qy0**	Qy1	Qy2	and so on until...
0x80	Qy27	Qy28	Qy29	Qy30	
0x84	Qy31	0x00	0x00	0x00	byte_order padding
0x88	0x00	0x00	0x00	0x80	and so on until...
0xBC	0x40	0x04	0x00	0x00	

13.4.2.8 KEYOUT command

Introduction

KEYOUT wraps (encrypts) a keystore key and stores the wrapped version in system memory. Because the wrapped key is encrypted with a secret wrapping key, it can be safely stored outside of ELS.

When a key is wrapped, the properties of the key, such as size and purpose, are also encrypted and stored in the wrapped version of the key.

KEYOUT is implemented using the algorithm described in the RFC3394 standard.

KEYOUT example sequence

1. Data preparation
   a. The correct key wrapping key must have been created in keystore.

      NOTE

      Key wrapping keys can be used to both wrap and unwrap.

   b. The key to be wrapped must exist in keystore.
2. Set the command parameters and start the command.

   The keystore index of the key to be wrapped, the keystore index of the wrapping key, and the address in system memory of the output (wrapped version) key, should all be defined.
3. After completion of the command, the output data will be available in system memory. The command wraps the material (key value) of the source key, plus the key properties which it gets from the key status register (KEY_STATUS0-KEY_STATUSn) that corresponds to the source key.

KEYOUT parameters

Table 276. KEYOUT parameters

Parameter	Register	Description
RES0_ADDR	DMA_RES0	Start address in system memory of the wrapped key that is output.
KIDX0	KIDX0	ELS keystore index of the wrapping key. The wrapping key must have the property UKWK = 1 or an ELS error will occur.
KIDX1	KIDX1	ELS keystore index of the key to be wrapped. The key must be wrappable (have property WRPOK = 1), or ELS error will occur.

KEYOUT data structures

For a description of how ELS handles endianness, see Endianness.

Table 277. KEYOUT data structures

Object	Description
Input key	KEYOUT takes as input a 128 or 256 bit key from keystore.
Wrapping key	The wrapping key must exist in keystore and must be a wrapping key.
Output key (also known as wrapped key or blob)	Wrapped key will be placed in system memory. Size of wrapped version of 128 bit key plus its properties is 256 bits. Size of wrapped version of 256 bit key plus its properties is 384 bits.

Keyout command: Additional checks

The following additional checks are specific to KEYOUT:

- The key to be wrapped must be active.
- The user must have sufficient rights to wrap the key. See Key access control checks for key wrapping and unwrapping.

13.4.2.9 KDELETE command

Introduction

KDELETE deletes a key from keystore. The purpose of this would be to discard keys that are no longer needed, and to free up keystore slots for new keys.

KDELETE example sequence

1. Data preparation: The key to be deleted must exist in keystore.
2. Set the command parameters and start the command. The keystore index of the base (first slot) of the key to be deleted should be defined.
3. The command when completed deletes the keystore key and zeroes the corresponding key status register(s) (KEY_STATUSn).

KDELETE parameters

Table 278. KDELETE parameters

Parameter	Register	Description
KIDX0	KIDX0	ELS keystore index of the key to be deleted. **NOTE** ELS attempts to delete an "empty" keyslot will be ignored by ELS, that is, it will not trigger an ELS error. An empty slot is a slot with property KACTV = 0, see Keystore key properties and status.

13.4.2.10 CKDF command

Introduction

CKDF creates a new keystore key from an existing keystore key. It does that by combining the existing key (the "derivation key"), with supplied data ("derivation data"), via a standard algorithm referred to as the key derivation function (KDF). CKDF supports 2 different KDF algorithms:

1. NIST SP 800-108: CMAC based KDF

   Based on key derivation as described in (NIST SP 800-108) with CMAC (NIST SP 800-38B) used as PRF (Pseudo-Random Function), that is, Derived_Key = CMAC (derivation key, derivation data).

   This option can create 2 key sizes: 128 bits and 256 bits.

   For derivation of a 256-bit key, the Key Derivation Function (KDF) is used in "Counter Mode". The PRF is repeated to create the second 128-bit chunk of the Derived Key, with the "counter" (which forms part of the derivation data), incremented by 1.

2. Based on key derivation as described in NIST SP800-56c, the derivation is done in two steps: extract and expand, where the extract key is used as an input to the expand step. CKDF must be called twice to complete the full derivation:

   a. Extract step: creates a 128-bit key

   b. Expand step: creates a 128-bit key

CKDF example sequence

1. Data preparation

   a. A suitable key derivation key must have been created in keystore. That is a key with usage property UCKDF = 1.

   b. Derivation data must have been placed in the system memory.

   c. Hardware derivation data

   NOTE

   This is not used in all cases. See the derivation data description in CKDF data structures below.

2. Set the command parameters and start the command. The keystore index of the derivation key, the keystore index of the derived (output) key, the requested properties of the derived key, the location in system memory of the derivation data, should all be defined.

3. After completion of the command, the output key will be available in the specified keystore slot.

CKDF parameters

Table 279. CKDF parameters

Parameter	Register	Description
CKDF_ALGO	CMDCFG0[13:12]	0 = NIST SP 800-108: CMAC based KDF 1 = SP 800-56C Two step CMAC KDF: expand step 2 = SP 800-56C Two step CMAC KDF: extract step **(caution: be aware that normally the extract step is run before the expand step)** 3 = not supported: triggers an ELS OPN ERROR
SRC0_ADDR	DMA_SRC0	• NIST SP 800-108 CMAC BASED KDF: start address in system memory of the software component of the derivation data input (SW_DRV_DATA). • SP 800-56C two-step CMAC KDF: extract step: ignored. • SP 800-56C two-step CMAC KDF: expand step: start address in system memory of the derivation data referred to as "L, {IV}, FixedInfo" in the standard.
SRC0_LEN	DMA_SRC0_LEN	• NIST SP 800-108 CMAC BASED KDF: ignored: (SW_DRV_DATA is fixed size). • SP 800-56C two-step CMAC KDF: extract step : ignored. • SP 800-56C two-step CMAC KDF: expand step: length in bytes of the derivation data before it is padded by the user. **NOTE** ELS needs the information about the size of the unpadded message, because according to the standard, messages that before padding are not an integer number of blocks require different processing compared to messages that before padding are an integer number of blocks.
SRC2_ADDR	DMA_SRC2	• NIST SP 800-108 CMAC BASED KDF: ignored • SP 800-56C two-step CMAC KDF: extract step: "start address of derivation data, referred to as "SALT" in the standard. **NOTE** This address must be 32-bit word aligned. Unaligned address will trigger an ELS OPN error. • SP 800-56C two-step CMAC KDF: expand step: ignored
SRC2_LEN	DMA_SRC2_LEN	• NIST SP 800-108 CMAC BASED KDF: ignored • SP 800-56C two-step CMAC KDF: extract step: length in bytes of the derivation data referred to in SRC2_ADDR description above. Can be 16 or 32 bytes (128 or 256 bits). Any other length triggers an ELS OPN error. • SP 800-56C two-step CMAC KDF: expand step: ignored.
KIDX0	KIDX0	• NIST SP 800-108 CMAC BASED KDF: keystore index of the input (Derivation) Key. • SP 800-56C two-step CMAC KDF: extract step: keystore index of the Derivation Key input for the extract step (referred to as "Z" in the standard). • SP 800-56C two-step CMAC KDF: expand step: keystore index of the Derivation Key (referred to as "Key Derivation Key" or "KDK" in the standard).

Table continues on the next page...

Table 279. CKDF parameters (continued)

Parameter	Register	Description
		NOTE In all cases above, the key size can be either 128 or 256 bits.
KIDX1	KIDX1	ELS keystore index of the Output Key
KPROPIN	KPROPIN	Requested properties of the output key • The bit positions of KPROPIN map to the list of key properties defined in Keystore key properties and status. • All bit positions of KPROPIN may be set or cleared, and in cases where KPROPIN is used in the derivation data, then the bits of KPROPIN are used as described in Table 280). • Size restriction when UHWO is set. When UHWO is set, only a 128-bit key may be created. KSIZE must be 0. Setting KSIZE to 1 in this case will trigger an ELS OPN error. • Some but not all bits of KPROPIN are copied to the Key status register of the created key. The list below describes that: — Access rights are copied. — Bit position DUK must be set to 0 in KPROPIN. Setting it to 1 will trigger an ELS OPN error. See Key creation rules: Setting the usage control properties. — Usage rights are copied. — Reserved fields are not copied. — Hardware feature status are not copied. — KSIZE IS copied. — KBASE and KACTV are not copied. Refer to the list of key properties that is defined in Keystore key properties and status.

CKDF data structures

For a description of how ELS handles endianness, refer to Endianness.

Table 280. CKDF data structures

Object	Description
Derivation Data	• NIST SP 800-108: CMAC based KDF 1. Derivation Data: The Derivation data that ELS uses as input to the algorithm has the following user-supplied components, which are optionally included depending on the properties of the derivation key. a. Software_derivation_data[95:0]: must be placed in system memory.

Table continues on the next page...

Table 280. CKDF data structures

Object	Description
	b. The requested properties of the new key: This is referred to below as requested_properties[31:0]. This is created from register field KPROPIN as follows: requested_properties[31:0] = KPROPIN[31:0] | 32'h000_0020. c. System hardware supplied: 64 bits are supplied by system hardware. **NOTE** ELS also adds internally generated fields to the derivation data described below. 2. The composition of the derivation data is: a. If the Derivation key has the property DUK = 1. Derivation data[255:0] = proprietary_modification(software_derivation_data[95:0] || hw_drv_data[63:0] || requested_properties[31:0] || length[31:0] || counter[31:0]). **NOTE** hw_drv_data[63:0] is supplied by system hardware. A proprietary modification is carried out on the derivation data before it is used in the KDF. b. Else if the Derivation key has the property DUK = 0. Derivation data[255:0] = software_derivation_data[95:0] || 64'h0 || requested_properties[31:0 || length[31:0] || counter[31:0]. **NOTE** In both cases "b.i" and "b.ii" above, "length" and "counter" are supplied by ELS. • SP 800-56C two-step CMAC KDF: extract step This corresponds to "SALT" as described in the standard: Used as the "key" in the CMAC operation. Supplied by the user and placed in system memory. Size can be either 128 or 256 bits. • SP 800-56C two-step CMAC KDF: expand step Supplied by the user, and placed in system memory Derivation data is a padded message: **All fields of the padded message are supplied by the user; none are supplied by ELS.** The user is responsible for padding the message according to SP 800-56C. The padded message can be any size but must be an integer number of AES blocks (a multiple of 16-bytes). The unpadded message must be an integer number of bytes. Example: derivation_data = SP 800-56C_PAD(Counter || Label || 0x00 || Context || Length).

Table continues on the next page...

Table 280. CKDF data structures (continued)

Object	Description
Derivation key	The derivation key must be a keystore key with property UCKDF = 1.
Output derived key	Derived key will be placed in keystore. • NIST SP 800-108: CMAC based KDF Size of derived key will be either 128 or 256 bits, depending what size was set in the requested properties. The user must ensure there are enough free keystore slots at the position pointed to by KIDX1. • SP 800-56C Two step CMAC KDF: extract step 128-bit keystore key. The user must ensure there are enough free keystore slots at the position pointed to by KIDX1. • SP 800-56C two-step CMAC KDF: expand step 128-bit keystore key. The user must ensure there is a free keystore slot at the position pointed to by KIDX1.

13.4.2.11 HKDF command

Introduction

HKDF creates a new keystore key from an existing keystore Key. It does that by combining the existing key (the derivation key), with supplied data (derivation data), via a standard algorithm referred to as the key derivation function (KDF). HKDF is provided to support the Device Identifier Composition Engine (DICE) standard. HKDF supports 2 different KDF algorithms

- RFC5869 standard "HMAC based key derivation".
- SP 800-56C: one step Key derivation as described in NIST SP 800-56C "Recommendation for Key-Derivation Methods in Key-Establishment Schemes".

  With the option H(x) = hash(x) chosen for the auxiliary function H(x) as described in section "Specification of Key-Derivation Functions" of the same document.

  And where hash(x) is as described in FIPS PUB 198-1 "The Keyed-Hash Message Authentication Code (HMAC)".

HKDF support for derivation data from Run Time Fingerprint (RTF)

HKDF with the RFC5869 algorithm has the option to use derivation data from either system memory, or the internal RTF dependent on the command switch RTFDRVDAT.

1. RFTDRVDAT = 0:
   a. ELS loads 256 bits of derivation data from system memory.

      ELS pads the derivation data according to the RFC5869 standard "HMAC based key derivation."

      NOTE

      RFC5869 padding is the same as HMAC padding.

   b. ELS runs RFC5869 HKDF on the padded derivation data.

      For RFC5869 compliance, the derivation data SW_DRV_DATA[255:0] must be composed as {label[247:0], counter[7:0]} from the RFC5869 standard. This interpretation does not affect the algorithm that ELS runs, or the padding that ELS adds, but is useful if modeling the operation outside of ELS to produce an expected result.

NOTE

The option RTFDRVDAT = 0 can also be used to achieve a simple HMAC KDF.

That is because the algorithm used by ELS HKDF command to implement RFC5869 COMPLIANT KDF is a simple HMAC, but with the DATA size enforced to be 256 bits.

ELS_KDF=HMAC(INPUT_KEY, PAD(DATA[255:0]))

a. If DATA[255:0] supplied by the user is composed of [label[247:0], counter[7:0]} then the KDF is RFC 5869 compliant.

b. If DATA[255:0] supplied by the user does not contain any counter component, then the KDF is a simple HMAC KDF.

In either case, the algorithm run by ELS is the same.

2. RTFDRVDAT = 1

   a. ELS loads the internal RTF and pads it according to the HMAC standard FIPS.198-1-1.

   b. ELS runs FIPS.198-1-1 HMAC on the padded RTF.

HKDF example sequence

1. Data preparation

   a. A suitable key derivation key must have been created in keystore. That is a key with usage property UHKDF = 1.

   b. Derivation data must have been stored in system memory (not needed if RTFDRVDAT = 1).

2. Set the command parameters and start the command. The keystore index of the derivation key, the keystore index of the derived (output) key , the requested properties of the derived key, and the location in system memory of the derivation data should all be defined.

3. After completion of the command, the output key will be available in the specified keystore slot.

HKDF parameters

Table 281. HKDF parameters

Parameter	Register	Description
HKDF_ALGO	CMDCFG0[1]	0 =RFC5869 standard "HMAC based key derivation". 1 = SP 800-56C "one step KDF" (using the SHA-256 hash algorithm).
SRC0_ADDR	DMA_SRC0	Start address in system memory of the derivation data: • HKDF_ALGO = RFC5869: "SW_DRV_DATA" (referred to in HKDF data structures below). • HKDF_ALGO = SP 800-56C: see entry "derivation data" in HKDF data structures below).
SRC0_LEN	DMA_SRC0_LEN	• HKDF_ALGO = RFC5869: ignored • HKDF_ALGO = SP 800-56C: Size in bytes of CUSTOM_PAD(counter || FixedInfo) 1. See HKDF data structures. "CUSTOM_PAD" is described there. 2. "counter" and "FixedInfo" are described in NIST SP 800-56C.
RES0_ADDR	DMA_RES0	• HKDF_ALGO = RFC5869: ignored. • HKDF_ALGO = SP 800-56C: start address in system memory where the output key will be stored.

Table continues on the next page...

Table 281. HKDF parameters (continued)

Parameter	Register	Description
KIDX0	KIDX0	ELS keystore index of the Derivation (input) Key.
KIDX1	KIDX1	• HKDF_ALGO = RFC5869: ELS keystore index of the Output Key. • HKDF_ALGO = SP 800-56C: ignored.
KPROPIN	KPROPIN	• HKDF_ALGO = RFC5869: requested properties of the output key. **NOTE** The KSIZE field of KPROPIN will be ignored. ELS will automatically create a 256-bit output key. • HKDF_ALGO = SP 800-56C: ignored. **NOTE** The bit positions of KPROPIN map to the list of key properties that is defined in Keystore key properties and status.
RTFDRVDAT	CMDCFG0[0]	• HKDF_ALGO = RFC5869: 0: do not include RTF in the derivation data. 1: include RTF in the derivation data. See HKDF data structures for a definition of how RTFDRVDAT affects derivation data. • HKDF_ALGO = SP 800-56C: ignored.

HKDF data structures

Table 282. HKDF data structures

Object	Description
Derivation data	• HKDF_ALGO = RFC5869 The derivation data size is 256 bits. There are 2 possible sources for the derivation data. Input parameter RTFDRVDAT selects between the 2 sources: — if (RTFDRVDAT = 1) Derivation data = RTF[255:0] — Else Derivation data = SW_DRV_DATA[255:0] • HKDF_ALGO = SP 800-56C The user must provide the following in system memory: CUSTOM_PAD(counter || FixedInfo) — The size of counter must be 4 bytes (32 bits). — The size of FixedInfo is any integer number of bits greater than or equal to 0. **Definition of "CUSTOM_PAD"**

Table continues on the next page...

Table 282. HKDF data structures (continued)

Object	Description
	— CUSTOM_PAD means "add SHA-256 padding, assuming an unpadded message size in bits of (256 plus "size in bits of counter" plus "size in bits of "FixedInfo"), **NOTE** "counter "is a 32-bit value, as described in the SP 800-56C standard, so "size in bits of counter" = 32. **Explanation of CUSTOM_PAD** — The KDF that ELS runs will be SHA-256_HASH(SHA-256_PAD(counter || keystore_key || FixedInfo). — The user is responsible for supplying the counter || FixedInfo part of the message and doing the padding (SHA-256_PAD operation). The user must do the padding because ELS is not capable of padding in this case. However while padding, the user must allow for the fact that during the HKDF operation ELS will insert a 256-bit keystore key into the padded message. Therefore, the user must supply padding for a message that is 256 bits larger than the parts of the message that are supplied by the user (counter || FixedInfo). — ELS is responsible for inserting the keystore_key into the data structure above. The resulting data structure will be SHA-256_PAD(counter || keystore_key || FixedInfo) and will be held internally to ELS and used as input to the operation SHA-256_HASH(SHA-256_PAD(counter || keystore_key || FixedInfo). Because of CUSTOM_PAD, the SHA-256 padding will be correct for the structure counter || keystore_key || FixedInfo. — keystore_key mentioned above, corresponds to Z, the shared secret in the formula K(i) = H(counter || Z || FixedInfo) from the SP 800-56C standard.
Derivation key	The derivation key must be a 256-bit keystore key with property UHKDF = 1. Key size other than 256 bits will trigger an ELS OPN error.
Output derived key	Size of the output key will be 256 bits. The user must ensure there is enough keystore or system memory space to hold the derived key. • HKDF_ALGO = RFC5869: derived key will be placed in keystore. • HKDF_ALGO = SP 800-56C: derived key will be placed in system memory.

13.4.2.12 TLS_INIT command

Introduction

TLS_INIT creates 4 session keys from a shared secret. TLS_INIT is provided to support the TLS 1.2 (Transport Layer Security) standard as described in the RFC5246 document by IETF (internet engineering task force).

The session keys can be used to transmit and receive data over a secure channel as described in TLS 1.2. The session keys are:

- CMK: Client Message Authentication Key (256 bits)
- SMK: Server message authentication key (256 bits)
- CEK: Client encryption key (128bits)

- SEK: Server encryption key (128 bits)

TLS_INIT example sequence

1. Initialize phase: creating the master secret
    a. Data preparation
        i. A suitable pre-master secret key must exist in keystore. That is a key with usage property UTLSPMS = 1. Use ECKXCH command to create a pre-master secret key.

NOTE

The master secret key that is output from initialize phase will occupy 4 slots, starting at the base key slot of the pre-master secret key. See Data structures.

NOTE

None of the 4 slots above can be either retention slots or hardware out slots. If this requirement is not met, an ELS OPN error will be triggered.

        ii. Derivation data (referred to as "The hello message" in RFC5246) must have been placed in system memory.
    b. Set the command parameters and start the command. The keystore index of the pre-master secret, the requested properties of the derived key, and the start address in system memory of the derivation data should be defined.
    c. The command when completed creates the master secret and writes that over the pre-master secret.
2. Finalize phase: creating the session keys
    a. Data preparation
        i. A suitable TLS master secret key must have been created in keystore. That is a key with usage property UTLSMS = 1.

NOTE

The master secret key occupies 4 slots, see Data structures.

        ii. 4 additional key slots must be freed from (KIDX0+4) TO (KIDX0+7).
        iii. None of the key slots used can be either a retention slot or a hardware out slot. If this requirement is not met, an ELS OPN error will be triggered.
    b. Set the command parameters and start the command. The keystore index of the master secret, and the requested properties of the derived keys should be defined.
    c. The command when completed creates four Session keys from the master secret and writes these over the master secret.

NOTE

At the end of the command, slots (KIDX0+6), (KIDX0+7) are left as empty slots.

TLS_INIT parameters

Table 283. TLS_INIT parameters

Parameter	Register	Description
SRC0_ADDR	DMA_SRC0	Location of Derivation data in system memory:

Table continues on the next page...

Table 283. TLS_INIT parameters (continued)

Parameter	Register	Description
		• Derivation data is required for both initialize and finalize steps. • Derivation data is the same, fixed size for both initialize and finalize steps. • The composition of the derivation data is different between initialize and finalize steps. Refer to Table 285 for detail about the derivation data.
KIDX0	KIDX0	Keystore index of the input key • FINALIZE = 0: The input key must be a TLS pre-master secret (UTLSPMS = 1). • FINALIZE = 1: The input key must be a master secret (UTLSMS = 1). KIDX0 also specifies the destination for the keys output by this command. • FINALIZE = 0: The output master secret key is written to KIDX0 through KIDX0+3. • FINALIZE = 1: The four output keys are written to the slots starting at KIDX0, in the order CEK, CMK, SEK, SMK. KIDX0 though KIDX0 + 7 must be available for the command, but the output session keys only occupy KIDX0 through KIDX0 + 5. KIDX0 + 6 and KIDX0 + 7 will be empty after the command completes.
KPROPIN	KPROPIN	Requested properties of the output key: only property UPPROT[1:0] is used, the rest are ignored. **NOTE** The bit positions of KPROPIN map to the list of key properties that is defined in Keystore key properties and status.
FINALIZE	CMDCFG0[10]	0 = Initialize 1 = Finalize

Key Properties that are set automatically by TLS_INIT

All properties except UPPROT are set automatically by the command. This means that except for UPPROT the user supplied property fields of KPROPIN are ignored.

Table 284. Key Properties that are set automatically by TLS_INIT

FINALIZE	OUTPUT KEY	Usage control properties	KSIZE	WRPOK	Comment
0	TLSMS	UTLSMS = 1	1	0	
1	CMK	UHMAC = 1	1	1	FINALIZE = 1 generates the 4 keys listed in a single step.
	SMK	UHMAC = 1	1	1	
	CEK	UAES = 1	0	1	
	SEK	UAES = 1	0	1	

Data structures

Table 285. TLS_INIT data structures

Object	Description
Derivation Data	The user should construct the derivation data in system memory according to standard RFC5246. 1. Size of the derivation data: 640 bits 2. Composition of the derivation data a. Initialize step: text string "master secret" || server random data || client random data || 0x800000 b. Finalize step: text string "key expansion" || client random data || server random data || 0x800000 **NOTE** For both the initialize and finalize steps above, the "0x800000" above is not part of the RFC5246 standard, but is the first 3 bytes of SHA padding. These 3 bytes must be supplied by the User. ELS will automatically add the rest of the SHA padding. See standard RFC5246 for more details.
TLSMS key	TLSMS key is the output of TLS_INIT(FINALIZE = 0). It occupies 4 slots located at (KIDX0) to {KIDX0+3). The command overwrites the 2 slots occupied by the input key, plus 2 additional slots. ELS does not currently support key sizes greater than 2 slots. Therefore TLSMS is written to keystore as 2, 2 slot keys. 1. TLSMS key part 0: KIDX0 - KIDX0+1 2. TLSMS key part 1: KIDX0+2 - KIDX0+3
Session Keys	The session keys are the output of TLS_INIT(FINALIZE = 1). They occupy 6 slots located at (KIDX0) to {KIDX0+5). The command overwrites the 4 slots occupied by the input key (TLSMS key), plus 2 additional slots. The locations of the keys are as follows: 1. CMK: KIDX0 - KIDX0+1 2. SMK: KIDX0+2 - KIDX0+3 3. CEK: KIDX0+4 4. SEK: KIDX0+5 **NOTE** In total, TLS_INIT requires 8 consecutive free key slots.

13.4.2.13 HASH command

HASH computes a hash of a message according to the NIST FIPS 180-4 secure hash standard.

SHA engines and supported algorithms

This version of ELS supports SHA2-224, SHA-256, SHA-384, and SHA-512 algorithms.

HASH example sequences

- Full mode: entire message is processed by a single call to HASH.
  1. Data preparation - The message must be padded in accordance with NIST.FIPS-180-4 and placed in system memory.
  2. Set the command parameters and start the command. The start address of the padded message in system memory and the start address in system memory where the hash output should be stored must be defined.

     ELS is supplied with the 512-bit SHA engine, the digest size must also be defined.

     Optionally, if the HASH output is being used to update the run time fingerprint (RTF), and/or output the RTF, additional switches must be defined.
  3. The command when completed computes a digest of the input message and optionally writes that to system memory or uses that to update the RTF.
- Partial processing: the message is split into chunks. Each chunk is processed by a call to HASH.
  1. Data preparation: same as for full mode above.
  2. Process the first chunk: set switches HASHINI and HASHOE, plus other parameters and start the command. The start address of the padded message in system memory and the start address in system memory where the hash state should be stored must be defined.

     After completion of the command, the output message (intermediate digest) will be available in system memory.
  3. Process additional chunks: set switches HASHLD and HASHOE, plus other parameters and start the command. The start address of the padded message in system memory and the start address in system memory where the hash state will be loaded from at the start of the command and stored to at the end of the command must be defined.

     After completion of the command, the output message (final digest) will be available in system memory.
  4. Process the final chunk: same settings as additional chunks above.

     The command when completed computes the final digest and writes that to system memory, or uses that to update the RTF.

HASH parameters

Table 286. HASH parameters

Parameter	Register	Description
SRC0_ADDR	DMA_SRC0	Start address in system memory of the padded message to be hashed.
SRC0_LEN	DMA_SRC0_LEN	Length in bytes of the message or message chunk to be hashed.
SRC1_ADDR	DMA_SRC1	Start address in system memory of the hash state (Partial message processing only).
RES0_ADDR	DMA_RES0	Start address in system memory of the output state or output digest. For algorithms SHA-224, SHA-256, the output will be 256 bits (8x 32-bit words). For algorithms SHA-384, SHA-512, the output will be 512 bits (16 x 32-bit words).
HASHMD	CMDCFG0[5:4]	Specifies what SHA algorithm will be used: 0 = SHA-256

Table continues on the next page...

Table 286. HASH parameters (continued)

Parameter	Register	Description
		1 = SHA-224 bits 2 = SHA-384 3 = SHA-512
HASHINI	CMDCFG0[2]	This option is required to support partial processing of the message via each call to HASH(). 0=Do not initialize the hash engine before starting the hash (resume from current internal state). 1=Initialize the internal state of the hash engine as per the initialization rules of the hash standard before starting the hash.
HASHLD	CMDCFG0[3]	This option is required to support partial processing of the message via each call to HASH(). **NOTE** HASHLD is ignored if HASHINI = 1. 0=do not load the hash engine state from system memory before starting the hash. 1=load the hash engine state from system memory before starting the hash.
HASHOE	CMDCFG0[6]	This option is required to support partial processing (processing of part of the message via each call to HASH()). **NOTE** HASHOE must be set if either (1) full mode in which the HASH command will process the entire message, or (2) Partial mode and the chunk processed by the hash command includes the last block of the message. 0=Do not output the hash engine state to system memory at the end of the hash. 1=Output hash engine state to system memory at the end of the hash.
RTFUPD	CMDCFG0[7]	RTF update 0=Do not update the RTF with the result of the hash. 1=RTF will be updated with the result of the Hash command: NEW_RTF = SHA-256(PAD(CURRENT_RTF || SHA-256(message_in))) • ELS will concatenate CURRENT_RTF to the hash of message_in, pad the concatenation in accordance with FIPS-180-4, and hash the padded concatenation. • When RTFUPD is set, HASHMD must = 'b00 (SHA-256). Any other value on HASHMD will trigger an ELS OPN trigger. • RTF update can only be done if the hash of the entire message is available. Therefore RTF update should only be enabled for hash command calls that generate the hash of the entire message. In the case of partial processing this means the hash command that processes the final chunk of the message. • A hash command creates the hash of the entire message must also output the hash. The combination of this and 3 means that if RTFUPD = 1, HASHOE should also be 1.

Table continues on the next page...

Table 286. HASH parameters (continued)

Parameter	Register	Description
		• The RTF can only be output if it has been selected to be updated (RTFUPD = 1). • See Table 287 for a description of how HASH command parameters control whether RTF is output or not.
RTFOE	CMDCFG0[8]	RTF output enable 0=Do not output RTF to system memory 1=Output RTF to system memory The value of RTF at the end of the "HASH and RTF update" operation is appended to the output digest **NOTE** If HASHOE = 0, RTFOE value is ignored, ELS behaves as if RTFOE = 0.

RTF output control

Table 287. RTF output control

RTFUPD	HASHOE	RTFOE	RTF is output
0	x	x	NO
x	0	x	NO
1	1	0	NO
1	1	1	YES

Data structures

For a description of how ELS handles endianness, see Endianness.

Table 288. HASH data structures

Object	Description
Input message: "the challenge"	The challenge must be an integer number of SHA2 blocks as defined by the SHA2 standard. SHA-224, SHA-256: block size is 512 bits, SHA-384, SHA-512: block size is 1024 bits
Output state or output digest	The size of this is SHA-224, SHA-256: 256 bits, SHA-384, SHA-512: 512 bits
RTF	RTF is appended to the output digest or output state if (RTFOE = 1 and RTFUPD = 1). Size of RTF is 256 bits.

13.4.2.14 HMAC command

Introduction

HMAC computes a message authentication code (MAC), compliant with FIPS 198-1 "The Keyed-Hash Message Authentication Code". A key is required for this operation. A single key size of 256 bits is supported. The key can be internal (keystore) or external (system memory).For internal key the user must use ELS commands to create a 256-bit key in keystore. ELS will pad that to 512 bits during the operation of the HMAC command. For external key, the user must pad a 256-bit key to 512 bits and provide that in system memory.

HMAC example sequence

1. Data preparation
    a. The message must be padded in accordance with the FIPS-180-4 Secure Hash Standard and placed in system memory.

> **NOTE**
> The Length field in the message padding must take account of the key as described in Data structures.

2. Set the command parameters and start the command. The start address of the padded message in system memory, or location of the external key, and the start address in system memory where output MAC will be stored, should be defined.
3. After completion of the command, the output message will be available in system memory.

> **NOTE**
> Partial processing is not supported for the HMAC command.

HMAC parameters

Table 289. HMAC parameters

Parameter	Register	Description
SRC0_ADDR	DMA_SRC0	Start address in system memory of the padded message to be processed.
SRC0_LEN	DMA_SRC0_LEN	Length in bytes of the padded message to be processed.
SRC2_ADDR	DMA_SRC2	If EXTKEY = 1, start address in system memory of the external plaintext key, else ignored. **NOTE** This address must be 32-bit word aligned. Unaligned address will trigger an ELS OPN error.
RES0_ADDR	DMA_RES0	Start address in system memory of the output MAC.
KIDX0	KIDX0	If EXTKEY = 0, keystore index of the HMAC Key, else ignored.
EXTKEY	CMDCFG0[13]	Use external Key: 0=use keystore key 1=use external key from system memory

Data structures

For a description of how ELS handles endianness, refer to Endianness

Table 290. HMAC data structures

Object	Description
HMAC key	1. EXTKEY = 0: a 256 bit keystore key must be used: Before using this in HMAC, ELS will automatically pad the key material create a 512 bit padded key as described in FIPS 198-1 Table1: The HMAC Algorithm, step1 - step3. See Keystore keys for information about how to create and use keystore keys. 2. EXTKEY = 1: A 512 bit key must be supplied in system memory. This must be prepared as described in FIPS 198-1 Table1: The HMAC Algorithm, step1 - step3.
Input message: the challenge	The message must be padded to an integer number of SHA-256 blocks, as described below: 1. Zero filling: Zero filling must be applied as described in FIPS-180-4 section 5.1 "padding the Message". 2. The length field of the padding: The length field must be formatted as described in FIPS-180-4 section 5.1 "padding the Message". 3. The value of the length field: This must be adjusted because as part of the HMAC operation, ELS will hash the concatenation of key prepended to message. The value of the length field must be set = length of the unpadded message plus 512 bits.

13.4.2.15 CMAC command

Introduction

CMAC computes a message authentication code (MAC), as defined by NIST SP 800-38b: "Recommendation for Block Cipher Operation:The CMAC Mode for Authentication". A Key is required for this operation. Key Sizes of 128 and 256 bits are supported. The key can optionally be supplied from either keystore or system memory. If the Key is supplied from system memory it must be supplied as plaintext. CMAC supports partial message processing as described in section Partial processing.

NOTE

ELS does not currently support Partial processing with CMAC state held internally. If using partial processing, the CMAC state must be written to/read from external memory.

CMAC example sequence

1. Data preparation
   a. The message must be padded in accordance with the NIST SP 800-38b standard and placed in system memory.
2. Set the command parameters and start the command: The start address of the message in system memory, The keystore key that will be the CMAC key, and the start address in system memory where output MAC will be stored, should be defined.
3. After completion of the command, the output message will be available in system memory.

CMAC parameters

Table 291. CMAC parameters

Parameter	Register	Description
SRC0_ADDR	DMA_SRC0	Start address in system memory of the padded message to be processed
SRC0_LEN	DMA_SRC0_LEN	Length in bytes of the message **before it was padded by the user**. **NOTE** ELS needs the information about the size of the unpadded message, because according to the standard, messages that before padding are not an integer number of blocks require different processing compared to messages that before padding are an integer number of blocks.
SRC1_ADDR	DMA_SRC1	IF (SIE = 1) && (INITIALIZE = 0) Start address in system memory of the input partial mode state. ELSE, ignored
SRC2_ADDR	DMA_SRC2	If EXTKEY = 1: start address in system memory of the external key. Else, ignored. **NOTE** This address must be 32-bit word aligned. Unaligned address will trigger an ELS OPN error.
SRC2_LEN	DMA_SRC2_LEN	If EXTKEY = 1 size in bytes of external key. **NOTE** The size must be one of 16, 32. Any other value will trigger an ELS OPN error.
RES0_ADDR	DMA_RES0	IF FINALIZE = 1: start address in system memory of the output MAC. IF FINALIZE = 0 AND SOE = 1: start address in system memory of the output partial mode state.
KIDX0	KIDX0	If EXTKEY = 0, keystore index of the CMAC Key.
INITIALIZE	CMDCFG0[0]	0=No extra processing 1=Perform the extra processing that is required when the message chunk includes the first block of the message. **NOTE** For processing the whole message in a single call to the command, both INITIALIZE and FINALIZE should be set.
FINALIZE	CMDCFG0[1]	0=No extra processing 1=Perform the extra processing that is required when the message chunk includes the last block of the message.

Table continues on the next page...

Table 291. CMAC parameters (continued)

Parameter	Register	Description
		NOTE Combination SOE = 1 AND FINALIZE = 1 is not supported. If this combination is chosen, incorrect output will be generated. This case does not trigger an ELS error. **NOTE** For processing the whole message in a single call to the command, both INITIALIZE and FINALIZE should be set.
SIE	CMDCFG0[3]	State In Enable For partial message processing, ELS does not currently support CMAC state held internally, only CMAC state held externally is supported. Setting SIE to 0, except in the case mentioned below will produce incorrect output. 1. INIITIALIZE = 0: SIE must be set to 1 2. INITIALIZE = 1: SIE is ignored: may be set to either 0 or 1 SIE is ignored, because CMAC state in is never required for the INITIALIZE operation 3. FINALIZE = 0: SIE must be set to 1 4. FINALIZE = 1: SIE must be set to 1
SOE	CMDCFG0[2]	State out Enable For partial message processing, ELS does not currently support CMAC state held internally, only CMAC state held externally is supported. Setting SOE to 0, except in the case mentioned below will produce incorrect output. 1. INIITIALIZE = 0: SOE must be set to 1 2. INITIALIZE = 1: SOE must be set to 1 3. FINALIZE = 0: SOE must be set to 1 4. FINALIZE = 1: SOE is ignored, may be set to either 0 or 1 SOE is ignored, because CMAC state out is never required for the FINALIZE operation.
EXTKEY	CMDCFG0[13]	0=key is taken from keystore key referenced by KIDX0. 1=key (plaintext) is taken from system memory, referenced by SRC2_ADDR.

Data structures

For a description of how ELS handles endianness, see Endianness.

Table 292. CMAC data structures

Object	Description
Internal key	A 128-bit or 256-bit keystore key must be used. See Keystore keys for information about how the keystore is organized.

Table continues on the next page...

Table 292. CMAC data structures (continued)

Object	Description
External key	A 128 or 256 bit plaintext key in system memory.
Input message	The input message must be padded in accordance with the NIST SP 800-38b standard. **NOTE** For CMAC partial processing, partial data chunks must have a size of 16 bytes or greater, and the size of partial data chunk must be a multiple of 16 bytes.
Output MAC	128 bits
Partial mode state	128 bits

Partial processing

CMAC supports partial processing. See section Partial processing.

For each partial chunk that is processed, the user **must** select via command parameters SOE, SIE, whether the state is:

1. Read from system memory at the start of processing
2. Written to system memory at the end of the command

Table 293. CMAC partial processing

Partial mode	Chunk type	INITIALIZE	FINALIZE	SIE	SOE	Description [1] [2]
Full	n/a	1	1	X	X	The chunk to be processed contains the entire message.
Partial	INITIAL (INITIALIZE = 1)	1	0	X	1	Processing a partial chunk, and the chunk includes the first block of the message. The first block of the message requires special processing.
Partial	INTERMEDIATE INITIALIZE = 0 && FINALIZE = 0	0	0	1	1	Processing a partial chunk, and the chunk includes neither the 1st block nor the last block of the message.
Partial	FINAL (FINALIZE = 1)	0	1	1	X	Processing a partial chunk, and the chunk includes the last block of the message, but does not include the first block.

1. X means that the value of the parameter is ignored.
2. Setting SIE = 1 only makes sense if the state was output at the end of processing the previous chunk.

13.4.2.16 RND_REQ command

Introduction

RND_REQ generates a random number of requested size and stores that in system memory. The user can select the source of the random number between:

- DRBG: Processed entropy
- TRNG: Raw entropy

RND_REQ can also be used to get ELS entropy level to "low" level. This is achieved via the LOW_ENT_INIT command switch. This feature allows the user to get ELS to "low" level entropy directly.

The LOW_ENT_INIT approach is quicker than the alternative "indirect" approach of getting ELS entropy level to "low" by running an ELS command that requires low entropy.

RND_REQ example sequence

1. Data preparation
    a. Free a section of system memory large enough for the output random number.
2. Set the command parameters and start the command. The start address of the output random number in system memory and the requested size in bytes of the random number should be defined.
3. After completion of the command, the random numbers will be available in system memory.

RND_REQ parameters

Table 294. RND_REQ parameters

Parameter	Register	Description
RES0_ ADDR	DMA_RES0	Start address in system memory of the output random number.
RES0_LEN	DMA_ RES0_LEN	• RND_RAW=DRBG: Requested size in bytes of the random number. Supported size range is 4 bytes to 2^16 bytes. Requesting a size outside the range will trigger an ELS OPN error. **NOTE** The requested size must be a multiple of 4. Using a size that is not a multiple of 4 will trigger an ELS OPN error. • LOW_ENT_INIT = 1 or RND_RAW = TRNG: RES0_LEN is ignored: It is not used and cannot trigger an ELS OPN error.
LOW_ENT_I NIT	CMDCFG0[0]	Control bit to force ELS state into a minimum LOW_ENTROPY state. 1 = Wait for TRNG low entropy and use that to initialize DRBG and PRNG. ELS is now initialized with low entropy (DRBG_ENT_LVL = low). **NOTE** When LOW_ENT_INIT = 1, ELS does not output a random number. 0 = Wait for TRNG high entropy, then initialize DRBG and output a random number from the source selected by RND_RAW.

Table continues on the next page...

Table 294. RND_REQ parameters (continued)

Parameter	Register	Description
RND_RAW	CMDCFG0[1]	Control bit to select the random number source 0 = DRBG (processed entropy) 1 = TRNG (raw entropy) **NOTE** Ignored if LOW_ENT_INIT = 1.

RND_REQ data structures

Table 295. RND_REQ data structures

Object	Description
Output random number	The output random number is written to system memory. See RES0_LEN above for supported size range of the random number.

RND_REQ error

When RND_REQ is run with LOW_ENT_INIT = 1, ELS does some additional internal checks. If the checks fail, an ELS FLT error is triggered.

13.4.2.17 DRBG_TEST command

Introduction

DRBG_TEST is provided to support FIPS CAVP testing. It has the following modes:

- DRBG_instantiate: Instantiate the DRBG using 256 bits of entropy fetched from system memory.
- DRBG_extract: Extract random data from the DRBG after it has been instantiated. This mode implements the same functionality as the RND_REQ command without the restriction that the ELS entropy level must be HIGH. DRBG_extract shall be used in conjunction with DRBG_instantiate to allow FIPS CAVP testing of the DRBG.

  NOTE

  For correct operation, DRBG_extract must be run after DRBG_instantiate with no other ELS commands in between.

- AES CTR test: Allows AES counter mode of the DRBG to be used to encrypt data provided by the system. It is provided to allow FIPS CAVP testing of the DRBG AES CTR mode.
- AES ECB test: Allows the AES ECB mode of the DRBG to be used to encrypt data provided by the system. It is provided to allow FIPS CAVP testing of the DRBG AES ECB Mode.

  NOTE

  Running DRBG_TEST command may temporarily modify the entropy of ELS using entropy from an external source.

For normal operation, ELS should use entropy from its own internal source. So after completing DRBG_TEST, ELS will automatically start the process of gathering entropy from its own internal source. At this point, the entropy level that ELS can immediately reach depends on the state of its internal entropy source. Therefore on completion of DRBG_TEST, the ELS entropy

level may be any of the 3 levels-- none, low or high. Because of that, the run time of commands that are run after DRBG_TEST may include a delay while ELS waits for the required entropy level for the command to be reached.

DRBG_TEST example sequence

1. Data preparation
    a. Free a section of system memory large enough for the data that is output from the command.
    b. Place the required input data in system memory.
2. Set the command parameters and start the command. See the parameters table below.
3. After completion of the command, the output message will be available in system memory.

DRBG_TEST parameters

Table 296. DRBG_TEST parameters

Parameter	Register	Description
SRC0_ ADDR	DMA_SRC0	Case (DRBG_TEST_MODE) • DRBG_instantiate: Start address in system memory of the input entropy data • DRBG_extract: Ignored • AES CTR test: Start address in system memory of the data to be encrypted • AES ECB test: Ignored
SRC0_LEN	DMA_SRC0_LE N	Case (DRBG_TEST_MODE) • DRBG_instantiate: ignored. The Size of the of the input entropy data is fixed, 32 bytes (256 bits), ELS does not need the parameter because it determines the size automatically. • DRBG_extract: ignored. • AES CTR test: Length in bytes of the data to be encrypted. Must be a non-zero multiple of 16. • AES ECB test: ignored.
SRC1_ADD R	DMA_SRC1	Case (DRBG_TEST_MODE) • DRBG_instantiate: ignored. • DRBG_extract: ignored. • AES CTR test: Start address in system memory of the Initialization Vector (IV) and Key. • AES ECB test: Start address in system memory of the Data to be encrypted and Key.
RES0_ ADDR	DMA_RES0	Case (DRBG_TEST_MODE) • DRBG_instantiate: ignored. • DRBG_extract: Start address in system memory of the output random number. • AES CTR test: Start address in system memory of the output encrypted data. • AES ECB test: Start address in system memory of the output encrypted data.

Table continues on the next page...

Table 296. DRBG_TEST parameters (continued)

Parameter	Register	Description
RES0_LEN	DMA_ RES0_LEN	Case (DRBG_TEST_MODE) • DRBG_instantiate: ignored. • DRBG_extract: Requested size in bytes of the output random number. Supported size range is 4 bytes to 2^16 bytes. Requesting a size outside the range will trigger an ELS OPN error. • AES CTR test: ignored. • AES ECB test: ignored.
DRBG_TEST _MODE	CMDCFG0[1:0]	• 0=DRBG_instantiate: Instantiate the DRBG using 256 bits of entropy fetched from system memory. • 1=DRBG_extract: Extract random data from the DRBG after it has been instantiated. This mode implements the same functionality as the RND_REQ command without the restriction that the ELS entropy level must be HIGH. DRBG_extract shall be used in conjunction with DRBG_instantiate to allow FIPS CAVP testing of the DRBG. • 2=AES CTR test: Allows AES counter mode of the DRBG to be used to encrypt data provided by the system. It is provided to allow FIPS CAVP testing of the DRBG AES CTR Mode. • 3=AES ECB test: Allows the AES ECB mode of the DRBG to be used to encrypt data provided by the system. It is provided to allow FIPS CAVP testing of the DRBG AES ECB Mode.

Data structures

Table 297. DRBG_TEST data structures

Object	Description
SRC0_DATA	Case (DRBG_TEST_MODE) • DRBG_instantiate: Input entropy data. This is fixed size 32 bytes (256 bits). The data should be placed so that the leftmost 32-bit word of the string as it appears in the CAVP vector is be placed at the **highest** system memory address. • DRBG_extract: N/A • AES CTR test: Data to be encrypted. The data should be placed so that the leftmost 32-bit word of the string as it appears in the CAVP vector is be placed at the **highest** system memory address. • AES ECB test: ignored.
SRC1_DATA	Case (DRBG_TEST_MODE) • DRBG_instantiate: N/A • DRBG_extract: N/A

Table continues on the next page...

Table 297. DRBG_TEST data structures (continued)

Object	Description
	• AES CTR test: The IV and Key. Both are of size 16 bytes (128 bits). They should be structured as follows: — The IV should be placed in the lower half of the structure, that is, at lower system memory addresses than the Key. — Both IV and Key should be placed so that the leftmost 32-bit word of the string as it appears in the CAVP vector is placed at the **highest** system memory address. • AES ECB test: The Data and Key. Both are of size 16 bytes (128 bits) for a total of 32 bytes. They should be structured as follows: — The Data should be placed in the lower half of the structure, that is, at lower system memory addresses than the Key. — Both Data and Key should be placed so that the leftmost 32-bit word of the string as it appears in the CAVP vector is placed at the **highest** system memory address.
DRBG OUTPUT DATA	Case (DRBG_TEST_MODE) • DRBG_instantiate: N/A • DRBG_extract: Size in bytes of the output entropy data will be as specified by DMA_RES0_LEN. The output data shall be placed so that the leftmost 32-bit word of the string as it appears in the CAVP vector is at the **lowest** system memory address. • AES CTR test: The output data shall be placed so that the leftmost 32-bit word of the string as it appears in the CAVP vector is at the **lowest** system memory address. The output data size is the same as the input data size specified by SRC0_LEN. • AES ECB test: The output data shall be placed so that the leftmost 32-bit word of the string as it appears in the CAVP vector is at the **lowest** system memory address. The output data size is fixed: 16 bytes (128 bits).

13.4.2.18 DTRNG_CFG_LOAD command

ELS automatically configures its internal TRNG during start-up. However, ELS supports overwriting the default (start-up) configuration of the TRNG, with a new configuration loaded from system memory. A scenario where that might be needed, is where silicon characterization shows that the default config of the internal RNG is not optimal. In that case, an improved config can be downloaded and used.

The overwrite action is implemented as the DTRNG_CFG_LOAD command.

NOTE

Any ELS reset will "revert" the TRNG to the default config. If a non-default configuration is required, that configuration must be reloaded, using DTRNG_CFG_LOAD, after any ELS reset.

DTRNG_CFG_LOAD example sequence

1. Data preparation
    a. Place the TRNG config to be loaded in system memory.
2. Set the command parameters and start the command.
    a. The start address in system memory of the TRNG config to be loaded, must be supplied

3. The command when completed starts the entropy gathering process of ELS. See Random numbers and entropy.

DTRNG_CFG_LOAD parameters

Table 298. DTRNG_CFG_LOAD parameters

Parameter	Register	Description
SRC0_ ADDR	DMA_SRC0	Start address in system memory of the TRNG config to load

DTRNG_CFG_LOAD data structures

For a description of how ELS handles endianness, see Endianness

Table 299. DTRNG_CFG_LOAD data structures

Object	Description
Config	The TRNG configuration to load. The TRNG configuration is NXP-supplied. Size in bytes is 84.

Integrity protection of TRNG configurations

TRNG configurations are integrity protected. Loading a configuration with incorrect protection will trigger an ELS ITG error.

Summary of errors that can be triggered by this command

Table 300. Errors triggered by DTRNG_CFG_LOAD

Event	Error category
Starting DTRNG_CFG_LOAD when the command is not supported, or not enabled	OPN
Loading a configuration with incorrect integrity protection	ITG
Failure of the ELS to reach high entropy. **NOTE** As stated above, DTRNG_CFG_LOAD triggers the ELS to start gathering entropy. If the entropy gathering process fails, an error will be triggered. The timing of that would be some time after DTRNG_CFG_LOAD has completed.	FLT

13.4.3 Endianness

The endianness of the data that ELS reads and writes from system memory is important for the correct operation of ELS. ELS has different requirements for the byte endianness, 32-bit word endianness and block endianness, and these are each described below.

Convention about "most significant" and "least significant": If a string of binary objects (bytes, 32-bit words, or blocks) is written in the form "object_0, object_1 ... object_n", then the convention used in this document is that object_0 is the most significant object, and object_n is the least significant object.

1. **Block endianness**
   a. Definition: If a binary string is expressed in blocks as "blk0, blk1... blkn", and the blocks are placed in system memory.
      i. If block_0, the most significant block, is stored at the lowest memory location, that is block BIG endian.
      ii. If block_0, the most significant block, is stored at the highest memory location, that is block LITTLE endian.

      NOTE
      A "block" is an arbitrary length string of binary bits.

   b. Requirements: The user is responsible for ordering the blocks in system memory to comply with ELS block endianness requirements. ELS block endianness requirements are listed per command and for each data structure used by the command.
2. **32-bit word endianness**
   a. Definition: If a binary string is expressed in 32-bit binary words as "word_0, word_1... word_n", and the words are placed in system memory.
      i. If word_0, the most significant 32-bit word, is stored at the lowest memory address, that is 32-bit word BIG endian.
      ii. If word_0, the most significant 32-bit word, is stored at the highest memory address, that is 32-bit word LITTLE endian.
   b. Requirements: The user is responsible for ordering the 32-bit words in system memory to comply with ELS 32-bit word endianness requirements. ELS 32-bit word endianness requirements are listed per command and for each data structure used by the command.
3. **Byte endianness**
   a. Definition: If a binary string is expressed in 8 bit binary bytes as "byte_0, byte_1... byte_n", and the bytes are placed in system memory.
      i. If byte_0, the most significant byte, is stored at the lowest byte address in memory, that is byte BIG endian.
      ii. If byte_0, the most significant byte, is stored at the highest byte address in memory, that is byte LITTLE endian.
   b. Requirements: The user is free to use either little or big endian byte ordering in system memory. However, the user must inform ELS of the byte ordering via CTRL[BYTE_ORDER]. For BYTE_ORDER, 0=byte little endianness and 1=byte big endianness. ELS needs this information to ensure that the correct byte order is maintained for the data when it is transferred to ELS internal storage.

Endianness requirements by data structure

Table 301. Per data endianness settings

Command	Data structure	Block endianness	32-bit word endianness
CIPHER	MESSAGE IN MESSAGE OUT	BIG	BIG
AUTH_CIPHER	IV IN AAD IN MESSAGE IN	BIG	BIG

Table continues on the next page...

Table 301. Per data endianness settings (continued)

Command	Data structure	Block endianness	32-bit word endianness
	MESSAGE OUT TAG OUT		
ECSIGN, ECVFY	DIGEST/CHALLENGE	N/A	If Challenge is pre-hashed (is a digest) then: REVF = 0 Digest 32-bit word endianness is assumed to be LITTLE REVF = 1 Digest 32-bit word endianness is assumed to be BIG If Challenge is unhashed then ENDIANNESS=BIG
HASH HMAC CMAC	MESSAGE IN	BIG	BIG

Data example

The table below shows how endianness affects the storage of the 128 bit ascii string "GHIJKLMNOPQRSTUV". In this example the block size is 64 bits, so there are two, 32-bit words per block.

Table 302. Endianness data example

Block endian	32-bit word endian	Byte endian	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
little	little	little	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G
little	little	big	S	T	U	V	O	P	Q	R	K	L	M	N	G	H	I	J
big	big	little	J	I	H	G	N	M	L	K	R	Q	P	O	V	U	T	S
big	big	big	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V

13.4.4 Partial processing

Partial processing is the technique of splitting the input data for an ELS command into "chunks" and processing each chunk with a separate call to the command. ELS commands can only operate one at a time, cannot be executed in parallel, so ELS commands that take a long time can potentially block shorter commands that might be needed urgently. Partial processing avoids that by allowing a single ELS command that processes a lot of data to be split into a series of calls to the same command with each call processing a subset of the data.

Some commands that support partial processing have the concept of state, that is, before a command starts processing the next "chunk" of data, it must have access to "state" information resulting from the previously processed chunk. This is managed in one of two ways, the choice of which is selectable by the user.

- **State held internally:** At the end of chunk processing, the state information is held internally. In this case there is no need to load the state information from system memory before starting the next chunk.

  This approach will give the best performance.

  NOTE

  — Not all ELS commands support "state held internally" see the table below.

  — The user must take care not to run other ELS commands that might overwrite the state. See the table below.

- **State I/O**: At the end of chunk processing, the state information is output to system memory. In this case, before starting to process the next chunk, the state must be reloaded from system memory. With this approach the user is free to run commands that overwrite the internal state.

Table 303. Partial Mode with state held internally: supported scenarios

Command	State held internally	Not supported command sequences (sequences that corrupt the internally stored state)
CIPHER	**Not supported**	
AUTH_CIPHER	**Not supported**	
HASH	Supported	1. HASH: a hash command that is unrelated to the in progress sequence of partial mode of HASH commands 2. HKDF HMAC 3. TLS_INIT
CMAC	**Not supported**	

13.4.5 Random numbers and entropy

ELS maintains random numbers for use either internally by ELS commands or externally. The user can request ELS to output random numbers to system memory. The random numbers can have three different entropy levels. The currently available entropy level is reported to the user via an ELS register field. The ELS commands require specific entropy levels in order to run. ELS will pause a command if the entropy level is too low to run the command.

Entropy level reporting

ELS entropy level is reported in STATUS[DRBG_ENT_LVL]. The encoding is shown in the table below.

Table 304. Entropy level reporting

Entropy level	STATUS[DRBG_ENT_LVL]
NONE	2'b00

Table continues on the next page...

Table 304. Entropy level reporting (continued)

Entropy level	STATUS[DRBG_ENT_LVL]
LOW	2'b01
HIGH	2'b10
This value is reserved/not used	2'b11

Getting random numbers from ELS

ELS can provide random numbers to the system as follows:

- Numbers with LOW level entropy can be read from the PRNG_DATOUT register.

  See PRNG for details.

- Numbers with HIGH level entropy can be output from ELS via the RND_REQ command.

  See RND_REQ command for details.

Entropy requirements of commands

Each ELS command has a minimum entropy level requirement. If a command is started when the entropy level is below its minimum requirement, the command will be paused until ELS has gathered enough entropy for the command to proceed.

The entropy requirements of ELS commands are shown in the tables below.

Table 305. Entropy requirements of commands

Minimum entropy requirement	Command
NONE	HKDF TLS_INIT HASH HMAC DRBG_TEST DTRNG_CFG_LOAD
LOW	CIPHER AUTH_CIPHER KEYIN KEYOUT ECVFY ECKXCH KDELETE CKDF CMAC RND_REQ(LOW_ENT_INIT = 1)

Table continues on the next page...

Table 305. Entropy requirements of commands (continued)

Minimum entropy requirement	Command
HIGH	ECSIGN KEYGEN RND_REQ(LOW_ENT_INIT = 0)

Explicitly waiting for entropy

In some use cases the user may want to explicitly wait for ELS entropy to reach a certain level. This is done as follows:

- Low Entropy: Use the RND_REQ command with the LOW_ENT_INIT switch set to 1.
- High Entropy: Execute any ELS command that needs high entropy.

Entropy gathering sequence

ELS will always attempt to raise the available entropy to the maximum level, which is "high".

ELS achieves this by:

1. Configuring the internal TRNG to gather entropy.

   The process of gathering TRNG is always incremental, that is, if the TRNG has no entropy, ELS configures it to gather low entropy. If the TRNG has low entropy, ELS configures it to gather high entropy.

   The process of gathering TRNG entropy happens in the background, that is, in parallel to ELS commands. However, ELS can only react to an increase in the available TRNG entropy when a foreground task is executed (that is, an ELS command).

2. Increasing the entropy level of the internal DRBG by seeding it with TRNG entropy.

   The "ELS entropy level" (DRBG_ENT_LVL) represents the entropy level of the DRBG.

   DRBG seeding can only be triggered by a foreground task (that is, by starting an ELS command). If at the start or end of an ELS command, TRNG entropy is available, with level greater than the current DRBG entropy, then DRBG will be re-seeded. As a result, by the time the command completes, ELS entropy level will have increased, either from none to low or low to high.

ELS entropy can also be reduced by some ELS operations of events described below

Table 306. Entropy gathering sequence

Event	Entropy level before event	Entropy level immediately after operation or event
Any reset	N/A	None
DTRNG_CFG_LOAD	None	None
	Low or high	Low
DRBG_TEST	N/A	See Introduction.

Effect of wait for entropy on the duration of ELS commands

As mentioned above, ELS commands will pause to gather entropy if the entropy is too low at the start of the command. The pause will affect the run time of the command.

13.4.5.1 PRNG

ELS has an internal PRNG, and it allows direct access to this via the PRNG_DATOUT register.

Suggested operation

1. To make PRNG data available, ELS must enable the internal PRNG, and seed it with random data. That can only happen when ELS DRBG has a minimum of low entropy. The user must follow the procedure below to ensure that this has happened.
    a. Enable ELS and wait for startup to complete.
    b. Check the ELS DRBG entropy level (STATUS[DRBG_ENT_LVL]).

    If DRBG_ENT_LVL is LOW or HIGH, then proceed to next step.

    Else if DRBG_ENT_LVL is NONE: run a RND_REQ command with parameter LOW_ENT_INIT = 1. At the end of the command, DRBG_ENT_LVL will be low, and PRNG will have been enabled and seeded.
    c. Wait for PRNG ready (STATUS[PRNG_RDY] = 1).
2. As soon as PRNG data is available (PRNG_RDY = 1), good quality data can be read 32 bits at a time from PRNG_DATOUT.

> **NOTE**
> User must check PRNG_RDY = 1 before reading from PRNG_DATOUT. Reading PRNG_DATOUT while PRNG_RDY = 0 will trigger an ELS PRNG_ERR error.

13.4.6 Pointer registers

ELS has a number of pointer registers that indicate the system memory location of various data objects used by ELS commands.

The pointer registers are:

- DMA_SRC0
- DMA_SRC1
- DMA_SRC2
- DMA_RES0

The pointers are used as follows:

- Before starting a command, the user must write valid system memory addresses to whatever pointers are required by the command.
- Pointer usage is command dependent. Each pointer may be used for a different purpose by each command.

> **NOTE**
> The pointer usage for each command is described in Commands overview in the Parameters subsection of each command.

13.4.7 Startup phase

Startup phase is an internal sequence in which ELS re-initializes itself. Start up can be triggered by:

- Asynchronous reset followed by setting CTRL[ELS_EN]: See ELS enable.

> **NOTE**
> ELS can assert an interrupt when the startup time is complete. In order to make use of this feature, the INT_ENABLE[INT_EN] bit must be set before CTRL[ELS_EN] is set.

- Synchronous reset, (providing that ELS is already enabled when the synchronous reset occurs).

- ELS error categories with a level 1 or level 2 response.

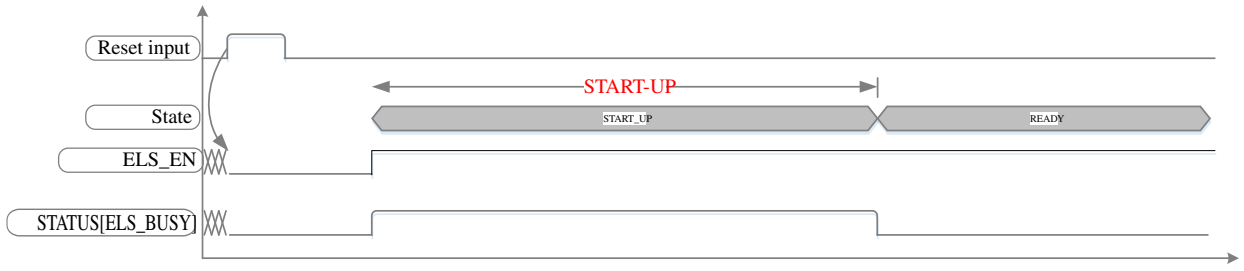


Figure 69. Startup following an asynchronous reset

The sequence of events within startup is as follows:

1. Access to all registers is allowed (takes effect immediately).
2. Register bit STATUS[ELS_BUSY] and async reset are set (takes effect immediately).
3. ELS does internal initialization.
4. Keystore clear

   NOTE

   This is described in Table 309.

   When startup is triggered by setting CTRL[ELS_EN] after an async reset Keystore is **NOT** cleared. No keys are deleted.

   When startup is triggered by a sync reset or an ELS error, Keystore **IS** cleared: all keys are deleted
5. On completion of internal initialization,
   a. STATUS[ELS_BUSY] cleared to 0.
   b. An ELS interrupt is triggered:

      NOTE

      If the interrupt was triggered while INT_ENABLE[INT_EN] was set, the interrupt must be cleared before starting an ELS command.
6. ELS is now ready to accept commands.

13.4.8 Clocking

Table 307. ELS clocks

Clock	Description
pclk	APB bus interface clock for the IP. Clocks the APB bus interface logic.
hclk	AHB bus interface clock for the IP. Clocks the AHB bus interface logic.
i_dtrng_ref_clk	TRNG reference clock. A fixed frequency reference clock that is used by the ELS TRNG submodule to set the duration of entropy gathering operations.

ELS has two levels of clock switching described in Power control and power domains.

13.4.8.1 Power control and power domains

ELS power is controlled by switching off the internal clocks. ELS has two levels of clock switching.

Global clock enable

To reduce power when ELS is not in use, ELS has a global clock control bit, CTRL[ELS_EN]. The operation is described in the table below.

Table 308. ELS_EN operation

Value	Behavior
0	ELS internal clock is switched off. AHB clock to ELS AHB bus master is switched off within ELS. Internal clock to ELS public registers is switched off within ELS. Most ELS public registers are **inaccessible**: Access to these will result in an APB slave error. There are some exceptions, see ELS enable.
1	All clocks within ELS are switched on. All ELS public registers are accessible.

Clock control of ELS internal submodules

To reduce run-time power, ELS also switches off clocks to individual submodules when they are not in use. That control is transparent to the user.

13.4.9 Reset

ELS has three reset sources:

- Asynchronous hardware reset (referred to from now on as **async reset**).
- Synchronous reset (referred to from now on as **sync reset**). Activated by setting CTRL[ELS_RESET].
- Synchronous reset triggered by ELS error (referred to from now on as **error reset**). Activated by detection of an error.

Each reset source triggers one or more actions, described in the table below.

See ELS enable for a description on how to enable ELS.

Table 309. Reset sources and actions

Action	Async reset	Sync reset	Error reset
Any running ELS command is stopped	YES	YES	YES
ELS is busy until the reset completes (see ELS busy and idle states)	NO	YES	YES
ELS public registers are reset	YES	YES, except INT_ENABLE	YES, except INT_ENABLE

Table continues on the next page...

Table 309. Reset sources and actions (continued)

Action	Async reset	Sync reset	Error reset
ELS enable state after the reset	Disabled (CTRL[ELS_EN] must be set to enable ELS)	Enabled	Enabled
Is ELS in ERROR state after the reset Refer to the description in Error state.	NO	NO	YES
Keystore state	All keys are invalidated. The KACTV field in the key status registers cleared. All other key status register fields are also cleared, except for the "hardware feature" fields.	All keys are invalidated. The KACTV field in the key status registers cleared. All other key status register fields are also cleared, except for the "hardware feature" fields.	Depends on error level. See Table 311.
TRNG and Entropy	1. TRNG configuration is replaced by internally supplied low entropy configuration (effect of any previous DTRNG_CFG_LOAD is removed). 2. ELS entropy gathering restarts 3. After release of reset, user must enable ELS to trigger startup. After startup, ELS will go idle. At that point DRBG_ENT_LVL will be "none".	1. TRNG configuration is replaced by internally supplied low entropy configuration (effect of any previous DTRNG_CFG_LOAD is removed). 2. ELS entropy gathering restarts 3. Sync reset will trigger startup. After startup ELS will go idle. At that point DRBG_ENT_LVL will be "none".	Depends on error level. See Table 311.
Sync or async reset during error reset	Async reset will terminate the response that is in progress due to the error reset. Resulting ELS response is the same as the actions already described in this column.	Startup will be triggered by the sync reset, with all the corresponding actions described in this column, This response startup will replace the response that had previously been triggered by the error reset. As a result The duration of ELS busy will be extended. ELS error is not cleared by sync reset, so after startup ELS will be in error state.	N/A

Table continues on the next page...

Table 309. Reset sources and actions (continued)

Action	Async reset	Sync reset	Error reset
RTF	RTF is restored to its default value	RTF is restored to its default value	depends on error level: refer to Table 311.

13.4.9.1 ELS enable

ELS may be enabled and disabled to manage its power consumption.

- ELS is disabled by async reset. This happens because the ELS global enable bit, CTRL[ELS_EN], is cleared by async reset. After release of async reset, ELS may be enabled by setting CTRL[ELS_EN].

  After startup has completed, ELS may be disabled by clearing CTRL[ELS_EN] and re-enabled by setting CTRL[ELS_EN].
- Enabling ELS for the first time causes a start up. See Startup phase for details.
- After the first start-up has happened, ELS can be disabled and re-enabled without causing another start-up (clearing and setting CTRL[ELS_EN]).
- ELS can assert an interrupt when the startup time is complete. To make use of this feature, INT_ENABLE[INT_EN] must be set before CTRL[ELS_EN] is set.
- When ELS is disabled:
  - All internal clocks are gated off. This includes the internal parts of the APB and AHB bus clocks.
  - Access to all registers is blocked, (except for CTRL and INT_ENABLE). For the registers that are not blocked, both read access and write access (if supported normally), are permitted. The read only access to SESSION_ID is also permitted. Access to a blocked register triggers bus error but does not trigger an ELS error.
  - Setting CTRL[ELS_START] or CTRL[ELS_RESET] has no effect.

13.4.10 ELS busy and idle states

When ELS is not running it is idle. When ELS is running an operation, it is busy, and when it completes the operation, it goes back to idle. The valid operations are:

1. Any ELS command
2. Startup phase
3. Sync reset: see Reset.
4. Error triggered sync reset: see Table 311.

ELS indicates that it is busy by setting STATUS[ELS_BUSY].

ELS indicates idle by clearing STATUS[ELS_BUSY].

NOTE

In addition to the busy flags, ELS interrupt can also be used to flag the transition from busy to idle. See Interrupts.

ELS behavior during Busy

While ELS is busy the following rules apply:

1. Starting an operation (by setting CTRL[ELS_START]) is not permitted. Starting an operation will cause an ELS OPN error.
2. Writes to ELS registers during busy will trigger an ELS OPN error.

   Exceptions to this rule are:

   a. Software reset by setting CTRL[ELS_RESET] is permitted, will reset ELS, and will not cause an error.

13.4.11 Interrupts

ELS has an interrupt that can be used to flag some ELS events. The interrupt is flagged via STATUS[ELS_IRQ].

1. ELS interrupt means the following:
    a. Interrupt status bit STATUS[ELS_IRQ] is set.
2. ELS interrupt occurs in the following cases:
    a. Completion of any ELS command.
    b. Completion of startup phase: see Startup phase.
    c. Completion of sync reset: see Reset.
    d. Completion of error triggered sync reset: see Table 311.
    e. Software triggered interrupt (see below).
3. **Software Triggered Interrupt**: The user can also cause an ELS interrupt by setting INT_STATUS_SET[INT_SET]. This action has the same effect as a hardware triggered interrupt.
4. **Interrupt Clear:** ELS interrupt can only be cleared by software. This is done by writing 1 to INT_STATUS_CLR[INT_CLR]. Clearing an interrupt clears STATUS[ELS_IRQ].

NOTE

If INT_ENABLE[INT_EN] was set when the interrupt was triggered, then starting an ELS command without clearing the interrupt will trigger an ELS OPN error.

13.4.12 DMA

ELS DMA is under direct control of the ELS internal state machine. ELS uses DMA to move data to and from system memory as part of the execution of ELS commands. User interaction with ELS DMA is limited to setting the target addresses in system memory that the DMA will access during the execution of ELS commands. You can control these addresses via parameters that are passed to ELS commands. See DMA final address readback for more information.

13.4.13 Errors

ELS errors are categorized. Each category has an associated status bit in the ERROR_STATUS register. When an error within the category is detected, the corresponding error status bit is set. The purpose of the categories is:

- to determine how ELS will respond and
- to allow the ELS user to decide if any additional response is needed.

The below table describes the ELS error categories.

Table 310. Error categories (error status bits)

Error categories (error status bits)	Description
FLT_ERR	Fault error: ELS has detected an internal fault that represents a deliberate attempt to interfere with its operation.
ITG_ERR	Integrity error: An integrity check on ELS internal data has failed.
ALG_ERR	Algorithm error: An internal algorithm has produced an unexpected result. This can happen occasionally and is caused by dependencies between some internal algorithms and the

Table continues on the next page...

Table 310. Error categories (error status bits) (continued)

Error categories (error status bits)	Description
	random data that is used by them. In cases when this error occurs, the error will likely not recur if the ELS command is repeated because a different set of random data will be used on the second attempt.
OPN_ERR	Operational error. ELS has been incorrectly operated. For example user has attempted to start an ELS command while ELS is busy.
BUS_ERR	Bus access error
PRNG_ERR	User read of PRNG_DATOUT when STATUS[PRNG_RDY] = 0.
DTRNG_ERR	Triggered when TRNG configuration cannot generate entropy. 1. If DTRNG_ERR is triggered after reset and before any DTRNG_CFG_LOAD has been run, this will not trigger an ELS error, but will set the DTRNG_ERR register bit. 2. If DTRNG_ERR is triggered after any DTRNG_CFG_LOAD has been run, this will trigger an ELS error. In either case above, DTRNG_ERR can be cleared by the usual ELS error clear procedure. In addition, in case 1 above user should be aware that running DTRNG_CFG_LOAD will clear DTRNG_ERR.

Error responses

ELS has 3 levels of response to errors, with level 2 being the highest level and level 0 the lowest. The response to an error is determined by the error category. This is described in the table below.

NOTE

After an error response has been triggered, the level of the response that occurred can be read from ERR_STATUS[ERR_LVL].

Table 311. Error response levels

Response Level	Associated error categories	Response sequence
Level 0	PRNG_ERR	1. STATUS[ELS_ERR] is set 2. If a command is executing, it **DOES NOT** terminate 3. ELS is **NOT** reset 4. KEYSTORE and RTF are preserved

Table continues on the next page...

Table 311. Error response levels (continued)

Response Level	Associated error categories	Response sequence
		5. Entropy level is preserved 6. ELS enters ERROR state
Level 1	OPN_ERR BUS_ERR ALG_ERR	1. STATUS[ELS_ERR] is set 2. If a command is running, it terminates immediately 3. ELS error reset occurs: see Reset **NOTE** ELS interrupt will occur when error reset completes. 4. KEYSTORE and RTF are preserved 5. Entropy level is preserved 6. On completion of reset, ELS enters ERROR state
Level 2	FLT_ERR ITG_ERR DTRNG_ERR after DTRNG_CFG_LOAD has been run	1. STATUS[ELS_ERR] is set 2. If a command is running, it terminates immediately 3. ELS error reset occurs: Refer to Reset **NOTE** ELS interrupt will occur when error reset completes. 4. KEYSTORE is cleared: All keys deleted 5. RTF is reset to default value 6. Entropy level returns to low 7. On completion of reset, ELS enters ERROR state

Error state

As stated above, the final step of any level of ELS error response is for ELS to enter ERROR state. In ERROR state:

- ELS will not respond to further commands.
- ELS error status bit will remain set.
- Writes to registers (except CTRL[ELS_RESET], and ERR_STATUS_CLR[ERR_CLR] will be ignored.

NOTE

This means:

1. It is not possible to start an ELS command when in error state.
2. It is not possible to clear an interrupt when in error state. To clear the interrupt, first exit error state.

Exit from error state and error flag clearing

Error state should exit by:

1. Wait for STATUS[ELS_BUSY] to clear.
2. Clear the error by setting ERR_STATUS_CLR[ERR_CLR].

NOTE

Multiple or continuous writes to ERR_CLR without confirming that the error has cleared (see next step below) can result in the error **not** being cleared correctly.

3. Poll STATUS[ELS_ERR] to confirm that the error has been cleared.

NOTE

Because more than one clock domain is present on ELS, error clearing can take a variable number of PCLK cycles. For this reason after clearing STATUS[ELS_ERR] the user must poll to confirm that the error has cleared.

4. Clear the outstanding interrupt by setting INT_STATUS_CLR[INT_CLR].

On exit from error state:

1. All error status flags are cleared.
2. ELS again respond to commands.

NOTE

Asserting a hardware reset will also cause an exit from error state, but the recommended way to exit is by setting ERR_CLR.

NOTE

A synchronous reset during error state will cause ELS to be reset, but when the reset completes, ELS will remain in error state.

The figure below shows the sequence for entry and exit of error state.

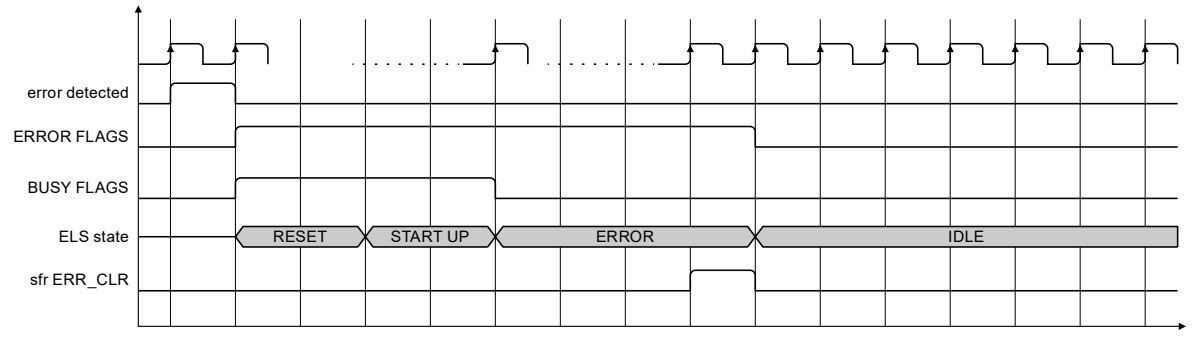


Figure 70. Error state: entry and exit

Error Flags

The ELS Error flags are the Error Status bits in register ERR_STATUS. The error flags are normally at 0. When an error occurs, one or more of the error status bits in register ERR_STATUS is set. The Error Status bits describe the type of error. Errors are categorized to enable the user of ELS to decide on a response.

Operation [OPN] errors

The table below lists the cases that will set cause an OPN error.

Table 312. Scenarios that trigger OPN Errors

Rule	Scenario
Only 1 ELS operation can run at a time.	Command start is requested while ELS is busy.
Only supported commands must be run.	Command start is requested for an unsupported command.
If INT_ENABLE[INT_EN] was set when an ELS interrupt was triggered, the interrupt must be cleared before starting an ELS command.	After an ELS interrupt has been triggered with INT_ENABLE[INT_EN] set, a command is started without first clearing the interrupt.
Only Valid TRNG configurations may be downloaded.	TRNG configuration download has detected an invalid configuration.
KEYIN(FMT=PUF) can only be run one time between ELS resets.	Attempting to re-run KEYIN(FMT=PUF) without first resetting ELS.
Only active keys can be used. An active key is a key slot with the properties KBASE, and KACTV set, in its base slot.	An invalid key has been used. One of the key indexes used by a command point to an invalid key.
Only a key with valid properties can be used. Refer to Key usage rules.	A key has been used for a purpose not permitted by its key properties. For example, CIPHER command. The key referenced by KEY_IDX0 has a usage property other than UAES set.
When a key is created by an ELS command, the properties requested for the created key must comply with Key creation rules: Setting the usage control properties.	
ELS must not be disabled while it is busy.	Clearing CTRL[ELS_EN] while ELS busy indicators are set.
Register writes while ELS is busy are not permitted (with some exceptions), refer to ELS behavior during Busy.	
Address range of an ELS data structure must be between 0 and (2^32)-1.	Running an ELS command with parameters set such that the address range exceeds the limits during the command.
Commands that use external keys. Start address of the key must be 32-bit word aligned.	Supplying a 32-bit word unaligned address for the external key.

[BUS] Errors

A BUS category Error will be triggered by any of the following cases:

- Errors on the ELS APB slave bus (system accesses to ELS registers)

  - Access to an unmapped block address
  - Write to a read-only register
  - Read from a write only register
- Errors on the ELS AHB master Bus (ELS access to system addresses)

13.4.14 Access control and access rights

When ELS is used it captures the access rights of the user and uses these as follows:

- In cases when the user access causes ELS to access the system AHB bus, ELS will propagate the users access rights to the system AHB bus accesses.
- Keystore keys are access controlled. When a user attempts to either use, or create a keystore key, ELS will allow or block the operation, depending on the access rights of the user and the access right settings of the key.
- Access to some ELS commands is access right controlled.

Access rights that are used by ELS

Access rights are communicated to the ELS via an internal hardware connection. To help with understanding, the access right information is referred to below as PPROT[1:0].

Access rights contain two pieces of information: privilege level and security level.

NOTE

Privilege level and security level are independent of each other, and when compared (or checked), a separate comparison is done for each type.

The possible values of privilege level and security level are described in the table below.

Table 313. Access right definitions

Bit	Meaning	Values
PPROT[0]	Privilege level	0: access is non-privileged 1: access is privileged
PPROT[1]	Security level	0: access is secure 1: access is non-secure

Capture of user access rights

ELS captures user access rights by sampling the PPROT[1:0], in the cycle when an ELS command is started. Once captured, the access rights are valid until the next ELS command is started.

Access control to keystore keys

This is described in section Key access control (PPROT).

Access control to ELS commands

The DTRNG_CFG_LOAD command can only be run when user access rights are secure and privileged.

13.4.15 Keys

ELS uses cryptographic keys for many of its commands. There are 2 types of keys:

- **External**: Stored outside ELS, in system memory. External keys can be accessed by the CPU. These keys can exist in 2 forms.
  - Plaintext: These keys are not protected against discovery by the system.
  - Wrapped: Key that has been encrypted by ELS. Wrapped keys are protected against discovery by the system.
- **Keystore:** Stored in ELS, in a bank of internal 128 bit registers (each register is called a key slot). The Keystore keys are deliberately isolated from the system outside of ELS. Specifically, their values (the Key material) cannot be accessed by the system. This is achieved by making them private. They do not appear in the ELS public interface (the register map). The system can reference them via a key indexing scheme that is described later. Also, their properties (for example, size and purpose) are directly accessible via the ELS register map.

Keystore keys

Keystore keys are structured as an array. Each array element contains 128 bits of key material. These are referred to as **key slots**. When a specific Keystore Key is referred to in this document, the notation used is KEY0 the first key, KEY1 the second key, etcetera.

When a Keystore key is used by an ELS command, the command parameters will include a key index parameter that allows software to specify which Keystore Key to use.

To prevent improper use of keys, the Keystore Keys have associated properties, and fixed rules that apply to the properties. More detail will be given in the sections below.

NOTE

Keystore keys **cannot** be accessed directly from the external bus. They are created via ELS commands, and accessed for use by the key index parameters described above.

NOTE

Not all commands use a Keystore Key. Some commands do not use any key. Others may fetch a plaintext external key and use it directly instead of using a Keystore Key.

Hardware out slots

Some keystore slots are connected directly to system hardware. The purpose of this is to allow ELS to export plaintext keys directly to the system hardware. To support key isolation, no Keystore key is directly accessible by system software, but the hardware out slots, are available for direct use by system hardware. See the chip-specific section for details on hardware slot usage on this device.

Hardware out slots are also flagged in the key status registers by the key feature field FHWO = 1.

NOTE

The misuse of hardware out slots is covered by both explicit and implicit protection:

- Explicit protection: Either writing a UHWO = 1 key in a FWHO = 0 slot, or writing a UHWO = 0 key in a FWHO = 1 slot is not allowed and will trigger an ELS OPN error.
- Implicit protection: A key derived for a hardware out slot should have usage control property UHWO = 1. If the user attempts to create a key with UHW0 = 0 in the slot, the key material for the bad key will be different and when the keys is used, the operation it is used for will fail.

Keystore key sizes

As stated above Key slots are a fixed size, 128 bits each. For 128 bit keys, each key uses one key slot. For larger keys, more than 1 key slot is used. Currently the largest supported key size is 512 bits or 4 slots.

The table below is provided to demonstrate the use of more than one slot to store larger keys. It shows how the key sizes are supported by key slots. It also shows how key properties are used to indicate the sizes of keys and which key slots they occupy. See Keystore key properties and status.

Table 314. Key sizes, key slots, and properties

KEY SIZE (bits)	KEY LOCATION	Key property KACTV	Key property KBASE	Key property KSIZE
512	SLOT_N	1	1	11b
	SLOT_N+1	1	0	11b
	SLOT_N+2	1	0	11b
	SLOT_N+3	1	0	11b
256	SLOT_N	1	1	01b
	SLOT_N+1	1	0	01b
128	SLOT_N	1	1	00b

Keystore key creation

A Keystore key is created by running an ELS command. There are several ELS commands that create keys.

- ECKXCH: Key exchange command outputs a shared secret that is written to keystore and treated as a key.
- KEYGEN: ECC key generation command creates an asymmetric key pair, and the private part is written to keystore.
- KEYIN (KFMT=PUF): Transfers device unique key from external PUF to keystore.
- KEYIN (KFMT=RFC3394: Decrypts an RFC3394 encrypted key and writes the decrypted key to keystore.
- CKDF, HKDF: Derives a child key from a master key and writes the new key to keystore.
- TLS_INIT: Either derives a master key from a pre-master key or derives child keys from a master key.

A key index parameter of the command selects which key slot(s) will receive the new key.

NOTE

If the new key will occupy multiple key slots, the key index should refer to the lowest (base) slot.

Key creation: Overwriting an existing key is an error

Occupied key slots must be freed before re-use by deleting them with the KDELETE command.

It is an error to overwrite an existing key. When a key is created, if any of the Keystore slots that it will be written to is occupied by an existing key, an OPN error will be triggered.

Key creation: Setting the properties of a new key and the KPROPIN request field

When a key is created, its properties must be set to ensure that is used correctly. Properties may be set either:

- **By user request:** The user requests properties of the new key by setting the relevant bits of the KPROPIN register.

NOTE

The bits of KPROPIN match exactly to the bit allocations of the Keystore key properties and status table that is defined in a later section.

NOTE

There are restrictions on some combinations of command and property. See next entry, **"Automatic"**.

- **Automatic:** For some combinations of key creation command and key property, the user is not permitted to set the property. In that case, ELS will automatically set the property.

  NOTE

  For details on automatically setting properties, refer to the detailed command descriptions. Each command has a subsection describing which properties are set automatically.

- **Inherent:** The "hardware feature" key properties are inherent to the underlying key slot and cannot be changed by user request.

Keystore key use

Keystore keys are referred to by the index parameters of ELS commands that require keys. The key index parameter selects one of the keys for use with the current command. When a command needs to access 2 keys, a second key index parameter is also used.

For each key that is used for a command, ELS will check that it is being used correctly. Refer to key properties and key rules below.

The size of the used key(s) must match the required key sizes for the ELS command. ELS will check this and return an error if there is a mismatch.

Keystore key indexes

Keystore keys are organized in 128-bit "slots". For example, a 128 bit key would require 1 slot, and a 256 bit key would require 2 slots. Key index values align with such slots, so for example a 256 bit key transferred using a destination key index of 0 would be stored in slot0 and slot1 respectively.

The key indexes are contained in registers KIDX0 and KIDX1. See ELS register descriptions.

Keystore key properties and status

Each Keystore Key has associated properties. Some properties give information about the key (for example key size 128 or 256 bit). Other properties describe the intended usage of the key.

The usage properties are interpreted by ELS to ensure that the keys are used properly (for example, a key used to derive another key should never be used to encrypt/decrypt a message). If a key is used improperly, ELS will flag an error.

The properties of a key are **set** when a key is created.

The properties are visible by reading the KSn registers. Each key slot has an associated key status register.

If a key uses more than one slot, the properties of the key can be read from the status register of any slot that the key occupies. The KBASE property is an exception to this; KBASE is set to 1 for the lowest (base) slot of a key and cleared to 0 for all other slots.

The properties are listed in the table below.

Table 315. Key properties

BIT	Key property	Property type	Description
[31:30]	UPPROT	Access control	In order for a key to be used, the user access rights of the user must be greater than or equal to the access rights required by UPPROT[1:0]. Follow carefully the procedure for comparing these, which is described in Key access control (PPROT).

Table continues on the next page...

Table 315. Key properties (continued)

BIT	Key property	Property type	Description
29	DUK	Identifier	Device Unique Key: A key whose value is unique per die and is secret, that is, not known outside of ELS. It is used as a Master Key. Other keys and secrets are derived from it. This property is used to infer some rules that are specific to DUK.
28	WRPOK	Access control	The key can be wrapped.
27	UHWO	Usage control	To be used in a hardware out slot, that is, a slot that has FHWO set.
26	UKGSRC	Usage control	Can provide the key material input for a KEYGEN command.
25	UTLSMS	Usage control	TLS master secret key: to be used as the input key to command TLS_INIT(FINALIZE = 1).
24	UTLSPMS	Usage control	TLS pre-master secret key: to be used as the input key to command TLS_INTI(FINALIZE = 0).
23	UKUOK	usage control	Key Unwrapping only key: can be used to unwrap another key but not to wrap another key. To be used in the command KEYIN(fmt = rfc3394).
22	UKWK	Usage control	Key wrapping key: to be used to either wrap **OR** unwrap another key.
21	UHMAC	Usage control	HMAC key: to be used as the key for the HMAC command.
20	UAES	Usage control	AES key: to be used as the key for CIPHER and AUTH_CIPHER commands.
19	UECDH	Usage control	ECC Diffie Hellman Private Key: to be used as the private key for Diffie Hellman Key exchange to create a shared secret.
18	UECSG	Usage control	ECC signing key: to be used as the private key for ECSIGN command.
17	UHKDF	usage control	HMAC based derivation key: to be used as the Master Key in the HKDF command.

Table continues on the next page...

Table 315. Key properties (continued)

BIT	Key property	Property type	Description
16	UCKDF	Usage control	CMAC based derivation key: to be used as the master key in the CKDF command.
15	URTF	Usage control	RTF signing key: allows a key to do a 'with RTF signing' operation. With RTF signing is an option of the ECSIGN command.
14	UKSK	Usage control	Key signing key: allows a key to sign a keystore key. A key signing key is used in the KEYGEN command when the "sign the output public key" option is enabled.
13	UCMAC	Usage control	CMAC key: to be used as the key for the CMAC command.
12	UTECDH	Usage control	Private key that can be only used with a trusted public key (UKPUK): cannot be used with a public key from system memory. Also, can only be used in a key exchange (ECKXCH command).
11	UKPUK	Usage control	Trusted public key
10	Reserved	Usage control	
9	FHWO	Hardware feature status	Hardware out slot: 1 indicates the slot is connected to a key mux.
8	FRTN	Hardware feature status	Retention slot: 1 indicates that the slot will retain its value when the system is put in power down mode.
7	FGP	Hardware feature status	General Purpose: 1 indicates that the slot is neither a retention slot nor a hardware out slot.
6	KBASE	Usage status	Key Base Slot: 1 indicates that the key is the base slot of a key that uses more than 1 slots.
5	KACTV	Usage status	Active Slot: 1 indicates that the slot is active and is in use by a key.
[4:2]	Reserved	n/a	
[1:0]	KSIZE	Usage status	key size: 0 = 128 bits (1 slot) 1 = 256 bits (2 slots)

Table continues on the next page...

Table 315. Key properties (continued)

BIT	Key property	Property type	Description
			2 = RFU 3 = 512 bits (4 slots) **NOTE** For a 256 bit key, KACTV will be set to 1 in both slots.

Keystore key rules - overview

ELS enforces usage rules based on the properties of keys. Before an ELS command is run, the key rules are checked and if a rule is broken, an OPN error will be flagged, and ELS error response will be triggered. For a description of the error response, see Errors.

For rules about overwriting keys, see Key creation: Overwriting an existing key is an error.

Key creation rules: Setting the usage control properties

The table below shows which usage control properties can be set when a key is created. The rules depend also on what command is being used to create the key. The first section of the table takes the form of a black list, that is, it explicitly describes the cases that are **not allowed.** All other cases can be assumed allowed.

Table 316. Key creation rules: usage control

Command	DUK	UCKDF	UHKDF	UECSG	UECDH	UAES	UHMAC	UKWK	UKUOK	UTLSPMS	UTLSMS	UKGSRC	UHWO	URTF	UKSK	UCMAC	UKPUK	UTECDH
KEYIN(RFC 3394)	Not ok							Not ok					Not ok	Not ok	Not ok			
CKDF from DUK	Not ok																Not ok	
CKDF from non-DUK	Not ok							Not ok					Not ok	Not ok	Not ok		Not ok	Not ok
HKDF FROM non-DUK	Not ok							Not ok					Not ok	Not ok	Not ok		Not ok	Not ok
ECKXCH	Not ok							Not ok					Not ok	Not ok	Not ok		Not ok	Not ok

Table continues on the next page...

Table 316. Key creation rules: usage control (continued)

Command	DUK	UCKDF	UHKDF	UECSG	UECDH	UAES	UHMAC	UKWK	UKUOK	UTLSPMS	UTLSMS	UKGSRC	UHWO	URTF	UKSK	UCMAC	UKPUK	UTECDH
The table section below describes the usage properties that might be set (dependent on command params) by ELS commands in which property setting is determined automatically by ELS, rather than being determined by user request.																		
TLS_INIT						Might be set	Might be set		Might be set		Might be set							
KEYGEN				Might be set	Might be set									Might be set	Might be set			Copied from KPROPIN if KGTYPEDH = 1
KEYIN(FMT=PUF)	Set	Set																
KEYIN(FMT=PBK)																	Set	
The table section below describes the usage control properties that are set automatically by ELS, on the keys that are output by TLS_INIT.																		
Output key																		
TLS MS											1							
TLS CEK						1			1									
TLS SEK						1												
TLS SMK							1											
TLS CMK							1											

Key creation rules: setting the WRPOK property

This section describes when key property WRPOK may be set. The rule depends on what usage control properties are being set.

It is OK to set WRPOK in the following commands that create a key:

- ECKXCH
- CKDF from DUK
- CKDF from non-DUK
- HKDF

For KEYIN(RFC3394), WRPOK is not checked when a key is created via keyin. That is because ELS RFC3394 wrap/unwrap operation is trusted. If a key is unwrapped, it must previously have been wrapped, that is, WRPOK must have been set. So, assuming no other errors occurred during KEYIN, WRPOK = 1 is allowed for any key that can be unwrapped by KEYIN.

For commands, TLS_INIT and KEYGEN, that do not accept user input for WRPOK, WPROK is set as follows:

- TLS_INIT:
  - FINALIZE = 0, WRPOK is set to 0
  - FINALIZE = 1, WRPOK is set to 1
- KEYGEN: WRPOK is set to 1 in all cases

Key Creation Rules: Size and position checks

When a key or keys are created by an ELS command the following are checked.

1. Overrun: The highest slot needed by the key must be smaller than the highest available slot of keystore.
2. Overwrite: All slots needed by the key must be free. The user is expected to free the slots via KDELETE command, before writing new keys via other commands.

Key creation rules: Hardware out slots

Only a key with UHWO = 1 can be written to a slot that supports hardware out (feature property FHWO = 1).

Also, a key with UHWO = 1 cannot be written to a slot with FHWO = 0.

NOTE

The hardware out slot keys have to be 128 bits.

Key usage rules

For the KEYGEN command, if KGSIGN = 1 and KGSRC = internal, then usage will be checked on two keys. This is the only case where this happens, and it affects the rules for KEYGEN. The two keys checked are:

1. Key Material: Key material check
2. Signing Key: Signing key check. That is described on a separate row of the table below.

Some Commands require a specific usage right property to be set on the Key that is used. This is indicated as **req** in the table below. The usage check will fail if the key does not have this property set.

For a given command, if req appears in more than one column command it means that at least one of the properties must be set, but both may be set.

- ECSIGN:
  - If SIGNRTF = 0, only UECSG may be set, URTF must not be set.
  - **RNOTE1**: If SIGNRTF = 1, at least one of UECSG, URTF must be set.
- KEYGEN: KGSIGN = 1, signing key check

— **RNOTE2**: At least one of UECSG, UKSK must be set.

In general, a key should only have one usage right. However there are exceptions. These are indicated as additional **ok** or **req** entries in a table row below and explained by the notes below.

NOTE

Occasionally **nok** will appear in the table. This is to highlight a special case where a property must not be set.

A blank entry in the table also indicates that a property may not be set.

- CIPHER, AUTH_CIPHER, KEYIN(RFC3394)
  - **ENOTE1**: UAES = 1 AND UKUOK = 1 is allowed
- ECSIGN
  - **ENOTE2**: UECSG = 1, AND URTF = 1 is allowed: (see above)
- KEYGEN key material check
  - If (KGSRC = internal) then UKGSRC is required; it must be set.
  - **ENOTE3**: In addition, none of, or one of, but no more than one of UHKDF, URTF, UKSK may be set .
  - UHKDF set is allowed to cover the case of the DICE KEY CDI, which must be both an EC signing key and a HKDF derivation key.
- HKDF
  - **ENOTE4:** UHKDF = 1 AND UKGSRC = 1 is allowed. This is to cover the case of the DICE KEY CDI, which must be both an EC signing key and a HKDF derivation key.
- USE OF INTERNAL PUBLIC KEYS
  - **RNOTE3**: When ECVFY or ECKXCH are run with 'Use internal public key' selected (EXTKEY = 0), the keystore key that is used must have UKPUK set.
  - **ENOTE4**: When ECKXCH is run, and the private key has UTECDH set, then the public key must be fetched from keystore (command parameter EXTKEY must be 0).

For information on usage rules for DUK and WRPOK see Key usage: DUK property and Key usage: WRPOK property.

Table 317. Key usage rules

Command	UCKDF	UHKDF	UECSG	UECDH	UAES	UHMAC	UKWK	UKUOK	UTLSPMS	UTLSMS	UKGSRC	UHWO	URTF	UKSK	UCMAC	UKPUK	UTECDH
CIPHER					req			Ok, ENOTE1									
AUTH_ CIPHER					req			Ok, ENOTE1									
ECSIGN			req														

Table continues on the next page...

Table 317. Key usage rules (continued)

Command	UCKDF	UHKDF	UECSG	UECDH	UAES	UHMAC	UKWK	UKUOK	UTLSPMS	UTLSMS	UKGSRC	UHWO	URTF	UKSK	UCMAC	UKPUK	UTECDH
SIGNRTF = 0 or ECHASHCHL = 0																	
ECSIGN SIGNRTF = 1			req, RNOTE1, ENOTE2										req, RNOTE1, ENOTE2				
ECKXCH				req												req, RNOTE3	ENOTE4
ECVFY																req, RNOTE3	
KEYGEN Key material check		Ok, ENOTE3									req		Ok, ENOTE3	Ok, ENOTE3			
KEYGEN Signing key Check			req, RNOTE2											req, RNOTE2			
KEYIN (FMT=RFC3394)					Ok, ENOTE1		req	req									

Table continues on the next page...

Table 317. Key usage rules (continued)

Command	UCKDF	UHKDF	UECSG	UECDH	UAES	UHMAC	UKWK	UKUOK	UTLSPMS	UTLSMS	UKGSRC	UHWO	URTF	UKSK	UCMAC	UKPUK	UTECDH
KEYIN (FMT=PBK)																req	
KEYOUT							req										
CKDF	req								(DUK = 1): Not ok, Else: Ok								
HKDF		req									Ok, ENOTE4						
TLS_INIT (FINALIZE = 0)	Ok								req								
TLS_INIT (FINALIZE = 1)										req							
HMAC						req											
CMAC															req		

Key usage: DUK property

DUK = 1 is only permitted for the CKDF command

Key usage: WRPOK property

In general, WRPOK = 1 is permitted for any key and any ELS command. An exception to this is the CKDF command when DUK = 1. In that case, WRPOK must not be set.

Key usage size rules

When a key is used, the following size rules are checked:

- HKDF: Key size must be 256 bits.
- CKDF: Size of the input key must be greater than or equal to the size of the output key.
- If the output key is a hardware out key (UHWO = 1), the output key size must be 128 bits.

Key access control (PPROT)

User access control

System hardware communicates to ELS, the access rights of the user. A decode of the possible settings of the user access right information is given for convenience here using the arbitrary label "PPROT". See Access control and access rights.

PPROT[0]: 0=non-privileged access; 1= privileged access

PPROT[1]: 0=secure access; 1=non-secure access

Key access controls

Keystore keys have corresponding usage access control bits UPPROT[1:0]. The bits are public readable and located in the Key Status registers (as stated above, there is one key status register per key slot).

Setting key access controls during Key creation

When a key is created by an ELS command, the access control value for the key must be set. This is done according to the following procedure.

1. The access control value UPPROT[1:0] of a created key is set by:
    a. Request: setting the bits of KPROPIN register that correspond to UPPROT[1:0] in Table 315.
    b. Unwrapped: see Key access control checks for key wrapping and unwrapping.

    ,
2. Before the Key is created, the access control value is checked against the access rights of the user. The access rights of the user must be greater than or equal to the access control value of the created key.
3. The checking rule is similar to the access control during key usage check described in a previous paragraph. The access rights of the user must be greater than or equal to the access control value of the created key.
4. If the check passes, the command continues. Else, the command aborts, no key is created, and an ELS OPN error is triggered.

Key access control checks during key usage

The bits define the access rights that a user must have in order to use the key. The rule is: the access rights of the user must be **greater than or equal to** the user access control level of the key.

> **NOTE**
>
> The bits are **independent** of each other, so the ELS implementation is that the above rule is checked separately for each bit. So from user perspective, each of the 2 bits can be treated separately, and each must comply with the above rule.

Example: UPPROT of the key requested for use is secure, privileged, and the users access rights is secure, unprivilege. The key requires secure, privileged, or higher, and the user is secure, non-privileged. The user's request fails due to the check on the privileged/non-privileged part of the access right.

Key access control checks for key deletion

1. The access control value of the key which will be deleted or overwritten is checked against the access rights of the user.

2. The access rights of the user must be greater than or equal to the access control value of the created key.
3. Violating the rule will trigger an ELS OPN error.

Key access control checks for key wrapping and unwrapping

- When a key is wrapped by the KEYOUT command, a key access control check is triggered.
  - The access rights of the user must be greater than or equal to the access control value of key to be wrapped.
  - In addition, the access control value of the wrapping key must be greater than or equal to the access control value of the key to be wrapped.
  - Violating the rules will trigger an ELS OPN error.
- Unwrapping (KEYOUT): When a key is unwrapped by KEYIN with format set to RFC3394, the key properties are encrypted along with the key material. The access control value (UPPROT) is restored by decrypting the Key. In this case, key access control checks are triggered.
  - The access rights of the user must be greater than or equal to the access control value of the key that is unwrapped.
  - In addition, the access control value of the key that is used to unwrap must be greater than or equal to the access control value of the key that is unwrapped.
  - Violating the rules will trigger an ELS OPN error.

External key locations and sizes

The location of an external key is normally specified in a parameter to the ELS Command that accesses that key.

The size of an external key is specified by a size parameter that is associated with the location parameter above. For example:

1. The CIPHER command receives the external key location via parameter SRC2_ADDR.
2. Correspondingly, CIPHER receives the external key size (in bytes) via parameter SRC2_LEN.

13.4.16 Run Time Fingerprint (RTF)

RTF is a 256-bit internal register that can be used to support flows that attest the system hardware and firmware.

An example of such a sequence is given below.

- ELS can be configured to output the result of the HASH command to RTF. When configured in this way, the current value of RTF is included in the HASH input data, so the output of each successive hash with RTF depends on the results of all previous hash with RTF. In this way, HASH with RTF can be used to build a cumulative digest from multiple sections of system memory. This can be used as a representative digest of system firmware. The creator of the firmware should be able to independently recreate the same digest value.
- To additionally prove that the attested firmware is running on trusted hardware, ELS also supports gathering of a "hardware signature", the hardware attestation data (HAD) from the system.
- The actual attestation consists of a challenge response step.
  - A verifier creates a challenge, which is sent to ELS. ELS concatenates the challenge, the RTF and the HAD, signs the result with a trusted signing key and returns the signature together with the HAD, to the verifier. The verifier has the challenge and HAD and can recreate RTF. So the verifier can independently recreate the concatenated message and use that to verify the signature returned by ELS.

ELS features to support RTF

To support RTF, ELS has the following features:

- RTF is located in an internal ELS register that is not accessible via the register map (referred to from now on as RTF).

NOTE

The access restriction mentioned above is to prevent attempts to produce a fake copy of RTF, that is, one that has not been created by hashing the system memory.

- RTF is initialized to the constant value 256'h0 by any ELS reset.
- RTF can be updated via the ELS HASH command, as follows:
  1. The input message to HASH is hashed as normal, then the resulting digest is appended to the current value of RTF and hashed again. Refer to the RTFUPD parameter in HASH parameters for details of the algorithm that is used to combine RTF with the hashed message.
  2. The result is stored in RTF.

NOTE

This approach allows more than one section of system memory to be hashed if required, with the resulting RTF value being determined by the combined hash of all the sections.

- After RTF has been updated from Hashes of system memory, it can be "added" to a message to be signed (Challenge), and the result signed. Because the resulting signature depends on the value of RTF which in turn depends on the system memory contents, a correct signature provides proof that the system memory contents are as expected.

  Refer to the SIGNRTF parameter in ECSIGN parameters for details of the algorithm that is used to add RTF to a message to be signed by ECSIGN command.
- For debug purposes the current value of RTF can be output to system memory by the HASH command. Refer to the RTFOE parameter in HASH parameters.

13.4.17 Public key import

ELS has the ability to import an ECC public key and store it in keystore. Trusted public keys may be used as:

- The public key input to an ECKXCH command.
- The public key input to an ECVFY command.

To be importable to ELS, the public key must be embedded in a data structure. The data structure can be a public key certificate in a standard format, but does not have to be. For the rest of this chapter, the data structure containing the public key to be imported is referred to as the **certificate**, and the public key contained in certificate is referred to as the **embedded public key**. Once imported from the certificate to ELS, the key becomes a **trusted public key**.

To be importable by ELS, the certificate must have been signed with the private key of **ELS ECC root key pair**. ELS ECC root key pair is created offline by NXP trust provisioning team. The private and public parts of ELS ECC root key pair are referred to in this chapter as **root private key** and **root public key**.

Importing the embedded public key

Embedded Public keys are imported by the KEYIN command with PBK import enabled (KEYIN(PBK)). As stated above, the certificate must have been signed with root private key. To confirm this, before importing embedded private key, KEYIN(PBK) first verifies the certificate, using root public key and a signature that must be supplied by the user.

NOTE

Root public key must already be in keystore when KEYIN(PBK) is started.

To import root public key to keystore, it must be imported from an RFC3394 blob using KEYIN command with RFC3394 enabled (KEYIN(RFC3394)). That is a separate operation which must happen before KEYIN(PBK).

Importing the embedded public key: required user steps

Data preparation

1. The certificate must be placed in system memory.

2. Separately, the signature, must be placed in system memory (signature is the result of signing the certificate with the root private key).
3. Root public key is a provisioned key. As stated above, it is created offline by NXP trust provisioning.

   As stated above, the root public key must be imported to keystore from an RFC3394 blob using KEYIN(RFC3394). That operation must have happened before the embedded public key is imported.

Setting KEYIN command parameters for importing the embedded public key

1. Set up ELS pointers to the certificate:
   a. SRC0_ADDR specifies the base address (byte address).
   b. SRC0_LEN specifies the size in bytes.
2. Set up an ELS pointer to indicate the **address offset** of the embedded public key within the certificate. SRC2_ADDR is used.

> **NOTE**
>
> The value written to SRC2_ADDR should be the **byte offset** from certificate base to embedded public key base.
>
> For example, if the system memory base address (bytes) of the certificate is 0x800, and the system memory base address of the public key is 0x840, then 0x040 should be written to SRC2_ADDR.

> **NOTE**
>
> See Table 264 for an example of the data structure of a certificate.

3. Set the ELS pointer to the base address of signature: SRC1_ADDR is used.
4. Set the ELS index to the output key (the trusted public key): KIDX1 is used.
5. Set the ELS index to the root public key: KIDX2 is used.
6. Set an ELS pointer to R that is output by KEYIN(PBK): RES0_ADDR is used.

   R is the R component of the ECC signature. It is calculated by the ECC verify sub-operation of KEYIN(PBK) and output to system memory.

   ELS outputs R to system memory to allow the user to do an independent comparison of R against the expected value of R contained in the signature.

> **NOTE**
>
> ELS also automatically compares the internally calculated R against the expected value of R as part of KEYIN(PBK).

Checking the status of the public key import

After the import (KEYIN(PBK)) has run:

1. The internal checks run by ELS during the command should have passed. As with any ELS command, the error status of ELS should be checked after the command has completed.
2. Optionally, the user may also check that the calculated ECC signature calculated during KEYIN(PBK) matches the expected ECC signature. As stated above, ELS does this check automatically during the command, but also outputs the calculated signature to allow the user to do the check independently.

Importing the embedded public key: integrity check

Due to capacity limitations of ELS, it is not possible to download the entire certificate to ELS. This means that the embedded public key is transferred to ELS in a separate operation after the certificate has been verified.

To ensure that the embedded public key is not modified between verification of the certificate, and transfer to ELS, the embedded public key is CRC'd internally by ELS during both operations, and the 2 CRCs are compared.

Using a trusted public key

A trusted public key is a key that has been imported from a certificate using KEYIN(PBK). A trusted public key is a key that will have usage property UKPUK = 1. A trusted public key can optionally be used by the ECKXCH and ECVFY commands. These commands can select whether to use a trusted public key from keystore or an untrusted key from system memory.

- ECKXCH
  - The command parameter EXTKEY controls whether the public key used by ECKXCH is loaded from system memory (untrusted public key) or from keystore (trusted public key).
  - If a trusted public key is used, the command parameter KIDX2 specifies the base index of the key in keystore.
  - A trusted public key must have usage property UKPUK set. Otherwise, an OPN error is triggered.
  - The private key that is used by ECKXCH may require to be used with a trusted public key. This is controlled by the UTECDH property of the private key.

    When ECKXCH command is executed with a private key which has the UTECDH property then the public key must be fetched from Keystore (EXTKEY = 0) otherwise an OPN error is triggered.
- ECVFY
  - The command parameter EXTKEY controls whether the public key used by ECVFY is loaded from system memory (untrusted public key) or from keystore (trusted public key).
  - If a trusted public key is used, the command parameter KIDX2 specifies the base index of the key in keystore.
  - A trusted public key must have usage property UKPUK set. If not, an OPN error is triggered.

13.4.18 Hardware semaphore

ELS has a Hardware Semaphore to support systems that use Multi CPU or Threaded software. The Semaphore will allow ELS to be shared in a multiprocessor or threaded software environment and to be allocated to a single software process or specific system CPU at any specific time.

Master ID

The ELS hardware semaphore functionality defines a system master ID which identifies the master that is accessing ELS. See the chip-specific information for details on ELS master ID values used for this device.

ELS semaphore lock / unlock

ELS semaphore feature is unlocked by default, that is, after asynchronous or synchronous reset. When UNLOCKED ELS freely accepts register accesses and command executions from any SW process or master, that is, regardless of the value Master ID value. ELS is LOCKED by a read from the SESSION_ID register while ELS is in UNLOCKED state.

Register fields used by semaphore

Table 318. Register fields used by semaphore

Register	Access	Field	Description
STATUS	Read only	ELS_LOCKED	HW semaphore locked status 0=unlocked, 1=locked
SESSION_ID	Read Write	SESSION_ID	Session ID
MASTER_ID	Read Write	MASTER_ID	One time write register containing the privileged

Table continues on the next page...

Table 318. Register fields used by semaphore (continued)

Register	Access	Field	Description
			master ID which can be used to unlock ELS without the session ID. Reset by Sync or Async Reset.

High level operation sequence

This describes a token-based (Session ID) solution in which system CPUs or threads must follow the protocol described below:

1. Thread reads SESSION_ID register of ELS.
2. If UNLOCKED, ELS issues a 32-bit nonzero session ID back to thread (as part of the read above).
3. Thread writes ELS registers to configure and start an ELS command.
4. Operations are repeated as required by the thread (that is, a sequence of one or more ELS register accesses takes place).
5. When thread has finished its sequence, it unlocks ELS by writing the original session ID back to SESSION_ID.

Detailed operation: Actions, Events and States

- Reset action: On any ELS reset
  - The session state (referred to as "state" from now on) is initialized to UNLOCKED.
  - The session ID (register field SESSION_ID) is reset to a deterministic non-zero value.
- Lock action
  - Procedure
    - Read SESSION_ID when the session state is UNLOCKED. The read will return the current value of the session ID.
  - Effects
    - State changes to LOCKED
    - Further read accesses to the SESSION_ID register will return zero as long as the STATE remains LOCKED.
- Unlock action
  - Procedure #1: Normal operation unlock
    - The SESSION_ID register must be written with the session ID value that was read when semaphore was locked.

      AND
    - The transaction PPROT must match the value of PPROT that was latched by ELS, when it was locked.

      AND
    - The transaction master ID (value on port i_sem_master_id) must match the master ID value that was recorded when ELS was locked.
  - Procedure #2: Privileged master unlock
    - The SESSION_ID register must be written, but the value can be any value.

      AND
    - The transaction PPROT must be "secure, privileged".

AND

- The transaction master ID (i_sem_master_id port) must match MASTER_ID[MASTER_ID] value OR, if the privileged master ID = 'h1F, transaction master ID can be any 5 bit value.

— Effects

- The state changes from LOCKED to UNLOCKED.
- A new session ID value is generated by ELS.

NOTE

The session ID changes each time ELS is unlocked. If session ID stayed the same, a previous master could unlock the current session because the current session ID would be the same as the previous session ID.

- Session ID error EVENT
  - — Description: A session ID error is an APB bus error caused by an incorrect access to the session ID.
  - — Triggering condition: A session ID error is triggered by the following:
    - Incorrect write to session ID when state is LOCKED.

      That is, any write in which neither the conditions of normal operation unlock nor privileged master unlock are met (see descriptions above).
    - Write to any register without matching the values of PPROT and MASTER_ID to the values used when ELS was locked.

      NOTE

      CTRL[ELS_RESET] is an exception: writing to CTRL[ELS_RESET] will not trigger a session ID error.
    - READ of any ELS register with PPROT and MASTER_ID values that do not match the value used when ELS was locked.

      Exception: Reads from STATUS and SESSION_ID registers are permitted and will not trigger a session ID error.
  - — Behavior: A session ID error appears as a normal APB bus error. It does not trigger an ELS error.

    The APB bus write that triggered the session ID error will not be blocked. The target register will be written by the APB bus write.

    If ELS is LOCKED when this error occurs, it will remain LOCKED.
- Privileged master ID access error EVENT
  - — Description: A privileged master ID access error is an APB bus error caused by an incorrect access to the privileged master ID.
  - — Triggering condition: After reset, privileged master ID can be written one time. After it is written, an attempt to write it again without first resetting ELS will trigger the error.

    NOTE

    As stated in "session ID error event" above, writing any register, including privileged master ID, when ELS is locked will trigger a session ID error.
  - — Behavior: A privileged master ID error does not trigger an APB bus error. Nor does It trigger an ELS error. However, in case of a privileged master ID error, the master ID is not changed.

    If ELS is LOCKED when this error occurs, it will remain LOCKED.

Other error EVENTS

- APB Write of SESSION_ID when unlocked.

- Read of any **NON** "general access" ELS registers without matching PPROT and MASTER_ID when locked.

  General access registers are defined as registers accessible to any process these are :

  — STATUS

  — SESSION_ID

NOTE

Any of the errors listed above triggers a level 0 error (APB pslverr) which is no reaction internal to ELS. If the ELS is locked it will remain locked until unlocked by the locking process or it is reset.

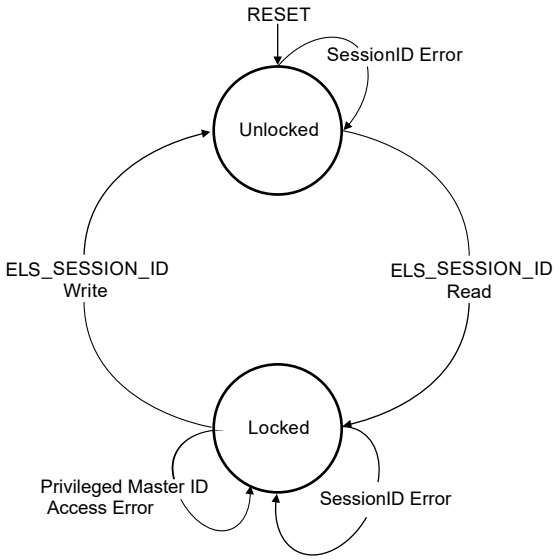


Figure 71. Hardware semaphore state diagram

Checking for semaphore unlocked

The user has the option to check if the semaphore is locked by reading STATUS[ELS_LOCKED]. The difference between this action, and reading SESSION_ID is that reading STATUS[ELS_LOCKED] will not lock the semaphore, whereas reading SESSION_ID will lock the semaphore (if it is unlocked when the read happens).

Reading STATUS[ELS_LOCKED] gives the user the option of either:

- Using ELS while ELS is unlocked, or
- Locking ELS (by reading SESSION_ID) before using ELS.

13.4.19 Security features

Random AES start delay

A random delay can be programmed for AES calculations.

- A different random delay is applied before the start of each AES calculation within an ELS command.
- The maximum value of the random delay is set in CFG[ADCTRL].
- The delay applies for AES calculations within the following ELS commands: CIPHER, AUTH_CIPHER, KEYIN, KEYOUT, CKDF, and CMAC.
- The table below describes how CFG[ADCTRL] values map to maximum value of the random delay.

Table 319. AES start delay control

ADCTRL	Delay before AES start (cycles)
1xxxxxxxxx	0 to 1023
01xxxxxxxx	0 to 511
001xxxxxxx	0 to 255
0001xxxxxx	0 to 127
00001xxxxx	0 to 63
000001xxxx	0 to 31
0000001xxx	0 to 15
00000001xx	0 to 7
000000001x	0 to 3
0000000001	0 to 1
0000000000	0

Security: Command sequence CRC (cmd CRC)

This feature allows the protection of ELS command sequences, for example, secure boot, by performing a CRC operation of ELS commands under user software control. User software can compare the expected CMD CRC against the ELS calculated value to provide insurance that all of the expected ELS command have been executed and none have been skipped due to an external attack.

CMD CRC programming flow

To protect a Sequence of Commands software uses the CMD CRC as follows:

1. Reset CMD CRC: set CMDCRC_CTRL[CMDCRC_RST].

   NOTE

   CMDCRC is reset to its default value by sync or async ELS reset, or by setting CMDCRC_RST. See CRC Configuration (CMDCRC_CTRL).

2. Enable CMD CRC: set CMDCRC_CTRL[CMDCRC_EN] = 1.

   Purpose: all ELS commands that are run while CMD CRC is enabled will be captured by the CRC.

   NOTE

   CMD CRC reset and enable are independent of each other: see Table 320.

3. Execute a sequence of ELS commands
4. Read the updated cmd CRC value from the CMDCRC register.

CMD CRC can be enabled and disabled during CRC Sequences without resetting the CRC. This allows Software full controllability over protecting different parts of ELS command sequences. The enable and reset operations for command CRC are independent of each other. This is shown in the table below:

NOTE

Care must be taken to achieve the correct behavior because the CMD CRC enable and reset bits are both in the same register.

Table 320. Effect of command CRC reset and enable

CMDCRC_RST	CMDCRC_EN	Result of setting CMDCRC_RST, CMDCRC_EN to the values listed
0	0	Command CRC is not reset, and is disabled
0	1	Command CRC is not reset, and is enabled
1	0	Command CRC is reset, and is disabled
1	1	Command CRC is reset, and is enabled

CMD CRC algorithm

The algorithm used by CMD CRC is a 32-bit CRC with the polynomial function given below:

x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x1 + 1

The input data width is 40 bits, and is constructed as follows:

DATAIN = {ELS_CMD[4:0], CMDCFG0[31:0], 3'h0}

PSEUDO CODE:

```
DATAIN_SIZE=40;

CRC_POLY = 32'h04C11DB7

crc_tmp = current_crc;

for (i=0; i<DATAIN_SIZE; i++)
{
    if ((crc_tmp[31] ^ DATAIN[DATAIN_SIZE-1-i]) = 1) : xor_val = CRC_POLY
    else
        xor_val = 0
    crc_tmp = {crc_tmp[30:0],1'b0} ^ xor_val;
}
new_crc = crc_tmp;
```

DMA final address readback

The final DMA transfer address accessed by ELS during operations can be read from DMA_FIN_ADDR. This is intended to be used as a security check to verify that the final DMA transfer of each command has completed as expected.

OPERATION OF THE FEATURE

After each ELS command, the final address of the last DMA transfer that occurred during the command can be read from the DMA_FIN_ADDR register.

13.4.20 Version and configuration information

Version information for ELS is contained in the VERSION register.

Version information includes:

1. The internal version number of the ELS hardware.
2. The internal version number of the ELS software (microcode): applies only to ELS versions after hardware version 2.13.

The table below describes how the fields of VERSION register map to the release level and internal version number information for ELS.

Table 321. Version information

Field	Description
SW_X	Software major release version: possible values 1-9
SW_Y1	Software minor release version, digit 1: possible values 0-9
SW_Y2	Software minor release version, digit 2: possible values 0-9
SW_Z	Software "Extended Revision version": sometimes a minor release version has to be reworked. In this case the minor release version stays unchanged, and the extended revision version is incremented by 1. Possible values 0-9
X	Hardware major release version: possible values 1-9
Y1	Hardware minor release version, digit 1: possible values 0-9
Y2	Hardware minor release version, digit 2: possible values 0-9
Z	Hardware "Extended Revision version": sometimes a minor release version has to be reworked. In this case the minor release version stays unchanged, and the extended revision version is incremented by 1. Possible values 0-9

Example: ELS internal hardware version is 2.05.4: MAJOR=2, MINOR_DGT1 = 0, MINOR_DGT0 = 5, XTND_REV = 4.

See ELS register descriptions for a description of the register fields mentioned above.

13.4.21 Limitations

Concurrent operations are not supported

Only one ELS command can be active at a time. While ELS is running an operation, or when it is in start up, it is considered to be "busy". Attempts to start an operation while ELS is busy will cause an error. Refer to section Operation [OPN] errors.

Access to registers while ELS is busy

While ELS is busy, writes to registers will have no effect. An exception to this is CTRL[ELS_RESET] which can be written while ELS is busy.

While ELS is busy, reads from registers behave as normal.

13.5 External signals

This module has no external signals.

13.6 Initialization

See Startup phase.

13.7 ELS register descriptions

13.7.1 ELS memory map

ELS_alias3 base address: 4005_7000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Status Register (STATUS)	32	R	See section
4h	Control Register (CTRL)	32	RW	0000_0000h
8h	Command Configuration (CMDCFG0)	32	RW	0000_0000h
Ch	Configuration Register (CFG)	32	RW	0000_0000h
10h	Keystore Index 0 (KIDX0)	32	RW	0000_0000h
14h	Keystore Index 1 (KIDX1)	32	RW	0000_0000h
18h	Key Properties Request (KPROPIN)	32	RW	0000_0000h
20h	DMA Source 0 (DMA_SRC0)	32	RW	0000_0000h
24h	DMA Source 0 Length (DMA_SRC0_LEN)	32	RW	0000_0000h
28h	DMA Source 1 (DMA_SRC1)	32	RW	0000_0000h
30h	DMA Source 2 (DMA_SRC2)	32	RW	0000_0000h
34h	DMA Source 2 Length (DMA_SRC2_LEN)	32	RW	0000_0000h
38h	DMA Result 0 (DMA_RES0)	32	RW	0000_0000h
3Ch	DMA Result 0 Length (DMA_RES0_LEN)	32	RW	0000_0000h
40h	Interrupt Enable (INT_ENABLE)	32	RW	0000_0000h
44h	Interrupt Status Clear (INT_STATUS_CLR)	32	RW	0000_0000h
48h	Interrupt Status Set (INT_STATUS_SET)	32	RW	0000_0000h
4Ch	Error Status (ERR_STATUS)	32	R	0000_0000h
50h	Error Status Clear (ERR_STATUS_CLR)	32	RW	0000_0000h
54h	Version Register (VERSION)	32	R	0000_0000h
5Ch	PRNG SW Read Out (PRNG_DATOUT)	32	R	See section

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
60h	CRC Configuration (CMDCRC_CTRL)	32	RW	0000_0000h
64h	Command CRC Value (CMDCRC)	32	R	A5A5_A5A5h
68h	Session ID (SESSION_ID)	32	RW	CE2C_6AF1h
70h	Final DMA Address (DMA_FIN_ADDR)	32	R	0000_0000h
74h	Master ID (MASTER_ID)	32	RW	0000_001Fh
78h	Keystore Index 2 (KIDX2)	32	RW	0000_0000h
150h - 19Ch	Key Status (ELS_KS0 - ELS_KS19)	32	R	See section

13.7.2 Status Register (STATUS)

Offset

Register	Offset
STATUS	0h

Function

Contains ELS primary status information such as: run/idle status, error status, IRQ status, internal entropy level, busy/idle status of internal TRNG, ready status of internal PRNG, completion status of ECDSA verify command, and privilege level of the current command.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved															ELS_LOC...
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved			Reserved		DTRNG_B...	DRBG_ENT_LVL		PPROT		ECDSA_VFY_STATUS		PRNG_RDY	ELS_ERR	ELS_IRQ	ELS_BUSY
W																
Reset	u	u	u	u	u	0	0	0	1	0	0	0	0	0	0	0

Fields

Field	Function
31-17 —	Reserved
16 ELS_LOCKED	When set, indicates that ELS is locked by a master 0b - Not locked by master 1b - Locked by master
15-13 —	Reserved
12-11 —	Reserved
10 DTRNG_BUSY	When set, it indicates TRNG is gathering entropy 0b - Not gathering entropy 1b - Gathering entropy
9-8 DRBG_ENT_LVL	Entropy quality of the current DRBG instance. This value can change while executing an ELS command but remains stable when ELS is not busy. 00b - NONE 01b - LOW, DRBG generates random numbers of low quality entropy 10b - HIGH, DRBG generates random numbers of high quality entropy 11b - RFU, Reserved for Future Use
7-6 PPROT	Current command privilege level 00b - Secure, non-privileged 01b - Secure, privileged 10b - Non-secure, non-privileged 11b - Non-secure, privileged
5-4 ECDSA_VFY_STATUS	Signature verify result status 00b - No verify run 01b - Signature verify failed 10b - Signature verify passed 11b - Invalid, Error
3 PRNG_RDY	When set, indicates the internal PRNG is ready. SFR PRNG_DATOUT can be read only when this status flag is high. Reading SFR PRNG_DATOUT while this status is low triggers an ELS error. 0b - Internal PRNG not ready 1b - Internal PRNG ready

Table continues on the next page...

Table continued from the previous page...

Field	Function
2 ELS_ERR	When set, indicates the ELS has detected an internal error 0b - Internal error not detected 1b - Internal error detected
1 ELS_IRQ	When set, indicates the ELS has an active interrupt 0b - No active interrupt 1b - Active interrupt
0 ELS_BUSY	When set, indicates the ELS is executing a crypto sequence 0b - Crypto sequence not executing 1b - Crypto sequence executing

13.7.3 Control Register (CTRL)

Offset

Register	Offset
CTRL	4h

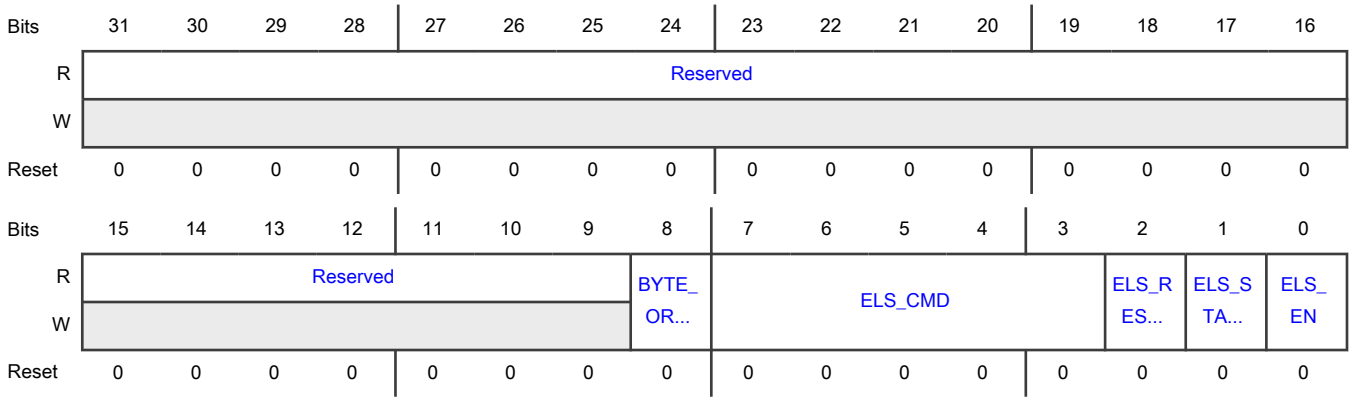
Function

Controls primary operation of ELS such as, enable ELS, reset ELS, and start an ELS command. The register also specifies which command to run and controls (in some cases) the endianness of data that is read and written by ELS.

NOTE

ELS_START and ELS_RESET read back as 0.

Diagram



Fields

Field	Function
31-9 —	Reserved
8 BYTE_ORDER	Defines endianness 0b - Little endian 1b - Big endian
7-3 ELS_CMD	ELS Command ID Defines which command will run when ELS_START is set. See Commands overview.
2 ELS_RESET	Write to 1 to perform an ELS synchronous reset. Writing 0 has no effect.
1 ELS_START	Write to 1 to start an ELS operation. Writing 0 has no effect.
0 ELS_EN	ELS enable 0b - Disabled 1b - Enabled

13.7.4 Command Configuration (CMDCFG0)

Offset

Register	Offset
CMDCFG0	8h

Function

Controls ELS command parameters that are specific to each command. The fields of this register have meanings that change depending on which command is being run.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CMDCFG0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CMDCFG0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 CMDCFG0	See Command specific parameters: CMDCFG0 register and its bit assignments

13.7.5 Configuration Register (CFG)

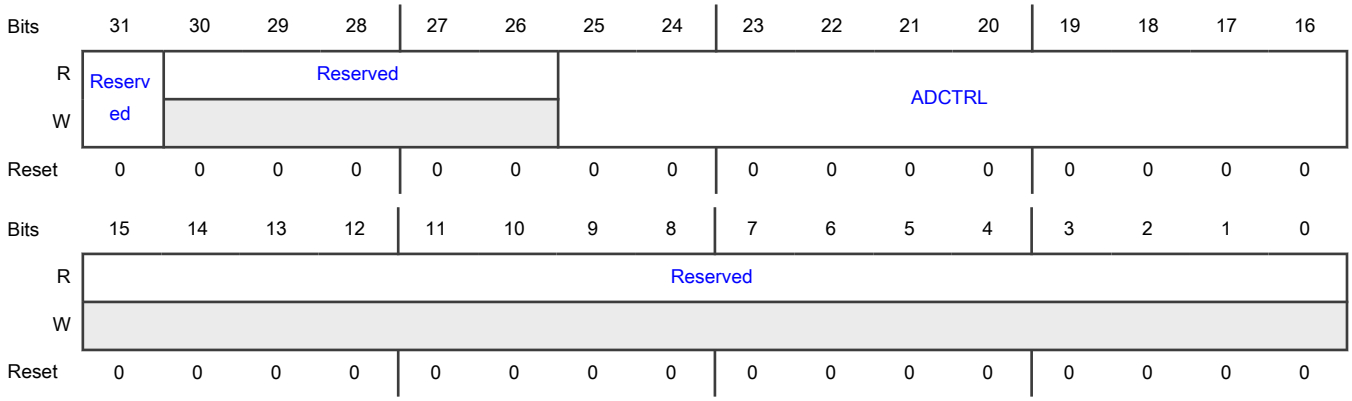
Offset

Register	Offset
CFG	Ch

Function

A general security countermeasure configuration register that controls maximum AES start delay (a security counter-measure. The register also enables SHA2 direct mode - direct access from the external APB bus to ELS internal SHA2 engine.

Diagram



Fields

Field	Function
31 —	Reserved
30-26 —	Reserved
25-16 ADCTRL	Controls the maximum value of a variable delay that will be applied before any ELS AES operation is started.
15-0 —	Reserved

13.7.6 Keystore Index 0 (KIDX0)

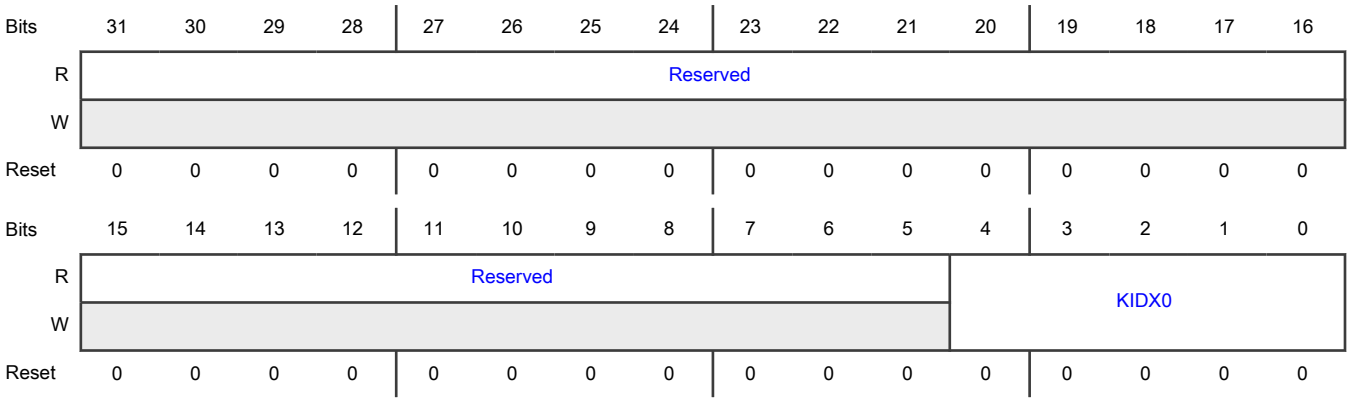
Offset

Register	Offset
KIDX0	10h

Function

This register is used for commands that access a single key. In cases where an ELS command uses one or more keys, the register specifies which key in ELS keystore will be used as the first key by the ELS command. The purpose for which the key is used is command-specific, and is described in the KIDX0 entry of the command description for each command in Commands overview.

Diagram



Fields

Field	Function
31-5 —	Reserved
4-0 KIDX0	Selects the base 128-bit section of a key in ELS keystore. ELS keystore is arranged as an array of indexable 128-bit key sections (key slots). Keys of size larger than 128 bits are constructed from multiple consecutive key slots. The value of KIDX0 specifies the index of the first (base) slot of a key.

13.7.7 Keystore Index 1 (KIDX1)

Offset

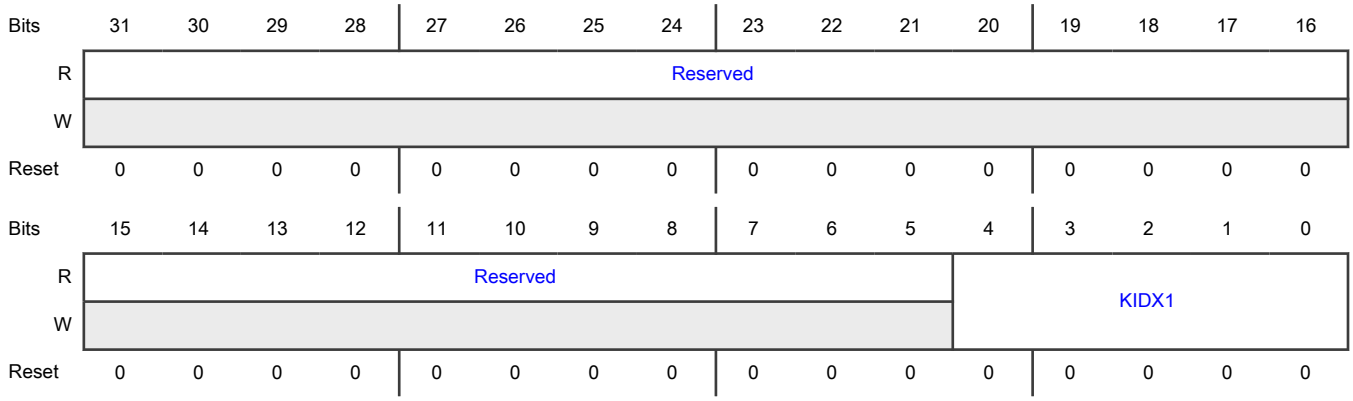
Register	Offset
KIDX1	14h

Function

This register is used for commands that access two keys. In cases where an ELS command requires two or more keys, KIDX1 controls which key will be used as the second key. KIDX1 contains the base index of the ELS keystore key that will be used

as the second key. The purpose for which the key is used is command-specific, and is described in the KIDX1 entry of the command description for each command in Commands overview.

Diagram



Fields

Field	Function
31-5 —	Reserved
4-0 KIDX1	Selects the base 128-bit section of a key in ELS keystore. ELS keystore is arranged as an array of indexable 128-bit key sections (key slots). Keys of size larger than 128 bits are constructed from multiple consecutive key slots. The value of KIDX1 specifies the index of the first (base) slot of a key.

13.7.8 Key Properties Request (KPROPIN)

Offset

Register	Offset
KPROPIN	18h

Function

Specifies the properties of the key that is created by ELS command. See Keystore key properties and status for the bit allocations of Key properties.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KPROPIN															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	KPROPIN															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 KPROPIN	Specifies requested properties of the key created by ELS command.

13.7.9 DMA Source 0 (DMA_SRC0)

Offset

Register	Offset
DMA_SRC0	20h

Function

Specifies the base address in system memory that one of the accesses required by an ELS command will occur at. See DMA_SRC0 entries for each command in Commands overview.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ADDR_SRC0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ADDR_SRC0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 ADDR_SRC0	Defines the system address of the start of the data to be transferred to the ELS via DMA.

13.7.10 DMA Source 0 Length (DMA_SRC0_LEN)

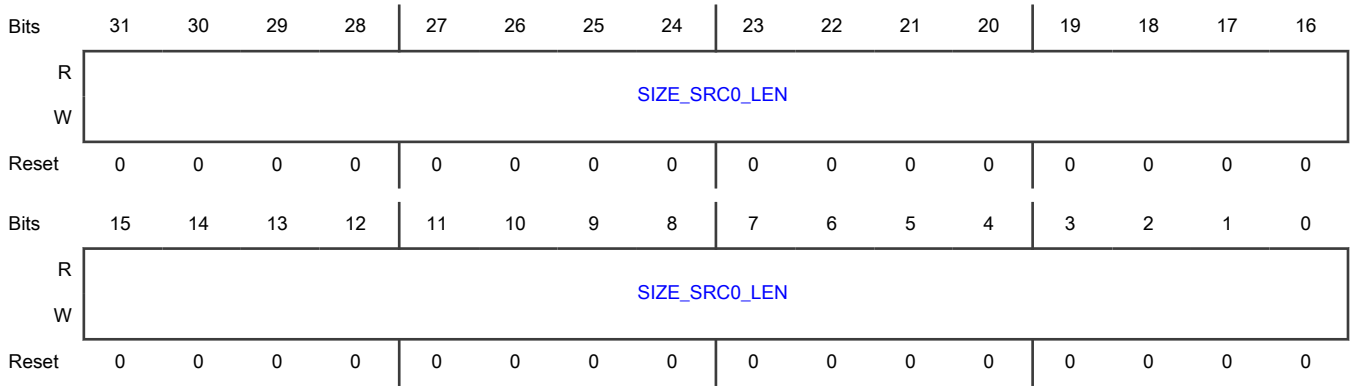
Offset

Register	Offset
DMA_SRC0_LEN	24h

Function

When an ELS command requires access to system memory, this register specifies the size (in bytes) of the data that will be transferred to or from system memory. See DMA_SRC0_LEN entries for each command in Commands overview.

Diagram



Fields

Field	Function
31-0 SIZE_SRC0_LEN	Size in bytes of the data to be transferred from the target defined in SFR DMA_SRC0.

13.7.11 DMA Source 1 (DMA_SRC1)

Offset

Register	Offset
DMA_SRC1	28h

Function

Specifies the base address in system memory that one of the accesses required by an ELS command will occur at. See DMA_SRC1 entries for each command in Commands overview.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ADDR_SRC1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ADDR_SRC1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 ADDR_SRC1	Defines the system address of the start of the data to be transferred to the ELS via DMA.

13.7.12 DMA Source 2 (DMA_SRC2)

Offset

Register	Offset
DMA_SRC2	30h

Function

Specifies the base address in system memory that one of the accesses required by an ELS command will occur at. See DMA_SRC2 entries for each command in Commands overview.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ADDR_SRC2															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ADDR_SRC2															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 ADDR_SRC2	Defines the system address of the start of the data to be transferred to the ELS via DMA.

13.7.13 DMA Source 2 Length (DMA_SRC2_LEN)

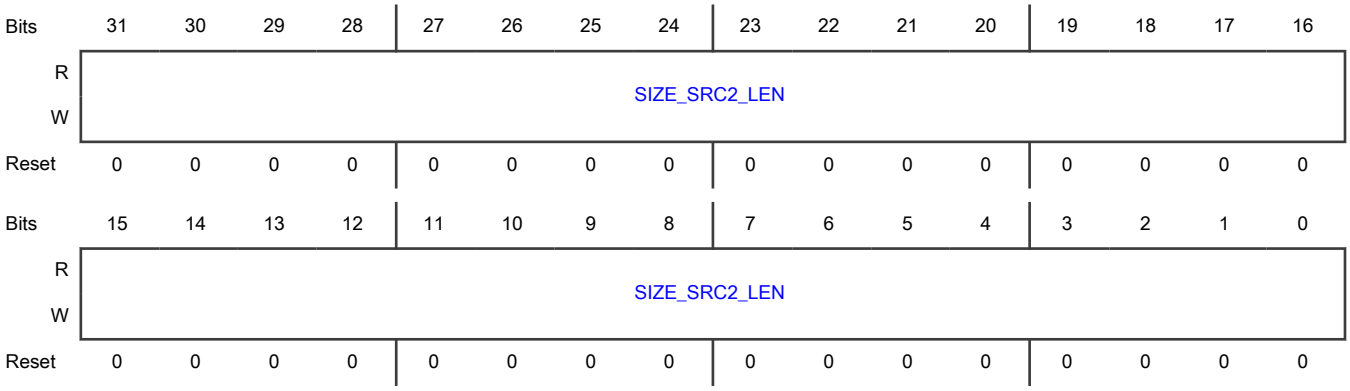
Offset

Register	Offset
DMA_SRC2_LEN	34h

Function

When an ELS command requires access to system memory, this register specifies the size (in bytes) of the data that will be transferred to or from system memory. See DMA_SRC2_LEN entries for each command in Commands overview.

Diagram



Fields

Field	Function
31-0 SIZE_SRC2_LEN	Size in bytes of the data to be transferred from the target defined in SFR DMA_SRC2.

13.7.14 DMA Result 0 (DMA_RES0)

Offset

Register	Offset
DMA_RES0	38h

Function

Specifies the base address in system memory that one of the accesses required by an ELS command will occur at. See DMA_RES0 entries for each command in Commands overview.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ADDR_RES0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ADDR_RES0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 ADDR_RES0	Defines the system start address where the result of the ELS operation is transferred via DMA.

13.7.15 DMA Result 0 Length (DMA_RES0_LEN)

Offset

Register	Offset
DMA_RES0_LEN	3Ch

Function

When an ELS command requires access to system memory, this register specifies the size (in bytes) of the data that will be transferred to or from system memory. See DMA_RES0_LEN entries for each command in Commands overview.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SIZE_RES0_LEN															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SIZE_RES0_LEN															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 SIZE_RES0_LEN	Size in bytes of the data to be transferred

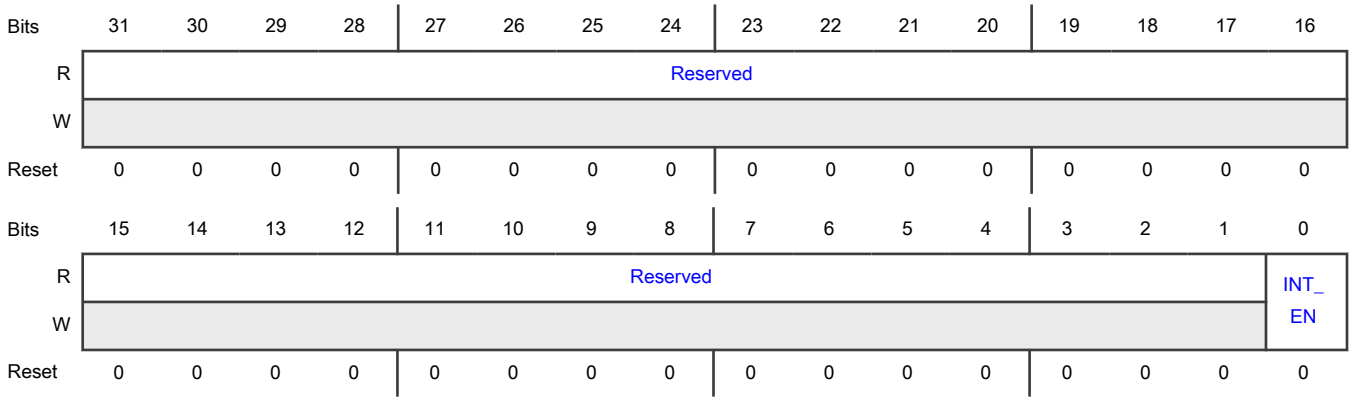
13.7.16 Interrupt Enable (INT_ENABLE)

Offset

Register	Offset
INT_ENABLE	40h

Function
Enables or disables the operation of the ELS interrupt output port.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_EN	Enables or disables the operation of the ELS interrupt output port. 0b - Disables 1b - Enables

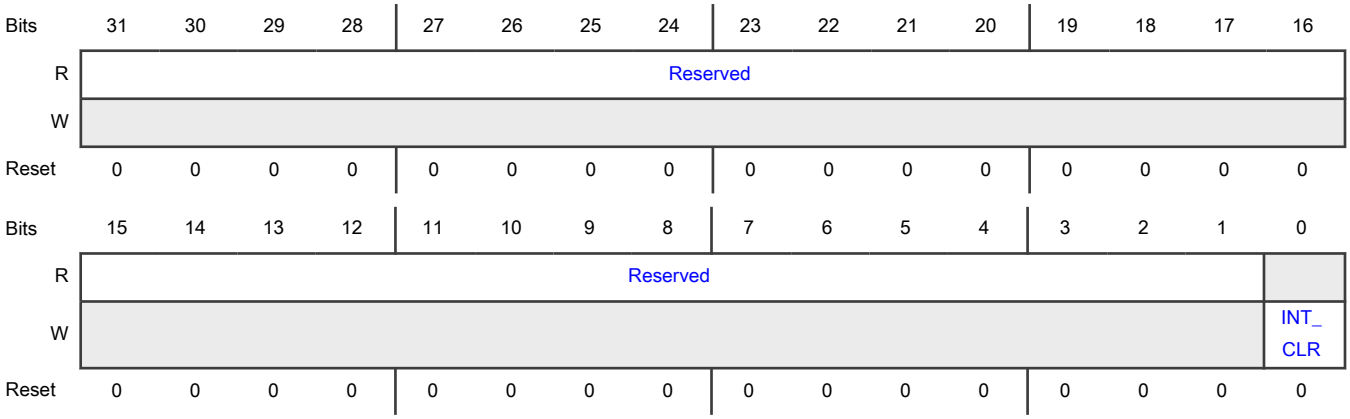
13.7.17 Interrupt Status Clear (INT_STATUS_CLR)

Offset

Register	Offset
INT_STATUS_CLR	44h

Function
Clears the ELS interrupt status flag.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_CLR	Interrupt status clear bit. Writing to 1 clears the ELS interrupt status flag. Writing to 0 has no effect.

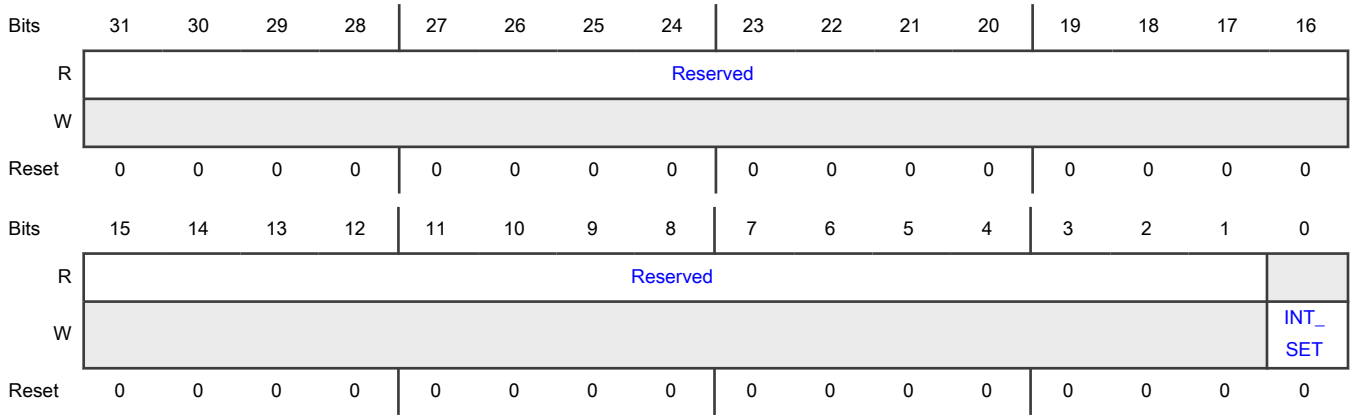
13.7.18 Interrupt Status Set (INT_STATUS_SET)

Offset

Register	Offset
INT_STATUS_SET	48h

Function
Allows the triggering of an ELS interrupt via software.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_SET	Software triggered interrupt bit. Writing to 1 triggers an ELS interrupt. Writing to 0 has no effect.

13.7.19 Error Status (ERR_STATUS)

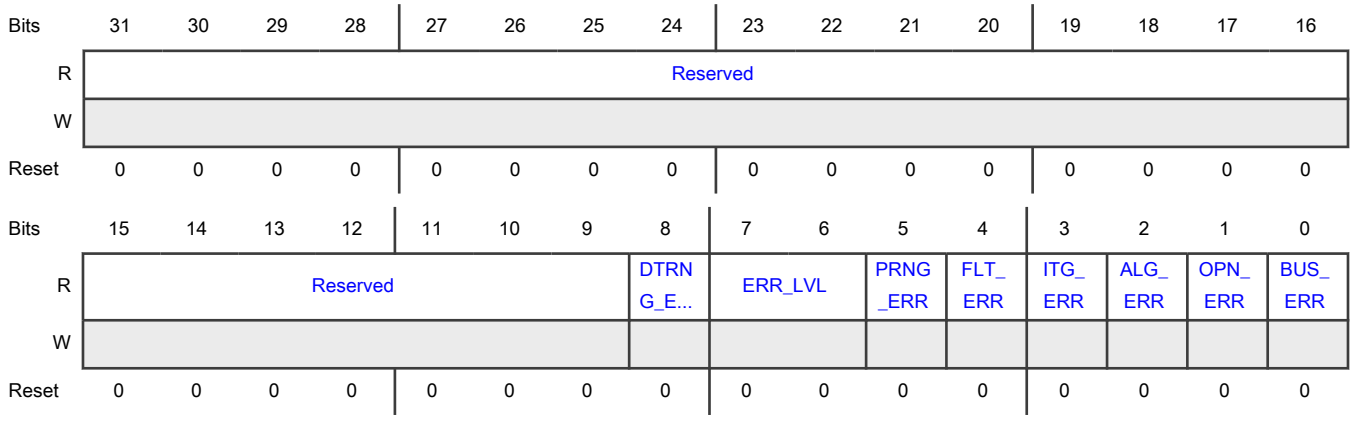
Offset

Register	Offset
ERR_STATUS	4Ch

Function

Contains status bits that indicate detailed error status of ELS, such as error level and error type.

Diagram



Fields

Field	Function
31-9 —	Reserved
8 DTRNG_ERR	TRNG unable to gather entropy with the current configuration. Provide TRNG with a new and valid configuration via ELS command DTRNG_CFG_LOAD. 0b - No error 1b - TRNG error occurred
7-6 ERR_LVL	Indicates the triggered error level: 0, 1 ,2.
5 PRNG_ERR	Indicates user read of PRNG_DATOUT when STATUS[PRNG_RDY] is 0. 0b - No error 1b - Error occurred
4 FLT_ERR	Indicates hardware fault error; an attempt to change the value of an internal register. 0b - No error 1b - Error occurred
3 ITG_ERR	Indicates data integrity error, that is, internal data integrity check has failed. 0b - No error 1b - Error occurred
2 ALG_ERR	Indicates algorithm error; an internal algorithm has produced an unexpected result. 0b - No error 1b - Error occurred
1 OPN_ERR	Indicates operational error, that is, ELS has been incorrectly operated. 0b - No error 1b - Error occurred
0 BUS_ERR	Indicates public or private bus access error. 0b - No error 1b - Error occurred

13.7.20 Error Status Clear (ERR_STATUS_CLR)

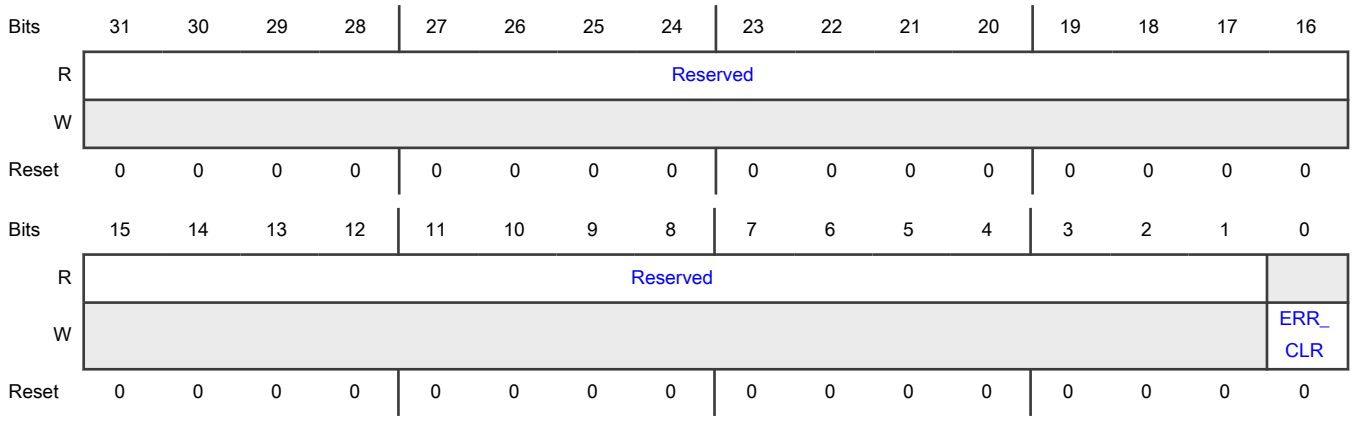
Offset

Register	Offset
ERR_STATUS_CLR	50h

Function

Controls clearing of the ELS error status. Clearing error status by the user is a required step in the ELS error clearing sequence.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 ERR_CLR	ELS error status bit 0b - Exits ELS error state 1b - Clears ELS error state

13.7.21 Version Register (VERSION)

Offset

Register	Offset
VERSION	54h

Function

Contains ELS hardware and software version information.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SW_X				SW_Y1				SW_Y2				SW_Z			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	X				Y1				Y2				Z			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-28 SW_X	Specifies the software major release version; possible values are from 1-9.
27-24 SW_Y1	Specifies the software minor release version digit1; possible values are from 0-9.
23-20 SW_Y2	Specifies the software minor release version digit0; possible values are from 0-9.
19-16 SW_Z	Specifies the software extended revision version; possible values are from 0-9.
15-12 X	Specifies the major release version; possible values are from 1-9.
11-8 Y1	Specifies the minor release version digit1; possible values are from 0-9.
7-4 Y2	Specifies the minor release version digit0; possible values are from 0-9.
3-0 Z	Specifies the extended release version digit1; possible values are from 0-9.

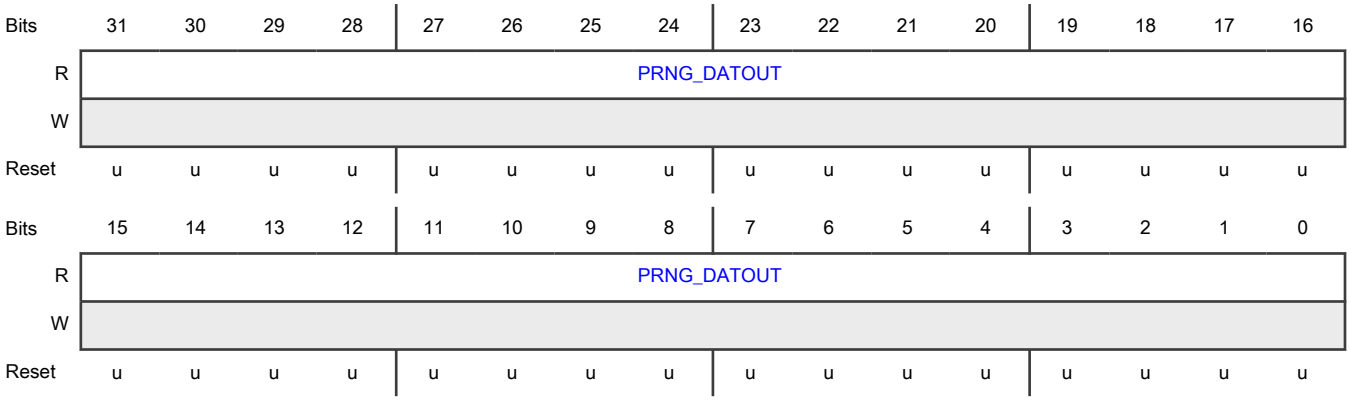
13.7.22 PRNG SW Read Out (PRNG_DATOUT)

Offset

Register	Offset
PRNG_DATOUT	5Ch

Function
Data out register that allows read access to pseudo random data from the ELS internal PRNG.

Diagram



Fields

Field	Function
31-0 PRNG_DATOUT	32-bit wide pseudo-random number. Successive reads from this field will access successive words of pseudo random data from the ELS internal PRNG.

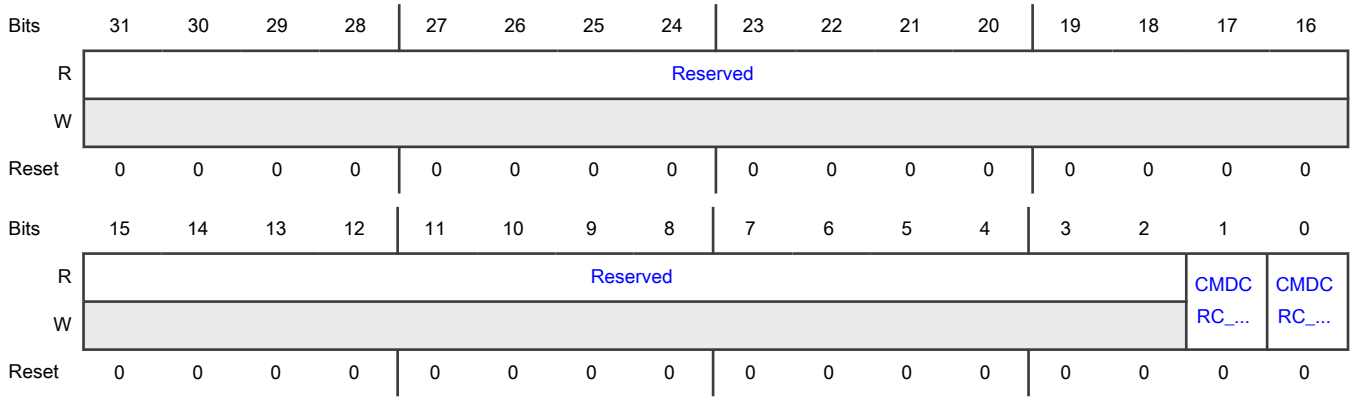
13.7.23 CRC Configuration (CMDCRC_CTRL)

Offset

Register	Offset
CMDCRC_CTRL	60h

Function
Supports operation of the ELS command CRC which is a security feature.

Diagram



Fields

Field	Function
31-2 —	Reserved
1 CMDCRC_EN	CRC enable bit 0b - Disables the CRC command CRC. The CRC command will not be updated on completion of each ELS command. 1b - Enables the CRC command. The CRC command will be updated on completion of each ELS command.
0 CMDCRC_RST	CRC reset to initial value **NOTE** CMDCRC_EN and CMDCRC_RST fields act independently. Both fields can be changed simultaneously via a single write to the register. 0b - No effect 1b - Resets the CRC command to its default value

13.7.24 Command CRC Value (CMDCRC)

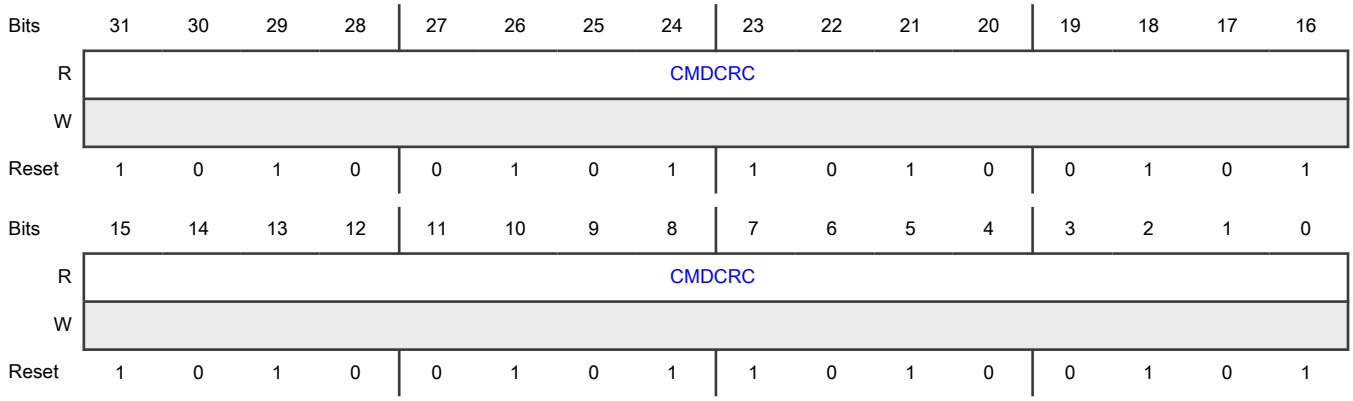
Offset

Register	Offset
CMDCRC	64h

Function

Indicates the current value of the CRC ELS command.

Diagram



Fields

Field	Function
31-0 CMDCRC	Indicates the current CRC value.

13.7.25 Session ID (SESSION_ID)

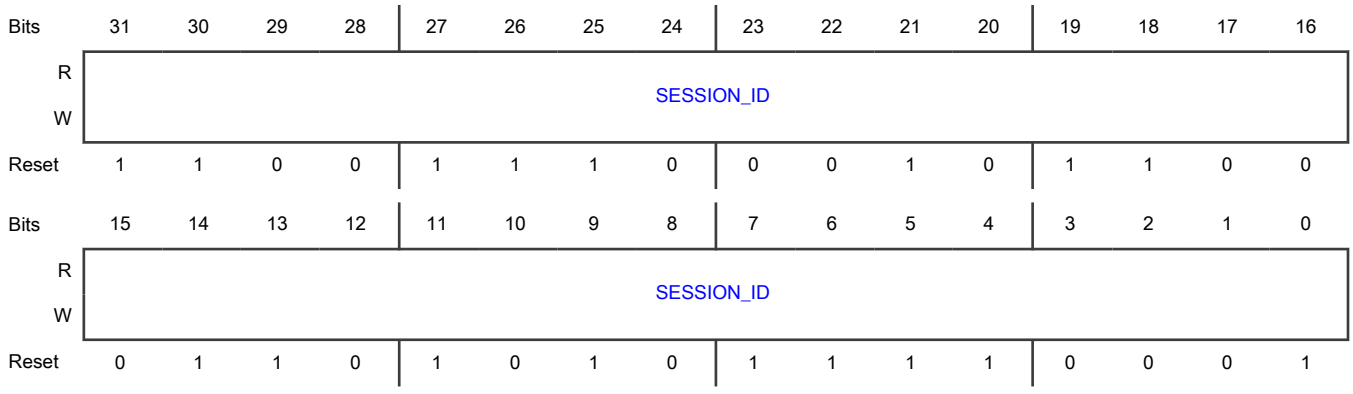
Offset

Register	Offset
SESSION_ID	68h

Function

Hardware semaphore session ID register. Indicates the current value of the session ID. See Hardware semaphore.

Diagram



Fields

Field	Function
31-0 SESSION_ID	Indicates the current value of the session ID.

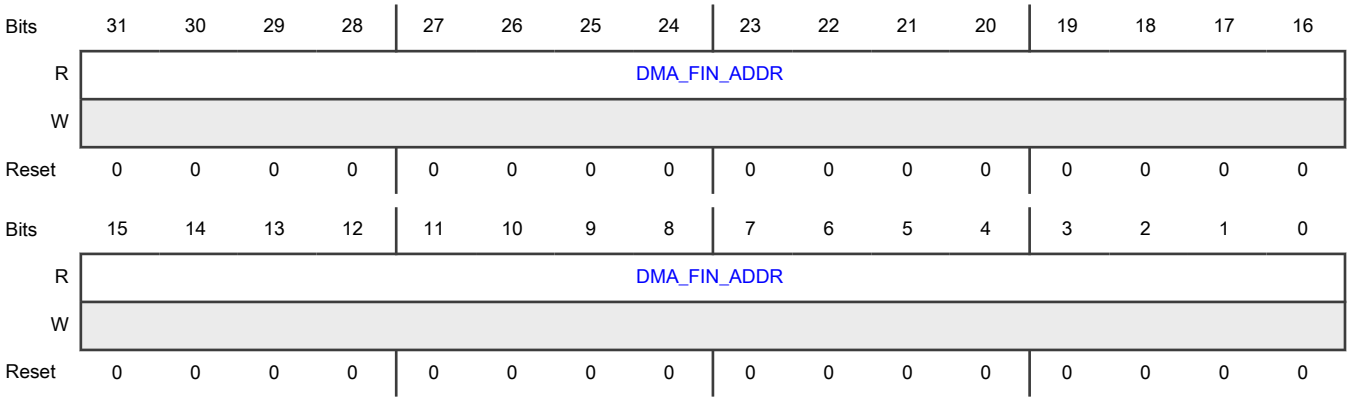
13.7.26 Final DMA Address (DMA_FIN_ADDR)

Offset

Register	Offset
DMA_FIN_ADDR	70h

Function
A security feature that indicates the final address of system memory that was accessed by ELS during the last command.

Diagram



Fields

Field	Function
31-0 DMA_FIN_ADDR	Indicates the final address of system memory that was accessed by ELS during the last command.

13.7.27 Master ID (MASTER_ID)

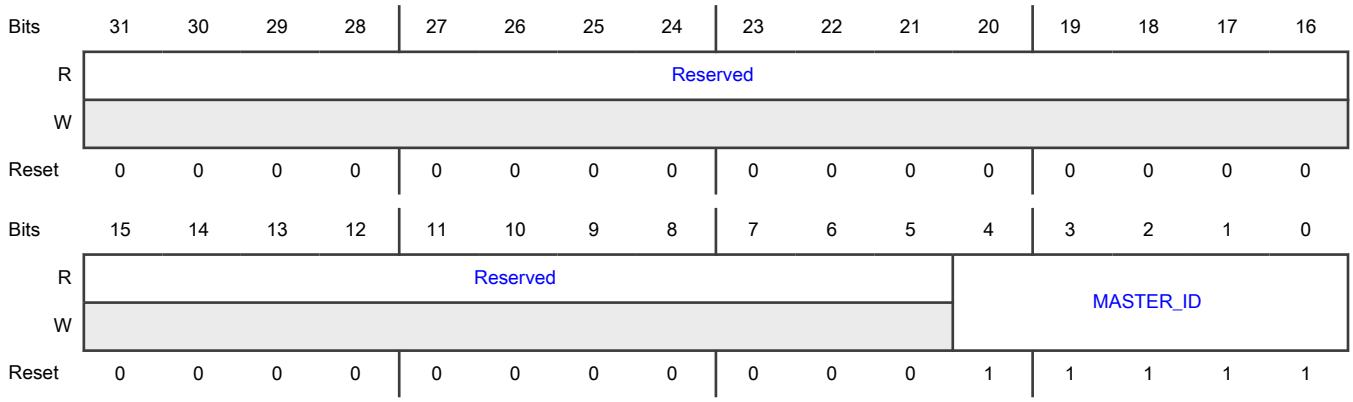
Offset

Register	Offset
MASTER_ID	74h

Function

Sets the privileged hardware semaphore master ID, which can be used to unlock ELS without the session ID. See Hardware semaphore for more details. This is a one-time write register. Write access to this register can only be restored by resetting ELS.

Diagram



Fields

Field	Function
31-5 —	Reserved
4-0 MASTER_ID	Sets the privileged master ID.

13.7.28 Keystore Index 2 (KIDX2)

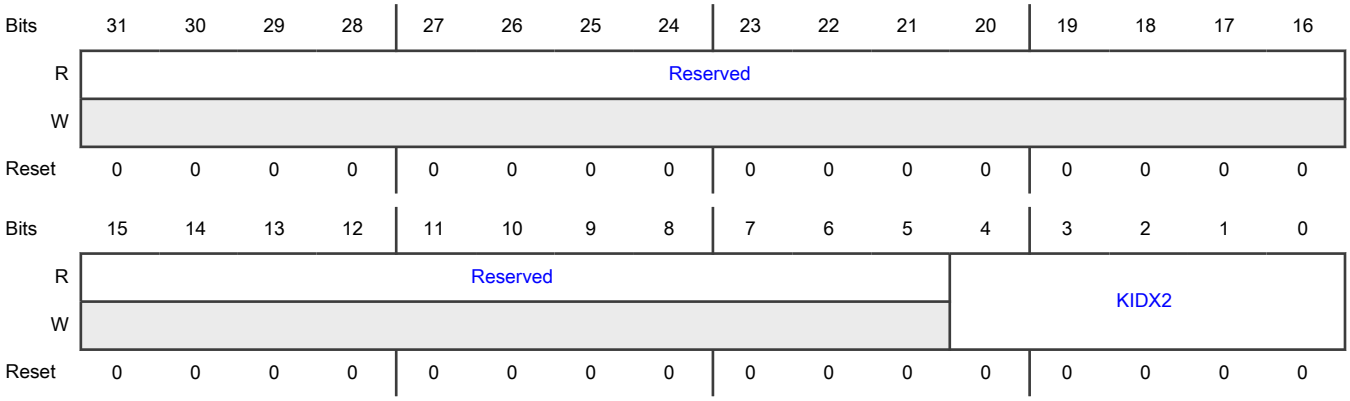
Offset

Register	Offset
KIDX2	78h

Function

In cases where an ELS command requires access to three keys, KIDX2 controls which key will be used as the third key. KIDX2 contains the base index of the ELS keystore key that will be used as the third key. The purpose for which the key is used is command-specific, and is described in the KIDX2 entry of the command description for each command in Commands overview.

Diagram



Fields

Field	Function
31-5 —	Reserved
4-0 KIDX2	Selects the base 128-bit section of a key in ELS keystore. ELS keystore is arranged as an array of indexable 128-bit key sections (key slots). Keys of size larger than 128 bits are constructed from multiple consecutive key slots. The value of KIDX2 specifies the index of the first (base) slot of a key.

13.7.29 Key Status (ELS_KS0 - ELS_KS19)

Offset

For i = 0 to 19:

Register	Offset
ELS_KSi	150h + (i × 4h)

Function

Indicates the key status register for key slot *n*. There is a key status register corresponding to each key slot in ELS keystore. Reading key status register *n* returns the key status of key slot *n*. For a detailed description of the key status fields, see Keystore key properties and status

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	KSn_UPPROT		KSn_UDUK	KSn_UWR...	KSn_UHWO	KSn_UKG...	KSn_UTL...	KSn_UTL...	KSn_UKU...	KSn_UKWK	KSn_UHM...	KSn_UAES	KSn_UEC...	KSn_UEC...	KSn_UHK...	KSn_UCK...
W																
Reset	See Register reset values.															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	KSn_URTF	KSn_UKSK	KSn_UCM...	KSn_UTE...	KSn_UKP...	Reserved	KSn_FHWO	KSn_FRTN	KS0_FGP	KSn_KBA...	KSn_KACT	Reserved			KSn_KSIZE	
W																
Reset	See Register reset values.															

Register reset values

Register	Reset value
ELS_KS0–ELS_KS1	ELS_alias3: 8000_0100h
ELS_KS2–ELS_KS3	ELS_alias3: 8000_0300h
ELS_KS4–ELS_KS19	ELS_alias3: 8000_0080h

Fields

Field	Function
31-30 KSn_UPPROT	Indicates privilege level key. User access rights are required to use this key. 00b - Non-privileged, secure 01b - Non-privileged, non-secure 10b - Privileged, secure 11b - Privileged, non-secure
29 KSn_UDUK	Indicates device unique key whose value is unique per die, and secret, that is, not known outside of ELS. It is used as a Master Key where other keys and secrets are derived from it. This property is used to infer rules that are specific to DUK. 0b - Key is not a device unique key 1b - Key is a device unique key
28 KSn_UWRPOK	Indicates wrap key that can be RFC3394 wrapped by another keystore key. 0b - Key cannot be RFC3394 wrapped 1b - Key can be RFC3394 wrapped
27 KSn_UHWO	Indicates hardware out key that only exists in a hardware out slot, that is a slot that has status bit FHWO set. 0b - Key can exist in any slot

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - Key can only exist in a hardware out slot
26 KSn_UKGSRC	Supplies key material source for deterministic ECC KEYGEN. A key with UKGSRC set provides the key material input to the KEYGEN command. 0b - Key cannot be used as a key material source for deterministic ECC keygen 1b - Key can be used as a key material source for deterministic ECC keygen
25 KSn_UTLSMS	Indicates TLS master secret key that is used as the input key to the TLS_INIT command with input parameter “FINALIZE” set to 1. 0b - Key is not a TLS master secret key 1b - Key is a TLS master secret key
24 KSn_UTLSPMS	Indicates TLS pre master secret key that is used as the input key to the TLS_INIT command with input parameter “FINALIZE” set to 0. 0b - Key is not a TLS pre master secret key 1b - Key is a TLS pre master secret key
23 KSn_UKUOK	Indicates RFC3394 key unwrap only Key that is used as the unwrapping key for the KEYIN(RFC3394=true) command, but not as the wrapping key for the KEYOUT command. 0b - Key is not a RFC3394 key unwrap only key 1b - Key is a RFC3394 key unwrap only key
22 KSn_UKWK	Indicates RFC3394 key wrap and unwrap key that is used as the unwrapping key for the KEYIN(RFC3394=true) command, and as the wrapping key for the KEYOUT command. 0b - Key is not a RFC3394 key wrap plus unwrap only key 1b - Key is a RFC3394 key wrap plus unwrap only key
21 KSn_UHMAC	Indicates HMAC key that is used as the input key for the HMAC command. 0b - Key is not a HMAC key 1b - Key is a HMAC key
20 KSn_UAES	Indicates AES key that is used as the input key for the CIPHER or AUTH_CIPHER commands. 0b - Key is not a AES key 1b - Key is a AES key
19 KSn_UECDH	Indicates Diffie Hellman key exchange private key that is used as the input private key for the ECKXCH command. 0b - Key is not a Diffie hellman key exchange private key 1b - Key is a Diffie hellman key exchange private key
18 KSn_UECSG	Indicates ECC signing key that is used as the input private key for the ECSIGN command. 0b - Key is not a ECC signing key 1b - Key is a ECC signing key

Table continues on the next page...

Table continued from the previous page...

Field	Function
17 KSn_UHKDF	Indicates HKDF key that is used as the input key for the HKDF command. 0b - Key is not a HKDF key 1b - Key is a HKDF key
16 KSn_UCKDF	Indicates CMAC based derivation key that is used as the input key for the CKDF command. 0b - Key is not a CMAC based derivation key 1b - Key is a CMAC based derivation key
15 KSn_URTF	Indicates ECC signing key that is used as the input private key to the ECSIGN command when SIGNRTF command parameter is set. 0b - Key is not a ECC signing key that can be used as the input private key to ECSIGN command when SIGNRTF command parameter is set 1b - Key is a ECC signing key that can be used as the input private key to ECSIGN command when SIGNRTF command parameter is set
14 KSn_UKSK	Indicates public key signing key that is used to sign the ECC public key generated by the KEYGEN command. 0b - Key is not a public key signing Key 1b - Key is a public key signing key
13 KSn_UCMAC	Indicates CMAC key that is used as input key for the CMAC command. 0b - Key is not a CMAC key 1b - Key is a CMAC key
12 KSn_UTECDH	Indicates a private key that can be only used with a trusted public key (key with UKPUK=1). It cannot be used with a public key from system memory. Also, it can only be used in a key exchange (see ECKXCH command). 0b - Key is not a UTECDH key 1b - Key is a UTECDH key
11 KSn_UKPUK	Indicates trusted public key that can be used as the public key in an ECSIGN command in which the private key is a UTECDH key. 0b - Key is not a trusted public key 1b - Key is a trusted public key
10 —	Reserved
9 KSn_FHWO	Indicates the hardware out type key slot. See UHWO. 0b - Slot is not a hardware out type key slot 1b - Slot is a hardware out type key slot

Table continues on the next page...

Table continued from the previous page...

Field	Function
8 KSn_FRTN	Specifies the retention type key slot. A slot with this property has been constrained during manufacturing so that it retains its key material in chip power down modes. 0b - Slot is not a retention type key slot 1b - Slot is a retention type key slot
7 KS0_FGP	Specifies the general purpose type key slot. A flag is set on slots that are neither retention nor hardware out slots. 0b - Slot is not a general purpose type key slot 1b - Slot is a general purpose type key slot
6 KSn_KBASE	Indicates the base key slot of a multi-slot key. 0b - Key is not a base slot of a multi slot key 1b - Key is a base slot of a multi slot key
5 KSn_KACT	Indicates that a slot is in use by a key. 0b - Key slot is not in use by a key 1b - Key is in use by a key
4-2 —	Reserved
1-0 KSn_KSIZE	Indicates the key size 00b - Key size is 128 bits (1 slot) 01b - Key size is 256 bits (2 slots) 10b - Reserved 11b - Key size is 512 bits (4 slots)

Chapter 14
Physically Unclonable Function (PUF)

14.1 Chip-specific PUF information

Table 322. Reference links to related information[1]

Topic	Related module	Reference
Full description	PUF	PUF
System memory map		See the section "System memory map"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

NOTE

The ITRC module can assert a zeroize signal into PUF.

14.1.1 Module instances

This device has one instance of the PUF module with four aliased base addresses.

14.1.2 Security considerations

The PUF module includes user context inputs as part of the Key code header used for Get Key and Wrap operations. The user_context_0 word includes an access level for the key that determines the secure/non-secure and privileged/non-privileged access for the key.

The PUF module is instantiated to use four module slots—PUF, PUF_alias1, PUF_alias2, and PUF_alias3. At the Secure AHB controller, each PUF slot can be configured for a different secure/non-secure and privileged/non-privileged access. This allows the PUF module to be used by any access level using the appropriate slot, while the Secure AHB controller's MISC_CTRL_REG[DISABLE_STRICT_MODE] option is configured for strict mode. The PUF's per-key access levels are then used to restrict access to specific keys according to the level used by that particular PUF slot.

14.1.3 Zerozie operation

PUF can be permanently disabled at system level, by using the disable pins, or by permanently making zeroize = 1. The HW zeroize signal is connected to one of the ITRC outputs (OUT2). Apart from ITRC based handling, SW can issue PUF_ZEROIZE command whenever it detects a system level threat condition not detected by ITRC input signal.

14.1.4 PUF module integration

The PUF module is tightly integrated with the PUF Key Context Management (PUF_CTRL). The PUF_CTRL implements access protections for the PUF module that can be used to lock the PUF to a particular access level. This is intended to be used as a temporary lock that keeps one master from modifying registers while another master is in the middle of a PUF operation.

The PUF module also has a direct connection to the EdgeLock Secure Subsystem (ELS). The ROM will execute commands at boot to load the 256-bit device unique key (NXP_DIE_MK_SK) into ELS key slots 0 and 1.

14.2 Overview

PUF assigns a unique key to each device that exists in the device based on the unique characteristics of PUF SRAM. They are virtually impossible to duplicate, clone or predict making them very suitable for applications such as secure key generation and storage, device authentication, flexible key provisioning and chip asset management.

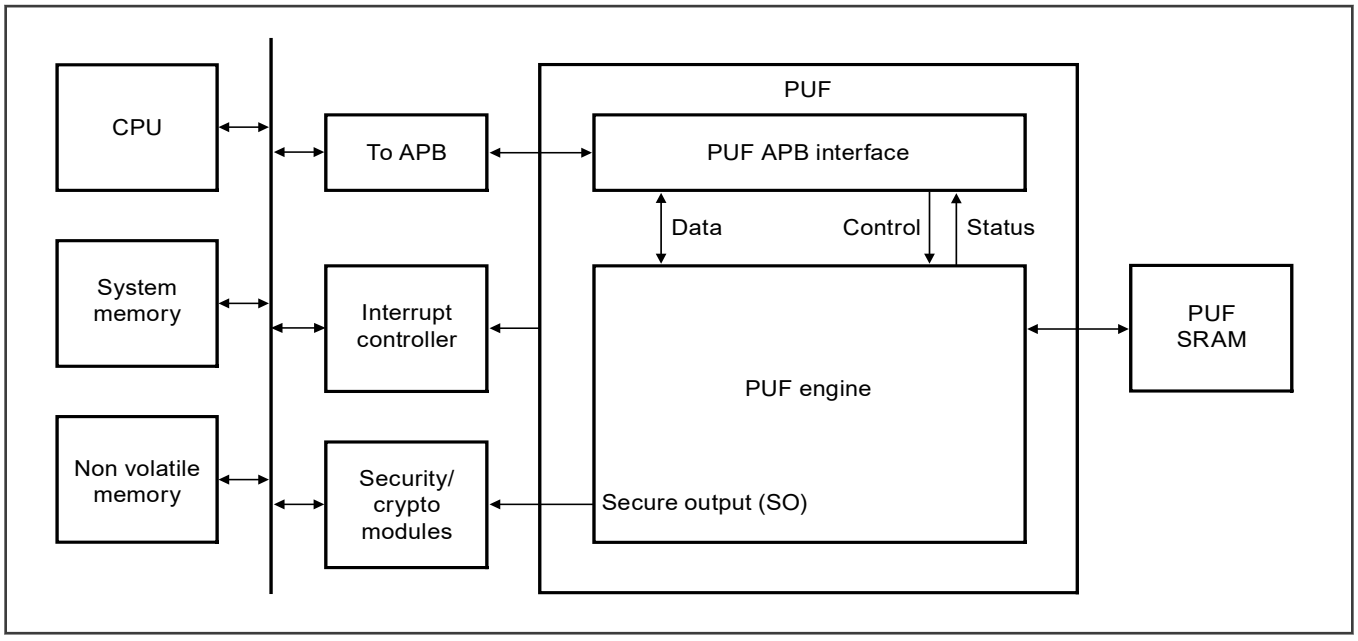
Due to deep sub-micron process variations in the production process, every transistor in an SRAM cell has slightly random electric properties. This randomness is expressed in the startup values of uninitialized SRAM. These values form a unique chip fingerprint, called the SRAM PUF response.

An SRAM PUF response is a noisy fingerprint, and turning it into a high-quality and secure key vault requires further processing done with the PUF. PUF reliably reconstructs the same cryptographic key under all environmental circumstances. It generates an Activation Code, which in combination with the SRAM startup behavior, is used to reconstruct keys, on demand, in real time, without the need to store the key directly. When the key is no longer needed, it can be removed from memory. When it is needed later it can be reconstructed. The device unique root key (DUK) is used as root key to derive further application keys and to wrap pre-shared keys. A key protected by PUF is integrity-protected and can be retrieved only on the same device while it would be meaningless on other devices.

14.2.1 Features

- Key strength of 256-bits
  - A 256-bit strength device-unique PUF root key using the digital fingerprint of a device derived from SRAM and error correction data called Activation Code (AC)
  - The AC generated during enrollment process is stored in the non-volatile memory in the system
- Generation, storage, and reconstruction of keys
- Key sizes from 64-bits to 4096-bits
  - PUF controller combines keys with digital fingerprint of device to generate key codes
  - Key codes should be provided to the controller to reconstruct the original key
  - Key codes stored on external, non-volatile memory device in the system
- Key output via dedicated hardware interface or through register interface
- PUF quality test with a score done during Enroll, Start and Reconstruct operations.

14.2.2 Block diagram



14.2.3 Security strength and key length

PUF's security strength is 256 bit. It refects the strength of its internal secrets, and hence determines the highest level of cryptographic security that any of PUF's cryptographic operations can offer. Depending on the operation and the provided parameters, the effective security strength can be lower, but never higher than 256 bit.

Shorter keys can reduce the security strength, for example, the security strength of a 128-bit key cannot be greater than 128 bit. Greater key lengths cannot increase the security strength beyond the maximum, for example, a 512-bit key generated by PUF still has a maximum security strength of 256 bit.

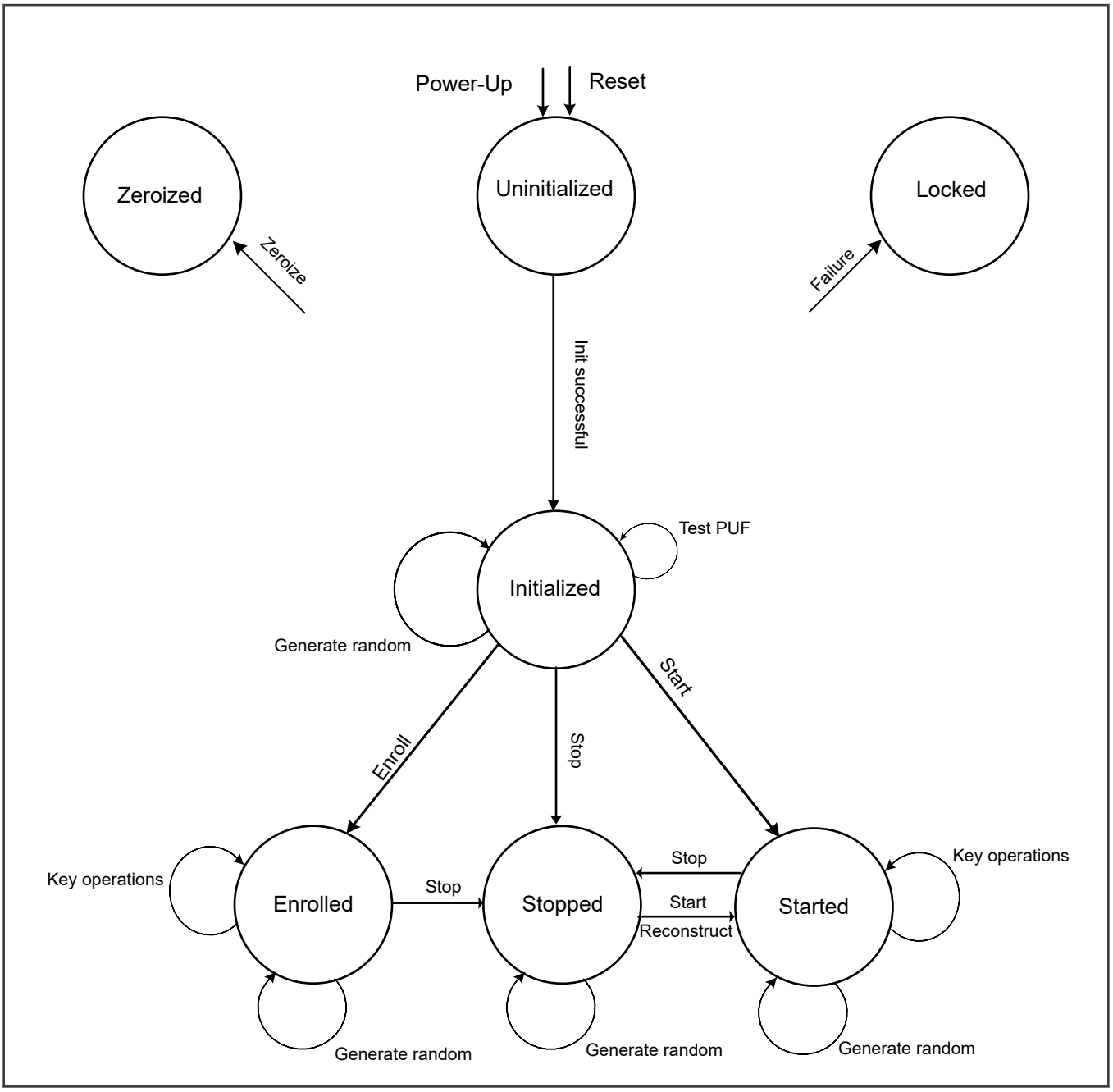
Key length can differ from security strength for various reasons. In elliptic-curve cryptography, a NIST P-521 private key has a 521-bit key length and a 256-bit security strength. PUF's 256-bit security strength is therefore suffcient to safely wrap P-521 private keys.

14.3 Functional Description

14.3.1 PUF states

The functionality of PUF is captured in various operations, which can be triggered by giving the corresponding commands to PUF. Each operation can transition PUF to a different state, and each state has a different set of possible future operations.

This section first describes the possible state sequences as a result of the operations PUF executes.



PUF operations and states

- Initialized
- Enroll/Enrolled
- Start/Started
- Stop/Stopped

Key operations:

- Get Key
- Wrap
- Wrap Generated Random

- Unwrap

See Order of PUF Operations for more details.

Possible state sequences as a result of the operations PUF executes:

- After power-up or reset, PUF begins in the Uninitialized state, and runs its initialization sequence. This is indicated by SR[BUSY] = 1.
- Reset takes precedence over all PUF functionality. As long as reset = 0, PUF stays in the Uninitialized state and commands or other pins have no effect.
- When initialization fnishes successfully, PUF moves to the Initialized state, while it moves to the Locked state on failure. In the Initialized state, a Generate Random, an Enroll, a Start or a Stop operation can be performed.

NOTE

An operation can only be performed when it is not disabled. This is valid in all states. Refer to the PUF command blocking configuration (CONFIG) register in the "PUF Key Context Management" section.

- After a successful Enroll operation PUF is in the Enrolled state. In this state it can do a Generate Random operation, Key operations or a Stop operation (when no further actions need to be done at the moment).
- After a successful Start operation PUF is in the Started state. In this state a Generate Random operation, Key operations or a Stop operation can be done.
- A Stop operation brings PUF to the Stopped state. In this state no sensitive data is present in the hardware. In the Stopped state, a Generate Random or a Start operation can be performed.
- When in Started or Enrolled state, Key operations can be performed. After such an operation is complete, PUF returns to the state it was in before the operation.
- In any of the Initialized, Enrolled, Started and Stopped states, random data can be generated with the Generate Random command
- The Zeroize command (erases all critical security parameters and prevents PUF from executing any more commands by going to the Zeroized state. The only way to leave this state is power-cycling the device which puts PUF in the Uninitialized state and starts initialization. This command can be given via the CR register or via the zeroize operation. See the Zeroize section for more details.
- If an operation is unsuccessful, this is indicated with an error (SR[ERROR] = 1). In this case PUF will return to the state in which the command was started.
- If a failure (unrecoverable error) occurs during any of the above-mentioned operations (including initialization and Zeroize), PUF goes to the Locked state. In this state, no commands can be executed except Zeroize. After a reset PUF will attempt to initialize.

The Locked and Zeroized states are indicated with the SR[OK], SR[ERROR], and SR[ZEROIZED] bits, as shown in Table 323.

Table 323. Locked and Zeroized state details

	Locked state	Zeroized state
ok	0	1
Error	1	0
Zeroized	1	1

Table 324. Operations overview

Operation	Input	Result via SR and ORR	Output

Table continues on the next page...

Table 324. Operations overview (continued)

Initialization	-	yes	-
Enroll	-	yes	Activation Code PUF Score
Start	Activation Code	yes	PUF Score
Stop	-	yes	-
Reconstruct	Activation Code	yes	PUF Score
Get Key	Key Destination Context for keys	yes	Key
Wrap Generated Random	Context for keys	yes	Key Code
Wrap	Context for keys User defined key	yes	Key Code
Unwrap	Key destination Key Code	yes	Key
Generate Random	Context for random	yes	Random data
Zeroize	-	yes	-
Test Memory	-	yes	
Test PUF	-	yes	PUF Score

Please see Order of PUF Operations for more details.

14.3.2 Operations

During chip startup, the PUF first tests the PUF SRAM for defects. When defects are found, the PUF SRAM is disqualifed, and PUF does not allow any operations. This prevents security breaches due to a malfunctioning PUF.

Enrollment is done to obtain the device's intrinsic PUF key, and to create helper data (the Activation Code or AC). This must be stored in Non-Volatile Memory (NVM), like fash, EEPROM, hard-disk, cloud, etc. From then on, the device's AC can be provided to start the intrinsic PUF key.

When the intrinsic PUF key is available (which is the case after enrollment and after successful reconstruction), Key operations can be performed.

When all Key operations have been completed for the moment, a Stop command removes all key material from PUF. When more Key operations need to be performed later, this can be enabled by a new start.

14.3.2.1 Order of PUF Operations

Programming 1 to a command bit in the Control register(CR), starts the corresponding operation of PUF. Each operation follows the same protocol.

When the chip programs the command bit to 1 and if it is allowed (indicated by the corresponding allow bit in the Allow register), PUF Engine starts the operation and sets the following status:

- SR[BUSY] = 1
- SR[OK] = 0
- SR[ERROR] = 0

- SR[ALLOW_*] = = 0

After the operation is finished, PUF sets the following status:

- SR[BUSY] = 0
- SR's OK and ERROR bits show the result of the operation
- SR[ALLOW_*] = bits indicate which commands are allowed next

When the result is successful (SR[OK] = 1, SR[ERROR] = 0), PUF moves to the next state. If the result is not successful (SR[OK] = 0, SR[ERROR] = 1), then the software will have to decide what to do next. With result codes (provided via ORR[RESULT_CODE]), PUF gives additional information about the cause of an error.

SR[REJECTED] indicates that in the past a command was not accepted. It is intended as a signal to software that something has gone wrong, and specifc actions may be required. PUF sets SR[REJECTED] = 1 in the following cases:

- When more than one command bit is 1 at a time
- When a new command is given, while the current command is still running
- When a not-allowed command is given
- When a command with incorrect or not-allowed parameters is given

SR[REJECTED] is a write one to clear bit. It stays set until a "1" is written to the bit. While SR[REJECTED] = 1, it is still possible to give commands, and when these commands are valid, PUF will accept them and start the corresponding operation.

NOTE

After reset or power-up PUF starts with initialization: It is busy and no commands are allowed. If a command is given during initialization, it will be rejected.

The Zeroize command deviates from the standard protocol: It can be given at any time and will never be rejected, independent of whether an operation (or initialization) is or is not already running.

14.3.2.1.1 Initialization

Initialization brings the PUF logic in a consistent state after each reset and power cycle. During initialization after a power cycle, PUF runs a memory test on the PUF SRAM.

The memory test automatically resumes when interrupted by a reset, and is not repeated until the next power cycle once completed.

1. System reset or power cycle starts PUF initialization (SR[BUSY] = 1).
2. Wait for SR[BUSY] = 0.
3. Check SR[ERROR] = 0 to verify initialization completed correctly.

NOTE

If the PUF was in the zeroized state before initialization, then it will remain in the zeroized state after initialization.

14.3.2.1.2 Enroll

During the Enroll operation, the SRAM startup values are read, an intrinsic PUF key is generated and the corresponding Activation Code (AC) is generated. The AC must be stored in Non-Volatile Memory (NVM) for future use.

During enrollment, PUF verifies that the PUF SRAM contains qualitative startup data. If that is not the case, enrollment will be terminated and PUF will respond with an error indication. The (partially) provided AC must then be discarded.

After successful completion of the Enroll operation, the diagnostic information about the PUF quality is provided in PSR[PUF_SCORE].

The Activation Code is not sensitive (it does not contain any information on the corresponding intrinsic PUF key) and is also unique for every device and every enrollment. It can be stored in non-secure NVM outside of the chip, or anywhere else where it is accessible to the chip.

The AC and the intrinsic PUF key are linked to a device, and cannot be used on another device. Every time an enrollment is performed, a new intrinsic PUF key is generated together with its AC.

The intrinsic PUF key always stays inside PUF and is never output to the rest of the system.

In principle, enrollment needs to be performed only one time for each device. To prevent unauthorized processes from repeating the Enroll operation, the *disable_enroll* pin can be set to 1. See PUF command blocking configuration (CONFIG) register in the PUF Key Context Management chapter for details.

The following steps should be performed during the Enroll operation:

1. Allocate space for AC (1000 bytes)
2. Set CR[ENROLL] = 1. This will cause the SR[BUSY] bit to set indicating the enroll is in progress.
3. Read the AC from DOR register
    a. Wait for SR[DO_REQUEST] to set.
    b. Read 32-bits of the AC from the DOR register.
    c. Repeat steps a-b until full AC (1000 bytes) has been read.
4. Wait until SR[BUSY] = 0
5. Check SR[ERROR] = 0 to verify enroll completed successfully.
6. Evaluate the PUF score in PSR[PUF_SCORE].
7. When successful: Store AC in NVM.

14.3.2.1.3 Start

During the Start operation, the SRAM start-up values and the AC that was generated during Enroll are used to reconstruct the intrinsic PUF key. When the AC of another device is used, the reconstruction will fail, and PUF will respond with an error indication.

After successful completion of the Start operation, diagnostic information about the PUF quality is provided in PSR[PUF_SCORE].

NOTE

The PUF scores returned after Start and Enroll (or Test PUF) can differ, respectively, because these operations each measure the PUF quality in a different way.

Start operates in two phases, each requiring the AC. Therefore the AC must be sent twice during a Start operation.

The following steps should be performed:

1. Set CR[START] = 1. This will cause the SR[BUSY] bit to set indicating the start is in progress.
2. Write the AC to the DIR register the first time:
    a. Write 32-bits of the AC to the DIR register.
    b. Wait for SR[DI_REQUEST] = 1.
    c. Repeat steps a-b until the entire AC (1000 bytes) has been written.
3. Write the AC to DIR a second time. Using the same procedure used to first the AC the first time (see previous step).
4. Wait until SR[BUSY] = 0.
5. Check SR[ERROR] = 0 to verify start completed successfully.
6. Evaluate the PUF score in PSR[PUF_SCORE].

14.3.2.1.4 Reconstruct operation

The Reconstruct operation is the same as the Start operation, except that Reconstruct does not include the anti-aging phase. This operation can be used instead of Start in the following cases:

- When the PUF SRAM is powered off.
- When the time from power-on or reset to key operations is time-critical. In other cases it is recommended to use the Start operation.

During the Reconstruct operation, the SRAM startup values and the AC that was generated during Enroll are used to reconstruct the intrinsic PUF key. When the AC of another device is used, the reconstruction will fail, and PUF will respond with an error indication.

After successful completion of the Reconstruct operation, diagnostic information about the PUF quality is provided in PSR[PUF_SCORE].

NOTE

The PUF scores returned after Reconstruct (or Start) and Enroll (or Test PUF) can differ, respectively, because these operations each measure the PUF quality in a different way.

The following steps should be performed:

1. Set CR[RECONSTRUCT] = 1, SR[BUSY] = 1.
2. Write the AC to DIR
    a. Write 32-bits of the AC to the DIR register.
    b. Wait for SR[DI_REQUEST] = 1.
    c. Repeat steps a-b until the entire AC (1000 bytes) has been written.
3. Wait until SR[BUSY] = 0.
4. Check SR[ERROR] = 0 to verify start completed successfully.
5. Evaluate the PUF score in PSR[PUF_SCORE].

If the AC can be read directly from NVM, then no memory has to be allocated and the location of the AC in NVM can directly be used as source.

14.3.2.1.5 Stop

The Stop operation removes all key material from PUF and PUF SRAM, and sets PUF to the Stopped state. After this command has finished, Key operations can no longer be performed. This can be used when no Key operations are expected for an extended period of time (e.g. when the system goes into sleep mode), and ensures that the intrinsic PUF key is only present in the system when it is needed, which improves security.

If Key operations need to be performed again after a Stop, then a Start operation needs to be completed first.

The following steps should be performed:

1. Set CR[STOP]= 1. This will cause the SR[BUSY] bit to set indicating the stop is in progress.
2. Wait until SR[BUSY] = 0, PUF is in STOP operation.
3. Check SR[ERROR] = 0 to verify stop completed successfully.

14.3.2.1.6 Get Key

The Get Key operation derives a key from the intrinsic PUF key and externally provided context. The context also defines the length of the derived key. The key is provided to the defined destination interface. When an invalid destination is defined (all bits 0, or more than one bit 1), the command will be rejected. The following steps should be performed when the key destination is the DOR register:

1. Set DATA_DEST[DEST_DOR] = 1.
2. Allocate space for the key.
3. Set CR[GET_KEY] = 1. This will cause the SR[BUSY] bit to set indicating the get key is in progress.

4. Write the key context to the DIR register. See Context specification for Key operations.
    a. Write 32-bits of the KC to the DIR register.
    b. Wait for SR[DI_REQUEST] = 1.
    c. Repeat steps a-b until the entire key context has been written.
5. Read the key.
    a. Wait for SR[DO_REQUEST] to set.
    b. Read 32-bits of the key from the DOR register.
    c. Repeat steps a-b until full key has been read.
6. Wait until SR[BUSY] = 0.
7. Check SR[ERROR] = 0 to verify get key completed successfully.
8. When successful: Pass the key to its destination.

The following steps should be performed when the key destination is the KO interface:

1. Set KEY_DEST[DEST_KO] = 1.
2. Set CR[GET_KEY] = 1. This will cause the SR[BUSY] bit to set indicating the get key is in progress.
3. Write the key context to the DIR register. See Context specification for Key operations.
    a. Write 32-bits of the KC to the DIR register.
    b. Wait for SR[DI_REQUEST] = 1.
    c. Repeat steps a-b until the entire key context has been written.
4. Wait until SR[BUSY] = 0.
5. Check SR[ERROR] = 0 to verify get key completed successfully.

14.3.2.1.7 Wrap Generated Random

The Wrap Generated Random operation wraps a random key into a Key Code (KC). Key Codes start with the provided context. A Key Code is not sensitive (it does not reveal information on the corresponding key) and is linked to the AC that was used during the last Enroll or Start command. It can be stored in non-secure Non-Volatile Memory (NVM) outside of the chip, or anywhere else where it is accessible to the chip. The following steps should be performed:

1. Set CR[WRAP_GENERATED_RANDOM] = 1, This will cause the SR[BUSY] bit to set indicating the wrap generated random is in progress.
2. Write the key context to the DIR register. See Context specification for Key operations.
    a. Write 32-bits of the key context to the DIR register.
    b. Wait for SR[DI_REQUEST] = 1.
    c. Repeat steps a-b until the entire key context has been written.
3. Read the key code from the DOR register:
    a. Wait for SR[DO_REQUEST] to set.
    b. Read 32-bits of the key from the DOR register.
    c. Repeat steps a-b until full key has been read.
4. Wait until SR[BUSY] = 0.
5. Check SR[ERROR] = 0 to verify wrap generated random completed successfully.
6. When successful: Pass the key code to its destination.

14.3.2.1.8 Wrap

The Wrap operation wraps a user defined key into a Key Code (KC). Key Codes start with the provided context. A Key Code is not sensitive (it does not reveal information on the corresponding key) and is linked to the AC that was used during the last Enroll or Start command. It can be stored in non-secure Non-Volatile Memory (NVM) outside of the chip, or anywhere else where it is accessible to the chip.

NOTE

Key Codes are always different, even when the same user defined key and context is used.

The following steps should be performed:

1. Set CR[WRAP] = 1. This will cause the SR[BUSY] bit to set indicating the wrap is in progress.
2. Write the key context to the DIR register. See Context specification for Key operations.
    a. Write 32-bits of the key context to the DIR register.
    b. Wait for SR[DI_REQUEST] = 1.
    c. Repeat steps a-b until the entire key context has been written.
3. Alternate writing the key to the DIR and reading the key code from the DOR.
    a. Write 32-bits of the key to the DIR register.
    b. Wait for SR[DO_REQUEST] = 1.
    c. Read 32-bits of the key code from the DOR register.
    d. Wait for SR[DI_REQUEST] = 1.
    e. Repeat steps a-d until the entire key has been written, and the entire key code has been read out.
4. Wait until SR[BUSY] = 0.
5. Check SR[ERROR] = 0 to verify wrap completed successfully.
6. When successful: Pass the key code to its destination.

14.3.2.1.9 Unwrap

The Unwrap operation unwraps the key from a previously created Key Code. Depending on the state of PUF and the key scope that is included in the Key Code, the allowed destinations may be limited. When an invalid destination is defined (all bits 0, or more than one bit 1), the command will be rejected.

The following steps should be performed when the key destination is the DOR register:

1. Set DATA_DEST[DEST_DOR] = 1 (other bits 0).
2. Set CR[UNWRAP] = 1. This will cause the SR[BUSY] bit to set indicating the unwrap is in progress.
3. Alternate writing the key code to the DIR and reading the key from the DOR.
    a. Write 32-bits of the key code to the DIR register.
    b. Wait for SR[DO_REQUEST] = 1.
    c. Read 32-bits of the key from the DOR register.
    d. Wait for SR[DI_REQUEST] = 1.
    e. Repeat steps a-d until the entire key code has been written, and the entire key has been read out.
4. Wait until SR[BUSY] = 0.
5. Check SR[ERROR] = 0 to verify unwrap completed successfully.
6. When successful: Pass the key to its destination.

The following steps should be performed when the key destination is the KO interface:

1. Set DATA_DEST[DEST_SO] = 1 (other bits 0).
2. Prepare receiver to accept key via SO interface.
3. Set CR[UNWRAP] = 1. This will cause the SR[BUSY] bit to set indicating the unwrap is in progress.
4. Write the key code to the DIR register:
    a. Write 32-bits of the key code to the DIR register.
    b. Wait for SR[DI_REQUEST] = 1.
    c. Repeat steps a-b until the entire key code has been written.
5. Wait until SR[BUSY] = 0.
6. Check SR[ERROR] = 0 to verify unwrap completed successfully.

14.3.2.1.10 Generate Random

The Generate Random operation outputs the requested amount of random data as specifed in a provided context.

The following steps should be performed when the data destination is the DOR register:

1. Set DATA_DEST[DEST_DOR] = 1 (other bits 0).
2. Set CR[GENERATE_RANDOM] = 1. This will cause the SR[BUSY] bit to set indicating the generate random is in progress.
3. Write the context input to the DIR register. See Context specification for Key operations.
    a. Write 32-bits of the context to the DIR register.
    b. Wait for SR[DI_REQUEST] = 1.
    c. Repeat steps a-b until the entire context has been written.
4. Read the random data from DOR.
    a. Wait for SR[DO_REQUEST] to set.
    b. Read 32-bits of the random data from the DOR register.
    c. Repeat steps a-b until requested amount of random data has been read.
5. Wait until SR[BUSY] = 0.
6. Check SR[ERROR] = 0 to verify generate random completed successfully.
7. When successful: Use the random data as needed.

The following steps should be performed when the data destination is the SO interface:

1. Set DATA_DEST[DEST_SO] = 1 (other bits 0).
2. Set CR[GENERATE_RANDOM] = 1. This will cause the SR[BUSY] bit to set indicating the generate random is in progress.
3. Write the context input to the DIR register. See Context specification for Key operations.
    a. Write 32-bits of the context to the DIR register.
    b. Wait for SR[DI_REQUEST] = 1.
    c. Repeat steps a-b until the entire context has been written.
4. Wait until SR[BUSY] = 0.
5. Check SR[ERROR] = 0 to verify generate random completed successfully.
6. When successful: Use the random data as needed.

14.3.2.1.11 Test Memory operation

With the Test Memory operation, the PUF SRAM is tested for defects.

If the test fails, an error is indicated and PUF goes to the Locked state.

The following steps should be performed during the Test Memory operation:

1. Set CR[TEST_MEMORY] = 1
2. Wait until SR[BUSY] = 0.
3. Evaluate result (SR or ORR register)

14.3.2.1.12 Zeroize

The Zeroize operation scrambles the PUF and removes all key material from PUF Engine and PUF SRAM. This disables all commands until the next power cycle. The Zeroize command can be used when a security breach is detected.

When the Zeroize operation is successful, PUF goes to the Zeroized state. If it fails, PUF goes to the Locked state.

See the "Zeroize operation" section in the "Chip-specific PUF information" section for more details.

Zeroize takes precedence over all other activities of PUF. This means it can also be started during initialization, or while another operation is active. Zeroize cancels the current operation, including all input and output activities.

NOTE

When Zeroize occurs during an operation, ongoing data transfers via the DIR or DOR registers may be interrupted. This may cause APB errors and exceptions, depending on the hardware integration and software configuration. Software must ensure that these events can be handled.

Transfers to ELS through HW bus may also be interrupted.

The following steps should be performed:

1. Set CR[ZEROIZE] = 1. This will cause SR[BUSY] to set indicating the zeroize operation is in progress.
2. Wait until SR[BUSY] = 0.
3. Verify the zeroize state (SR[ZEROIZE] = 1).

Instead of software programming the zeroize bit, an external module can start a Zeroize operation by setting the pin zeroize = 1.

NOTE

The Zeroize command (either via the CR register or via the zeroize pin) only has effect when a clock is present and no reset is applied.

14.3.2.1.13 Test PUF

With the Test PUF operation, diagnostics about the PUF quality are collected and presented in a PUF score. When the operation is successful, the result is shown in PSR[PUF_SCORE].

NOTE

The Test PUF operation collects the same diagnostic information as Enroll.

NOTE

Test PUF can be run one time after a power cycle, or reset .

The following steps should be performed during the Test PUF operation:

1. Set CR[TEST_PUF] = 1. This will cause SR[BUSY] to set indicating the zeroize operation is in progress.
2. Wait until SR[BUSY] = 0.
3. Verify the results (SR).

14.3.3 Data formats

14.3.3.1 AC format

The Activation Code is 1000 bytes and includes the following fields:

- Header (64 bits): Used by PUF to check that the AC is intended for this product version; it contains the following fields:
  - Header ID (16 bits)
  - Maj (8 bits): Major product version
  - Min (8 bits): Minor product version
  - Hardware ID (32 bits): Same value as in the HW_ID register
- Payload: The actual data used by PUF
- Checksum (32 bits): Used by PUF to check for transfer errors; can also be used by software to verify that the AC is stored correctly in NVM.

The indicated bit numbers are for the big endian data format (default). When the little endian data format is used, the bytes have to be swapped per word.

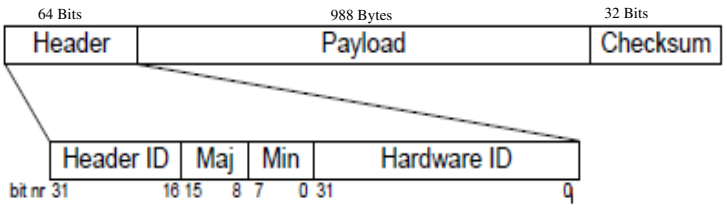


Figure 72. Format of Activation Code (big endian data format)

14.3.3.2 KC format

The Key Code includes the following fields:

- Header (128 bits): Defines the product version and the context provided during the Wrap or Wrap Generated Random operation; it contains the following fields:
  - Maj (4 bits): Major product version
  - Min (4 bits): Minor product version
  - Context: Words 0 to 3 of the provided context (refer to the following subsections for details).
- Payload: The actual data used by PUF
- Checksum (32 bits): Used by PUF to check for transfer errors; can also be used to verify that the KC is stored correctly in NVM.

The indicated bit numbers are for the big endian data format (default). When the little endian data format is used, the bytes have to be swapped per word.

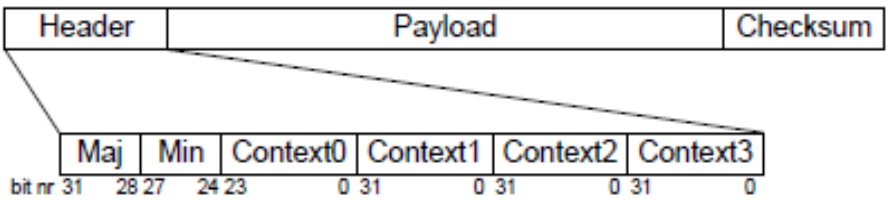


Figure 73. Format of Key Code (big endian data format)

See Table 328 for the key code lengths for different key sizes.

14.3.3.2.1 Context specification for Key operations

The context input for Key operations is used to specify the properties of a key, and enables the explicit differentiation of keys that have distinct purposes. Under no circumstance must the same context be used on the same device for keys with different properties or purposes.

For Get Key operations, the combination of the intrinsic PUF key and the context uniquely defines the key that is output. If the same context is used, the same key is produced.

For Wrap and Wrap Generated Random operations, the combination of the intrinsic PUF key and context uniquely determines which keys are used to protect the wrapped data. Each key code is still unique, even when the same data and context are used.

Word	Bit	Field	Description
0	23-16	context_type	0x10: Context for Key operations Other values: Reserved
	15-13	Reserved	Must be 0
	12-0	key_length	Length of the key in bits; the following lengths are supported: 64 - 1024 bits in 64-bit increments 2048 bits 3072 bits 4096 bits

Table continues on the next page...

Table continued from the previous page...

			NOTE The key length is not necessarily the same as the security strength.
1	31-16	key_type	0x0000: Generic Other values: Reserved
	15-8	key_scope_started	Defines the allowed key destinations, when PUF is in the Started state: bit 8: Key can be made available via the DOR register bit 9: Key can be made available to ELS through HW bus Others: Reserved
	7-0	key_scope_enrolled	Defines the allowed key destinations when PF is in the enrolled state: bit 0: Key can be made available via the DOR register bit 1: Key can be made available to ELS through HW bus Others: Reserved
2 (*Also see* HW_RUC0 *register*.)	31:28	access_level	Restrict the key access based on TrustZone security level. b'0000 - Key is usable even when access level protection feature is not enabled. b'0001 - Key is usable at non-secure user and above levels b'001x - key is usable at non-secure privilege level and above. b'01xx - Key is usable at secure user level and above

Table continues on the next page...

Table continued from the previous page...

			b'1xxx - Key is usable at secure privilege level only.
	27	Reserved	Must be 0.
	26	dsp_debug	Disable key access when debugger is attached to DSP after power-up. Set this bit to 1 during key code creation (wrap step). When set the key cannot be unwrapped if a debug session is established with the DSP core on device. The debug state of the device is reset only on PoR/BoR/ SWR_reset reset.
	25	coolflux_debug	Must be 0.
	25	Reserved	Must be 0.
	23:8	Boot_state	Temporal boot state. The boot-up process of the device can be broken in to 16 stages which tracked by monotonic counter register in SYSCON block. Each stage is represented by the bit position. Set the bitfield to inverted value of the encoded boot stages state. Boot state is encoded as below: • 0x00 - Stage 0: Early boot stage in ROM • 0x01 - Stage 1: NXP signed firmware stage (used for provisioning) • 0x03 - Stage 2: OEM boot loader or PRoT(Platform Root of Trust) • 0x07 … 0xFF - Stages defined by OEM boot & security policy. For example to restrict a key to "Stage 1" and below then set the value of this field to

Table continues on the next page...

Table continued from the previous page...

			0xFE. The key code can't be unwrapped in stage 2 and beyond.
	7:0	lc_state	Life cycle state based restrictions. Set the bitfield to inverted value of the encoded life cycle state value to restrict the key usage to specific life cycle state. Life cycle state is encoded as below: • 0000_0011 OEM Open • 0000_0111 OEM Secure World • 0000_1111 OEM Closed • 0001_1111 Field Return OEM • 0011_1111 OEM recommission • 0011_1111 Field Return NXP • 1100_1111 OEM Locked (no return) • 1111_1111 Shredded To restrict key usage to multiple states this field should be set to inverted ORed result of encoded value. For example to restrict a key to *OEM Open*, *OEM Secure world* and *OEM Closed* states then set the values of this field to 1111_0000. The key code can't be unwrapped in states beyond *OEM closed* state. For example to restrict a key to *OEM Open*, *OEM Secure world, OEM Closed* and *OEM Locked* states then set the values of this field to 0011_0000. The key code

Table continues on the next page...

Table continued from the previous page...

			can't be unwrapped in states beyond *OEM closed* state.
3 (*Also see* HW_RUC1 *register*.)	31-0	APP_CTX	Application customizable context. Product designer can sub-divide this context word to implement custom key policy on top of the mechanism provided by context 0 to 3 word. The content of this word is influenced by the APP_CTX_MASK register. For example, Platform Root of Trust (PRoT) module running at secure-privilege level can change the APP_CTX_MASK register when application task switch happens. Each application task can be allocated each a bit in APP_CTX_MASK. • Task 0 mask will be 0xFFFFFFFE • Task 1 mask will be 0xFFFFFFFD • Task 2 mask will be 0xFFFFFFFB • … • Task 31 mask will be 0x7FFFFFFF Then keys can be isolated per each application. • Task 0 keys should have this field set to 0x0000001. • Task 1 keys should have this field set to 0x0000002. • Task 2 keys should have this field set to 0x0000004. • …

Table continues on the next page...

Table continued from the previous page...

			• Task 31 keys should have this field set to 0x8000000. Similarly another example would be to use APP_CTX_MASK to isolate keys by usage algorithms.

For a Get Key operation, key_length defines the length of the requested key. For a Wrap or Wrap Generated Random operation, key_length defines the length of the key that must be wrapped.

The key_scope_* fields can be used to limit the accepted destinations of the key. This can be used when a key must be shared with an application, but in the field the key may not be exposed to the microcontroller.

For security reasons, user context bits that are not required for the intended application also should not be available to other parties. Therefore, it is recommended to limit the number of bits that can be used in the user context fields as much as possible. This can be done with the restrict_user_context_0 and restrict_user_context_1 pins.

See the PUF Context Key Management section of the PUF Key management chapter for more details.

14.3.3.2.2 Context specification for Generate Random operations

The context for Generate Random operations is used for the internal random generation.

Table 325.

Word	Bit	Field	Description
0	31-24	Reserved	Must be 0
	23-16	*context_type*	0x00: Context for Generate Random Other values: Reserved
	15-13	Reserved	Must be 0
	12-0	*data_length*	Length of the requested data in bits; the following lengths are supported: • 64 - 1024 bits in 64-bit increments • 2048 bits • 3072 bits • 4096 bits

14.3.3.2.3 Examples of using context for Key operations

The way the context for Key operations can be used is explained with the following example. The example assumes that a key of 256 bits must be shared with a central server during production/end test, and that the key may not be exposed when the product in is the field so they can only be sent to the Security/Crypto module).

14.3.3.2.3.1 Example for Get Key

In this example, the key is derived by PUF.

- During production or end testing (after Enroll), define a key and send to the central server:

1. Define the key length: 256 bits, 0x00000100
2. Define the key scope to 0x00000203: Not restricted after Enroll, restricted to KO after Start
3. Define the user context for the first key
4. Program the key destination to Register (KEY_DEST = 0x00000001)
5. Run the Get Key operation
6. The key will be made available via the DOR register
7. Send the key to the central server

- During the production/end test, the key must also be used by the Security/Crypto modules, the key is derived again and now sent via the KO interface:
  1. Define the key length: 256 bits
  2. Define the key scope to 0x00000203: Not restricted after Enroll, restricted to KO after Start
  3. Define the user context for the key
  4. Program the key destination to KO interface (KEY_DEST = 0x00000002)
  5. Run the Get Key operation
  6. The key will be made available to the Security/Crypto modules via the KO interface
- During use in the field (after Start); the same procedure is used as the last step during production/end test:
  1. Define the key length: 256 bits
  2. Define the key scope to 0x00000203: Not restricted after Enroll, restricted to KO after Start
  3. Define the user context for the key
  4. Program the key destination to KO interface (KEY_DEST = 0x00000002)

NOTE

KEY_DEST = 0x00000001 cannot be used here, because that was disallowed by the key scope

  5. Run the Get Key operation
  6. The key will be made available to the Security/Crypto modules via the KO interface

14.3.3.2.3.2 Example for Wrap

In this example, the key is not derived by PUF, but defined and provided by the user.

During production or end testing (after Enroll):

1. Wrap key (e.g. obtained from a central server):
   a. Define the key length: 256 bits, 0x00000100
   b. Define the key scope started and key scope enrolled fields to 0x00000202: Restricted to KO after Enroll and after Start
   c. Define the user context for the key
   d. Run the Wrap operation, provide the key that must be wrapped: key_1
   e. Store the Key Code in NVM as KC_1
2. When during the production/end test, the key must also be used by the Security/Crypto modules, the previously wrapped key is unwrapped and sent via the KO interface. Because the context is part of the Key Code, it does not have to be defined during the unwrap operation.
   a. Program the key destination to KO interface (KEY_DEST = 0x00000002)

NOTE

KEY_DEST = 0x00000001 cannot be used, because that was disallowed by the key scope

b. Run the Unwrap operation, provide the Key Code: KC_1 (or KC_2)

c. The key will be made available to the Security/Crypto modules to ELS only through HW bus

3. During use in the field (after Start); the same procedure is used as the last step during production/end test. Because the context is part of the Key Code, it does not have to be defined during the unwrap operation.

a. Program the key destination to KO interface (KEY_DEST = 0x00000002)

NOTE

KEY_DEST = 0x00000001 cannot be used, because that was disallowed by the key scope

b. Run the Unwrap operation, provide the Key Code: KC_1 (or KC_2)

c. The key will be made available to the Security/Crypto modules to ELS only through HW bus

14.3.3.3 PUF characteristics

14.3.3.3.1 Number clocks per operation

An indication of the length of each operation is shown in table below.

Table 326. Indication of the maximum number of clock cycles and time per operation

Operation	#clock cycles
Initialization (after power cycle)	22700
Initialization (after warm reset)	1800
Enroll	35200
Start (12.5% PUF noise)	57200
Start (worst case PUF noise)	61550
Reconstruct (nominal PUF noise)	39400
Reconstruct (worst case PUF noise)	41650
Stop	1700
Get Key (256 bits)	1200
Wrap Generated Random (256 bits)	6050
Wrap (256 bits)	4450
Unwrap (256 bits)	2800
Generate Random (256 bits)	1800
Test Memory	12600
Test PUF	2450
Zeroize	1500

NOTE

The number of clock cycles is based on immediate availability and acceptance of data when PUF requests or provides data.

14.3.3.3.2 Length of Activation Code

Table 327. Length of Activation Code

Parameter	Number of Bytes
Activation code	1000

14.3.3.3.3 Length of Key Code

The length of the key code can be calculated from the key length in bits as follows:

```
KC_length(bytes) = 36 + key_length(bits)/8 + 16 x [key_length(bits)/384]
```

Table 328. Key code lengths for the accepted key lengths

Key length (bits)	Key length (bytes)	Key code length (bytes)
64	8	60
128	16	68
192	24	76
256	32	84
320	40	92
384	48	100
448	56	124
512	64	132
576	72	140
640	80	148
704	88	156
768	96	164
832	104	188
896	112	196
960	120	204
1024	128	212
2048	256	388

Table continues on the next page...

Table 328. Key code lengths for the accepted key lengths (continued)

Key length (bits)	Key length (bytes)	Key code length (bytes)
3072	384	548
4096	512	724

14.3.4 Clocks

The AHB clock is the clock source for PUF.

14.3.5 Interrupts

The PUF block supports the following interrupts:

- Busy
- Ok
- Error
- Zeroized
- Rejected
- Data In request
- Data out requset

14.4 Memory Map and register definition

This section includes the PUF module memory map and detailed descriptions of registers.

There are 4 aliased addresses, all 4 can access the same registers.

14.4.1 PUF register descriptions

14.4.1.1 PUF memory map

PUF base address: 4002_C000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Control (CR)	32	RW	0000_0000h
4h	Operation Result (ORR)	32	R	0000_0000h
8h	Status (SR)	32	RW	0000_0001h
Ch	Allow (AR)	32	R	0000_0000h
10h	Interrupt Enable (IER)	32	RW	0000_0000h
14h	Interrupt Mask (IMR)	32	RW	0000_0000h
18h	Interrupt Status (ISR)	32	RW	0000_0000h
20h	Data Destination (DATA_DEST)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
24h	Data Source (DATA_SRC)	32	RW	0000_0000h
A0h	Data Input (DIR)	32	W	0000_0000h
A8h	Data Output (DOR)	32	R	0000_0000h
C0h	Miscellaneous (MISC)	32	RW	0000_0001h
D0h	Interface Status (IF_SR)	32	RW	0000_0000h
DCh	PUF Score (PSR)	32	R	0000_000Fh
E0h	Hardware Restrict User Context 0 (HW_RUC0)	32	R	See section
E4h	Hardware Restrict User Context 1 (HW_RUC1)	32	R	0000_0000h
F4h	Hardware Information (HW_INFO)	32	R	2100_0000h
F8h	Hardware Identifier (HW_ID)	32	R	2104_2000h
FCh	Hardware Version (HW_VER)	32	R	0003_0800h
300h	SRAM Configuration (SRAM_CFG)	32	RW	See section
304h	Status (SRAM_STATUS)	32	R	See section
3D8h	Interrupt Enable Clear (SRAM_INT_CLR_ENABLE)	32	W	See section
3DCh	Interrupt Enable Set (SRAM_INT_SET_ENABLE)	32	W	See section
3E0h	Interrupt Status (SRAM_INT_STATUS)	32	R	See section
3E4h	Interrupt Enable (SRAM_INT_ENABLE)	32	R	See section
3E8h	Interrupt Status Clear (SRAM_INT_CLR_STATUS)	32	W	See section
3ECh	Interrupt Status set (SRAM_INT_SET_STATUS)	32	W	See section

14.4.1.2 Control (CR)

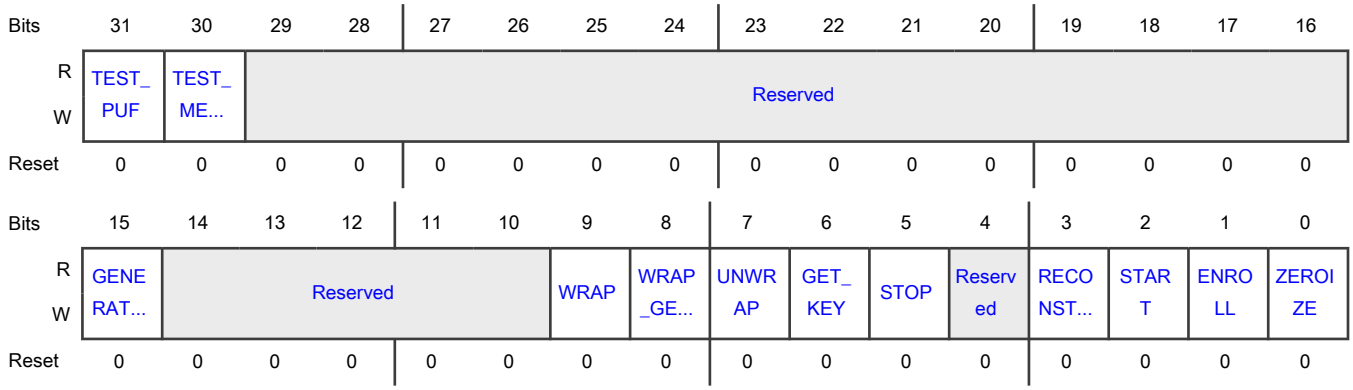
Offset

Register	Offset
CR	0h

Function

The Control register defines which command must be executed next. When the command is accepted or rejected, the bits automatically revert to 0. Only one command bit may be written with 1 at a time, with the exception of zeroize. Writing zeroize with 1 takes precedence over all other commands.

Diagram



Fields

Field	Function
31 TEST_PUF	Test PUF operation 0b - Test PUF operation is disabled. 1b - Test PUF operation is enabled.
30 TEST_MEMORY	Test memory operation 0b - Test memory operation is disabled. 1b - Test memory operation is enabled.
29-16 —	Reserved
15 GENERATE_RANDOM	Generate Random operation 0b - Generate Random operation disabled 1b - Generate Random operation enabled
14-10 —	Reserved
9 WRAP	Wrap operation 0b - Wrap operation disabled 1b - Wrap operation enabled
8 WRAP_GENERATED_RANDOM	Wrap Generated Random operation 0b - Wrap generated random operation disabled 1b - Wrap generated random operation enabled
7 UNWRAP	Unwrap operation 0b - Unwrap operation disabled

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - Unwrap operation enabled
6 GET_KEY	Get Key operation 0b - Get Key operation disabled 1b - Get Key operation enabled
5 STOP	Stop operation 0b - Stop operation disabled 1b - Stop operation enabled
4 —	Reserved
3 RECONSTRUCT	Reconstruct operation 0b - Reconstruct operation disabled 1b - Reconstruct operation enabled
2 START	Start operation 0b - Does not begin start operation (disabled). 1b - Begins the start operation (enabled).
1 ENROLL	Enroll operation 0b - Does not begin enroll operation (disabled). 1b - Begins the enroll operation (enabled).
0 ZEROIZE	Zeroize operation 0b - Does not begin zeroize operation (disabled). 1b - Begins the zeroize operation (enabled).

14.4.1.3 Operation Result (ORR)

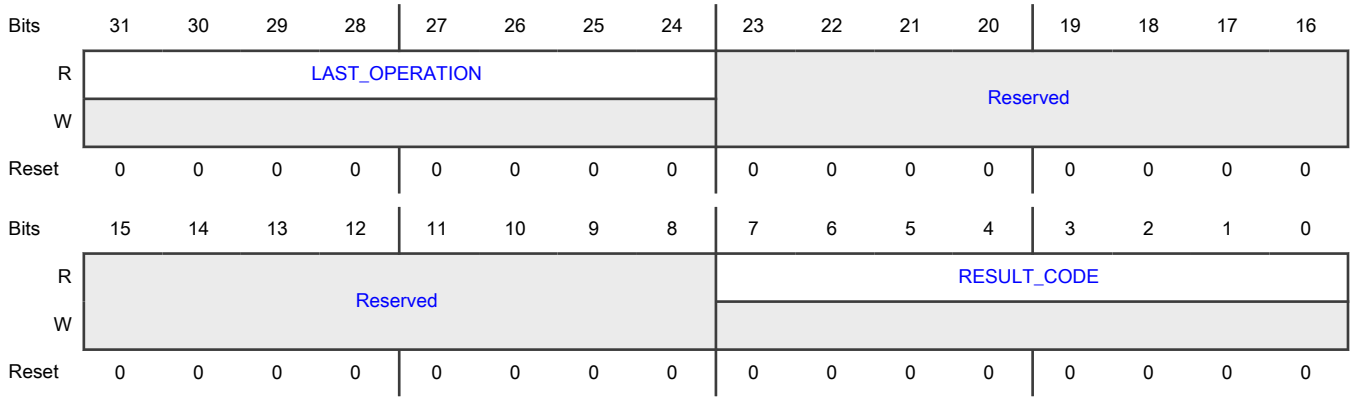
Offset

Register	Offset
ORR	4h

Function

The Operation Result register provides the result code of the last operation (Initialization, Generate Random). The value in this register is only valid when busy = 0.

Diagram



Fields

Field	Function
31-24 LAST_OPERATION	Last operation type 0000_0000b - Indicates that the operation is in progress. 0000_0001b - Indicates that the last operation was Enroll. 0000_0010b - Indicates that the last operation was Start. 0000_0011b - Indicates that the last operation was Reconstruct 0000_0101b - Indicates that the last operation was Stop. 0000_0110b - Indicates that the last operation was Get Key. 0000_0111b - Indicates that the last operation was Unwrap. 0000_1000b - Indicates that the last operation was Wrap Generated Random. 0000_1001b - Indicates that the last operation was Wrap. 0000_1111b - Indicates that the last operation was Generate Random. 0001_1110b - Indicates that the last operation was Test Memory. 0001_1111b - Indicates that the last operation was Test PUF. 0010_0000b - Indicates that the last operation was Initialization. 0010_1111b - Indicates that the last operation was Zeroize.
23-8 —	Reserved
7-0 RESULT_CODE	Result code of last operation 0000_0000b - Indicates that the last operation was successful or operation is in progress. 1111_0000b - Indicates that the AC is not for the current product/version. 1111_0001b - Indicates that the AC in the second phase is not for the current product/version. 1111_0010b - Indicates that the AC is corrupted.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1111_0011b - Indicates that the AC in the second phase is corrupted. 1111_0100b - Indicates that the authentication of the provided AC failed. 1111_0101b - Indicates that the authentication of the provided AC failed in the second phase. 1111_0110b - Indicates that the SRAM PUF quality verification fails. 1111_0111b - Indicates that the incorrect or unsupported context is provided. 1111_1000b - Indicates that a data destination was set that is not allowed according to other settings and the current PUF state. 1111_1111b - Indicates that the PUF SRAM access has failed.

14.4.1.4 Status (SR)

Offset

Register	Offset
SR	8h

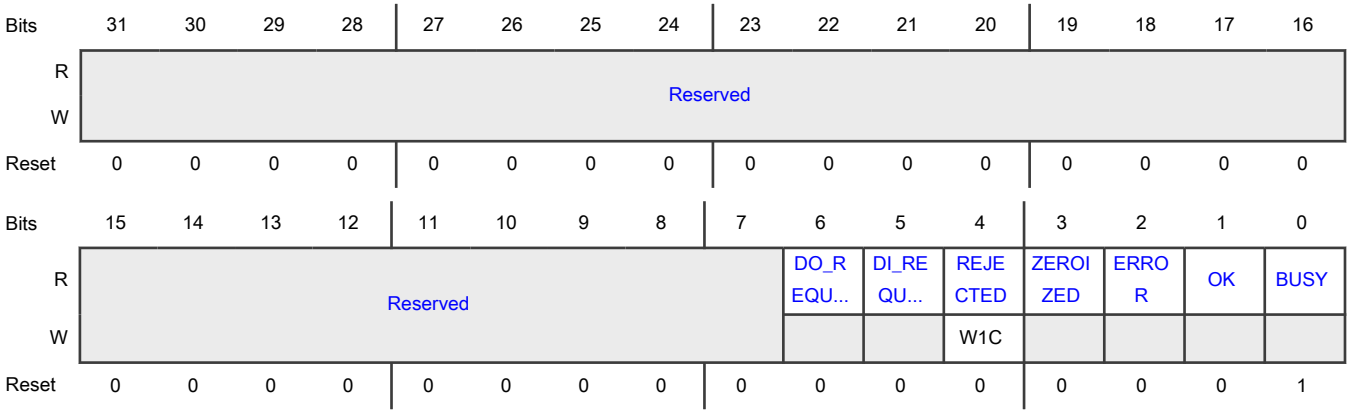
Function

The Status register shows the current status of PUF and indicates whether a data transfer is requested.

NOTE

The reset value 0x0000_0001 is present immediately after reset. After finishing initialization, the busy bit will go to 0 and, depending on the result of initialization, either the OK bit or the ERROR and ZEROIZED bits will go to 1.

Diagram



Fields

Field	Function
31-7 —	Reserved
6 DO_REQUEST	Indicates the request for data out transfer via the DOR register 0b - Indicates no request for data out transfer through DOR register 1b - Indicates the request for data out transfer through DOR register
5 DI_REQUEST	Indicates the request for data in transfer via the DIR register 0b - Indicates no request for data in transfer through DIR register 1b - Indicates the request for data in transfer through DIR register
4 REJECTED	Operation rejected Read: Indicates that the last command was rejected Write 1: Clears this bit 0b - Indicates that the last command was not rejected 1b - Indicates that the last command was rejected
3 ZEROIZED	Zeroized or Locked state 0b - Indicates that the state is not locked 1b - Indicates that the state is zeroized or locked
2 ERROR	Last operation failed 0b - Resets value, no special meaning 1b - Indicates that the last operation failed
1 OK	Last operation successful 0b - Resets value, no special meaning 1b - Indicates that the last operation is successful
0 BUSY	Operation in progress 0b - Indicates no operation is performed and the hardware is idle 1b - Indicates the operation is in progress and the hardware is busy

14.4.1.5 Allow (AR)

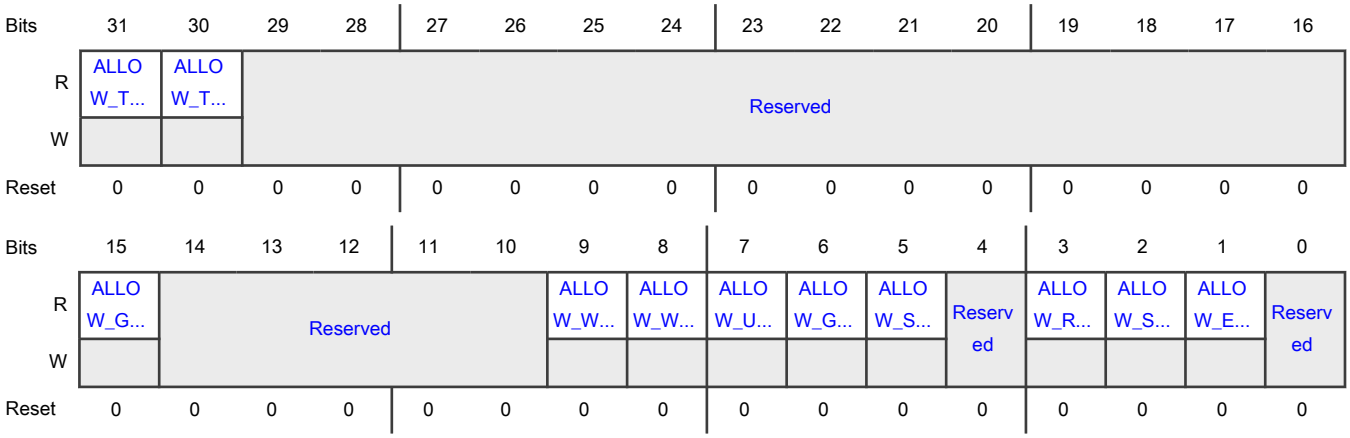
Offset

Register	Offset
AR	Ch

Function

The Allow register indicates which operations are currently allowed.

Diagram



Fields

Field	Function
31 ALLOW_TEST_PUF	Test PUF operation 0b - Test PUF operation is not allowed 1b - Test PUF operation is allowed
30 ALLOW_TEST_MEMORY	0b - Indicates that the Test Memory operation is not allowed 1b - Indicates that the Test Memory operation is allowed
29-16 —	Reserved
15 ALLOW_GENERATE_RANDOM	Generate Random operation 0b - Indicates that the Generate Random operation is not allowed 1b - Indicates that the Generate Random operation is allowed
14-10 —	Reserved
9 ALLOW_WRAP	Wrap operation 0b - Indicates that the Wrap operation is not allowed 1b - Indicates that the Wrap operation is allowed
8	Wrap Generated Random operation 0b - Indicates that the Wrap Generated Random operation is not allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
ALLOW_WRAP_GENERATED_RANDOM	1b - Indicates that the Wrap Generated Random operation is allowed
7 ALLOW_UNWRAP	Unwrap operation 0b - Indicates that the Unwrap operation is not allowed 1b - Indicates that the Unwrap operation is allowed
6 ALLOW_GET_KEY	Get Key operation 0b - Indicates that the Get Key operation is not allowed 1b - Indicates that the Get Key operation is allowed
5 ALLOW_STOP	Stop operation 0b - Indicates that the Stop operation is not allowed 1b - Indicates that the Stop operation is allowed
4 —	Reserved
3 ALLOW_RECONSTRUCT	Reconstruct operation 0b - Indicates that the Reconstruct operation is not allowed 1b - Indicates that the Reconstruct operation is allowed
2 ALLOW_START	Start operation 0b - Indicates that the Start operation is not allowed 1b - Indicates that the Start operation is allowed
1 ALLOW_ENROLL	Enroll operation 0b - Indicates that the Enroll operation is not allowed 1b - Indicates that the Enroll operation is allowed
0 —	Reserved

14.4.1.6 Interrupt Enable (IER)

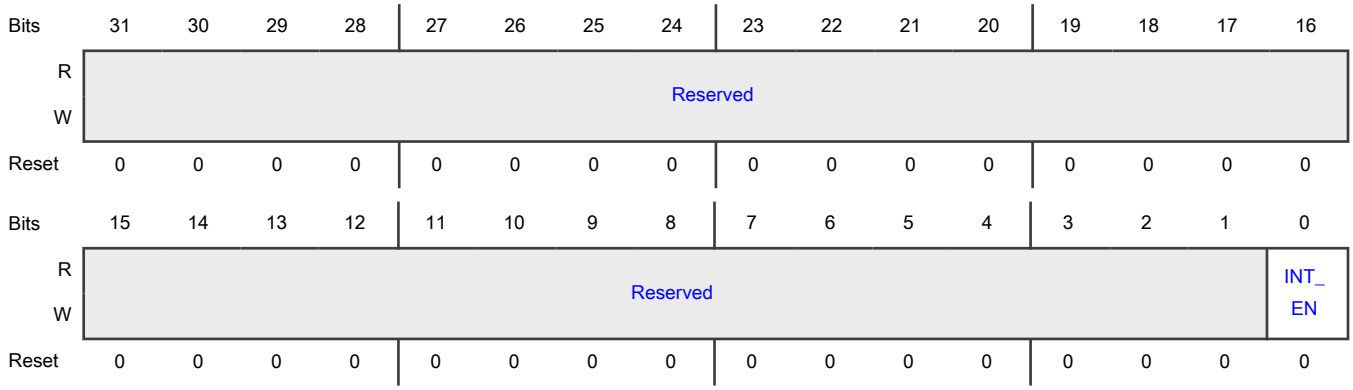
Offset

Register	Offset
IER	10h

Function

With the Interrupt Enable register all PUF interrupts can be enabled or disabled, without changing the Interrupt Mask register.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_EN	Interrupt enable 0b - Disables all PUF interrupts 1b - Enables all PUF interrupts that are enabled in the Interrupt Mask register

14.4.1.7 Interrupt Mask (IMR)

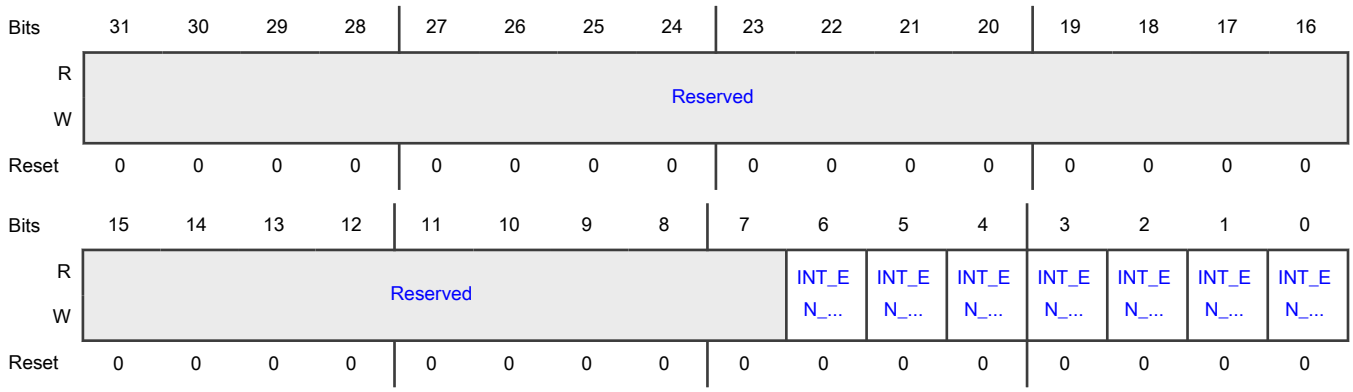
Offset

Register	Offset
IMR	14h

Function

Individual PUF interrupts can be enabled or disabled with the Interrupt Mask register.

Diagram



Fields

Field	Function
31-7 —	Reserved
6 INT_EN_DO_REQUEST	Data out request interrupt 0b - Disables the data out request interrupt 1b - Enables the data out request interrupt
5 INT_EN_DI_REQUEST	Data in request interrupt 0b - Disables the data in request interrupt 1b - Enables the data in request interrupt
4 INT_EN_REJECTED	Rejected interrupt 0b - Disables the rejected interrupt 1b - Enables the rejected interrupt
3 INT_EN_ZEROIZED	Zeroized interrupt 0b - Disables the zeroized interrupt 1b - Enables the zeroized interrupt
2 INT_EN_ERROR	Error interrupt 0b - Disables the error interrupt 1b - Enables the error interrupt
1 INT_EN_OK	Ok interrupt 0b - Disables the Ok interrupt 1b - Enables the Ok interrupt
0 INT_EN_BUSY	Busy interrupt 0b - Disables the busy interrupt 1b - Enables the busy interrupt

14.4.1.8 Interrupt Status (ISR)

Offset

Register	Offset
ISR	18h

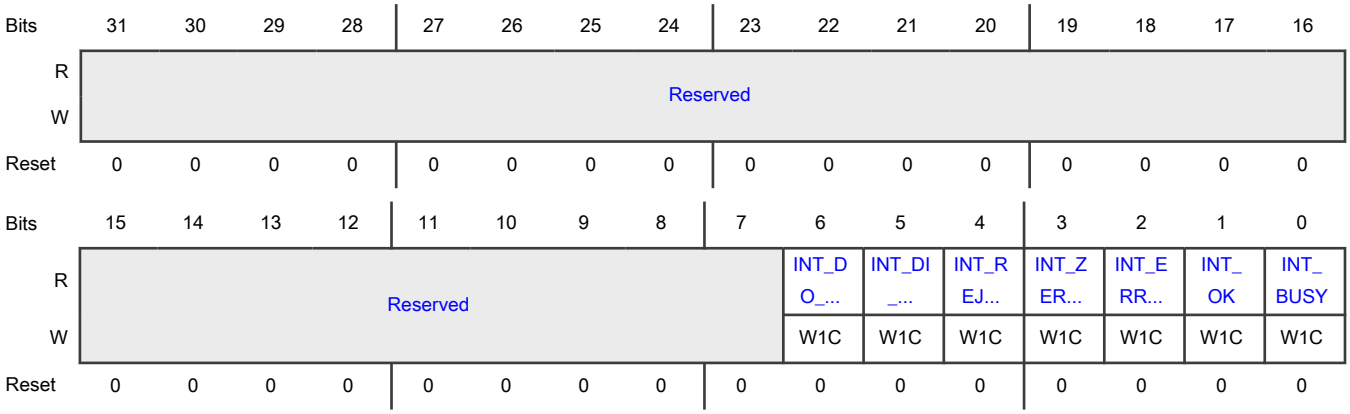
Function

The Interrupt Status register indicates which interrupt occurred.

Multiple interrupts may occur at the same time. Busy interrupt occurs together with one of the ok and error interrupts.

Writing 1 to a bit, clears the corresponding bit.

Diagram



Fields

Field	Function
31-7 —	Reserved
6 INT_DO_REQUEST	Positive edge occurred on do_request Indicates that positive edge occurred on do_request, which means that a data out transfer is requested via the DOR register 0b - Does not have do_request interrupt 1b - do_request (data out transfer is requested through the DOR register) interrupt appears
5 INT_DI_REQUEST	Positive edge occurred on di_request Indicates that positive edge occurred on di_request, which means that a data in transfer is requested via the DIR register 0b - Does not have di_request interrupt 1b - di_request (data in transfer is requested through the DIR register) interrupt appears
4 INT_REJECTED	Positive edge occurred on Rejected Indicates that positive edge has occurred on Rejected, which means that a command was rejected 0b - Does not have rejected interrupt 1b - rejected (a command was rejected) interrupt appears
3 INT_ZEROIZED	Positive edge occurred on Zeroized Indicates that positive edge has occurred on Zeroized, which means that PUF has moved to the Zeroized or Locked state 0b - Does not have zeroized interrupt 1b - zeroized (PUF has moved to the zeroized or locked state) interrupt appears

Table continues on the next page...

Table continued from the previous page...

Field	Function
2 INT_ERROR	Positive edge occurred on Error Indicates that positive edge has occurred on Error, which means that an operation has failed 0b - Does not have error interrupt 1b - int_error (an operation failed) interrupt appears
1 INT_OK	Positive edge occurred on Ok Indicates that positive edge has occurred on Ok, which means that an operation has successfully completed 0b - Does not have OK interrupt 1b - int_ok (an operation has successfully completed) interrupt appears
0 INT_BUSY	Negative edge occurred on Busy Indicates that negative edge has occurred on busy, which means that an operation has completed 0b - Does not have busy interrupt 1b - int_busy (an operation has completed) interrupt appears

14.4.1.9 Data Destination (DATA_DEST)

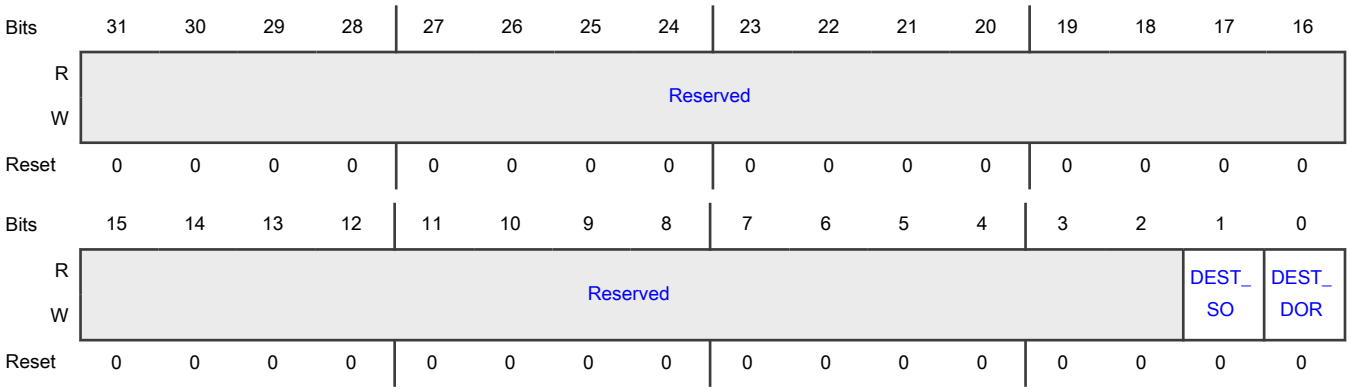
Offset

Register	Offset
DATA_DEST	20h

Function

The Data Destination register defines via which interface the data is made available by PUF. Only one bit at a time may be 1.

Diagram



Fields

Field	Function
31-2 —	Reserved
1 DEST_SO	Key available to ELS Defines that the key will be made available to ELS only through HW bus 0b - Key is unavailable through HW bus to ELS. If DEST_SO is set to 0, you can set DEST_DOR to 1 to make key available 1b - Key is available through HW bus to ELS
0 DEST_DOR	Key available via the DOR register Defines that the key will be made available via the DOR register 0b - Key is unavailable through DOR register. If DEST_DOR is set to 0, you can set DEST_SO to 1 to make key available 1b - Key is available through DOR register

14.4.1.10 Data Source (DATA_SRC)

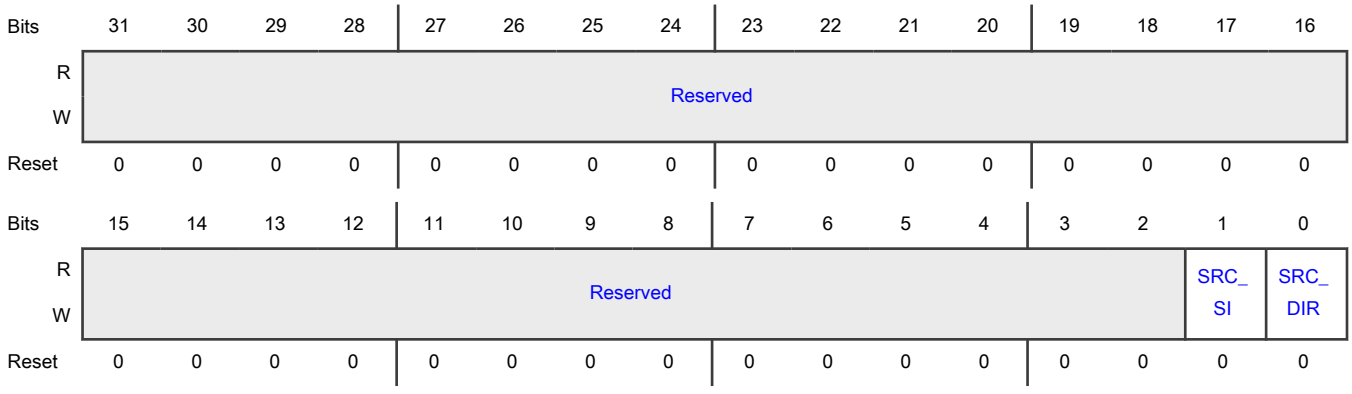
Offset

Register	Offset
DATA_SRC	24h

Function

The Data Source register defines via which interface data must be provided to PUF. Combinations of bits may be 1 at a time.

Diagram



Fields

Field	Function
31-2 —	Reserved
1 SRC_SI	Data provided via the SI interface Defines that the data must be provided via the SI interface 0b - Does not provide data through SI interface. If SRC_SI is set to 0, you can set SRC_DIR to 1 to provide the data 1b - Provides data through SI interface
0 SRC_DIR	Data provided via the DIR register Defines that the data must be provided via the DIR register 0b - Does not provide data through DIR register. If SRC_DIR is set to 0, you can set SRC_SI to 1 to provide the data 1b - Provides data through DIR register

14.4.1.11 Data Input (DIR)

Offset

Register	Offset
DIR	A0h

Function

The Data Input register is used to transfer data to PUF. It can only be written when input data is requested (indicated by the SR[DI_REQUEST]). When written at any other moment, or when read, an APB slave error is generated. Depending on the way the system is configured and connected, this may generate an exception.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	DI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	DI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 DI	Input data Transfers the input data to PUF

14.4.1.12 Data Output (DOR)

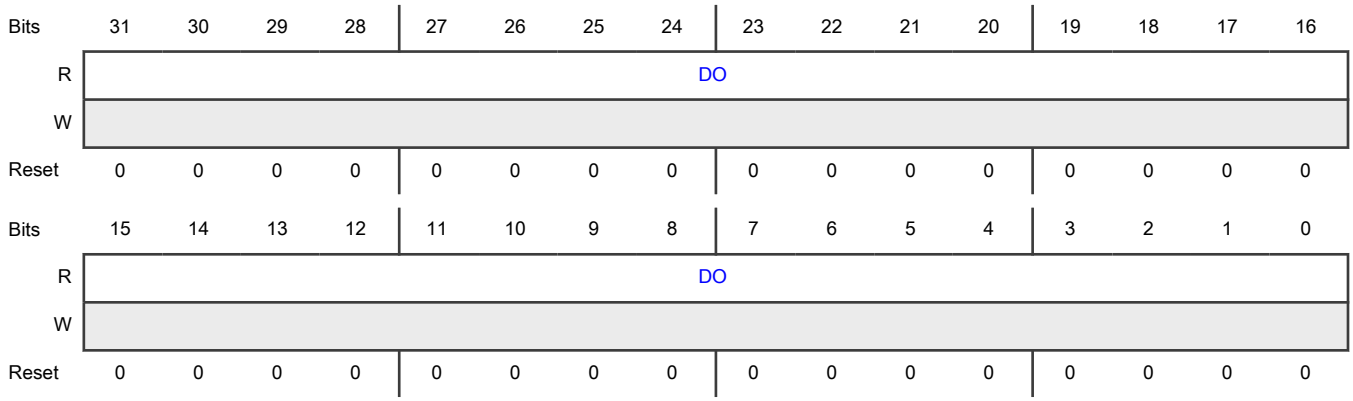
Offset

Register	Offset
DOR	A8h

Function

The Data Output register is used to transfer data from PUF. It can only be read when input data is available (indicated by the SR[DO_REQUEST]). When read at any other moment, or when written, an APB slave error is generated. Depending on the way the system is configured and connected, this may generate an exception.

Diagram



Fields

Field	Function
31-0 DO	Output data Transfers the output data from PUF

14.4.1.13 Miscellaneous (MISC)

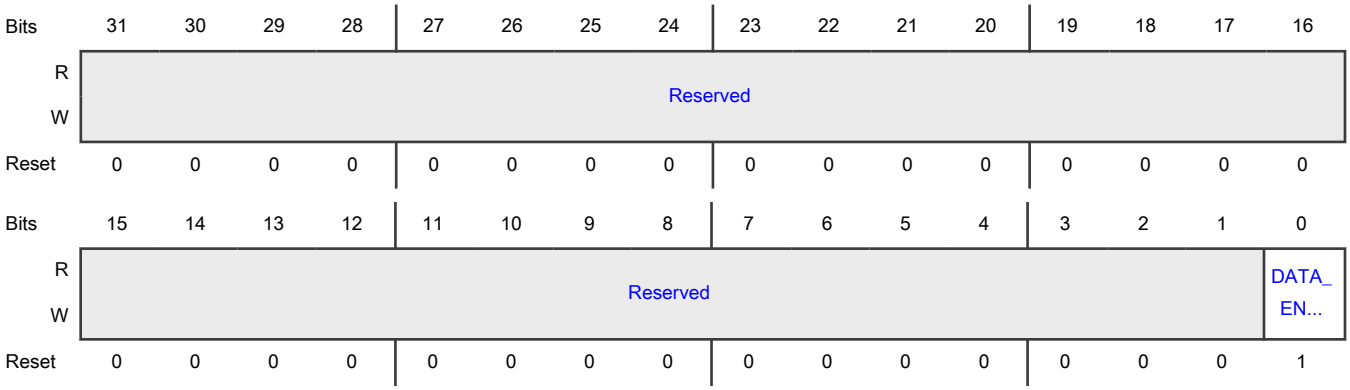
Offset

Register	Offset
MISC	C0h

Function

The Miscellaneous register defines miscellaneous settings for PUF. These usually need to be configured only once after power up or reset.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 DATA_ENDIANNESS	Defines the endianness of data in DIR and DOR: 0b - Little endian 1b - Big endian (default)

14.4.1.14 Interface Status (IF_SR)

Offset

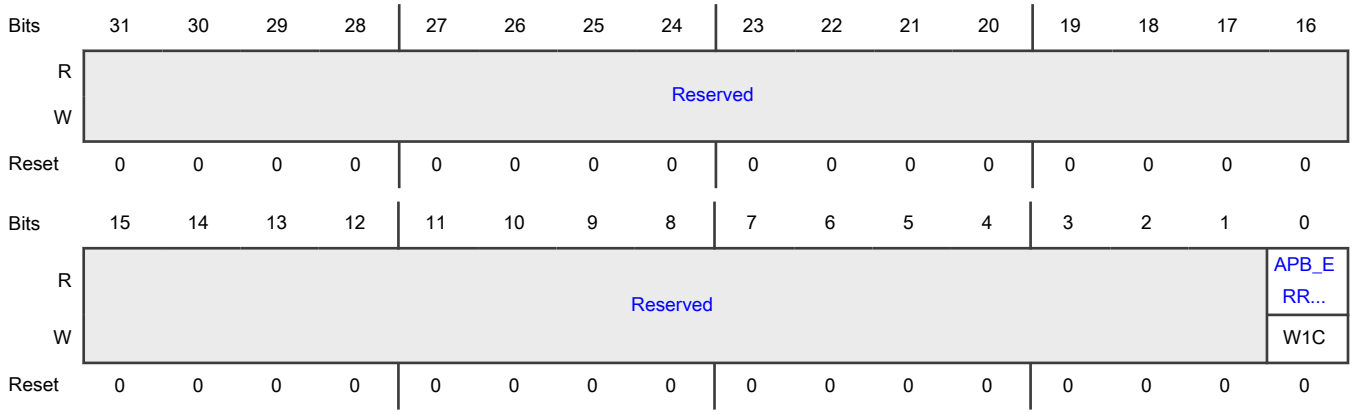
Register	Offset
IF_SR	D0h

Function

The status of the interface can be monitored with the Interface Status register.

Writing 1 to a bit, clears the corresponding bit.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 APB_ERROR	APB error Indicates that an APB error has occurred 0b - No APB error has occurred 1b - Indicates an APB error has occurred

14.4.1.15 PUF Score (PSR)

Offset

Register	Offset
PSR	DCh

Function

The PUF Score register shows the value of the PUF Score that was obtained during the last Test PUF, Enroll, Reconstruct or Start operation.

To determine the PUF quality, the controller performs statistical analysis on various aspects of the PUF, like randomness and noise. This is done during the Test PUF, Enroll, Start and Reconstruct operations, and results in a PUF score with a value from 0x0 to 0x8. A lower PUF score indicates a better PUF quality.

The PUF score can be read from the QK_PSR register after the operation successfully completes. If the operation is not successful (qk_error = 1), then the PUF score is not presented, and the register contains 0xF instead.

The PUF scores returned after Enroll (or Test PUF) give an indication of the PUF quality, while the scores after Start (or Reconstruct) provide an indication of the noise level of the PUF. Therefore the scores can be different.

Possible causes of a high PUF score (low PUF quality or high noise level) during Test PUF, Enroll, Start and Reconstruct operations are, for example:

- Extreme operating conditions

- The power up curve for the SRAM PUF was not according to specification
- Overwriting of SRAM PUF startup values by sources other than the controller, e.g. test logic

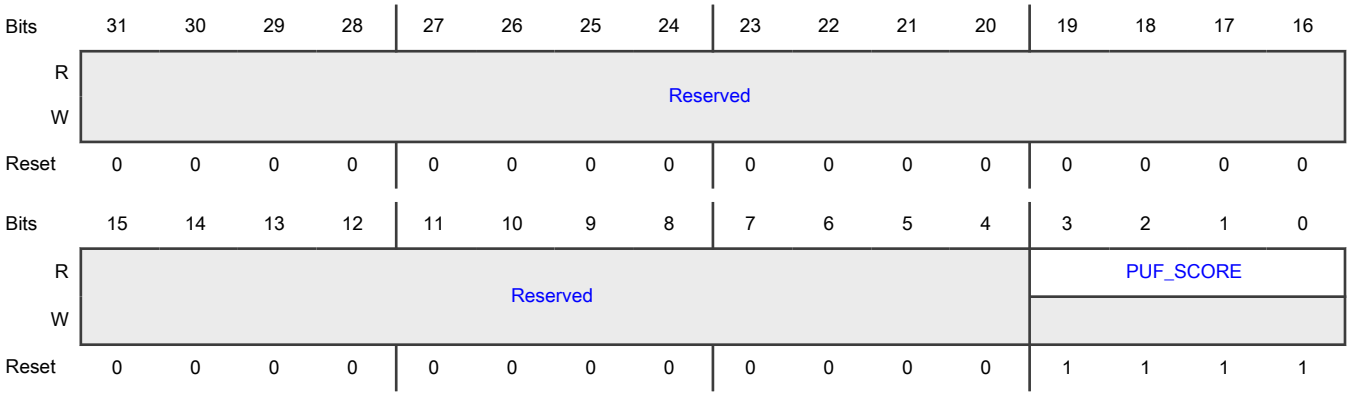
High or increasing PUF scores during Start or Reconstruct operations can indicate that, for example:

- The operating conditions have changed, either suddenly or gradually
- One or more system components are (gradually) degrading, e.g. power supply

When a high PUF score is unexpected, the system level software must decide if actions are required, and what they must be.

When a high PUF score is unexpected, the system level software must decide if actions are required, and what they must be.

Diagram



Fields

Field	Function
31-4 —	Reserved
3-0 PUF_SCORE	Provides the PUF score obtained during the last Test PUF, Enroll or Start operation. 0x0 .. 0x7 - The PUF quality is within the normal range 0x8 - The PUF quality is outside the normal range. Although controller is fully operational, investigation is recommended 0x9 .. 0xE - Reserved 0xF - The PUF quality is too low; Enroll cannot take place

14.4.1.16 Hardware Restrict User Context 0 (HW_RUC0)

Offset

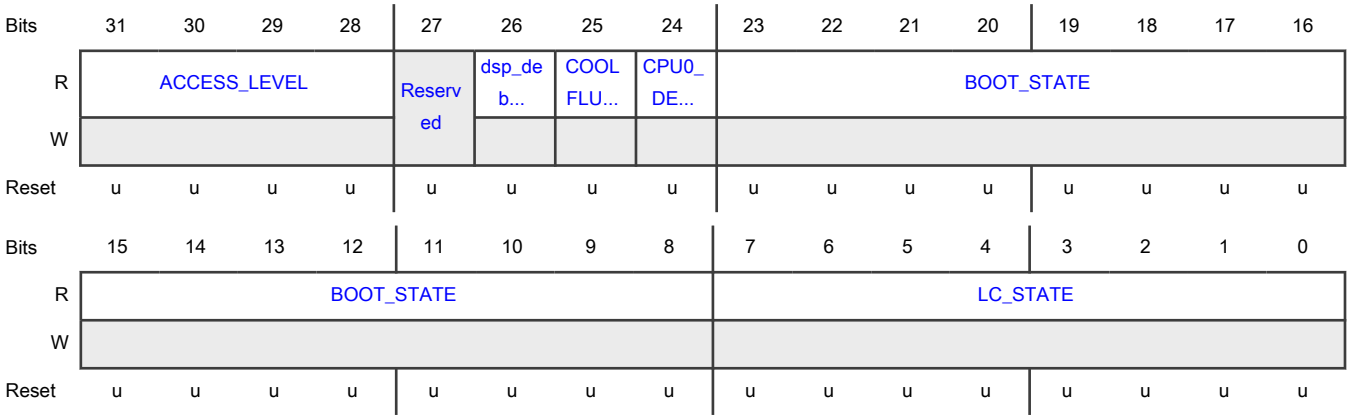
Register	Offset
HW_RUC0	E0h

Function

The Hardware Restrict User Context 0 register shows the values of the restrict_user_context_0 signal on the pin interface.

Refer the “Context Word 2" table in the "Context data" section of "PUF Key management" chapter for more details.

Diagram



Fields

Field	Function
31-28 ACCESS_LEVEL	Restrict the key access based on TrustZone security level 0000 - Key is usable even when access level protection feature is not enabled 0001 - Key is usable at non-secure user and above levels 001x - key is usable at non-secure privilege level and above 01xx - Key is usable at secure user level and above 1xxx - Key is usable at secure privilege level only
27 —	Reserved
26 dsp_debug	DSP debug status. If set then an external debugger has connected to the device at least once after power on reset has occurred. 0b - No external debugger is connected to the device at least once after power on reset has occurred 1b - Indicates an external debugger is connected to the device at least once after power on reset has occurred
25 COOLFLUX_DEBUG	Disable key access when debugger is attached to COOLFLUX after power-up This bit must be 0.
24 CPU0_DEBUG	Disable key access when debugger is attached to CPU0 after power-up Set this bit to 1 during key code creation (wrap step). When set the key cannot be unwrapped if a debug session is established with Cortex-M33 on the device. The debug state of the device is reset only on PoR/BoR/SWR_reset reset.

Table continues on the next page...

Table continued from the previous page...

Field	Function
23-8 BOOT_STATE	Temporal boot state The boot-up process of the device can be broken in to 16 stages which tracked by monotonic counter register in SYSCON block. Each stage is represented by the bit position. Set the bitfield to inverted value of the encoded boot stages state. Boot state is encoded as below: • 0x00 - Stage 0: Early boot stage in ROM • 0x01 - Stage 1: NXP signed firmware stage (used for provisioning) • 0x03 - Stage 2: OEM boot loader or PRoT • 0x07 … 0xFF - Stages defined by OEM boot and security policy For example to restrict a key to "Stage 1" and below then set the value of this field to 0xFE. The key code can't be unwrapped in stage 2 and beyond.
7-0 LC_STATE	Life cycle state based restrictions Set the bitfield to inverted value of the encoded life cycle state value to restrict the key usage to specific life cycle state. Life cycle state is encoded as below. To restrict key usage to multiple states this field should be set to inverted ORed result of encoded value. For example to restrict a key to OEM Develop, OEM Develop 2, OEM In-field and OEM Locked states then set the values of this field to 0011_0000. The key code can't be unwrapped in states beyond OEM In-field locked state. 0000_0011b - OEM Develop 0000_0111b - OEM Develop 2 0000_1111b - OEM In-field 0001_1111b - OEM Field return 0011_1111b - NXP Field Return/Failure Analysis 1100_1111b - In-field Locked 1111_1111b - Bricked

14.4.1.17 Hardware Restrict User Context 1 (HW_RUC1)

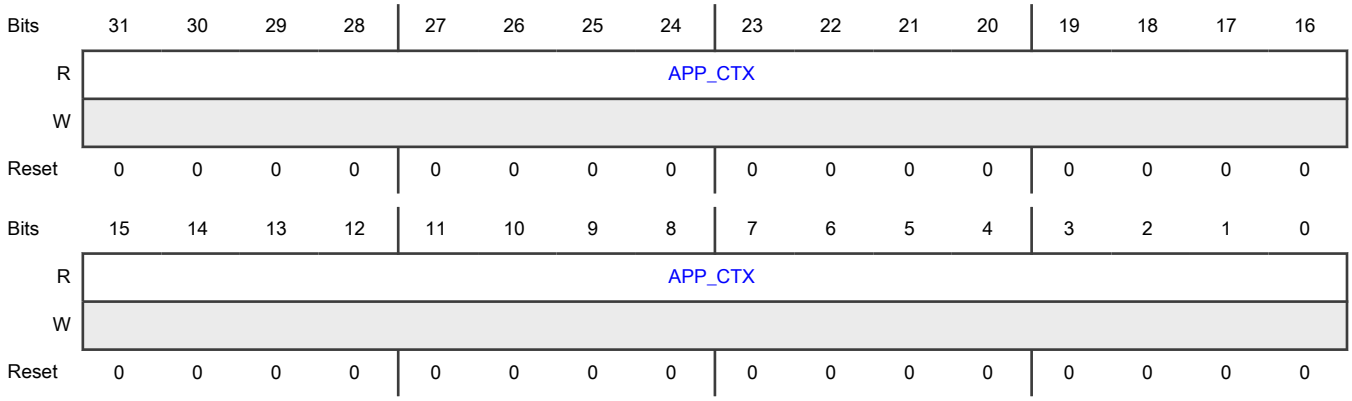
Offset

Register	Offset
HW_RUC1	E4h

Function

The Hardware Restrict User Context 1 register shows the values of the restrict_user_context_1 signal on the pin interface.

Diagram



Fields

Field	Function
31-0 APP_CTX	Application customizable context This context word can be sub-divided to implement custom key policy on top of the mechanism provided by context 0 to 3 word. The content of this word is influenced by the APP_CTX_MASK register. For example, Platform Root of Trust (PRoT) module running at secure-privilege level can change the APP_CTX_MASK register when application task switch happens. Each application task can be allocated each a bit in APP_CTX_MASK. Task 0 mask will be 0xFFFFFFFE Task 1 mask will be 0xFFFFFFFD Task 2 mask will be 0xFFFFFFFB … Task 31 mask will be 0x7FFFFFFF Then keys can be isolated per each application. Task 0 keys should have this field set to 0x0000001 Task 1 keys should have this field set to 0x0000002 Task 2 keys should have this field set to 0x0000004 … Task 31 keys should have this field set to 0x8000000 Similarly another example would be to use APP_CTX_MASK to isolate keys by usage algorithms.

14.4.1.18 Hardware Information (HW_INFO)

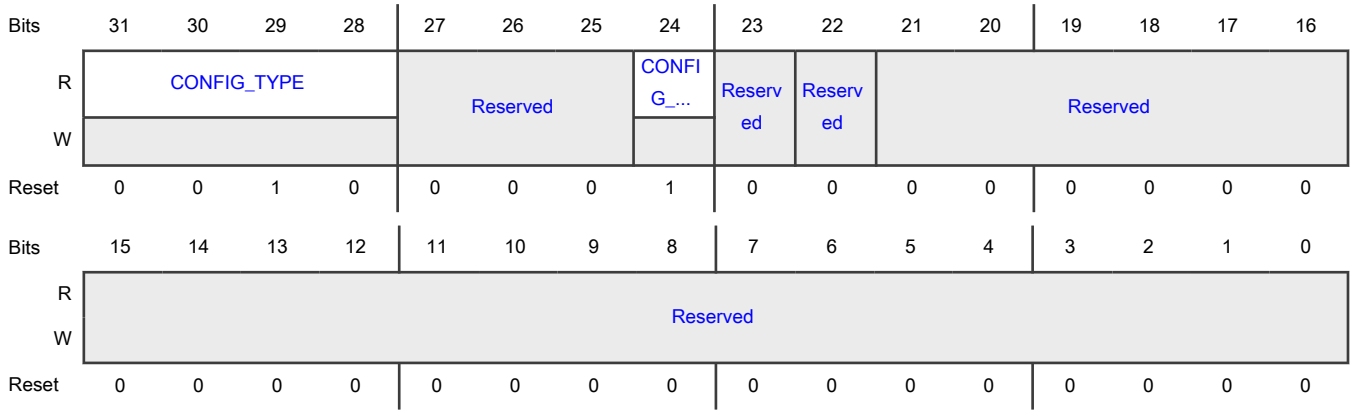
Offset

Register	Offset
HW_INFO	F4h

Function

The Hardware Information register indicates which configuration of PUF is used for this module and which options it includes.

Diagram



Fields

Field	Function
31-28 CONFIG_TYPE	PUF configuration 0001b - Indicates that PUF configuration is Safe. 0010b - Indicates that PUF configuration is Plus.
27-25 —	Reserved
24 CONFIG_WRAP	Wrap configuration 0b - Indicates that Wrap is not included 1b - Indicates that Wrap is included
23 —	Reserved 0b - Plus 1b - Safe
22 —	Reserved
21-0 —	Reserved

14.4.1.19 Hardware Identifier (HW_ID)

Offset

Register	Offset
HW_ID	F8h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HW_ID															
W																
Reset	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	HW_ID															
W																
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 HW_ID	Provides the hardware identifier. IP level has ID, which has 32 bits to record current hardware identifier

14.4.1.20 Hardware Version (HW_VER)

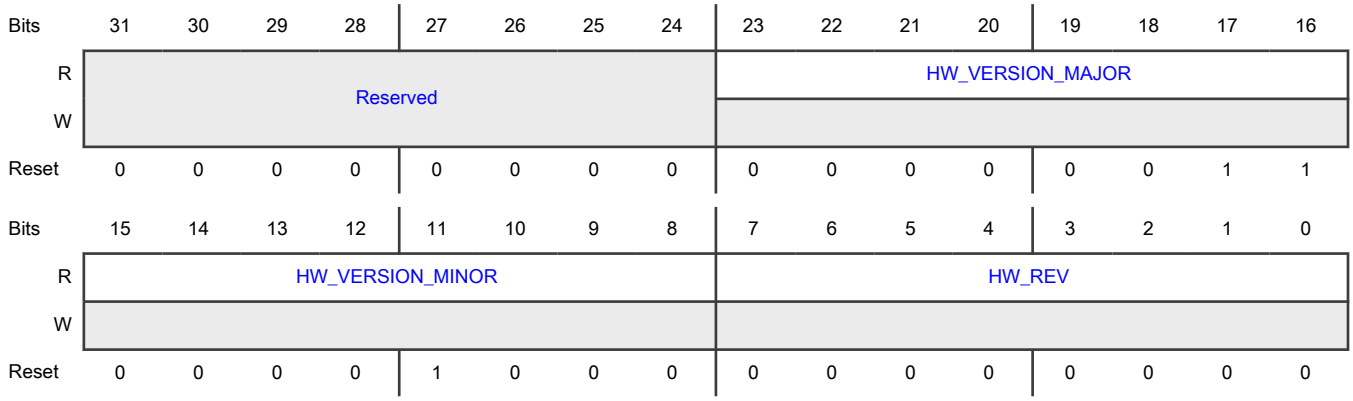
Offset

Register	Offset
HW_VER	FCh

Function

The Hardware Version register provides the version of the module.

Diagram



Fields

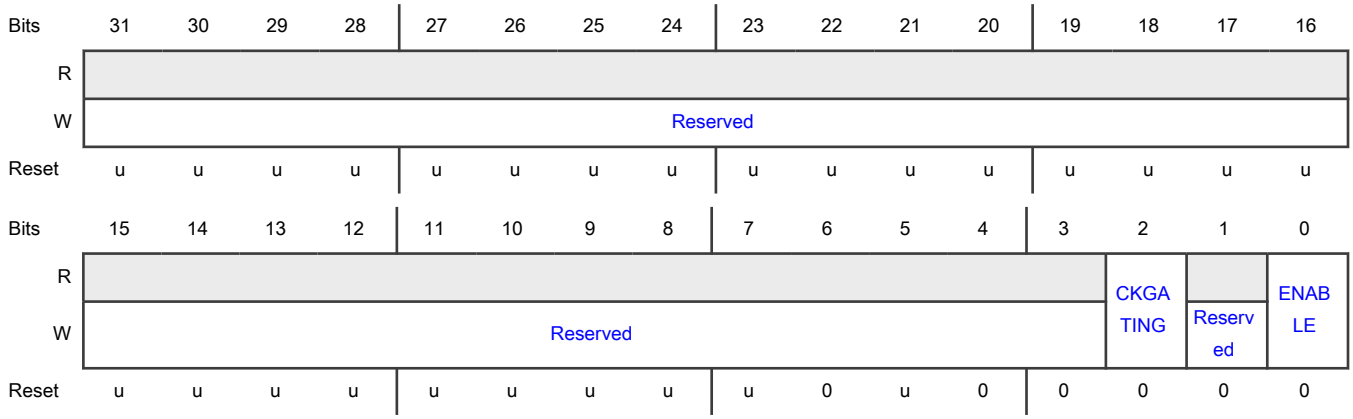
Field	Function
31-24 —	Reserved
23-16 HW_VERSION_MAJOR	Provides the hardware version, major part
15-8 HW_VERSION_MINOR	Provides the hardware version, minor part
7-0 HW_REV	Provides the hardware version, patch part

14.4.1.21 SRAM Configuration (SRAM_CFG)

Offset

Register	Offset
SRAM_CFG	300h

Diagram



Fields

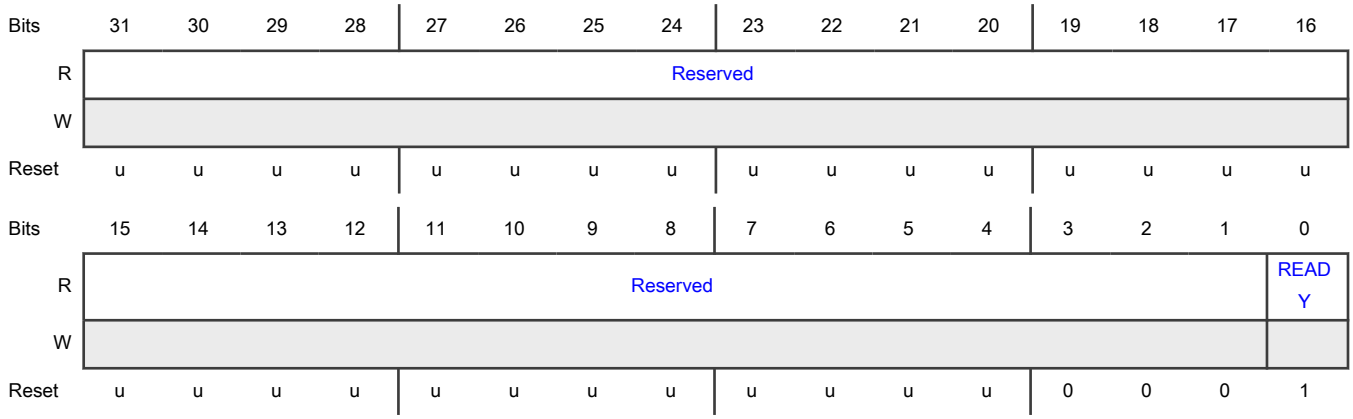
Field	Function
31-3 —	Reserved
2 CKGATING	PUF SRAM Clock Gating control 0b - Disable clock gating function. Clock will be enabled 1b - Enable clock gating function. Clock will be disabled
1 —	Reserved
0 ENABLE	PUF SRAM Controller activation 0b - Disable SRAM controller activation function 1b - Enable SRAM controller activation function

14.4.1.22 Status (SRAM_STATUS)

Offset

Register	Offset
SRAM_STATUS	304h

Diagram



Fields

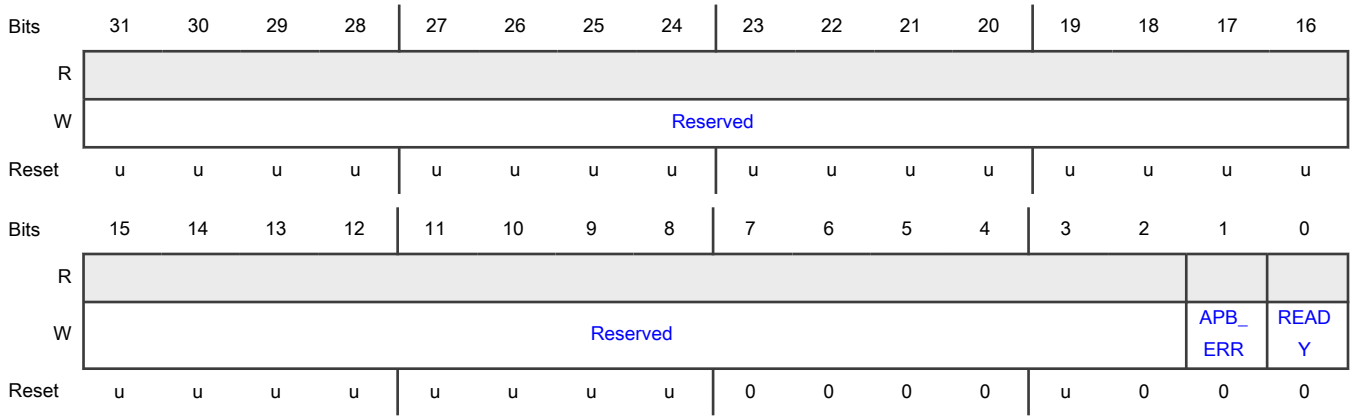
Field	Function
31-1 —	Reserved
0 READY	PUF SRAM Controller State 0 - Not ready. PUF cannot access PUF SRAM. 1 - Ready. PUF can access PUF SRAM.

14.4.1.23 Interrupt Enable Clear (SRAM_INT_CLR_ENABLE)

Offset

Register	Offset
SRAM_INT_CLR_ENABLE	3D8h

Diagram



Fields

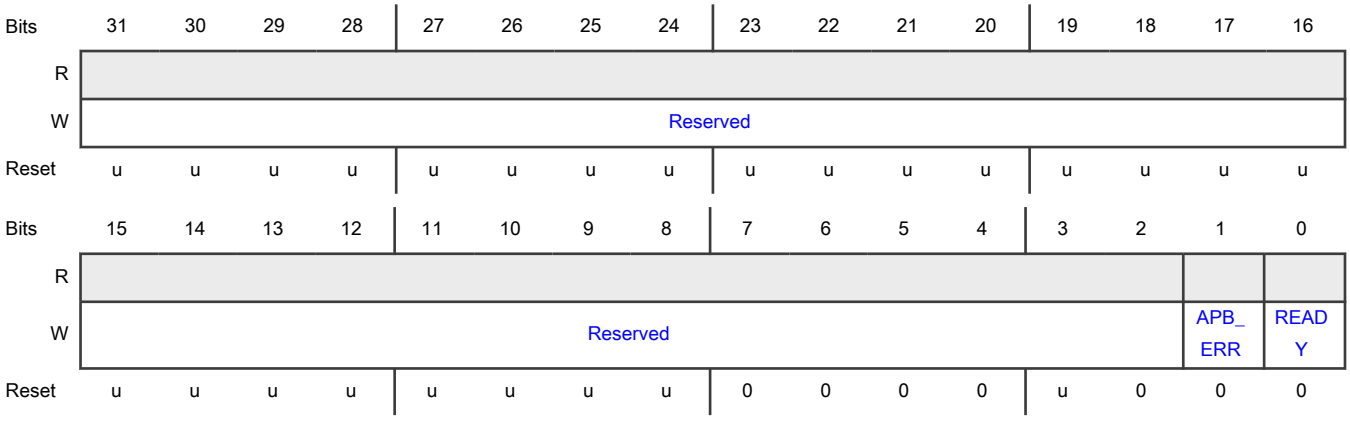
Field	Function
31-2 —	Reserved
1 APB_ERR	APB_ERR Interrupt Enable clear 0 - No effect. 1 - Clears the APB_ERR bit field in register INT_ENABLE. Automatically reset by the Hardware.
0 READY	READY Interrupt Enable clear 0 - No effect. 1 - Clears the READY bit field in register INT_ENABLE. Automatically reset by the Hardware.

14.4.1.24 Interrupt Enable Set (SRAM_INT_SET_ENABLE)

Offset

Register	Offset
SRAM_INT_SET_ENABLE	3DCh

Diagram



Fields

Field	Function
31-2 —	Reserved
1	APB_ERR Interrupt Enable set

Table continues on the next page...

Table continued from the previous page...

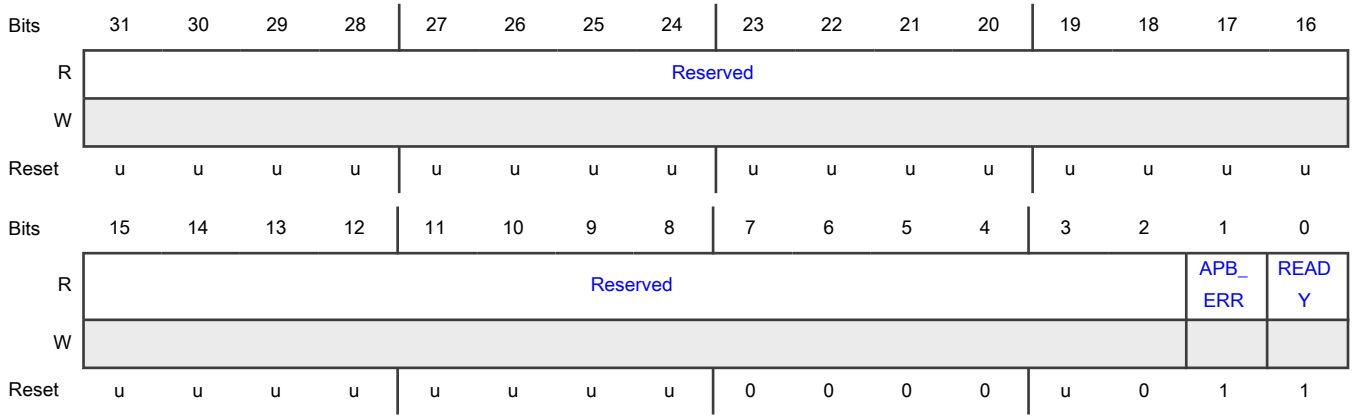
Field	Function
APB_ERR	0 - No effect. 1 - Sets the APB_ERR bit field in register INT_ENABLE. Automatically reset by the Hardware.
0 READY	READY Interrupt Enable set 0 - No effect. 1 - Sets the READY bit field in register INT_ENABLE. Automatically reset by the Hardware.

14.4.1.25 Interrupt Status (SRAM_INT_STATUS)

Offset

Register	Offset
SRAM_INT_STATUS	3E0h

Diagram



Fields

Field	Function
31-2 —	Reserved
1 APB_ERR	APB_ERR Interrupt Status Set when an access to registers has been denied in application of the access security rules determined by pprot, pprot_mask and pprot_match. 0 - Not pending. 1 - Pending.

Table continues on the next page...

Table continued from the previous page...

Field	Function
0 READY	READY Interrupt Status Set at the end of the SRAM initialization phase (whatever the value of the READY bit in INT_ENABLE register),indicating that can PUF (Quiddykey) can use the SRAM. An interrupt is generated only when both this bit and the READY bit in INT_ENABLE register are set. 0 Not pending. 1 Pending.

14.4.1.26 Interrupt Enable (SRAM_INT_ENABLE)

Offset

Register	Offset
SRAM_INT_ENABLE	3E4h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved															
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved														SRAM_AP...	READY
W																
Reset	u	u	u	u	u	u	u	u	0	0	0	0	u	0	0	0

Fields

Field	Function
31-2 —	Reserved
1 SRAM_APB_ERR	APB_ERR Interrupt Enable 0b - Disabled 1b - Enabled
0 READY	READY Interrupt Enable 0b - Disabled 1b - Enabled

14.4.1.27 Interrupt Status Clear (SRAM_INT_CLR_STATUS)

Offset

Register	Offset
SRAM_INT_CLR_STATUS	3E8h

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	Reserved															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	Reserved														APB_ERR	READY
Reset	u	u	u	u	u	u	u	u	0	0	0	0	u	0	0	0

Fields

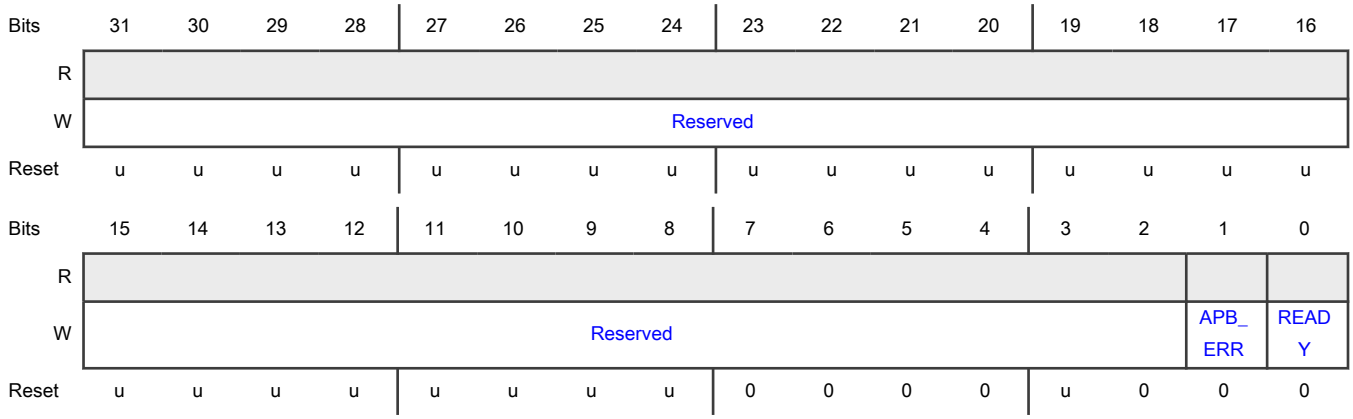
Field	Function
31-2 —	Reserved
1 APB_ERR	APB_ERR Interrupt Status Clear 0b - No effect 1b - Clears the APB_ERR bit field in register INT_STATUS. Automatically reset by the Hardware
0 READY	READY Interrupt Status clear 0 - No effect. 1 - Clears the READY bit field in register INT_STATUS. Automatically reset by the Hardware.

14.4.1.28 Interrupt Status set (SRAM_INT_SET_STATUS)

Offset

Register	Offset
SRAM_INT_SET_STATUS	3ECh

Diagram



Fields

Field	Function
31-2 —	Reserved
1 APB_ERR	APB_ERR Interrupt Status Set 0b - No effect 1b - Clears the APB_ERR bit field in register INT_STATUS. Automatically reset by the Hardware
0 READY	READY Interrupt Status set 0 - No effect. 1 - Sets the READY bit field in register INT_STATUS. Automatically reset by the Hardware.

14.5 Glossary

Table 329. Defnitions, acronyms and abbreviations

AC	Activation Code
AMBA	Advanced Microcontroller Bus Architecture
APB	(AMBA) Advanced Peripheral Bus
BIST	Built-In Self Test
DMA	Direct Memory Access
KC	Key Code
KO	Key Output
NVM	Non-Volatile Memory

Table continues on the next page...

Table 329. Defnitions, acronyms and abbreviations (continued)

PUF	Physical Unclonable Function
SD	Startup Data
SRAM	Static Random Access Memory

1. References

a. AMBA APB Protocol, Version 2.0, Specifcation, ARM IHI 0024C (ID041610) Issue C, 13 April 2010

Chapter 15
PUF Key Context Management

15.1 Chip-specific PUF_CTRL information

Table 330. Reference links to related information[1]

Topic	Related module	Reference
Full description	PUF_CTRL	PUF_CTRL
System memory map		See the section "System memory map"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

15.1.1 Module instances

This device has one instance of the PUF_CTRL module with four aliased base addresses.

15.1.2 Security considerations

The PUF_CTRL module implements Register Access Protection for the PUF module that can be used to lock the PUF to a particular access level. This is intended to be used as a temporary lock that keeps one master from modifying registers while another master is in the middle of configurating a PUF operation.

The PUF_CTRL module is instantiated to use four module slots—PUF_CTRL, PUF_CTRL_alias1, PUF_CTRL_alias2, and PUF_CTRL_alias3. At the Secure AHB controller, each PUF_CTRL slot can be configured for a different secure/non-secure and privileged/non-privileged access. This allows the PUF_CTRL module to be used by any access level using the appropriate slot, while the Secure AHB controller's MISC_CTRL_REG[DISABLE_STRICT_MODE] option is configured for strict mode. The PUF_CTRL's access protection feature can then be used to temporarily lock the PUF module to the current master.

15.2 Overview

The keys used for symmetric cryptographic and ECC P-256 asymmetric operation are protected by ELS S50 key management logic. Since ELS S50 only supports ECC P-256 curve for all other curves customer have to use PKC engine to accelerate the cryptographic operations. For keys used with PKC and application key which are provisioned to ELS S50 key store, it is recommended for software to use PUF to store the applications keys securely.

15.2.1 Features

To provide key isolation based on life cycle state, temporal boot state, debug state and Trustzone security level state on this device we provide key management wrapper for PUF which utilizes the key context feature of PUF.

- Key context data structure composed of four 32-bit values
- Software set context value while creating the key code for keys.
- Context tightly coupled to the key code allowing software to implement key policy management using the context data.

15.3 PUF Context Key Management

15.3.1 Context data

Context data is split in following ways to provide flexible key policy management:

Context word 0: The bit fields in this context word are cross checked by the key wrap and unwrap logic inside hardware and any tampering to the fields are detected.

Table 331. Context Word 0

Bits	Field name	Description
31:24	Reserved	Must be 0
23:16	context_type	0x10: Context for Key operationsOther values: Reserved
15:13	Reserved	Must be 0
12:0	key_length	Length of the key in bits; the following lengths are supported:- 64 .. 1024 bits in 64-bit increments- 2048 bits- 3072 bits- 4096 bits

Context word 1: The bit fields in this context word are used by the key wrap and unwrap logic inside hardware and any tampering to the fields are detected.

Table 332. Context Word 1

Bits	Field name	Description
31:16	key_type	0x0000: Generic Other values: Reserved
15:8	key_scope_started	Defines the allowed key destinations, when PUF is in the Started state: • bit 8: Key can be made available via the QK_DOR register • bit 9: Key can be made available to ELS S50 • Others: Reserved
7:0	key_scope_enrolled	Defines the allowed key destinations, when PUF is in the Enrolled state: • bit 0: Key can be made available via the QK_DOR register • bit 1: Key can be made available to ELS S50 • Others: Reserved

Context word 2: The key management wrapper in the chip provides the device context to key wrap and unwrap state machine to enforce key policy.

Bits	Field name	Description
31:28	access_level	Restrict the key access based on TrustZone security level. b'0000 - Key is usable even when access level protection feature is not enabled. b'0001 - Key is usable at non-secure user and above levels b'001x - key is usable at non-secure privilege level and above. b'01xx - Key is usable at secure user level and above b'1xxx - Key is usable at secure privilege level only.
27	Reserved	Must be 0.
26	dsp_debug	Disable key access when debugger is attached to DSP after power-up. Set this bit to 1 during key code creation (wrap step). When set the key cannot be unwrapped if a debug session is established with the DSP core on device. The debug state of the device is reset only on PoR/BoR/SWR_reset reset.
25	coolflux_debug	Must be 0.
24	cpu0_debug	Disable key access when debugger is attached to CPU0 after power-up. Set this bit to 1 during key code creation (wrap step). When set the key cannot be unwrapped if a debug session is established with Cortex-M33 on the device. The debug state of the device is reset only on PoR/BoR/SWR_reset reset.
23:8	Boot_state	Temporal boot state. The boot-up process of the device can be broken in to 16 stages which tracked by monotonic counter register in SYSCON block. Each stage is represented by the bit position. Set the bitfield to inverted value of the encoded boot stages state. Boot state is encoded as below:

Table continues on the next page...

Table continued from the previous page...

Bits	Field name	Description
		• 0x00 - Stage 0: Early boot stage in ROM • 0x01 - Stage 1: NXP signed firmware stage (used for provisioning) • 0x03 - Stage 2: OEM boot loader or PRoT(Platform Root of Trust) • 0x07 … 0xFF - Stages defined by OEM boot & security policy. For example to restrict a key to "Stage 1" and below then set the value of this field to 0xFE. The key code can't be unwrapped in stage 2 and beyond.
7:0	lc_state	Life cycle state based restrictions. Set the bitfield to inverted value of the encoded life cycle state value to restrict the key usage to specific life cycle state. Life cycle state is encoded as below: • 0000_0011 OEM Open • 0000_0111 OEM Secure World • 0000_1111 OEM Closed • 0001_1111 Field Return OEM • 0011_1111 OEM recommission • 0011_1111 Field Return NXP • 1100_1111 OEM Locked (no return) • 1111_1111 Shredded To restrict key usage to multiple states this field should be set to inverted ORed result of encoded value. For example to restrict a key to *OEM Open*, *OEM Secure world* and *OEM Closed* states then set the values of this field to 1111_0000. The key code can't be unwrapped in states beyond *OEM closed* state. For example to restrict a key to *OEM Open*, *OEM Secure world, OEM Closed* and *OEM Locked* states then set the

Table continues on the next page...

Table continued from the previous page...

Bits	Field name	Description
		values of this field to 0011_0000. The key code can't be unwrapped in states beyond *OEM closed* state.

Context word 3:

Table 333. Context Word 3

Bits	Field name	Description
31:0	APP_CTX	Application customizable context. Product designer can sub-divide this context word to implement custom key policy on top of the mechanism provided by context 0 to 3 word. The content of this word is influenced by the APP_CTX_MASK register. For example, Platform Root of Trust (PRoT) module running at secure-privilege level can change the APP_CTX_MASK register when application task switch happens. Each application task can be allocated each a bit in APP_CTX_MASK. • Task 0 mask will be 0xFFFFFFFE • Task 1 mask will be 0xFFFFFFFD • Task 2 mask will be 0xFFFFFFFB • … • Task 31 mask will be 0x7FFFFFFF Then keys can be isolated per each application. • Task 0 keys should have this field set to 0x0000001. • Task 1 keys should have this field set to 0x0000002. • Task 2 keys should have this field set to 0x0000004. • … • Task 31 keys should have this field set to 0x8000000. Similarly another example would be to use APP_CTX_MASK to isolate keys by usage algorithms.

15.3.2 Register Access Protection

To use the register access protection behavior described in this section the peripheral checker for PUF in AHB security control block should be set to non-secure user level (2b'00).

- During register access pprot[1:0] represents the transaction access level:
  - Bit 0: Normal (0) or privileged (1)
  - Bit 1: Secure (0) or non-secure (1)
- With the pprot_mask and pprot_match bits the allowed types of access can be defined. Access to the registers is granted when the following is true:

  (pprot AND pprot_mask) == (pprot_match AND pprot_mask)
  - Where AND is a bitwise logic AND operation.
  - pprot_mask = (SEC_LOCK[3:0] == 4'b0000)? 3'b100 : {1'b1, cfgsecext, 1'b1};
  - Cfgsecext is set to 1 when TrustZone-M is enabled for CM33 pprot_match = {1'b0, ~SEC_LEVEL[1], SEC_LEVEL[0]}

Table 334. Register access protection

SEC_LOCK[3:0]	Uc_sec_mask[3:0]	Description
4b'00_00	4b'1111	PUF register access is not locked to a particular level. No access limits, any master in any mode has access to the PUF registers.
b'11_00	4b'1110	Only normal access (non-secure user level) is granted access to the PUF registers.
b'10_01	4b'1100	Only privileged access(non-secure privilege level) is granted access to the PUF registers.
b'01_10	4b'1000	Only secure user level is granted access to the PUF registers.
b'00_11	4b'0000	Only secure privileged access is granted access to the PUF registers.

15.4 Memory Map and register definition

This section includes the PUF Key Context Management module memory map and detailed descriptions of registers.

15.4.1 PUF Key Context Management register descriptions

15.4.1.1 PUF_KEY_CONTEXT memory map

PUF_CTRL base address: 4002_C000h

Offset	Register	Width (In bits)	Access	Reset value
100h	PUF command blocking configuration (CONFIG)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
104h	Security level lock (SEC_LOCK)	32	RW	0000_0000h
108h	Application defined context mask (APP_CTX_MASK)	32	RW	0000_0000h

15.4.1.2 PUF command blocking configuration (CONFIG)

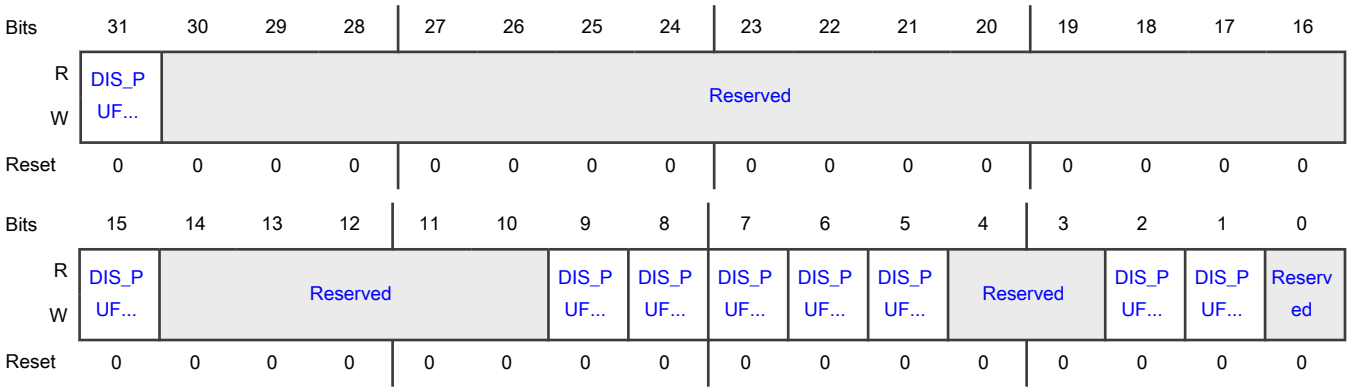
Offset

Register	Offset
CONFIG	100h

Function

Each bit in this register represents a PUF command which can be blocked from usage until the chip is power-cycled (PoR or BoD reset). Each bit can be set once. Once the bit is set it can be reset only on PoR/BoD reset condition.

Diagram



Fields

Field	Function
31 DIS_PUF_TEST	Disable PUF test command 0b - Command enabled 1b - Command disabled
30-16 —	Reserved
15	Disable PUF generate and wrap key command

Table continues on the next page...

Table continued from the previous page...

Field	Function
DIS_PUF_GEN_RANDOM_NUMBER	0b - Command enabled 1b - Command disabled
14-10 —	Reserved
9 DIS_PUF_WRAP_KEY	Disable PUF wrap key command 0b - Command enabled 1b - Command disabled
8 DIS_PUF_GEN_WRAP_KEY	Disable PUF generate and wrap key command 0b - Command enabled 1b - Command disabled
7 DIS_PUF_UNWRAP_KEY	Disable PUF unwrap key command 0b - Command enabled 1b - Command disabled
6 DIS_PUF_GET_KEY	Disable PUF get key command 0b - Command enabled 1b - Command disabled
5 DIS_PUF_STOP	Disable PUF stop command 0b - Command enabled 1b - Command disabled
4-3 —	Reserved
2 DIS_PUF_START	Disable PUF start command 0b - Command enabled 1b - Command disabled
1 DIS_PUF_ENROLL	Disable PUF enroll command 0b - Command enabled 1b - Command disabled
0 —	Reserved

15.4.1.3 Security level lock (SEC_LOCK)

Offset

Register	Offset
SEC_LOCK	104h

Function

Lock the security level of PUF block until key generate, wrap or unwrap operation is completed.

This register can used as semaphore mechanism between security levels tasks to lock the block, since key operations involve multiple register writes and read transactions. Once the lock is set only task at same security level or secure-privilege level task can release the lock by writing 0xAC50 to this register.

To lock PUF access at security level of the task, software should follow below sequence:

- Read the register and check it blank. If lock is set wait until lock gets cleared by other task.
- Write the security level of task by setting the fields as described below.
- Read the register back and check if the security level is same as the level written.
- Issue desired PUF command through register access. All other tasks will be blocked from accessing the PUF registers.
- After the commands complete, release the lock by writing a 0xAC50.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PATTERN												ANTI_POLE_SEC_L...		SEC_LEVEL	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-16 —	Reserved
15-4 PATTERN	Pattern This field must be written as 0xAC5 in order to change the value of SEC_LOCK in bits [3:0].
3-2	Anti-pole of security level

Table continues on the next page...

Table continued from the previous page...

Field	Function
ANTI_POLE_SEC_LEVEL	If this field does not match the inverted value of bits [1:0] then access level could be locked at wrong level. Please refer AHB Secure Controller for more details. 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
1-0 SEC_LEVEL	Security Level This register allows configuring security level for PUF. Please refer AHB Secure Controller for more details. 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master

15.4.1.4 Application defined context mask (APP_CTX_MASK)

Offset

Register	Offset
APP_CTX_MASK	108h

Function

This register is used in conjunction with key context word 3. It is only modifiable by task running at secure-privilege level.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	APP_CTX_MASK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	APP_CTX_MASK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 APP_CTX_MASK	Application defined context If a bit in this register is 1 then the corresponding bit in the context 3 word should be 0.

Chapter 16
Public-key Cryptography Accelerator (PKC)

16.1 Chip-specific PKC information

Table 335. Reference links to related information[1]

Topic	Related module	Reference
Full description	PKC	PKC
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

NOTE

Refer Control PKC RAM Interleave Access (RAM_INTERLEAVE) for details on PKC RAM control.

16.1.1 Module instances

This device has one instance of the PKC module.

16.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

16.1.3 Parity error check

PKC supports parity checking on it's RAM. Any parity errors that are detected are reported through the Error Recording Module (ERM). See the ERM chapter in MCX Nx4x Reference Manual.

16.2 Overview

PKC (Public-Key Crypto Coprocessor) is a security IP hardware, which can perform basic arithmetic and logical operations on multi-precision integers. Its purpose is to provide hardware acceleration to the software when executing public-key cryptography computations. The interfacing is done mainly using special function registers (SFRs) that provide the following functionality:

- Control and configuration SFRs to configure and start a PKC calculation
- Status SFRs to report status information (busy, carry, zero-result, fault detected)
- Layer 0 (L0): parameter set registers to define the calculation mode and operands (via start address and length)
- Layer 1 (L1): micro-coding SFRs to configure and start a PKC calculation via its micro-coding interface
- Layer 2 (L2): universal pointer fetch registers to define the DMA pointer addresses and amount of calculations that are executed in a series

The PKC kernel consists of the arithmetic unit (layer 0 (L0)) and a micro-coding state machine (layer 1 (L1)). The arithmetic unit supports the multiplication of a PKC word (64 bits) with a multi-precision integer (MPI) and addition, subtraction, shift operation and logical functions (AND, OR, XOR) of two multi-precision integers. The micro-coding state machine is used to define more complex

calculation functions, like full-size multiplication or modular multiplication. Via the universal-pointer fetch complete sequences of L0 and L1 operations can be executed.

For any kind of PKC operation the data needs to be first loaded into the PKC RAM. Afterward, software can configure the PKC via the SFR-IF and start the operation. During the execution of an operation PKC automatically fetches data via the RAM and the AHB L2 interfaces. Once the operation is completed the results can be read from the PKC RAM.

16.3 Block diagram

The PKC mainly consists of an interface shell (PKC control) and the PKC kernel. The interface shell connects the PKC kernel to the SFRs and implements the hardware for the universal-pointer fetch.

The PKC is interfaced via its advanced peripheral bus (APB) using special function registers (SFRs).

The PKC implements an AHB master to provide DMA access for the universal pointer unit in layer 2 (L2). This AHB interface requires only read accesses. A second read/write DMA to the PKC accessible RAM (PKC RAM) is required for arithmetic calculation of the PKC kernel itself in layer 0 (L0).

The PKC provides further error and interrupt signals that need to be connected to the relevant system controller. A basic overview of the PKC interfaces and its typical integration into the system is shown in Figure 74.

The PKC kernel consists of the arithmetic unit in L0 and a micro-coding state machine in layer 1 (L1). The arithmetic unit supports the multiplication of a 64 bit PKC word with a multi-precision integer (MPI) and addition, subtraction, shift/rotate operation and logical functions (AND, OR, XOR only) of two multi-precision integers. The micro-coding state machine is used to define more complex calculation functions, like full-size multiplication or modular multiplication. Via the universal-pointer fetch complete sequences of L0 and L1 operations can be executed.

The following is the PKC block diagram:

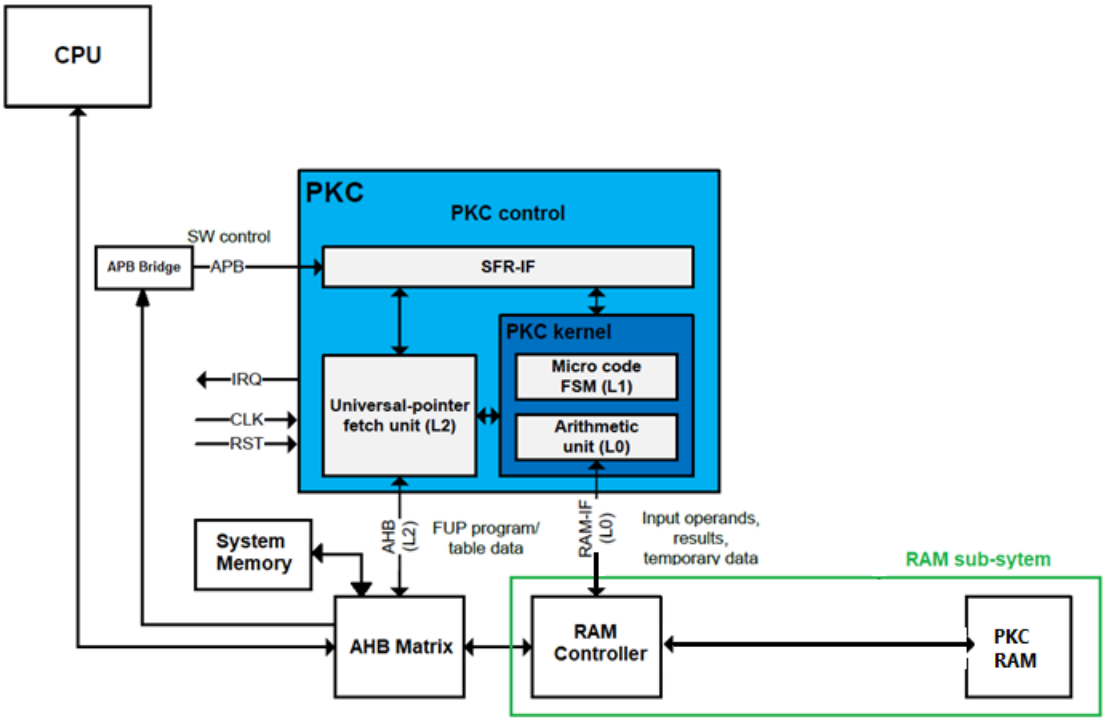


Figure 74. Overview of the PKC interfaces and its typical integration into the system.

16.4 Functional description

A PKC calculation is executed using the following steps, e.g., for a single addition R=Y+Z using the direct programming mode (L0):

1. Release PKC coprocessor from reset (could also be done later but increases the noise source during parameter and operand loading)
2. Load addition operands to PKC accessible RAM
3. Prepare parameter set for PKC addition, e.g. using parameter set 1
    a. Calculation mode: addition (PKC_MODE1)
    b. Start address of Y operand (PKC_XYPTR1[YPTR]
    c. Start address of Z operand (PKC_ZRPTR1[ZPTR])
    d. Start address of result area (PKC_ZRPTR1[RPTR])
    e. Length of operands (PKC_LEN1[LEN])
    f. Start calculation by setting the 'go' bit for the chosen parameter set: PKC_CTRL[GOD1]=1
4. Load addition operands to PKC accessible RAM
5. Poll for end of PKC calculation using the combined 'busy' flag: PKC_STATUS[ACTIV]
6. Read result of addition from RAM at address PKC_ZRPTR[RPTR]
7. Check PKC status register for carry (PKC_STATUS[CARRY]), and zero (PKC_STATUS[ZERO])

16.4.1 PKC control parameter sets

The PKC Control implements two parameter sets to define the PKC calculation. The parameter set contains all required information to define a single calculation: calculation mode, location of all inputs, length of inputs. Two calculations can be defined using the two dedicated parameter sets to support fast switching between two calculations.

A parameter set contains the following fields:

- PKC_MODEi[MODE] (8 bits) - calculation mode or start address for micro code (MC)
- PKC_XYPTRi[XPTR] (16 bits) - start address for X operand
- PKC_XYPTRi[YPTR] (16 bits) - start address for Y operand
- PKC_ZRPTRi[ZPTR] (16 bits) - start address for Z operand or constant operand for CONST or shift and rotate calculations (lower 8 bits only)
- PKC_ZRPTRi[RPTR] (16 bits) - start address for calculation result R
- PKC_LENi[LEN] (16 bits) - length of Y operand, defines also length of Z and R (may differ from length defined in SFR but depends on the chosen calculation mode, e.g. for op-code 0x01 (multiply-accumulate) Z is one PKC word bigger and R is two PKC words bigger than Y)
- PKC_LENi[MCLEN] (16 bits) - loop counter for MC execution, e.g. for MC pattern (MC sequence) for plain multiplication

The following is an example of plain multiplication (fixed MC pattern 0x13, parameter set 1):

```
@ RAM address 0x2004_0000
X: 0xE9794B15AAF151CF991D686D31D95C20263E5D725DE2CB9B9C51AEFBC167E8C2
@ RAM address 0x2004_0200
Y: 0x5A4D00195A39B947F732CB60CE97B0CA83D54C982ED5EC06ADB0785842550D31
@ address 0x2004_0400
R: 0x525ADDF852BEDFD7E81B1E7BCAF241DBC9582598B710677515542B839D462C87_
     531FC208AF12E6D76066B522F209506E228A834A83067B6F632C3B34981F6722

PKC_MODE1.MODE = 0x013 --> start address of Plain multiplication
PKC_XYPTR1.XPTR = 0x0000 --> start address of X operand in RAM
PKC_XYPTR1.YPTR = 0x0200 --> start address of Y operand in RAM
PKC_ZRPTR1.ZPTR --> not used/required
PKC_ZRPTR1.RPTR = 0x0400 --> start address of result R in RAM
PKC_LEN1.LEN = 0x0020 --> length of Y operand (32 bytes = 4 PKC-words of 64 bits)
```

```
PKC_LEN1.MCLEN = 0x0020 --> length of X operand (32 bytes = 4 PKC-words of 64 bits)
PKC_CTRL.GF2CONV = 0 --> execute in non-GF(2)

Calculation is started with PKC_CTRL.GOM1 --> MC pattern calculation with parameter set 1
```

The granularity of the pointers is one byte. The implementation is such that the pointers used by the CPU can be reused.

Example:

- CPU address of data word in PKC RAM: 0x2204_12EF
- PKC pointer value, e.g., for PKC_XYPTRi[XPTR]: 0x12EF

As the minimum operand size of the PKC coprocessor is 64 bits the least significant 3 bits are ignored and not forwarded to the PKC kernel. The same holds for the most significant bits that are not required due to the limitation in PKC RAM. Even though the bits are redundant for the PKC all bits are stored within the PKC SFRs.

All operands in RAM and the pointers and length registers need to be aligned to the word size of the PKC kernel.

Table 336. Alignment for operands in RAM, pointers and length registers

PKC_CTRL[REDMUL]	Multiplier size / PKC word	Security alarm in case PKC_LENi[LEN] (or MCLEN if used) is ...	Operand and register (pointer/length) alignment
00b	default (64 bit)	< 0x08	default value referring to native multiplier size indicated via PKC_VERSION[MULSIZE] = 10b (64 bit)
01b	-	-	Reserved (triggers an error)
10b	64 bit	< 0x08	64 bit, least significant 3 bits [2:0] of pointer/length registers are ignored, for shift/rotate operation only PKC_ZPTRi[5:0] is evaluated, the upper bits are ignored (no error is triggered if set)
11b	-	-	Reserved (triggers an error)

The security alarm for PKC_LENi[MCLEN] is only triggered in a L1 operation when DecrTBNZ is used with a zero MCLEN value.

16.4.1.1 Layer 0: Operating PKC through parameter set interface (L0)

With L0 the basic operations of the PKC coprocessor can be started, e.g. simple addition, shift and logical operations or multiplication of a single PKC word with a long integer. A single calculation is started by setting the write only PKC_CTRL[GOD1] bit for parameter set 1 and PKC_CTRL[GOD1] for parameter set 2. The PKC_STATUS[GOANY] flag is set automatically when a calculation start has been triggered and cleared by HW as soon as the PKC kernel has acknowledged the start of the calculation. The end of the calculation is indicated by PKC_STATUS[ACTIV]=0 assuming there is no other pending calculation.

See PKC arithmetic unit (L0) for details of Layer 0 calculation modes supported by the PKC arithmetic unit.

16.4.1.2 Layer 1: Operating PKC through micro code interface (L1)

PKC micro codes can be used to execute complex calculations (e.g. modular multiplication or addition) that consist of a combination of single PKC Control and L0 calculations. Beside the L0 calculations, loops, conditional and unconditional jumps and simple pointer adaptations (increment/decrement) are supported. The operand pointers are defined using one of the two available parameter sets. The micro code can be started either using parameter set 1 or parameter set 2 via the dedicated control flags PKC_CTRL[GOM1] and PKC_CTRL[GOM2].

Execution of a PKC micro code is done by defining the 8 bit start address (via the PKC_MODE1 SFR) that is used as a pointer within the MC state machine. The PKC coprocessor kernel provides hard-coded calculation modes to implement commonly used

operations, for example, a full size multiplication, modular addition, subtraction, reduction and multiplication in in Z_N (which is normal computation modulo N) and in Galois Field GF(2) (which are computations in binary field).

User programmable micro code is not supported in this version of the IP as the size of the flexible MC table is zero. Attempts to read or write the PKC_MCDATA register will trigger a security alarm.

The calculation itself is started using a dedicated write-only control flag PKC_CTRL[GOM1] or PKC_CTRL[GOM2]. The PKC_STATUS[GOANY] flag is set when GOM1 or GOM2is set and cleared by HW after the micro execution has been started. Setting a GOM bit while GOANY is set will trigger a security alarm. The currently used parameter set can be updated after associated STATUS[LOCKED] bit is cleared. The parameter set is latched, such that pointer modifications (increment/decrement) during the MC execution do not overwrite the initial parameter set.

The end of a micro code execution is indicated by ACTIV=0.

In addition to the parameter-set registers, two extra internal pointer registers S and T are available during micro-code execution. The two pointer registers S and T can be used, e.g., as temporary storage (i.e., loaded with value from other pointer register via a MC instruction) and are reset to zero with the start of each L1 calculation triggered by setting either GOM1 or GOM2.

The parameter set registers including the start address and loop counter are latched with the start of a MC operation and can be overwritten once the MC start has been acknowledged.

For detailed information on the Micro Code Layer see Micro code (MC) interface (L1).

16.4.1.3 Layer 2: Operating PKC through universal pointer fetch unit (L2)

The PKC Control is able to perform a predefined number of calculations with different parameters without interaction of the CPU. To achieve this, two universal pointers (PKC_UPTRand PKC_UPTRT) and a counter register (PKC_ULEN) have been implemented. By this a flow of PKC calculations can be defined and executed in a series to implement complex cryptographic operations, e.g. EC point arithmetic. Using the universal pointer fetch single PKC calculation (L0) and PKC micro codes (L1) can be executed in any order.

The universal pointer, a 32 bit address pointer, is used to fetch the parameters for the single PKC operation. The parameters can be located in any embedded memory (ROM, RAM, Flash) that is accessible via the L2 DMA. All memory access restrictions that are valid for the L2 DMA are also true for the universal pointer fetch. SFRs cannot be accessed using the PKC universal pointer fetch unit. In case of an invalid L2 DMA access (e.g. access to an invalid memory area) triggered by the PKC L2 calculation and detected by the system a security alarm is triggered and PKC_ACCESS_ERR[AHB] error flag is set.

The pipe operation is started by setting the PKC_CTRL[GOU] control bit. The end of the pipe calculation is indicated via the PKC_STAT[ACTIV] status flag. Between start and end of the pipe operation no further CPU interaction is required. The operand length is taken from parameter set 1 or parameter set 2 depending on CALCparam0[6] and needs to be defined upfront. The length SFRs cannot be overwritten while GOU is set. A violation triggers a security alarm. The loop counter (MCLEN) for the MC execution needs to be initialized via SFR write before starting the universal pointer fetch operation as well. The loop counter must not be overwritten during the complete universal pointer fetch calculation.

A single PKC kernel calculation (L0 or L1) triggered by universal pointer fetch can be repeated up to 15 times using the parameter CALCparam0[3:0].

PKC_UPTR, PKC_UPTRT and PKC_ULEN must not be updated during the L2 calculation (while GOANY=1). Updates of these SFRs during a calculation are not covered by HW but must be ensured by the SW operating the PKC.

The PKC pipe operation is based on the two universal pointer registers PKC_UPTR, PKC_UPTRT and PKC_ULEN that defines the number of calculations in a row. Starting a universal pointer fetch with PKC_ULEN equal 0 will trigger a security alarm.

PKC_UPTR points at the start address of the PKC Universal Pointer fetch program (short FUP program). The FUP program parameters for a single calculation have to be stored and are fetched in the following order:

- CALCparam0, CALCparam1, XPTRind, YPTRind, ZPTRind, RPTRind (= data entry) or
- CALCparam0, CALCparam1, CRC32[0], CRC32[1], CRC32[2], CRC32[3] (= CRC entry (when CRC mode is enabled))

CALCparam0 and CALCparam1 define the calculation mode and potential universal pointer control options for future use. The bit fields of CALCparam0 are described in Table 337

Table 337. CALCparam0 bit fields.

Bits	Description
3:0	Repeat counter that defines how often the operation/pattern is repeated (redundant for CRC entry).
4	0 → Indicates a data entry. The 4 bytes following CALCparam1 define the table indexes for the operand pointer (X, Y, Z, R). 1 → Indicates a CRC entry. The CRC32 preload value is provided with the 4 bytes following CALCparam1. CALCparam1 is redundant for a CRC entry.
5	Reserved
6	0 → Operand length and micro code loop counter are taken from parameter set 1 (redundant for CRC entry). 1 → Operand length and micro code loop counter are taken from parameter set 2 (redundant for CRC entry).
7	0 → Direct PKC calculation (L0) is executed. CALCparam1 defines the calculation mode. 1 → Micro code execution (L1) is started. CALCparam1 defines the start address.

Each parameter of the FUP program has a size of one byte. The index parameters (XPTRind,…) are multiplied by two and added to PKC_UPTRT to fetch the selected parameters from the FUP table. PKC_UPTR is incremented automatically by one for each byte that is fetched via the L2 DMA. It is mandatory that the FUP table and program pointers and as such also the underlying parameters are 16-Bit word aligned as the least significant bit of PKC_UPTR and PKC_UPTRT is ignored.

PKC_UPTRT is used as a pointer to the start address of the FUP table. The FUP table contains the pointers, lengths and operands (for CONST) for the internal parameter registers used for the pipe operation. The table entries must have a size of two bytes. The two bytes variables have to be stored with the endianness that corresponds to the CPU architecture. Up to 256 successive 2-bytes variables are supported. PKC_UPTRT remains unchanged during the complete pipe operation.

NOTE

When modifying the content of the FUP program it is recommended to update the PKC_UPTR register (via an SFR write access) prior to starting a new L2 calculation to clear the internal code-fetch buffer.

NOTE

If the cache is enabled and the FUP table content is updated, it is recommended to invalidate the cache prior to starting a new L2 calculation.

The PKC_ULEN SFR defines the number of FUP parameter sets (FUP program entries) which shall be used for calculation in serial order without interaction of the CPU. After each parameter set fetch this register is decremented, the calculations are stopped when the PKC_ULEN register has been decremented to zero and the last operation has been completed.

The write-only PKC_CTRL[GOU] control bit is used to start the parameter fetch via L2 DMA using the universal pointer. The calculation starts immediately after the complete parameter set is loaded. During the calculation of the PKC new parameters for the next operation are fetched from ROM, RAM or Flash as long as PKC_ULEN is not zero.

The following steps are needed to perform a PKC pipe operation:

1. Store all needed operands in the RAM (accessible by PKC coprocessor)
2. Calculate the pointers and lengths for the parameter sets and define the MODE for the calculation.
3. Store these 2 byte parameters (word aligned) in the FUP table in any embedded memory.
4. Store the FUP program in the order described above in any embedded memory.
5. Calculate the checksum of the FUP program and store it at the end of the FUP program parameter bytes (optional: only when FUP program integrity check via CRC checksum is used)

6. Set PKC_ULEN to the number of calculations to be done.
7. Set PKC_UPTR to the start address of the FUP program and PKC_UPTRT to the start address of the FUP table.
8. Set the PKC_CTRL[GOU] bit to start the parameter fetch and the calculation.

When PKC_ULEN has been decremented to zero, GOANY is cleared by the PKC Control after the start of the last operation in the series. The end of the calculation is indicated by clearing the flag PKC_STATUS[ACTIV] and setting the PKC_INT_STATUS[INT_PDONE] bit. Setting one of the GOx bits while GOANY is set will trigger a security alarm.

To reduce the amount of required DMA accesses by the PKC Control a cache is implemented to latch the last used parameter values. With this, the parameters for the next calculations can be written during the running calculation to minimize the gap between two calculations.

The usage of the cache is configurable and can be disabled completely such that for each single PKC operation (either atomic macro function or micro code instruction) all calculation parameters are fetched via the universal pointer fetch. The cache can be cleared using a dedicated write only control bit (PKC_CTRL[CLRCACHE]). The cache is also invalidated if the SFR PKC_UPTRT is written or the cache is disabled via PKC_CTRL[CACHE_EN].

In case the FUP table content in RAM or Flash is updated it is mandatory to invalidate the cache to ensure that old calculation parameters available in the cache are no longer used.

NOTE

When executing two FUP programs from consecutive memory locations, one could in principle omit the update of the PKC_UPTR pointer prior to starting the second FUP program (as PKC_UPTR is auto incremented and will already point to the correct location). However, it is important to note that this will not work if the cache is invalidated in between (i.e., by asserting PKC_CTRL[CLRCACHE] or writing to SFR PKC_UPTR).

For integrity protection of the fetched FUP program a CRC32 checksum calculation can be enabled. Bit 4 in CALCparam0 indicates if the entry is a data entry (bit 4 is 0) or a CRC entry (bit 4 is 1).

In case of a CRC entry, all other bits of CALCparam0 and CALCparam1 are redundant and ignored. The remaining four bytes of the entry contain the CRC-32 pre-load value in little endian notation.

The CRC-32 is also known as AUTODIN-II (e.g., used in Ethernet, AAL5) with the following generator polynomial: x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x+1. CRC input data order is lsb first. The CRC check is disabled with each start of a new L2 calculation, triggered by setting PKC_CTRL[GOU]. The first fetch of a CRC entry activates the CRC calculation. This allows implementing FUP programs with and without CRC check (security/performance trade-off).

If the CRC calculation is not active and a CRC entry is fetched, the pre-load value is loaded into the CRC checksum register in the CRC unit. At the end of the FUP program (UPLEN equal to 0) the CRC checksum register should contain zero.

If the CRC calculation is already active and a CRC entry is fetched, the new pre-load value is XORed to the CRC checksum register (expected to be zero) such that the zero check is only needed at the very end of the FUP program.

The input bytes for the CRC calculation are processed in the same order as stored in the FUP program, resp. read and processed during the L2 calculation. In case of a data entry the byte order is: CALCparam0, CALCparam1, XPTRind, YPTRind, ZPTRind, RPTRind .

For CRC entries (CALCparam0.4=1) CALCparam0 and CALCparam1 are not used for the CRC checksum calculation but skipped, only the data entries are considered to (re-)initialize the CRC checksum. This integrity-protection concept supports multiple entry and exit points in the FUP program, easing extendability and reuse of FUP programs.

Example - initialization with pre-calculated CRC init value

1. CRC entry (init to 0x663da777): 0x10 0x00 0x77 0xA7 0x3D 0x66
2. FUP calculation 1 (addition): 0x00 0x0A 0x01 0x23 0x45 0x67
3. FUP calculation 2 (shift right): 0x00 0x15 0x89 0xab 0xcd 0xef
4. Final CRC32 equal 0x00000000

Example - init with zero, adding expected CRC32 at the end of the FUP program:

1. CRC entry 1 (init to 0x00000000): 0x10 0x00 0x00 0x00 0x00 0x00

2. FUP calculation 1 (addition): 0x00 0x0A 0x01 0x23 0x45 0x67
3. FUP calculation 2 (shift right): 0x00 0x15 0x89 0xab 0xcd 0xef
4. CRC entry 2 (zero checksum using the calculated CRC32 (0x3AE03616)): 0x10 0x00 0x16 0x36 0xE0 0x3A
5. Final CRC32 equal 0x00000000

16.4.2 PKC Kernel Description

The PKC coprocessor kernel is divided into a PKC arithmetic unit and the PKC interface shell.

The PKC arithmetic unit supports the multiplication of a PKC word (64 bits) with a multi-precision integer (MPI) and addition, subtraction, shift operation and the logical functions (AND, OR, XOR) of two multi-precision integers.

The PKC interface shell connects the arithmetic unit to the interface register sets. Furthermore a micro coding (MC) interface is provided to define more complex calculation functions, like full-size multiplication or modular multiplication.

16.4.2.1 PKC arithmetic unit (L0)

The PKC arithmetic unit is the main building block for the arithmetic operations to support public key cryptography. Table 338 lists all the calculation modes (L0 operations) supported by the PKC arithmetic unit.

Table 338. L0 calculation modes supported by the PKC arithmetic unit

Op-Code	Symbol	Operation	Description
0x00	MUL	R = X_0 · Y	Pure multiplication of a PKC word X_0 by a MPI Y
0x02	MAC	R = X_0 · Y + Z	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z
0x03	MAC_NEG	R = X_0 · Y - Z	Multiply-Subtract of a PKC word X_0 with a MPI Y and a MPI Z
0x04	MUL_GF2	R = X_0 · Y {GF(2)}	Pure multiplication of a PKC word X_0 by a MPI Y over GF(2)
0x06	MAC_GF2	R = X_0 · Y + Z {GF(2)}	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z over GF(2)
0x09	NEG	R = -Z	Two's complement of MPI Z
0x0a	ADD	R = Y + Z	Addition of a MPI Y with a MPI Z
0x0b	SUB	R = Y - Z	Subtraction of a MPI Y with a MPI Z
0x0d	AND	R = Y AND Z	Logical AND of a MPI Y with a MPI Z
0x0e	OR	R = Y OR Z	Logical OR of a MPI Y with a MPI Z
0x0f	XOR	R = Y XOR Z	Logical XOR of a MPI Y with a MPI Z
0x10	MAC_CONST_GF2	R = X_0 · Y + CONST {GF(2)}	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST over GF(2)
0x12	MAC_CONST	R = X_0 · Y + CONST	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST
0x13	MAC_NEG_CONST	R = X_0 · Y - CONST	Multiply-Subtract of a PKC word X_0 with a MPI Y and a CONST
0x14	SHL	R = Y << CONST	Shift left of a MPI Y by CONST positions
0x15	SHR	R = Y >> CONST	Shift right of a MPI Y by CONST positions

Table continues on the next page...

Table 338. L0 calculation modes supported by the PKC arithmetic unit (continued)

Op-Code	Symbol	Operation	Description
0x16	ROTL	R = rotateleft (Y) by CONST	Rotate left of a MPI Y by CONST positions
0x17	ROTR	R = rotateright (Y) by CONST	Rotate right of a MPI Y by CONST positions
0x1a	ADD_CONST	R = Y + CONST	Addition of a MPI Y with a CONST
0x1b	SUB_CONST	R = Y - CONST	Subtraction of a MPI Y with a CONST
0x1d	AND_CONST	R = Y AND CONST	Logical AND of a MPI Y with a CONST
0x1e	OR_CONST	R = Y OR CONST	Logical OR of a MPI Y with a CONST
0x1f	XOR_CONST	R = Y XOR CONST	Logical XOR of a MPI Y with a CONST
0x20	MUL1	Reg_i = $X_0 \cdot Y_0$ mod 2^wordsize	Plain multiplication of 1 PKC word modulo word size
0x22	MACC	R = $X_0 \cdot Y$ + {c,Z}	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z extended by carry overhead
0x24	MUL1_GF2	Reg_i = $X_0 \cdot Y_0$ mod 2^wordsize {GF(2)}	Plain multiplication of 1 PKC word modulo word size over GF(2)
0x26	MACC_GF2	R = $X_0 \cdot Y$ + Z {GF(2)}	Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z over GF(2)
0x2A	ADDC	R = (c,Y) + Z	Addition of a MPI Y incl. carry overhead with a MPI Z
0x2B	SUBC	R = (c,Y) - Z	Subtraction of a MPI Y incl. carry overhead with a MPI Z
0x2D	LSB0s	T = LSB0s(Z)	Determines the number of consecutive zero bits (up to 8) of MPI Z starting from LSB
0x2F	MSB0s	T = MSB0s(Z)	Determines the number of consecutive zero bits (up to 8) of MPI Z starting from MSB
0x3E	CONST	R = CONST	Initializes memory with a CONST
0x4B	CMP	Y-Z	Compares two MPIs by subtracting MPI Z from MPI Y
0x62	MACCR	R = (Reg_i $\cdot$ Y + {c,Z}) >> wordsize	Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z extended by carry overhead
0x66	MACCR_GF2	R = (Reg_i $\cdot$ Y + Z) >> wordsize {GF(2)}	Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z over GF(2)
0x6A	ADD_Z0	R = Y + Z_0	Addition of a MPI Y with a single PKC word Z_0
0x6F	XOR_Z0	R = Y XOR Z_0	XOR of a MPI Y with a single PKC word Z_0

Detailed description of the L0 calculation modes

Table 339. Plain Multiplication

Pure multiplication of a PKC word X_0 by a MPI Y		
$R = X_0 \cdot Y$	Op-code = 0x00	Symbol: MUL
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
Result	RLEN = YLEN + 1 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR and RPTR == YPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x04 - pure multiplication over GF(2) / MUL_GF2	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 340. Plain Multiplication with Addition

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z. MPI Z must be one PKC word longer than Y.		
$R = X_0 \cdot Y + Z$	Op-code = 0x02	Symbol: MAC
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	

Table continues on the next page...

Table 340. Plain Multiplication with Addition (continued)

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z. MPI Z must be one PKC word longer than Y.		
$R = X_0 \cdot Y + Z$	Op-code = 0x02	Symbol: MAC
	ZLEN = YLEN + 1 $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN + 2 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR, RPTR == YPTR and RPTR == ZPTR	
Updated status flags[1]	CARRY=1 in case of overflow (equals lsb(R[RLEN-1])), ZERO	
GF2 conversion opcode	0x06 - multiply-accumulate over GF(2) / MAC_GF2	
Execution time	4·(YLEN+2) PKC clock cycle	

1. Carry overflow is stored in carry flag and in RAM, in the most significant word of the result (R[RLEN-1]).

Table 341. Plain Multiplication with Subtraction

Multiply-Subtract of a PKC word X_0 with a MPI Y and a MPI Z. MPI Z must be one PKC word longer than Y.		
$R = X_0 \cdot Y - Z$	Op-code = 0x03	Symbol: MAC_NEG
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN + 1 $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN + 2 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR, RPTR == YPTR and RPTR == ZPTR	

Table continues on the next page...

Table 341. Plain Multiplication with Subtraction (continued)

Multiply-Subtract of a PKC word X_0 with a MPI Y and a MPI Z. MPI Z must be one PKC word longer than Y.		
$R = X_0 \cdot Y - Z$	Op-code = 0x03	Symbol: MAC_NEG
Updated status flags[1]	CARRY=1 in case result is negative (equals lsb(R[RLEN-1])), ZERO	
GF2 conversion opcode	0x06 - multiply-accumulate over GF(2) / MAC_GF2	
Execution time	4·(YLEN+2) PKC clock cycle	

1. Carry overflow is stored in carry flag and in RAM, in the most significant word of the result (R[RLEN-1]).
   A CARRY=1 indicates a negative result in two's complement notation.

Table 342. Plain Multiplication in GF2

Pure multiplication of a PKC word X_0 with a MPI Y over GF(2).		
$R = X_0 \cdot Y$ {GF(2)}	Op-code = 0x04	Symbol: MUL_GF2
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
Result	RLEN = YLEN + 1 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR and RPTR == YPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x04 (no impact)	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 343. Plain Multiplication with Addition in GF2

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z over GF(2). MPI Z must be one PKC word longer than Y.		
$R = X_0 \cdot Y + Z$ {GF(2)}	Op-code = 0x06	Symbol: MAC_GF2
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN + 1 $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN + 2 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR, RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x06 (no impact)	
Execution time	4·(YLEN+2) PKC clock cycle	

Table 344. Negation

Two's complement of MPI Z		
R = -Z	Op-code = 0x09	Symbol: NEG
Parameter	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Z in bytes
Operands	ZLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = ZLEN	

Table continues on the next page...

Table 344. Negation (continued)

Two's complement of MPI Z		
R = -Z	Op-code = 0x09	Symbol: NEG
	R = $R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == ZPTR	
Updated status flags[1]	CARRY=1 in case result is negative (only equal '1' if ZERO=1), ZERO	
GF2 conversion opcode	0x09 (no impact)	
Execution time	4·(YLEN+1) PKC clock cycle	

1. A CARRY=1 indicates a negative result in two's complement notation.

   Applying NEG(Z) on Z=0 will result in R=0 and CARRY=0

Table 345. Addition

Addition of a MPI Y and a MPI Z .		
R = Y + Z	Op-code = 0x0A	Symbol: ADD
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) Y = $Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN Z = $Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN R = $R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	

Table continues on the next page...

Table 345. Addition (continued)

Addition of a MPI Y and a MPI Z .		
R = Y + Z	Op-code = 0x0A	Symbol: ADD
GF2 conversion opcode	0x0f - logical xor / XOR	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 346. Subtraction

Subtraction of a MPI Y and a MPI Z .		
R = Y - Z	Op-code = 0x0B	Symbol: SUB
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags[1]	CARRY=1 in case of negative result, ZERO	
GF2 conversion opcode	0x0f - logical xor / XOR	
Execution time	4·(YLEN+1) PKC clock cycle	

1. A CARRY=1 indicates a negative result in two's complement notation.

Table 347. Logical AND/OR/XOR

Logical AND/OR/XOR of a MPI Y and a MPI Z .		
R = Y AND Z R = Y OR Z R = Y XOR Z	Op-code = 0x0D Op-code = 0x0E Op-code = 0x0F	Symbol: AND Symbol: OR Symbol: XOR
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x09 (no impact)	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 348. Plain Multiplication with Addition of a constant in GF2

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST over GF(2). CONST = {0,...,255}.		
$R = X_0 \cdot Y + CONST$ {GF(2)}	Op-code = 0x10	Symbol: MAC_CONST_GF2
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Constant operand
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes

Table continues on the next page...

Table 348. Plain Multiplication with Addition of a constant in GF2 (continued)

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST over GF(2). CONST = {0,...,255}.		
$R = X_0 \cdot Y + CONST$ {GF(2)}	Op-code = 0x10	Symbol: MAC_CONST_GF2
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[7:0]	
Result	RLEN = YLEN + 1 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR and RPTR == YPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x10 (no impact)	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 349. Plain Multiplication with Addition of a constant

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST. CONST = {0,...,255}.		
$R = X_0 \cdot Y + CONST$	Op-code = 0x12	Symbol: MAC_CONST
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Constant operand
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[7:0]	
Result	RLEN = YLEN + 2 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR and RPTR == YPTR	

Table continues on the next page...

Table 349. Plain Multiplication with Addition of a constant (continued)

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a CONST. CONST = {0,...,255}.		
$R = X_0 \cdot Y + CONST$	Op-code = 0x12	Symbol: MAC_CONST
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	0x10 - Multiply-Accumulate with a CONST in GF(2) / MAC_CONST_GF2	
Execution time	4·(YLEN+2) PKC clock cycle	

Table 350. Plain Multiplication with Subtraction of a constant

Multiply-Subtract of a PKC word X_0 with a MPI Y and a CONST. CONST = {0,...,255}.		
$R = X_0 \cdot Y - CONST$	Op-code = 0x13	Symbol: MAC_NEG_CONST
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Constant operand
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[7:0]	
Result	RLEN = YLEN + 2 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR and RPTR == YPTR	
Updated status flags[1]	CARRY=1 in case of negative result, ZERO	
GF2 conversion opcode	0x10 - Multiply-Accumulate with a CONST in GF(2) / MAC_CONST_GF2	
Execution time	4·(YLEN+2) PKC clock cycle	

1. A CARRY=1 indicates a negative result in two's complement notation.

Table 351. Shift and Rotate

Shift/Rotate left/right a MPI Y by CONST positions. CONST = {0,...,63}. Redundant most significant bits of CONST are ignored.		
R = shiftleft (Y) by CONST R = shiftright (Y) by CONST R = rotateleft (Y) by CONST R = rotateright (Y) by CONST	Op-code = 0x14 Op-code = 0x15 Op-code = 0x16 Op-code = 0x17	Symbol: SHL Symbol: SHR Symbol: ROTL Symbol: ROTR
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Shift/rotate factor, max. value depends on REDMUL setting
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[5:0] for REDMUL == 0x0 or 0x2	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR	
Updated status flags	CARRY=1 in case shifted/rotated out bit is '1', ZERO	
GF2 conversion opcode	no impact	
Execution time	SHL: 4·(YLEN+1) PKC clock cycle SHL, ROTL, ROTR: 4·(YLEN+2) PKC clock cycle	

NOTE

Internal register Reg_i is updated by SHR, ROTL and ROTR calculation with intermediate data:

- For SHR and CONST unequal zero $Reg_i = (Y << (wordsize - CONST))_{YLEN-1}$, with wordsize = 64-bit for REDMUL == 0x0 or 0x2. For CONST = 0 $Reg_i = Y_{YLEN-1}$
- For ROTL $Reg_i = (Y << CONST)_0$.
- For ROTR and CONST unequal zero $Reg_i = (Y << (wordsize - CONST))_0$, with wordsize = 64-bit for REDMUL == 0x0 or 0x2. For CONST = 0 $Reg_i = Y_{YLEN-1}$

Table 352. Addition with CONST

Addition of a MPI Y with a CONST. CONST = {0,...,255}.		
R = Y + CONST	**Op-code = 0x1A**	**Symbol: ADD_CONST**
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Value of constant operand CONST.
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[7:0]	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	
GF2 conversion opcode	0x1F - Logical XOR with constant / XOR_CONST	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 353. Subtraction with CONST

Subtraction of a MPI Y with a CONST. CONST = {0,...,255}.		
R = Y - CONST	**Op-code = 0x1B**	**Symbol: SUB_CONST**
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Value of constant operand CONST.
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	CONST = ZPTR[7:0]	
Result	RLEN = YLEN	

Table continues on the next page...

Table 353. Subtraction with CONST (continued)

Subtraction of a MPI Y with a CONST. CONST = {0,...,255}.		
R = Y - CONST	Op-code = 0x1B	Symbol: SUB_CONST
	$R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR	
Updated status flags[1]	CARRY=1 in case of negative result, ZERO	
GF2 conversion opcode	0x1F - Logical XOR with constant / XOR_CONST	
Execution time	4·(YLEN+1) PKC clock cycle	

1. A CARRY=1 indicates a negative result in two's complement notation.

Table 354. Logical AND/OR/XOR with CONST

Logical AND/OR/XOR of a MPI Y with a CONST. CONST = {0,...,255}.		
R = Y AND CONST R = Y OR CONST R = Y XOR CONST	Op-code = 0x1D Op-code = 0x1E Op-code = 0x1F	Symbol: AND_CONST Symbol: OR_CONST Symbol: XOR_CONST
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Value of constant operand CONST expanded to every byte.
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = LEN $CONST = ZPTR[7:0] \cdot 2^{8 \cdot (ZLEN-1)} + ZPTR[7:0] \cdot 2^{8 \cdot (ZLEN-2)} + ... + ZPTR[7:0] \cdot 2^8 + ZPTR[7:0]$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	no impact	

Table continues on the next page...

Table 354. Logical AND/OR/XOR with CONST (continued)

Logical AND/OR/XOR of a MPI Y with a CONST. CONST = {0,...,255}.		
R = Y AND CONST R = Y OR CONST R = Y XOR CONST	Op-code = 0x1D Op-code = 0x1E Op-code = 0x1F	Symbol: AND_CONST Symbol: OR_CONST Symbol: XOR_CONST
Execution time	4·(YLEN+1) PKC clock cycle	

Table 355. Plain Multiplication modulo PKC wordsize

Plain multiplication of a PKC word X_0 by a PKC word Y_0 modulo PKC wordsize. Result is stored in internal register Reg_i		
$Reg_i = X_0 \cdot Y_0 \bmod 2^{wordsize}$	Op-code = 0x20	Symbol: MUL1
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to operand Y_0: Byte address of Y_0
	LEN[1]	don't care --> set to 1 PKC word, depends on REDMUL configuration
Operands	X_0	
	Y_0	
Result[2]	Reg_i	
Result-in-place	n/a	
Updated status flags	no result flags updated	
GF2 conversion opcode	0x24 - plain multiplication of a PKC word modulo wordsize over GF(2) / MUL1_GF2	
Execution time	4 PKC clock cycle	

1. Operand length is fixed to one PKC word (64-bit for REDMUL == 0x0 or 0x2). PKC_LENi.LEN is ignored.
2. Result is written to internal register Reg_i.

Table 356. Multiplication with Addition incl. previous carry overhead

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z extended by carry overhead. MPI Y and MPI Z must have equal length.		
$R = X_0 \cdot Y + \{c,Z\}$	Op-code = 0x22	Symbol: MACC
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0

Table continues on the next page...

Table 356. Multiplication with Addition incl. previous carry overhead (continued)

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z extended by carry overhead. MPI Y and MPI Z must have equal length.		
$R = X_0 \cdot Y + \{c,Z\}$	Op-code = 0x22	Symbol: MACC
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $\{c,Z\} = CARRY \cdot b^{ZLEN} + Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN + 1 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR, RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	
GF2 conversion opcode	0x26 - multiply-accumulate over GF(2) / MACC_GF2	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 357. Plain Multiplication modulo PKC wordsize in GF2

Plain multiplication of a PKC word X_0 by a PKC word Y_0 modulo PKC wordsize in GF(2). Result is stored in internal register Reg_i		
$Reg_i = X_0 \cdot Y_0 \bmod 2^{wordsize}$ {GF(2)}	Op-code = 0x24	Symbol: MUL1_GF2
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to operand Y_0: Byte address of Y_0
	LEN[1]	don't care --> set to 1 PKC word, depends on REDMUL configuration
Operands	X_0	
	Y_0	
Result[2]	Reg_i	
Result-in-place	n/a	

Table continues on the next page...

Table 357. Plain Multiplication modulo PKC wordsize in GF2 (continued)

Plain multiplication of a PKC word X_0 by a PKC word Y_0 modulo PKC wordsize in GF(2). Result is stored in internal register Reg_i		
$Reg_i = X_0 \cdot Y_0 \bmod 2^{wordsize}$ {GF(2)}	Op-code = 0x24	Symbol: MUL1_GF2
Updated status flags	no result flags updated	
GF2 conversion opcode	no impact	
Execution time	4 PKC clock cycle	

1. Operand length is fixed to one PKC word (64-bit for REDMUL == 0x0 or 0x2). PKC_LENi.LEN is ignored.
2. Result is written to internal register Reg_i.

Table 358. Multiplication with Addition in GF2

Multiply-Accumulate of a PKC word X_0 with a MPI Y and a MPI Z . MPI Y and MPI Z must have equal length.		
$R = X_0 \cdot Y + \{c,Z\}$ {GF(2)}	Op-code = 0x26	Symbol: MACC_GF2
Parameter	XPTR	Pointer to operand X_0: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	X_0	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN + 1 $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == XPTR, RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	no impact	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 359. Addition incl. previous carry overhead

Addition of a MPI Y extended by carry overhead and a MPI Z.		
R = {c,Y} + Z	Op-code = 0x2A	Symbol: ADDC
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $\{c,Y\} = CARRY \cdot b^{YLEN} + Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	
GF2 conversion opcode	0x0f - logical xor / XOR	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 360. Subtraction incl. previous carry overhead

Subtraction of a MPI Y extended by carry overhead and a MPI Z.		
R = {c,Y} - Z	Op-code = 0x2A	Symbol: SUBC
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $\{c,Y\} = CARRY \cdot b^{YLEN} + Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN	

Table continues on the next page...

Table 360. Subtraction incl. previous carry overhead (continued)

Subtraction of a MPI Y extended by carry overhead and a MPI Z.		
R = {c,Y} - Z	Op-code = 0x2A	Symbol: SUBC
	$Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags[1]	CARRY=1 in case of overflow or negative result, ZERO	
GF2 conversion opcode	0x0f - logical xor / XOR	
Execution time	4·(YLEN+1) PKC clock cycle	

1. CARRY=1 represents an overflow only in case input carry c has been set. If input carry c=0 a set CARRY flag for this calculation represents a negative result in two's complement notation.

Table 361. Least significant zero bits

Determines the number of consecutive zero bits (up to 8) of MPI Z starting from lsb. No data is written to RAM, only flags and internal T-register are updated. The result T is in the range of 0 to 8. T can only be used with L1 calculations.		
T = LSB0s(Z)	Op-code = 0x2D	Symbol: LSB0s
Parameter	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	LEN	Length of operand Z in bytes
Operands	ZLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result[1]	T-register	
Result-in-place	n/a	
Updated status flags	CARRY=0, ZERO=1 if LSB0s(Z)=0 resp. least significant bit of Z operand is '1' --> Z is odd	
GF2 conversion opcode	no impact	
Execution time	16 PKC clock cycle	

1. In case this operation is used in a L1 operation, register T is assigned and stable only after a CondJump. This is required for all MC instructions following the LSB0s calculation that read or write the T pointer: InDecPtr(Ptr_Sel=T), PreLoadT, Execute(Mode={LSB0s, MSB0s, calculation modes with CONST + ConfLoad(ConstSrc=1)})

Table 362. Most significant zero bits

Determines the number of consecutive zero bits (up to 8) of MPI Z starting from msb. No data is written to RAM, only flags and internal T-register are updated. The result T is in the range of 0 to 8. T can only be used with L1 calculations.		
T = MSB0s(Z)	Op-code = 0x2E	Symbol: MSB0s
Parameter	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	LEN	Length of operand Z in bytes
Operands	ZLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result[1]	T-register	
Result-in-place	n/a	
Updated status flags	CARRY=0, ZERO=1 if MSB0s(Z)=0 resp. most significant bit of Z operand is '1'	
GF2 conversion opcode	no impact	
Execution time	16 PKC clock cycle	

1. In case this operation is used in a L1 operation T is assigned and stable only after a CondJump. This is required for all MC instructions following the MSB0s calculation that read or write the T pointer: InDecPtr(Ptr_Sel=T), PreLoadT, Execute(Mode={LSB0s, MSB0s, calculation modes with CONST + ConfLoad(ConstSrc=1)})

Table 363. Initialization function

Initializes memory with a CONST. CONST = {0,...,255}.		
R = CONST	Op-code = 0x3E	Symbol: CONST
Parameter	ZPTR	Value of constant operand CONST expanded to every byte
	LEN	Length of result area R in bytes
Operands	ZLEN = LEN $CONST = ZPTR[7:0] \cdot 2^{8 \cdot (ZLEN-1)} + ZPTR[7:0] \cdot 2^{8 \cdot (ZLEN-2)} + ... + ZPTR[7:0] \cdot 2^8 + ZPTR[7:0]$	
Result	RLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	n/a	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	no impact	

Table continues on the next page...

Table 363. Initialization function (continued)

Initializes memory with a CONST. CONST = {0,...,255}.		
R = CONST	Op-code = 0x3E	Symbol: CONST
Execution time	4·(YLEN+1) PKC clock cycle	

Table 364. Compare

Compares two MPIs by subtracting MPI Z from MPI Y. No result is written to RAM, only flags are updated. Start address of MPI Y and MPI Z must not be equal.		
Y - Z	Op-code = 0x4B	Symbol: CMP
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	LEN	Length of operand Y and Z in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	n/a - only CARRY and ZERO status flag is updated	
Result-in-place	n/a	
Updated status flags	CARRY=1 in case of Z > Y, ZERO in case Y == Z	
GF2 conversion opcode	no impact	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 365. Multiplication with Addition incl. previous carry overhead ignoring least significant result word

Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z extended by carry overhead ignoring least significant result word. MPI Z must be one word longer than Y. R is one PKC word bigger than Y but the least significant result word R[0] is withdrawn such that the resulting R has the same length as MPI Y.		
$R = Reg_i \cdot Y + \{c,Z\} >> wordsize$	Op-code = 0x62	Symbol: MACCR
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0

Table continues on the next page...

Table 365. Multiplication with Addition incl. previous carry overhead ignoring least significant result word (continued)

Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z extended by carry overhead ignoring least significant result word. MPI Z must be one word longer than Y. R is one PKC word bigger than Y but the least significant result word R[0] is withdrawn such that the resulting R has the same length as MPI Y.		
$R = Reg_i \cdot Y + \{c,Z\} >> wordsize$	Op-code = 0x62	Symbol: MACCR
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	Reg_i	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN+1 $\{c,Z\} = CARRY \cdot b^{ZLEN} + Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN} \cdot b^{RLEN-1} + R_{RLEN-1} \cdot b^{RLEN-2} + ... + R_2 \cdot b + R_1$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	
GF2 conversion opcode	0x66 - multiply-accumulate over GF(2) / MACCR_GF2	
Execution time	4·(YLEN+1) PKC clock cycle	

NOTE

Theoretically the result can have a carry overflow of 0x2 that is not covered in the carry status flag (CARRY=0 in this case). However in the typical use case of the Montgomery Multiplication and Reduction this can never happen. In case MACCR operation is used in a different context the max. possible result has to be considered.

An operation that writes to Reg_i (e.g. MUL1 (0x20)) has to be executed directly before executing MACCR. In case Reg_i is not initialized by a proper preceding calculation the result of the MACCR calculation may be undefined.

Table 366. Multiplication with Addition ignoring least significant result word in GF(2)

Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z ignoring least significant result word in GF(2). MPI Z must be one word longer than Y. R is one PKC word bigger than Y but the least significant result word R[0] is withdrawn such that the resulting R has the same length as MPI Y.		
$R = Reg_i \cdot Y + Z >>$ wordsize {GF(2)}	Op-code = 0x66	Symbol: MACCR_GF2
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0

Table continues on the next page...

Table 366. Multiplication with Addition ignoring least significant result word in GF(2) (continued)

Multiply-Accumulate of the internal register Reg_i with a MPI Y and a MPI Z ignoring least significant result word in GF(2). MPI Z must be one word longer than Y. R is one PKC word bigger than Y but the least significant result word R[0] is withdrawn such that the resulting R has the same length as MPI Y.		
$R = Reg_i \cdot Y + Z >>$ wordsize {GF(2)}	Op-code = 0x66	Symbol: MACCR_GF2
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	Reg_i	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN+1 $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = YLEN $R = R_{RLEN} \cdot b^{RLEN-1} + R_{RLEN-1} \cdot b^{RLEN-2} + ... + R_2 \cdot b + R_1$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	no impact	
Execution time	4·(YLEN+1) PKC clock cycle	

NOTE

An operation that writes to Reg_i (e.g. MUL1_GF2 (0x24)) has to be executed directly before executing MACCR_GF2. In case Reg_i is not initialized by a proper preceding calculation the result of the MACCR_GF2 calculation may be undefined.

Table 367. Addition with single PKC word Z_0

Addition of a MPI Y with a single PKC word Z_0 .		
$R = Y + Z_0$	Op-code = 0x6A	Symbol: ADD_Z0
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to single PKC word operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2)	

Table continues on the next page...

Table 367. Addition with single PKC word Z_0 (continued)

Addition of a MPI Y with a single PKC word Z_0 .		
$R = Y + Z_0$	Op-code = 0x6A	Symbol: ADD_Z0
	$Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	Z_0	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=1 in case of overflow, ZERO	
GF2 conversion opcode	0x6f - logical xor with single PKC word Z / XOR_Z0	
Execution time	4·(YLEN+1) PKC clock cycle	

Table 368. XOR with single PKC word Z_0

XOR of a MPI Y with a single PKC word Z_0 .		
$R = Y$ XOR Z_0	Op-code = 0x6F	Symbol: XOR_Z0
Parameter	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to single PKC word operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
Operands	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	Z_0	
Result	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR	
Updated status flags	CARRY=0, ZERO	
GF2 conversion opcode	no impact	

Table continues on the next page...

Table 368. XOR with single PKC word Z_0 (continued)

XOR of a MPI Y with a single PKC word Z_0 .		
R = Y XOR Z_0	Op-code = 0x6F	Symbol: XOR_Z0
Execution time	4·(YLEN+1) PKC clock cycle	

16.4.2.2 Micro code (MC) interface (L1)

The MC programming interface uses simple microcode instructions to:

- Initialize a PKC L0 calculation (increment/decrement pointer, select pointer for calculation)
- Start a PKC L0 calculation
- Decrement a loop counter and branch if not zero
- Load pointer into internal shadow T register (incl. auto-increment/decrement)
- Select parameter sets
- Unconditional jumps and conditional jumps based on carry and zero flag

Two internal pointers S (S-Ptr) and T (T-Ptr) are implemented to support pointer handling, e.g. to backup and recover a pointer. S and T pointer can be loaded and updated with the PreLoadT instruction and InDecPtr only (they are not directly accessible through the register memory map). The T pointer can furthermore be used as alternative to the CONST value, defined via ConfLoad. On top the result of the LSB0s and MSB0s operation is stored in T pointer automatically. S pointer is for storage only. With each start of a new L1 calculation the internal pointer registers S-Ptr and T-Ptr are reset to zero.

All pointer and length values used in the micro-code layer have word granularity that depends on the PKC_CTRL[REDMUL] setting, e.g.:

- X pointer is PKC_XYPTRi[XPTR] >> 3 (REDMUL == 0x0 or 0x2)
- operand length is PKC_LENi[LEN] >> 3 (REDMUL == 0x0 or 0x2)
- loop counter LC is PKC_LENi[MCLEN] >> 3 (REDMUL == 0x0 or 0x2)

The coding of the micro code instructions is shown in the Table 369.

Table 369. PKC micro code instructions

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	MC instruction
1	Mode (L0 operation)							Execute
0	1	Addr (relative address +31,...,-32)						DecrTBNZ
0	0	1	Const_Src	Conf_Load				ConfLoad
0	0	0	1	Incr	Ptr_Sel			InDecPtr
0	0	0	0	1	Cond			CondJump (uses both bytes)
Addr (relative address +127,...,-128)								
0	0	0	0	0	1	Reserved		Reserved

Table continues on the next page...

Table 369. PKC micro code instructions (continued)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	MC instruction
0	0	0	0	0	0	1	Const	PreLoadT (two bytes)
Exch	T2Ptr	AddEN	IncrEN	Add_Incr	Ptr_Sel			
8-bit Constant value								
0	0	0	0	0	0	0	1	Jump (two bytes)
Addr (relative address +127,...,-128)								
0	0	0	0	0	0	0	0	Exit

MC instruction: Execute

Starts execution of the L0 operation indicated by the parameter Mode. When trying to start a new operation while a previous one is still ongoing, the Execute instruction will block (until next operation can be started). Table 370 describes the usage of the different parameters of the Execute instruction.

Table 370. Description of the parameters used within the Execute instruction

Parameter	Description
Mode[6:0]	Op-code of native L0 operation that should be started (see PKC arithmetic unit (L0) for a list of all available L0 op-codes).

MC instruction: DecrTBNZ

The DecrTBNZ (Decrement, Test and Branch if Not Zero) instruction decrements the internal loop-counter register (LC) and performs a relative jump (i.e., +31 to -32 is added to program counter) in the MC table in case the loop counter is not zero. For example, if the DecrTBNZ instruction is located at address a and the offset value (relative address) is b (in two's complement representation), the final jump address where the next instruction will be fetched from is a + b. In case the loop counter is zero after the decrement, it is reloaded again with the preload value from PKC_LENi.MCLEN and increments the program counter by one (i.e., next instruction in table is executed afterwards). Table 371 describes the usage of the different parameters of the DecrTBNZ instruction. In case the DecrTBNZ instruction is executed on a zero loop counter value a security alarm is triggered.

Note: The DecrTBNZ has no relation to the internal T pointer of the MC program interface. The T pointer is neither read nor updated by the DecrTBNZ MC instruction.

Table 371. Description of the parameters used within the DecrTBNZ instruction

Parameter	Description
Addr[5:0]	Relative address that is used in case of a jump. The address is in the range of +31 to -32 and is added to the current program counter.

MC instruction: ConfLoad

The ConfLoad instruction configures the loading of the operands, i.e., how pointers from a register set are loaded into the PKC kernel. The main purpose of this configuration is to be able to use different pointers for different PKC kernel operands or the result, e.g., to be able to use an operand previously stored at RPTR as an input operand (X, Y, Z) for a following calculation or to write the result of a calculation to RAM address defined by an input operand pointer (e.g. ZPTR). A dedicated flag is used to indicate the source of the constant value provided to the kernel (either lower byte of ZPTRor T-Ptr is used). The specified configuration on

a ConfLoad instruction stays active until the next ConfLoad instruction is executed or the pattern is exited (default configuration is used). Table 372 describes the usage of the different parameters of the ConfLoad instruction.

Table 372. Description of the parameters used within the ConfLoad instruction

Parameter	Description
Const_Src	Define source of the constant value provided to the PKC kernel • 0: constant value is taken from lower byte of ZPTR • 1: constant value is taken from lower byte of T-Ptr
Conf_Load[3:0]	Selects a configuration that is used for loading operands into the PKC kernel (e.g., using configuration 0001 changes behavior of addition operation from R = Y + Z to R = R + Z). • 0000: XYZR ☐ XYZR • 0001: ZRZR ☐ XYZR • 0010: XYRR ☐ XYZR • 0011: XYXY ☐ XYZR • 0100: XXYX ☐ XYZR • 0101: XZRR ☐ XYZR • 0110: XRXR ☐ XYZR • 0111: XXYR ☐ XYZR • 1000: XZRZ ☐ XYZR • 1001: XYRY ☐ XYZR • 1010: YRZZ ☐ XYZR • 1011: Reserved (XYZR ☐ XYZR) • 1100: Reserved (XYZR ☐ XYZR) • 1101: Reserved (XYZR ☐ XYZR) • 1110: Reserved (XYZR ☐ XYZR) • 1111: Reserved (XYZR ☐ XYZR) • 1111: Reserved (XYZR ☐ XYZR)

MC Instruction: InDecPtr

The InDecPtr instruction increments or decrements a selected pointer/parameter. Table 373 describes the usage of the different parameters of the InDecPtr instruction.

Table 373. Description of the parameters used within the InDecPtr instruction

Parameter	Description
Incr	Selects increment or decrement 0: decrement pointer/parameter selected via Ptr_Sel 1: increment pointer/parameter selected via Ptr_Sel
Ptr_Sel[2:0]	Selects a pointer/parameter

Table continues on the next page...

Table 373. Description of the parameters used within the InDecPtr instruction (continued)

Parameter	Description
	• 000: select pointer X • 001: select pointer Y • 010: select pointer Z • 011: select pointer R • 100: select pointer S • 101: select pointer T • 110: select parameter LEN (operand length) • 111: select parameter LC (loop counter)

MC instruction: CondJump

This instruction performs a conditional jump and consists of two bytes. The first byte contains the jump condition; different jump conditions can be selected: carry flag, zero flag, combination of carry and zero flag. The second byte contains the relative address that is added to the program counter in case the defined jump condition is fulfilled. For example, if the CondJump instruction is located at address a and a + 1 and the offset value (relative address) is b (in two's complement representation), the final jump address where the next instruction will be fetched from is a + 1 + b. If the jump condition is not fulfilled, the program counter is incremented by one (i.e., next instruction in the MC table is executed). It is important to note that the CondJump instruction blocks until the current operation has completely finished (i.e., carry and zero flag of current operation are valid), which typically takes 8 extra PKC clock cycles. Table 374 describes the usage of the different parameters of the CondJump instruction.

Table 374. Description of the parameters used within the CondJump instruction

Parameter	Description
Cond[2:0]	Defines the jump condition (selection of flag(s) and state(s)) • 0xy: jump if zero flag = x and carry flag = y • 10y: jump if zero flag = y • 11y: jump if carry flag = y
Addr[7:0]	Relative address that is used in case jump condition is fulfilled. The address is in the range of +127 to -128 and is added to the current program counter.

MC instruction: PreLoadT

The PreLoadT instruction allows transferring data to and from the internal T register (T-Ptr) inside the MC layer and consists of two bytes. This includes also an option to load 8 bit constants into the T register. Data can also be manipulated during the transfer (increment/decrement, addition/subtraction) and exchanged with data from another pointer register/parameter. Table 375 describes the usage of the different parameters of the PreLoadT instruction.

Table 375. Description of the parameters used within the PreLoadT instruction

Parameter	Description
Const	Defines if second byte contains configuration options for transferring data between selected pointer/parameter and T register or an 8 bit constant value

Table continues on the next page...

Table 375. Description of the parameters used within the PreLoadT instruction (continued)

Parameter	Description
	• 0: second byte contains configuration options for transferring data • 1: second byte contains 8 bit constant value that is loaded into T register
Const[7:0]	8 bit constant value loaded into T register - padded with 0s (Const = 1)
Exch	Enables exchanging of data between T register and selected pointer/parameter, with optional data manipulation (T2Ptr is not considered). • 0: exchanging of data is disabled • 1: exchanging of data is enabled, i.e., in addition to normal data transfer Ptr = T (T2Ptr = 0) (e.g., T = Ptr - T; Ptr = T (T2Ptr = 0; AddEN = 1; IncrEN = X; Add_Incr = 0)) T = Ptr (T2Ptr = 1) (e.g., Ptr = T - Ptr; T = Ptr (T2Ptr = 1; AddEN = 1; IncrEN = X; Add_Incr = 0) Pointer assignments are done concurrently.
T2Ptr	Defines if data is loaded from selected pointer/parameter to T register or the other way around, with optional data manipulation • 0: data loaded into T register from selected pointer/parameter T = Ptr - T (AddEN = 1; IncrEN = X; Add_Incr = 0) T = Ptr + T (AddEN = 1; IncrEN = X; Add_Incr = 1) T = Ptr - 1 (AddEN = 0; IncrEN = 1; Add_Incr = 0) T = Ptr + 1 (AddEN = 0; IncrEN = 1; Add_Incr = 1) T = Ptr (AddEN = 0; IncrEN = 0; Add_Incr = X) • 1: data stored from T register to selected pointer/parameter Ptr = T - Ptr (AddEN = 1; IncrEN = X; Add_Incr = 0) Ptr = T + Ptr (AddEN = 1; IncrEN = X; Add_Incr = 1) Ptr = T - 1 (AddEN = 0; IncrEN = 1; Add_Incr = 0) Ptr = T + 1 (AddEN = 0; IncrEN = 1; Add_Incr = 1) Ptr = T (AddEN = 0; IncrEN = 0; Add_Incr = X)
AddEN	Enables/disables adding/subtracting T register or selected pointer/parameter • 0: adding/subtracting disabled • 1: adding (Add_Incr = 1) / subtracting (Add_Incr = 0) enabled
IncrEN	Enables/disables incrementing/decrementing T register or selected pointer/parameter. Ignored if AddEN = 1 • 0: incrementing/decrementing disabled • 1: incrementing (Add_Incr = 1) / decrementing (Add_Incr = 0) enabled

Table continues on the next page...

Table 375. Description of the parameters used within the PreLoadT instruction (continued)

Parameter	Description
Add_Incr	Selects added/incremented or subtracted/decremented operation • 0: subtracting/decrementing • 1: adding/incrementing
Ptr_Sel[2:0]	Selects pointer/parameter that should be used for transferring/exchanging data with T register • 000: select pointer X • 001: select pointer Y • 010: select pointer Z • 011: select pointer R • 100: select pointer S • 101: select pointer T • 110: select parameter LEN • 111: select parameter LC

The condition 'Ptr_Sel = T' is a special case as in this case the Exch and T2Ptr parameters are don't care. Table 376 shows how pointers are updated when PreLoadT is operated with 'Ptr_Sel != T' and 'Ptr_Sel = T'.

Table 376. Description of pointer update during PreLoadT instruction for Ptr_Sel != T and Ptr_Sel = T

Exch	T2Ptr	AddEN	IncrEN	Add_Incr	Ptr_Sel != T	Ptr_Sel = T
1	0	1	X	0	T = Ptr - T; Ptr = T	T = 0
1	0	1	X	1	T = Ptr + T; Ptr = T	T = 2T
1	0	0	1	0	T = Ptr - 1; Ptr = T	T = T - 1
1	0	0	1	1	T = Ptr + 1; Ptr = T	T = T + 1
1	0	0	0	X	T = Ptr; Ptr = T	T = T
0	0	1	X	0	T = Ptr - T	T = 0
0	0	1	X	1	T = Ptr + T	T = 2T
0	0	0	1	0	T = Ptr - 1	T = T - 1
0	0	0	1	1	T = Ptr + 1	T = T + 1
0	0	0	0	X	T = Ptr	T = T
0	1	1	X	0	Ptr = T - Ptr	T = 0
0	1	1	X	1	Ptr = T + Ptr	T = 2T
0	1	0	1	0	Ptr = T - 1	T = T - 1
0	1	0	1	1	Ptr = T + 1	T = T + 1
0	1	0	0	X	Ptr = T	T = T
1	1	1	X	0	Ptr = T - Ptr; T = Ptr	T = 0

Table continues on the next page...

Table 376. Description of pointer update during PreLoadT instruction for Ptr_Sel != T and Ptr_Sel = T (continued)

Exch	T2Ptr	AddEN	IncrEN	Add_Incr	Ptr_Sel != T	Ptr_Sel = T
1	1	1	X	1	Ptr = T + Ptr; T = Ptr	T = 2T
1	1	0	1	0	Ptr = T - 1; T = Ptr	T = T -1
1	1	0	1	1	Ptr = T + 1; T = Ptr	T = T + 1
1	1	0	0	X	Ptr = T; T = Ptr	T = T

MC instruction: Jump

This instruction performs an unconditional jump and consists of two bytes. The first byte contains only the op-code, the second byte contains the relative address that is added to the program counter. For example, if the Jump instruction is located at address a and a+1 and the offset value (relative address) is b (in two's complement representation), the final jump address where the next instruction will be fetched from is a + 1 + b. Table 377 describes the usage of the different parameters of the Jump instruction.

Table 377. Description of the parameters used within the Jump instruction

Parameter	Description
Addr[7:0]	Relative address (+127 to -128) that is added to the program counter

MC instruction: Exit

The Exit instruction marks the end of a pattern. The execution of the pattern stops. The Exit instruction has no parameters. When the execution of a pattern is stopped, configuration values (potentially changed during the execution of a pattern) are set back to default (e.g., Conf_Load is set to 0000 and Const_Src to 0).

Any MC instruction not listed above will trigger a security alarm.

16.4.2.3 Fixed L1 MC Patterns

The PKC provides the fixed MC patterns that are listed in Table 378.

Table 378. Fixed L1 MC patterns

Start-Addr	MC size [bytes]	Symbol	Operation	Description
0x00	19	MM	R = X · Y mod Z	Modular Multiplication of a MPI X with a MPI Y modulo a MPI Z (Montgomery Reduction)
0x00	19	MM_GF2	R = X · Y mod Z {GF(2)}	Modular Multiplication of a MPI X with a MPI Y modulo a MPI Z (Montgomery Reduction) over GF(2)
0x13	10	PM	R = X · Y	Plain Multiplication of a MPI X with a MPI Y
0x13	10	PM_GF2	R = X · Y {GF(2)}	Plain Multiplication of a MPI X with a MPI Y over GF(2)
0x1D	4	PMA	R = X · Y + Z	Plain Multiplication with Addition of a MPI X with a MPI Y and a MPI Z
0x1D	4	PMA_GF2	R = X · Y + Z {GF(2)}	Plain Multiplication with Addition of a MPI X with a MPI Y and a MPI Z over GF(2)

Table continues on the next page...

Table 378. Fixed L1 MC patterns (continued)

Start-Addr	MC size [bytes]	Symbol	Operation	Description
0x21	9	MA	R = Y + Z mod X	Modular Addition of a MPI Y with a MPI Z modulo a MPI X
0x2A	9	MS	R = Y - Z mod X	Modular Subtraction of a MPI Y with a MPI Z modulo a MPI X
0x33	32	MR	R = X mod Z	Modular Reduction of a MPI X and a MPI Z
0x33	32	MR_GF2	R = X mod Z {GF(2)}	Modular Reduction of a MPI X and a MPI Z over GF(2)
0x53	10	MMP2	$R = X \cdot Y \mod 2^{LEN(Y)}$	Modular Multiplication of a MPI X and a MPI Y modulo $2^{LEN(Y)}$
0x5A	10	MMAP2	$R = X \cdot Y + Z \mod 2^{LEN(Y)}$	Modular Multiplication with Addition of a MPI X with a MPI Y and a MPI Z modulo $2^{LEN(Y)}$
0x5D	64	MI	$R = Y^{-1} \cdot 2^k \mod X$	Modular Inversion of a MPI Y module a MPI X
0x5D	64	MI_GF2	$R = Y^{-1} \cdot 2^k \mod X$ {GF(2)}	Modular Inversion of a MPI Y module a MPI X over GF(2)
0x9D	10	PM_PATCH	$R = X \cdot Y$	Plain Multiplication of a MPI X with a MPI Y (patched version) NOTE: same functionality as PM, but using patched code that supports corner case: MPI X > 1 PKC word and MPI Y = 1 PKC word (slightly longer execution time)
0x9D	10	PM_PATCH_GF2	$R = X \cdot Y$ {GF(2)}	Plain Multiplication of a MPI X with a MPI Y over GF(2) (patched version) NOTE: same functionality as PM_GF2, but using patched code that supports corner case: MPI X > 1 PKC word and MPI Y = 1 PKC word (slightly longer execution time)
0xA7	28	GCD	R = GCD(Y, Z)	Greatest common divider (GCD) of MPI Y and MPI Z

Detailed description of the fixed L1 calculation modes

Table 379. Modular Multiplication

Modular multiplication of a MPI X with a MPI Y modulo a MPI Z. MPI Y and MPI Z must have equal length. Montgomery algorithm is used for reduction.		
$R = X \cdot Y \mod Z$	Start-Addr = 0x00	Symbol: MM / MM_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0

Table continues on the next page...

Table 379. Modular Multiplication (continued)

Modular multiplication of a MPI X with a MPI Y modulo a MPI Z. MPI Y and MPI Z must have equal length. Montgomery algorithm is used for reduction.		
R = X · Y mod Z	Start-Addr = 0x00	Symbol: MM / MM_GF2
		Multiplicative Inverse ~Z has to be stored in the PKC word directly before Z_0.
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
	$\sim Z = (-Z)^{-1} \bmod 2^{wordsize}$; wordsize = 64 for REDMUL == 0x0 or 0x2	
Result[2]	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN+1	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR. Not possible for RTPR == ZPTR.	
Updated status flags	CARRY=0, ZERO	
GF2 conversion	Modular Multiplication in GF(2) / MM_GF2	
Execution time	4·XLEN·(2·YLEN+3)+8 PKC clock cycle; In case of optional final reduction the cycle count is increased by (YLEN+1), not applicable for MM_GF2	

MC code	`ModMul :  ConfLoad(0,XYZR)`
	`  Execute(0x00)`
	`ModMul_Loop :  InDecPtr(0,Z)`
	`  ConfLoad(0,ZRZR)`
	`  Execute(0x20)`
	`  InDecPtr(1,Z)`
	`  ConfLoad(0,XZRR)`
	`  Execute(0x62)`
	`  DecrTBNZ(ModMul_Cont)`	

Table continues on the next page...

Table 379. Modular Multiplication (continued)

Modular multiplication of a MPI X with a MPI Y modulo a MPI Z. MPI Y and MPI Z must have equal length. Montgomery algorithm is used for reduction.		
R = X · Y mod Z	Start-Addr = 0x00	Symbol: MM / MM_GF2
	``` ModMul_CondSub : ConfLoad(0,ZRZR) CondJump(C=0) (ModMul_Exit) Execute(0x2B) ModMul_Exit : Exit() ModMul_Cont : InDecPtr(1,X) ConfLoad(0,XYRR) Execute(0x22) Jump() (ModMul_Loop) ```	

1. In case XLEN>1 and YLEN=1 the least-significant PKC word of the result R is read before it is written the very first time. This can lead to reading uninitialized memory. If this is an issue the least-significant PKC word of the result R needs to be initialized upfront before calling the operation.
2. The result R might be bigger than the modulus (R ≥ Z). The reduction only guarantees that the result fits to the wordsize of the modulus. → $R < 2^{YLEN \cdot wordsize}$

   The length of the result is equal RLEN(=YLEN) but for the calculation one extra PKC word on top of the result R is required. The width of the word depends on PKC_CTRL.REDMUL.

NOTE

There may happen a final reduction step at the end of the calculation that can be used for timing attacks. Refer to the PKC application note how to prevent side channel attacks based on this effect.

In case the operands X and Y have been transformed to Montgomery Space, also the result R is in Montgomery Space (represented by []):

$[X] \cdot [Y] \bmod Z = X \cdot Q \cdot Y \cdot Q \cdot Q^{-1} \bmod Z = R \cdot Q \bmod Z = [R]$; with $Q = 2^{MCLEN \cdot wordsize}$

This is only valid for MCLEN==LEN. If MCLEN ≠ LEN special care has to be taken. Refer to the PKC application note in this case.

Table 380. Plain Multiplication

Plain multiplication of a MPI X with a MPI Y		
R = X · Y	Start-Addr = 0x13	Symbol: PM / PM_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	

Table continues on the next page...

Table 380. Plain Multiplication (continued)

Plain multiplication of a MPI X with a MPI Y		
R = X · Y	Start-Addr = 0x13	Symbol: PM / PM_GF2
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
Result	RLEN = XLEN+YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR.	
Updated status flags[2]	CARRY=0, ZERO	
GF2 conversion	Plain Multiplication in GF(2) / PM_GF2	
Execution time	4·XLEN·(YLEN+1) PKC clock cycle;	

MC code	`PlainMul :  ConfLoad (0,XYZR)`
	`  Execute(0x00)`
	`PlainMul_Loop :  DecrTBNZ(PlainMul_Cont )`
	`PlainMul_Exit :  Exit()`
	`PlainMul_Cont :  InDecPtr(1,X)`
	`  InDecPtr (1,R)`
	`  ConfLoad (0,XYRR)`
	`PlainMac_Entry :  Execute(0x22 )`
	`  Jump()`
	`  (PlainMul_Loop)`	

1. In case XLEN>1 also YLEN needs to be >1 (XLEN and YLEN can still have different values). This can be achieved, e.g., by extending Y with another PKC word set to zero ($Y_1 = 0$) resulting in YLEN=2.
2. ZERO=1 indicates only a zero result in the last L0 calculation, resp. the last MACC (Execute(0x22)) for the most significant X word. In case XLEN>1 it can happen that ZERO is set even though the complete result R is unequal zero. To verify if for XLEN>1 the complete result R is zero a succeeding CMP calculation (op-code 0x4B) could be used.

Table 381. Plain Multiplication with Addition

Plain multiplication of a MPI X with a MPI Y and addition of a MPI Z. MPI Y and MPI Z must have equal length.		
R = X · Y + Z	Start-Addr = 0x1D	Symbol: PMA / PMA_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0

Table continues on the next page...

Table 381. Plain Multiplication with Addition (continued)

Plain multiplication of a MPI X with a MPI Y and addition of a MPI Z. MPI Y and MPI Z must have equal length.		
R = X · Y + Z	Start-Addr = 0x1D	Symbol: PMA / PMA_GF2
	LEN	Length of operand Y and Z in bytes
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = XLEN+YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR. Always possible for RPTR == ZPTR.	
Updated status flags[2]	CARRY=0, ZERO	
GF2 conversion	Plain Multiplication with Addition in GF(2) / PMA_GF2	
Execution time	4·XLEN·(YLEN+1) PKC clock cycle;	

MC code	`PlainMac :  ConfLoad(0,XYZR)`
	`Execute(0x2D) // to clear carry bit`
	`Jump()`
	`(PlainMac_Entry)`	

1. In case XLEN>1 also YLEN needs to be >1 (XLEN and YLEN can still have different values). This can be achieved, e.g., by extending Y with another PKC word set to zero ($Y_1 = 0$) resulting in YLEN=2.
2. ZERO=1 indicates only a zero result in the last L0 calculation, resp. the last MACC (Execute(0x22)) for the most significant X word. In case XLEN>1 it can happen that ZERO is set even though the complete result R is unequal zero. To verify if for XLEN>1 the complete result R is zero a succeeding CMP calculation (op-code 0x4B) could be used.

Table 382. Modular Addition

Modular Addition of a MPI Y with a MPI Z modulo a MPI X. MPI X, MPI Y and MPI Z must have equal length.		
R = Y + Z mod X	Start-Addr = 0x21	Symbol: MA
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand X, Y and Z in bytes
	MCLEN	don't care
Operands[1]	XLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = XLEN $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = XLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result[2]	RLEN = XLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR. Not possible for RPTR == XPTR.	
Updated status flags	CARRY=0, ZERO	
GF2 conversion	not supported --> behaves like L0 calculation with op-code 0x0f - logical xor / XOR	
Execution time	4·(YLEN+3 + n·(YLEN+3)) PKC clock cycle; with n equals amount of reduction loops (n=0 in case Y+Z < $2^{YLEN \cdot wordsize}$)	

MC code	`ModAdd :  ConfLoad(0,XYZR)`
	`Execute(0x0A)`
	`ConfLoad(0,XRXR)`
	`CondJump(C=0)`
	`(ModAdd_Exit)`
	`ModAdd_Sub :  Execute(0x2B)`
	`CondJump(C=1)`	

Table continues on the next page...

Table 382. Modular Addition (continued)

Modular Addition of a MPI Y with a MPI Z modulo a MPI X. MPI X, MPI Y and MPI Z must have equal length.		
R = Y + Z mod X	Start-Addr = 0x21	Symbol: MA
	(ModAdd_Sub) ModAdd_Exit : Exit()	

1. Carry overflow during addition is corrected by subtracting the modulus X from the result. If X is small a lot of subtraction loops might be required. X should be prepared accordingly, e.g. shifted so that the msb of X is '1'.
2. The result R might be bigger than the modulus (R ≥ X). The reduction only guarantees that the result fits to the word-size of the modulus. → $R < 2^{YLEN \cdot wordsize}$

Table 383. Modular Subtraction

Modular Subtraction of a MPI Y by a MPI Z modulo a MPI X. MPI X, MPI Y and MPI Z must have equal length.		
R = Y - Z mod X	Start-Addr = 0x2A	Symbol: MS
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand X, Y and Z in bytes
	MCLEN	don't care
Operands[1]	XLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = XLEN $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = XLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result[2]	RLEN = XLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Always possible for RPTR == YPTR and RPTR == ZPTR. Not possible for RPTR == XPTR.	
Updated status flags	CARRY=0, ZERO	

Table continues on the next page...

Table 383. Modular Subtraction (continued)

Modular Subtraction of a MPI Y by a MPI Z modulo a MPI X. MPI X, MPI Y and MPI Z must have equal length.		
R = Y - Z mod X	Start-Addr = 0x2A	Symbol: MS
GF2 conversion	not supported --> behaves like L0 calculation with op-code 0x0f - logical xor / XOR	
Execution time	4·(YLEN+3 + n·(YLEN+3)) PKC clock cycle; with n equals amount of reduction loops (n=0 in case Y-Z ≥ 0)	

MC Code	`ModSub : ConfLoad(0,XYZR)`
	`Execute(0x0B)`
	`ConfLoad(0,XRXR)`
	`CondJump(C=0)`
	`(ModSub_Exit)`
	`ModSub_Add : Execute(0x2A)`
	`CondJump(C=1)`
	`(ModSub_Add)`
	`ModSub_Exit : Exit()`	

1. Carry overflow in subtraction is reduced by adding the modulus X to the result. If X is small a lot of addition loops might be required. X should be prepared accordingly, e.g. shifted so that the msb of X is '1'.
2. In case the operand Y Is bigger than X and Z the result R might be bigger than the modulus (R ≥ X). If Y is smaller than X or Z the result is smaller than the modulus X.

Table 384. Modular Reduction

Modular Reduction of a MPI X and a MPI Z. Montgomery algorithm is used for reduction.		
R = X mod Z	Start-Addr = 0x33	Symbol: MR / MR_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0 Multiplicative Inverse ~Z has to be stored in the PKC word directly before Z_0.
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Z in bytes
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	ZLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
	$\sim Z = (-Z)^{-1} \bmod 2^{wordsize}$; wordsize = 64 for REDMUL == 0x0 or 0x2	
Result[2]	RLEN = ZLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	

Table continues on the next page...

Table 384. Modular Reduction (continued)

Modular Reduction of a MPI X and a MPI Z. Montgomery algorithm is used for reduction.		
R = X mod Z	Start-Addr = 0x33	Symbol: MR / MR_GF2
Result space	RLEN+1	
Result-in-place	Not possible for RPTR == XPTR and RTPR == ZPTR.	
Updated status flags	CARRY=0, ZERO	
GF2 conversion	Modular Reduction in GF(2) / MR_GF2	
Execution time	4·(XLEN·(2·ZLEN+6) + MAX(ZLEN+1, 4)) PKC clock cycle;	

MC Code	```
	ModRed : ConfLoad(1,XYZR)
	  Execute(0x3E)
	  PreLoadT(0)
	  (T = R)
	  PreLoadT(0)
	  (T = T + LEN)
	  PreLoadT(0)
	  (Y = T)
	  PreLoadT(1)
	  (T = 1)
	ModRed_Loop : PreLoadT(0)
	  (T = T + LEN; LEN = T)
	  PreLoadT(0)
	  (T <--> S)
	  ConfLoad(1,XYXY)
	  Execute(0x3E)
	  PreLoadT(0)
	  (T <--> S)
	  PreLoadT(0)
	  (T <--> LEN)
	  ConfLoad(0,XRXR)
	  Execute(0x6A)
	  InDecPtr(0,LEN)
	  InDecPtr(0,Z)
	  ConfLoad(0,ZRZR)
	  Execute(0x20)
	  InDecPtr(1,Z)
	  ConfLoad(0,XZRR)
	  Execute(0x62)
	  InDecPtr(1,X)
	  DecrTBNZ(ModRed_Loop)
	  Exit()
	``	

1. In case XLEN>1 and YLEN=1 the least-significant PKC word of the result R is read before it is written the very first time. This can lead to reading uninitialized memory. If this is an issue the least-significant PKC word of the result R needs to be initialized upfront before calling the operation.
2. The reduction ensures that the result is smaller or equal to the modulus (i.e., R ≤ Z).

   The length of the result is equal RLEN(=ZLEN) but for the calculation one extra PKC word on top of the result R is required. The width of the word depends on PKC_CTRL.REDMUL.

NOTE

MontRed(X) = MM(X,1) = $X \cdot Q^{-1} \bmod Z$; with $Q=2^{XLEN \cdot wordsize}$

To obtain the result in normal representation (without the Q^{-1} term), a Modular Montgomery multiplication with Q^2 is required

→ $X \bmod Z = MM(MontRed(X), Q^2 \bmod Z)$

Because Q^2 is typically only known for the $Q=2^{YLEN \cdot wordsize}$ the length of the X operand should be equal ZLEN or be adapted accordingly (refer to PKC application note).

Table 385. Modular Multiplication Modulo Power of 2

Modular multiplication of a MPI X with a MPI Y modulo 2^{YLEN}.		
$R = X \cdot Y \bmod 2^{YLEN}$	Start-Addr = 0x53	Symbol: MMP2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
Result[2]	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN+1	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR.	
Updated status flags[3]	CARRY=0, ZERO	
GF2 conversion	not supported	
Execution time	$4 \cdot XLEN \cdot (2 \cdot YLEN - XLEN + 3)/2$ PKC clock cycle;	

MC code	`ModMulP2 : Execute(0x00)`
	`ModMulP2_Loop : DecrTBNZ(ModMulP2_Cont)`
	`Exit()`
	`ModMulP2_Cont : InDecPtr(1,X)`
	`InDecPtr(1,R)`
	`InDecPtr(0,LEN)`
	`ConfLoad(0,XYRR)`	

Table continues on the next page...

Table 385. Modular Multiplication Modulo Power of 2 (continued)

Modular multiplication of a MPI X with a MPI Y modulo 2^{YLEN}.		
$R = X \cdot Y \bmod 2^{YLEN}$	Start-Addr = 0x53	Symbol: MMP2

	`ModMulAddP2_Entry : Execute(0x22)`
	`Jump()`
	`(ModMulP2_Loop)`	

1. This micro code program only works for YLEN ≥ XLEN.
2. The length of the result is equal RLEN(=YLEN) but for the calculation one extra PKC word on top of the result R is required. The width of the word depends on PKC_CTRL.REDMUL.
3. ZERO=1 indicates only a zero result in the last L0 calculation, resp. the last MACC(Execute(0x22)) for the most significant X word. In case XLEN>1 it can happen that ZERO is set even though the complete result R is unequal zero. To verify if for XLEN>1 the complete result is zero a succeeding CMP calculation (op-code 0x4B) could be used.

Table 386. Modular Multiplication with Addition Modulo Power of 2

Modular multiplication with addition of a MPI X with a MPI Y and a MPI Z modulo 2^{YLEN}. MPI Y and MPI Z must have equal length.		
$R = X \cdot Y + Z \bmod 2^{YLEN}$	Start-Addr = 0x5A	Symbol: MMAP2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y and Z in bytes (≥ MCLEN)
	MCLEN	Length of operand X in bytes
Operands[1]	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = YLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result[2]	RLEN = YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN+1	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR. Always possible for RPTR == ZPTR.	

Table continues on the next page...

Table 386. Modular Multiplication with Addition Modulo Power of 2 (continued)

Modular multiplication with addition of a MPI X with a MPI Y and a MPI Z modulo 2^{YLEN}. MPI Y and MPI Z must have equal length.		
R = X · Y + Z mod 2^{YLEN}	Start-Addr = 0x5A	Symbol: MMAP2
Updated status flags[3]	CARRY=0, ZERO	
GF2 conversion	not supported	
Execution time	4·XLEN·(2·YLEN-XLEN+3)/2 PKC clock cycle;	

MC code	`ModMulP2 : Execute(0x00)`
	`ModMulP2_Loop : DecrTBNZ(ModMulP2_Cont)`
	`Exit()`
	`ModMulP2_Cont : InDecPtr(1,X)`
	`InDecPtr(1,R)`
	`InDecPtr(0,LEN)`
	`ConfLoad(0,XYRR)`
	`ModMulAddP2_Entry : Execute(0x22)`
	`Jump()`
	`(ModMulP2_Loop)`	

1. This micro code program only works for YLEN ≥ XLEN.
2. The length of the result is equal RLEN(=YLEN) but for the calculation one extra PKC word on top of the result R is required. The width of the word depends on PKC_CTRL.REDMUL.
3. ZERO=1 indicates only a zero result in the last L0 calculation, resp. the last MACC(Execute(0x22)) for the most significant X word. In case XLEN>1 it can happen that ZERO is set even though the complete result R is unequal zero. To verify if for XLEN>1 the complete result is zero a succeeding CMP calculation (op-code 0x4B) could be used.

Table 387. Modular Inversion

Modular Inversion of a MPI Y modulo an MPI X. Almost Montgomery inversion algorithm used.		
R = $Y^{-1} \cdot 2^k$ mod X	Start-Addr = 0x5D	Symbol: MI / MI_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0 MPI X gets overwritten
	YPTR	Pointer to MPI operand Y: Byte address of Y_0 MPI Y gets overwritten
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0 MPI Z needs to be initialized to 1
	RPTR	Pointer to result area R: Byte address of R_0 k is stored in the PKC word directly before R_0 {MPI R, k} needs to be initialized to 0

Table continues on the next page...

Table 387. Modular Inversion (continued)

Modular Inversion of a MPI Y modulo an MPI X. Almost Montgomery inversion algorithm used.		
$R = Y^{-1} \cdot 2^k \bmod X$	Start-Addr = 0x5D	Symbol: MI / MI_GF2
	LEN	Length of operand X, Y and Z in bytes
	MCLEN	32 << 3 (REDMUL == 0x0 or 0x2) bytes
Operands	XLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = XLEN $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = XLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = XLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
	kLEN = 1 PKC word $k = k_0$	
Result space	RLEN+kLEN	
Result-in-place	No	
Updated status flags	CARRY=0, ZERO	
GF2 conversion	Modular Inversion in GF(2) / MI_GF2	
Execution time	data dependent	

NOTE

1. The following input relation has to be fulfilled: MPI Z = 1, {MPI R, k} = 0, MCLEN = 31 PKC words (the actual value depends on PKC_CTRL.REDMUL).
2. Output relation of this operation: MPI X and MPI Y get overwritten.

Table 388. Plain Multiplication (Patched version)

Plain multiplication of a MPI X with a MPI Y		
$R = X \cdot Y$	Start-Addr = 0x9D	Symbol: PM_PATCH / PM_PATCH_GF2
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0

Table continues on the next page...

Table 388. Plain Multiplication (Patched version) (continued)

Plain multiplication of a MPI X with a MPI Y		
R = X · Y	Start-Addr = 0x9D	Symbol: PM_PATCH / PM_PATCH_GF2
	YPTR	Pointer to MPI operand Y: Byte address of Y_0
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand Y in bytes
	MCLEN	Length of operand X in bytes
Operands	XLEN = MCLEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
Result	RLEN = XLEN+YLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Only possible if MCLEN == 1 PKC word for RPTR == XPTR and RPTR == YPTR.	
Updated status flags[1]	CARRY=0, ZERO	
GF2 conversion	Plain Multiplication in GF(2) / PM_PATCH_GF2	
Execution time	4·XLEN·(YLEN+2) PKC clock cycle;	

MC code	`PlainMulPatch :  Execute(0x00)`
	`PlainMulPatch_Loop :  DecrTBNZ(PlainMulPatch_Cont)`
	`PlainMulPatch_Exit :  Exit()`
	`PlainMulPatch_Cont :  InDecPtr(1,X)`
	`InDecPtr (1,R)`
	`ConfLoad (0,XYRR)`
	`PlainMacPatch :  Execute(0x20)`
	`Execute(0x22)`
	`Jump()`
	`(PlainMulPatch_Loop)`	

1. ZERO=1 indicates only a zero result in the last L0 calculation, resp. the last MACC (Execute(0x22)) for the most significant X word. In case XLEN>1 it can happen that ZERO is set even though the complete result R is unequal zero. To verify if for XLEN>1 the complete result R is zero a succeeding CMP calculation (op-code 0x4B) could be used.

Table 389. Greatest Common Divider

Greatest Common Divider (GCD) of MPI Y and MPI Z		
R = GCD(Y, Z)	Start-Addr = 0xA7	Symbol: GCD
Parameter	XPTR	Pointer to MPI operand X: Byte address of X_0 (needs to point to same address as YPTR)
	YPTR	Pointer to MPI operand Y: Byte address of Y_0 MPI Y gets overwritten
	ZPTR	Pointer to MPI operand Z: Byte address of Z_0 (needs to point to same address as RPTR) MPI Z gets overwritten
	RPTR	Pointer to result area R: Byte address of R_0
	LEN	Length of operand X, Y and Z in bytes
	MCLEN	don't care
Operands	XLEN = LEN >> 3 (REDMUL == 0x0 or 0x2) $X = X_{XLEN-1} \cdot b^{XLEN-1} + X_{XLEN-2} \cdot b^{XLEN-2} + ... + X_1 \cdot b + X_0$	
	YLEN = XLEN $Y = Y_{YLEN-1} \cdot b^{YLEN-1} + Y_{YLEN-2} \cdot b^{YLEN-2} + ... + Y_1 \cdot b + Y_0$	
	ZLEN = XLEN $Z = Z_{ZLEN-1} \cdot b^{ZLEN-1} + Z_{ZLEN-2} \cdot b^{ZLEN-2} + ... + Z_1 \cdot b + Z_0$	
Result	RLEN = XLEN $R = R_{RLEN-1} \cdot b^{RLEN-1} + R_{RLEN-2} \cdot b^{RLEN-2} + ... + R_1 \cdot b + R_0$	
Result space	RLEN	
Result-in-place	Yes (since RPTR == ZPTR).	
Updated status flags	CARRY, ZERO=1	
GF2 conversion	not supported	
Execution time	data dependent	

NOTE

1. The following input relation has to be fulfilled: XPTR = YPTR and ZPTR = RPTR.
2. Output relation of this operation: MPI X = MPI Y = MPI Z = MPI R (i.e., MPI Y and MPI Z get overwritten).

16.4.3 Security

This section provides a brief summary of the security related aspects of PKC, general considerations, error conditions triggered by PKC and software-configurable countermeasures.

Error Conditions

The following types of error conditions trigger a reset of the PKC-CTRL and the PKC coprocessor kernel. The error that triggered this reset is latched in the PKC_ACCESS_ERR register and is only cleared by setting PKC_ACCESS_ERR_CLR[ERR_CLR] or via a global reset (not (PKC_ACCESS_ERR_CLR[PKC_SOFT_RST[SOFT_RST]). The single error bits are accumulated. If a new error occurs while the previous one is still indicated the new error flag is set on top. This holds for all bits in PKC_ACCESS_ERR.

Bit definition for block specific PKC_ACCESS_ERR fields:

- PKCC - Error in PKC coprocessor kernel (e.g. state machine or illegal calculation mode (could be triggered by SW)) or L0 bus errors
- FDET - Error in PKC control state machine
- CTRL - Error in GO handling (could also be triggered by SW)
- UCRC - Error in UP CRC check (could also be triggered by SW)
- BLK_ERR[7:4] - reserved

Bit definition for AHB specific PKC_ACCESS_ERR:

- AHB - AHB master error (AHB_AMBA_ERROR) - L2

Bit definition for APB specific PKC_ACCESS_ERR fields:

- APB_WRGMD - wrong access rights or wrong read-write information: Access not allowed in the current mode or write access to a read-only register or vice versa. APB interface returns zero on data read port for invalid read access.

  NOTE

  Reading from write-only SFRs will not trigger an error, instead zero on data read port is returned.

- APB_NOTAV - unavailable address: There is no SFR or memory located at the accessed address. APB interface returns zero on data read port for invalid read access.

The APB master that triggered the first APB_WRGMD/APB_NOTAV error is stored in the field PKC_ACCESS_ERR.APB_MASTER.

The PKC_ACCESS_ERR[APB_WRGMD]/[NOTAV]/[MASTER] fields are not updated in case a second APB error triggers but only for the first occurrences until cleared via PKC_ACCESS_ERR_CLR or a global reset.

All other bit fields of the PKC_ACCESS_ERR SFR are reserved, and always read zero.

The following events will trigger a security alert that resets the PKC including the PKC kernel and asserts the pkc_err signal. All error outputs listed as output in the following table stay high until cleared by reset (only POR and chip/main reset) or PKC_ACCESS_ERR_CLR[ERR_CLR].

Table 390. List of PKC security alerts

Security alert	PKC_ACCESS_ERR flag
SFR access	
SFR access to not implemented SFR	APB_NOTAV
SFR write access to read_only SFR	APB_WRGMD
DMA access	
System feedback error as response to an AHB access. L2 only	AHB

Table continues on the next page...

Table 390. List of PKC security alerts (continued)

Security alert	PKC_ACCESS_ERR flag
PKC kernel errors	
Executing PKC kernel operation with illegal calculation mode	PKCC
Executing PKC kernel operation with illegal operand length LEN (zero or too short for selected multiplier size)	PKCC
Executing DecrTBNZ on zero loop counter value (e.g. input via MCLEN)	PKCC
REDMUL setting selects multiplier size that is bigger than native multiplier size, e.g. REDMUL=11b for 64-bit native multiplier.	PKCC
Error detection	PKCC
Executing CMP calculation with YPTR equal ZPTR.	PKCC
Executing Reserved MC instruction in L1 calculation.	PKCC
PKC RAM parity error.	PKCC
PKC-CTRL errors	
Error detection	FDET
Set PKC_CTRL[GOx] while GOANY is set.	CTRL
Write access to parameter set 1 (PKC_MODE1, PKC_XYPTR1, PKC_ZRPTR1, PKC_LEN1) while PKC_STATUS.LOCKED[0] is set.	CTRL
Write access to parameter set 2 (PKC_MODE2, PKC_XYPTR2, PKC_ZRPTR2, PKC_LEN2) while bit 1 of PKC_STATUS[LOCKED is set.	CTRL
More than one GO bit is set for CTRL write.	CTRL
PKC_CTRL[GOx] is set while PKC_CTRL[RESET] is set. (PKC_CTRL[RESET] is required to be cleared before setting GOx. RESET cannot be cleared in the same write that sets a GOx bit).	CTRL
Write access to PKC_CFG while PKC_CTRL[RESET] is cleared.	CTRL
Write to PKC_LEN1 or PKC_LEN2 during L2 calculation while PKC_CTRL[GOANY] is set.	CTRL
Set PKC_CTRL[GOU] while PKC_ULEN[LEN] is equal zero.	CTRL
Error in CRC integrity check of L2 calculation (universal pointer fetch).	UCRC
PKC Stop Error, attempt to change the PKC_CTRL[STOP] bit before it is stable (see Handling of PKC_CTRL[RESET] and PKC_CTRL[STOP]).	CTRL

Table continues on the next page...

Table 390. List of PKC security alerts (continued)

Security alert	PKC_ACCESS_ERR flag
PKC Reset Error, attempt to change the PKC_CTRL.RESET bit before it is stable (see Handling of PKC_CTRL[RESET] and PKC_CTRL[STOP]).	CTRL

16.4.4 Interrupt, exception and DMA

Interrupts

The PKC Control provides a level triggered interrupt to indicate the end of a calculation. The high active interrupt request signal `pkc_int` is raised in case the interrupt is enabled (PKC_INT_ENABLE[EN_PDONE] is set due to previous PKC_INT_SET_ENABLE[EN_PDONE] SFR write) and one of the following conditions is triggered:

- end of a native calculation (L0)
- end of microcode calculation (L1)
- end of universal pointer calculation (L2)
- Interrupt triggered by SW via SFR PKC_INT_SET_STATUS[INT_PDONE], e.g. for test purposes

The interrupt request signal stays high unless cleared by reset (all resets except the PKC kernel reset (PKC_CTRL[RESET])) or via SW using SFR PKC_INT_CLR_STATUS[INT_PDONE].

PKC_INT_STATUS[INT_PDONE] will be set by HW or SW independently from PKC_INT_ENABLE[EN_PDONE]. PKC_INT_ENABLE[EN_PDONE] shall only gate the interrupt request signal that is connected to the interrupt controller.

```
pkc_int = PKC_INT_STATUS.INT_PDONE && PKC_INT_ENABLE.EN_DONE
```

Exceptions

The PKC_CTRL notifies the system block on any security alarm, resp. misbehavior of the PKC. The system decides how to react, e.g. trigger global reset, raise a CPU exception or ignore. The possible security alarms are listed in table Table 390

DMA / Memory Interfaces

The PKC Control provides an AHB (L2) master port for the DMA access of the Universal Pointer Fetch (read-only). Typically two windows for two different memories (e.g. RAM and NV) are accessed during a PKC universal pointer fetch calculation, one for the program code and one for the calculation parameters.

The PKC kernel requires a direct and fast DMA access to the accessible RAM (read-write) via RAM-IF (L0).

16.4.5 Clocking

The IP is a fully synchronous design.

The PKC Control supports an APB clock (port name `clk_apb`) and a functional clock for the kernel (port name `clk_pkc`).

16.4.6 Reset

There is a single main hard reset connected from the system to PKC-CTRL

- POR/System reset (rst_main_sys_n)

POR reset is a Power-On-Reset and asserted once by the system during the start up. System reset is asserted when the system decides to reset the PKC-CTRL and PKC (usually due to a security violation). This resets PKC-CTRL and PKC completely.

There are two soft resets defined in the SFR Interface:

- PKC_SOFT_RST[SOFT_RST], which resets the PKC-CTRL including all sub-blocks (except SFR PKC_ACCESS_ERR),

- PKC_CTRL[RESET], which resets only the PKC kernel (incl. the internal carry (c) and result register Reg_i that are e.g. used as input for opcode 0x62 (R=(Reg_i * Y + {c,Z}) >> 64)).

In case of a security alarm the PKC_CTRL including all sub-blocks are reset as well. The behavior is the same as for the SOFT_RST. The error indicating SFR PKC_ACCESS_ERR is not reset. The security reset releases also the error source such that the PKC_CTRL is again functional after the reset was triggered.

Table 391. Resets and impact for PKC-CTRL and kernel

Reset type	Reset Signal	Impact on PKC_CTRL	Impact on PKC kernel
Global system reset	rst_main_sys_n	Full block is reset.	Kernel is reset (Note 1), due to reset of PKC-CTRL.RST, clk_pkc is stopped
Soft reset SW (PKC_SOFT_RST)	N/A	Reset full block except PKC_ACCESS_ERR SFR.	Kernel is reset, due to reset of PKC-CTRL.RST, clk_pkc is stopped
Security alarm reset	pkc_err	Reset full block except PKC_ACCESS_ERR SFR.	Kernel is reset, due to reset of PKC-CTRL.RST, clk_pkc is stopped
Kernel reset (PKC_CTRL[RESET])	N/A	Running PKC calculation is interrupted. Read-only PKC_STATUS SFR is reset.	Kernel is reset, clk_pkc is stopped

Note 1: Uninitialized registers are discharged to their intrinsic value in case power off was long enough, influenced by environment (PVT conditions).

Note 2: For safely interrupting any ongoing operation (i.e., ensuring all pending memory requests have been completed) and completely resetting the IP afterwards the following approach needs to be used:

1. Ensure PKC kernel clock is enabled by setting PKC_CTRL[STOP]=0.
2. Poll value of PKC_CTRL[STOP] until it has taken over internally the value (i.e., reads as 0). This ensures kernel clock has been properly enabled by the system (important if PKC-CTRL and PKC kernel block run at different clock speeds).
3. Interrupt ongoing operation via PKC_CTRL[RESET]=1.
4. Poll value of PKC_CTRL[RESET] until it has taken over internally the value (i.e., reads as 1). This ensures all pending memory requests have been completed and PKC kernel has entered reset state (important if PKC-CTRL and PKC kernel block run at different clock speeds).
5. Perform soft reset via PKC_SOFT_RST[SOFT_RST]=1
6. Clear any pending access errors via PKC_ACCESS_ERR_CLR[ERR_CLR]=1

16.5 Handling of PKC_CTRL[RESET] and PKC_CTRL[STOP]

The PKC internal reset and STOP features are controlled via the PKC_CTRL[RESET] and PKC_CTRL[STOP] bits respectively.

Internally the PKC_CTRL[RESET] is implemented as a synchronous reset of the PKC kernel, therefore careful handling of this control bit and the PKC_CTRL[STOP] is required.

To ensure that PKC_CTRL[RESET] and PKC_CTRL[STOP] bit writes have been appropriately propagated to the PKC kernel and ongoing memory accesses have successfully completed, the software should poll the modified bit until the changed value is read back from the PKC_CTRL register.

This requirement is due to the PKC kernel being fed internally from a different clock source than the register interface.

It is important to note that enabling of the STOP feature while changing the PKC Reset is not supported, the STOP feature should first be disabled before changing the PKC_CTRL[RESET] state.

Table Table 392 lists Invalid writes to the PKC_CTRL register modifying the RESET and STOP bit together which should be avoided.

Table 392. Invalid PKC_CTRL writes

Current State	Updated State
PKC_CTRL[RESET] = 1 PKC_CTR[STOP] = 1	PKC_CTRL[RESET] = 0 PKC_CTR[STOP] = 1
PKC_CTRL[RESET] = 0 PKC_CTR[STOP] = 1	PKC_CTRL[RESET] = 1 PKC_CTR[STOP] = 1
PKC_CTRL[RESET] = 1 PKC_CTR[STOP] = 0	PKC_CTRL[RESET] = 0 PKC_CTR[STOP] = 1

The PKC also ensures ongoing memory accesses at its L2 AHB and L0 RAM interfaces are completed before accepting a kernel reset (PKC_CTRL[RESET]=1) or clock-stop (PKC_CTRL[STOP]=1) request:

- Setting PKC_CTRL[RESET] flag: PKC will stop sending new read/write requests via its L2 AHB and L0 RAM interfaces and wait until any ongoing request has completed (i.e, wait until last AHB request has been acknowledged by AHB L2 slave via corresponding hready_l2=1).
- Setting PKC_CTRL[STOP] flag: PKC will stop sending new read/write request via its L0 RAM interface and wait until any ongoing request has completed.

16.6 PKC register descriptions

16.6.1 PKC memory map

PKC0.PKC base address: 4002_B000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Status Register (PKC_STATUS)	32	R	0000_0000h
4h	Control Register (PKC_CTRL)	32	RW	0000_0001h
8h	Configuration register (PKC_CFG)	32	RW	0000_0719h
10h	Mode register, parameter set 1 (PKC_MODE1)	32	RW	0000_0000h
14h	X+Y pointer register, parameter set 1 (PKC_XYPTR1)	32	RW	0000_0000h
18h	Z+R pointer register, parameter set 1 (PKC_ZRPTR1)	32	RW	0000_0000h
1Ch	Length register, parameter set 1 (PKC_LEN1)	32	RW	0000_0000h
20h	Mode register, parameter set 2 (PKC_MODE2)	32	RW	0000_0000h
24h	X+Y pointer register, parameter set 2 (PKC_XYPTR2)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
28h	Z+R pointer register, parameter set 2 (PKC_ZRPTR2)	32	RW	0000_0000h
2Ch	Length register, parameter set 2 (PKC_LEN2)	32	RW	0000_0000h
40h	Universal pointer FUP program (PKC_UPTR)	32	RW	0000_0000h
44h	Universal pointer FUP table (PKC_UPTRT)	32	RW	0000_0000h
48h	Universal pointer length (PKC_ULEN)	32	RW	0000_0000h
50h	MC pattern data interface (PKC_MCDATA)	32	RW	0000_0000h
60h	PKC version register (PKC_VERSION)	32	R	0000_00BEh
FB0h	Software reset (PKC_SOFT_RST)	32	RW	0000_0000h
FC0h	Access Error (PKC_ACCESS_ERR)	32	R	0000_0000h
FC4h	Clear Access Error (PKC_ACCESS_ERR_CLR)	32	RW	0000_0000h
FD8h	Interrupt enable clear (PKC_INT_CLR_ENABLE)	32	RW	0000_0000h
FDCh	Interrupt enable set (PKC_INT_SET_ENABLE)	32	RW	0000_0000h
FE0h	Interrupt status (PKC_INT_STATUS)	32	R	0000_0000h
FE4h	Interrupt enable (PKC_INT_ENABLE)	32	R	0000_0000h
FE8h	Interrupt status clear (PKC_INT_CLR_STATUS)	32	RW	0000_0000h
FECh	Interrupt status set (PKC_INT_SET_STATUS)	32	RW	0000_0000h
FFCh	Module ID (PKC_MODULE_ID)	32	R	E103_1280h

16.6.2 Status Register (PKC_STATUS)

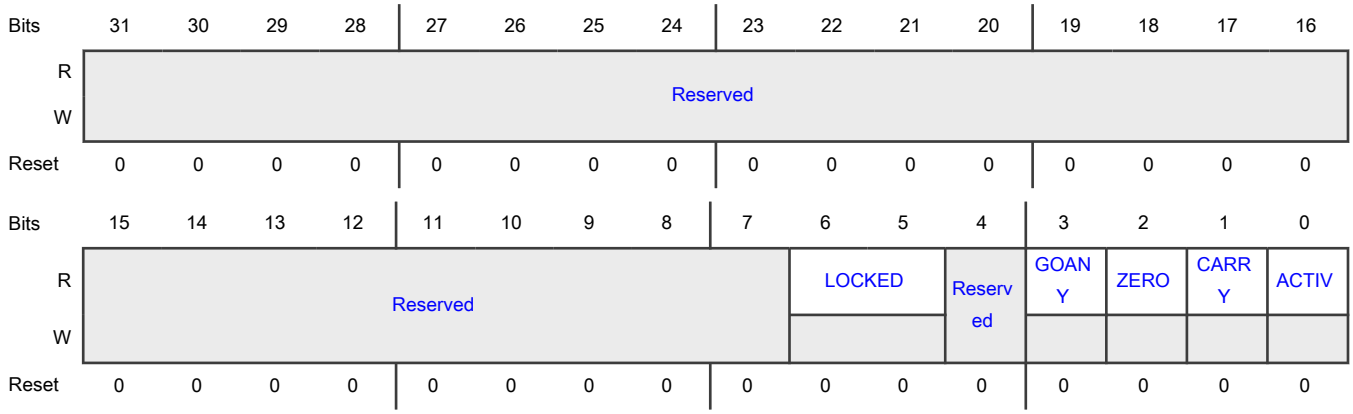
Offset

Register	Offset
PKC_STATUS	0h

Function

Contains PKC primary status information.

Diagram



Fields

Field	Function
31-7 —	Reserved
6-5 LOCKED	Parameter set locked Indicates if parameter set is locked due to a pending calculation start or can be overwritten. x1b- parameter set 1 is locked 1xb- parameter set 2 is locked LOCKED[0] is set in case a PKC calculation is started with parameter set 1 via PKC_CTRL[GOD1] or PKC_CTRL[GOM1]. In case one of the parameter set 1 SFRs PKC_MODE1, PKC_XYPTR1, PKC_ZRPTR1, PKC_LEN1 is updated while LOCKED[0] is set a PKC security alarm will be triggered (PKC_ACCESS_ERR[CTRL] is set). LOCKED[1] is set in case a PKC calculation is started with parameter set 2 via PKC_CTRL[GOD2] or PKC_CTRL[GOM2]. In case one of the parameter set 2 SFRs PKC_MODE2, PKC_XYPTR2, PKC_ZRPTR2, PKC_LEN2 is updated while LOCKED[1] is set a PKC security alarm will be triggered (PKC_ACCESS_ERR[CTRL] is set). Bits 1:0 are not set for a universal pointer fetch (layer2) PKC calculation started via PKC_CTRL[GOU]. All parameter set SFRs can be updated during a universal pointer fetch operation, except PKC_LEN1 and PKC_LEN2. Updating PKC_LEN1 or PKC_LEN2 during a layer2 calculation indicated by PKC_STATUS[GOANY] will trigger a PKC security alarm (PKC_ACCESS_ERR[CTRL] is set). Bits 1:0 are cleared with the start of the calculation in parallel with the 1-to-0 transition of PKC_STATUS[GOANY] and in case PKC_CTRL[RESET] is set.
4 —	Reserved
3 GOANY	Combined GO status flag Set in case either PKC_CTRL[GOD1], [GOD2], [GOM1], [GOM2] or [GOU] is set. The 1-to-0 transition of GOANY indicates that a calculation has been started and that a new GO bit can be set. If GOANY is cleared also all PKC_STATUS[LOCKED] bits are cleared to indicate that the parameter set can be updated. GOANY is always '0' in case PKC_CTRL.RESET is set.

Table continues on the next page...

Table continued from the previous page...

Field	Function
2 ZERO	Zero result flag Set by the PKC at the end of a calculation in case the result of the calculation is equal zero. ZERO is updated for each PKC calculation mode, except for MUL1 (opcode 0x20) and MUL1_GF2 (opcode 0x24). **NOTE** A set ZERO flag only indicates that the result written to PKC RAM is zero, but does not imply that there is no overflow (i.e., CARRY flag set). If PKC is in reset (PKC_CTRL.RESET=1), ZERO is set to logic 0 (default).
1 CARRY	Carry overflow flag Set by the PKC at the end of a calculation in case • an addition or multiplication with addition operation has been executed and an overflow in the most significant bit has occurred. • a subtraction or multiplication with subtraction operation has been executed with negative result in twos complement notation. • a shift or rotate operation has been executed (most/least significant shifted/rotated bit is stored in CARRY flag). • a LSB0s or MSB0s operation has been executed (CARRY indicates zero least/most-significant zero bits). • a compare operation have been executed with Z>Y. CARRY is updated for each PKC calculation mode, except for MUL1 (opcode 0x20) and MUL1_GF2 (opcode 0x24). This holds also for operation modes where CARRY is always zero, e.g. logical operations (AND, OR, ...). If PKC is in reset (PKC_CTRL.RESET=1), CARRY is set to logic 0 (default).
0 ACTIV	PKC ACTIV ACTIV = 1 signals that a calculation is in progress or about to start. At the end of a calculation, ACTIV is automatically reset to logic 0 in case no further GO bit is set. If the next PKC operation has been started by setting a GO bit during a calculation, ACTIV remains high. ACTIV is always '1' in case PKC_STATUS[GOANY] is set. ACTIV is always '0' in case PKC_CTRL_RESET register is set. While ACTIV is '1' the PKC is active also in using the PKC RAM. Concurrent accesses to PKC RAM via the CPU is still possible, but may reduce the performance of the PKC calculation. Write accesses to the operand and result area defined by the PKC pointer and length registers is prohibited during a running calculation and may result in wrong calculation results.

16.6.3 Control Register (PKC_CTRL)

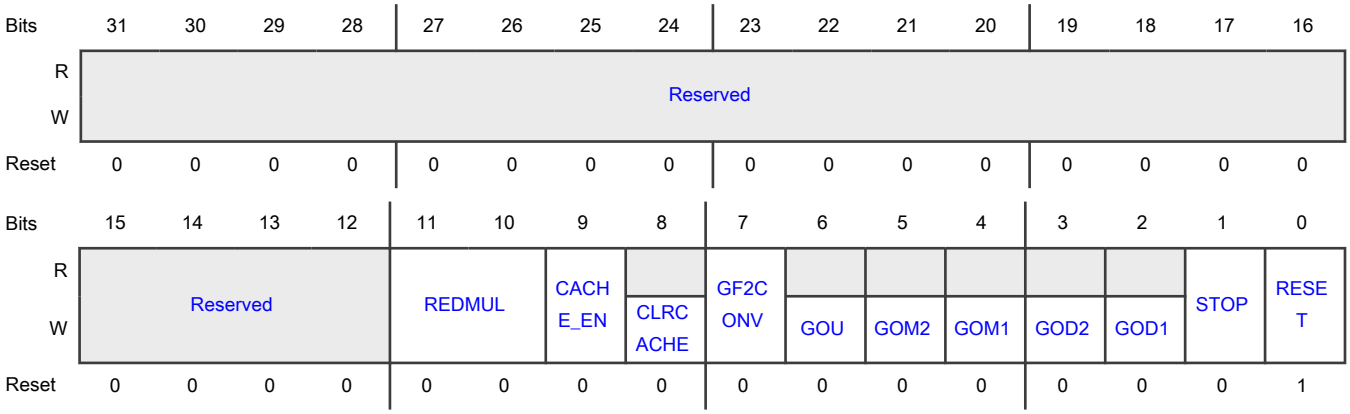
Offset

Register	Offset
PKC_CTRL	4h

Function

Controls primary operation of ELS.

Diagram



Fields

Field	Function
31-12 —	Reserved
11-10 REDMUL	Reduced multiplier mode Defines the operand width processed by the PKC coprocessor. This IP version only supports 64-bit operand width, i.e., REDMUL = 0x0 or 0x2. **NOTE** Although operand width is fixed the following points need to be considered: For hard coded layer1 calculation modular multiplication and modular reduction the pre-computed inverse of the module depends on the PKC-word size setting and as such is not the same for different REDMUL configurations. The operand pointers and length definitions are independent from the REDMUL setting as long as they are 64-bit aligned. The amount of ignored least significant bits of the pointer and length SFRs depends on REDMUL. As a consequence, also the minimum supported operand length is defined via bits 1 and 2 of PKC_LEN[LEN] and bits 1,2 if PKC_LEN[MCLEN] are affected by REDMUL. 00b - full size mode, 3 least significant bits of pointer and length are ignored, minimum supported length 0x0008 01b - Reserved - Error Generated if selected 10b - 64-bit mode, 3 least significant bits of pointer and length are ignored, minimum supported length 0x0008 11b - Reserved - Error Generated if selected
9 CACHE_EN	Enable universal pointer cache If CACHE_EN=1, the cache for the universal pointer parameters is enabled. In case a parameter value is found in the cache (from a previous fetch) no DMA access is triggered. As such the amount of DMA accesses for the parameter fetch vary between 0 and 4.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	To further optimize the cache utilization not used parameters, e.g. XPTR for a plain addition (opcode 0x0A), could be defined equal to a used one (e.g. equal YPTR or RPTR) or a previously fetched parameter. The universal pointer cache can store up to 6 16-bit parameter values. In case the cache is full the entry that has not been used longest is overwritten by the newly fetched parameter. The cache can be enabled and disabled at any point in time, even during a running universal pointer fetch calculation.
8 CLRCACHE	Clear universal pointer cache Invalidates the cache such that all previously fetched parameters are withdrawn and have to be fetched again via DMA accesses. CLRCACHE can be triggered at any point in time, even during a running universal pointer fetch calculation. CLRCACHE is write-only and always read '0'. CLRCACHE has no impact in case the cache is disabled via PKC_CTRL[CACHE_EN]=0 or in case no layer2 calculations are executed. Beside CLRCACHE the cache is also invalidated when it is disabled (PKC_CTRL[CACHE_EN]=0) or in case of any write access to PKC_UPTRT.
7 GF2CONV	Convert to GF2 calculation modes If GF2CONV is set, operations are mapped to their GF(2) equivalent operation modes. GF2CONV influences all PKC operation modes layer0/1/2. GF2CONV can be updated at any point in time, even during a running calculation. However the change is only applied when a new calculation is started via PKC_CTRL.GO*.
6 GOU	Control bit to start pipe operation If GOU is set PKC will start the pipe / layer2 operation (parameter fetch and calculation) described in section 'PKC Universal Pointer Fetch Operation'. Section 'Operating the PKC via the Universal Pointer Fetch Unit (layer 2)' describes how to start layer2 operations. GOU is a write-only bit and always read '0'. It is only allowed to set GOU when PKC_STATUS[GOANY] and PKC_CTRL[RESET] are cleared, PKC_ULEN>0 and PKC_UPTR(T) point to valid addresses (defined by MMU). Otherwise a security alarm is triggered (PKC_ACCESS_ERR[CTRL] or PKC_ACCESS_ERR[AHB] is set). An alarm is also triggered in case more than one GO bit is set for the PKC_CTRL SFR write access (PKC_ACCESS_ERR[CTRL] is set).
5 GOM2	Control bit to start MC pattern using parameter set 2 If GOM2 is set PKC will start a MC pattern / layer1 operation using parameter set 2 (PKC_MODE2, PKC_XYPTR2, PKC_ZRPTR2, PKC_LEN2). Section Layer 1: Operating PKC through micro code interface (L1) describes how to start layer1 operations. GOM2 is a write-only bit and always read '0'. It is only allowed to set GOM2 when PKC_STATUS[GOANY] and PKC_CTRL[RESET] are cleared. Otherwise a security alarm is triggered (PKC_ACCESS_ERR[CTRL] is set). An alarm is also triggered in case more than one GO bit is set for the PKC_CTRL SFR write access (PKC_ACCESS_ERR[CTRL] is set).
4 GOM1	Control bit to start MC pattern using parameter set 1 If GOM1 is set PKC will start a MC pattern / layer1 operation using parameter set 1 (PKC_MODE1, PKC_XYPTR1, PKC_ZRPTR1, PKC_LEN1). Section 'Operating the PKC via the Micro Code interface (layer 1)' describes how to start layer1 operations.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	GOM1 is a write-only bit and always read '0'. It is only allowed to set GOM1 when PKC_STATUS[GOANY] and PKC_CTRL[RESET] are cleared. Otherwise a security alarm is triggered (PKC_ACCESS_ERR[CTRL] is set). An alarm is also triggered in case more than one GO bit is set for the PKC_CTRL SFR write access (PKC_ACCESS_ERR[CTRL] is set).
3 GOD2	Control bit to start direct operation using parameter set 2 If GOD2 is set PKC will start a direct / layer0 operation using parameter set 2 (PKC_MODE2, PKC_XYPTR2, PKC_ZRPTR2, PKC_LEN2). Section 'Operating the PKC via the Parameter Set interface (layer 0)' describes how to start layer0 operations. GOD2 is a write-only bit and always read '0'. It is only allowed to set GOD2 when PKC_STATUS.GOANY and PKC_CTRL.RESET are cleared. Otherwise a security alarm is triggered(PKC_ACCESS_ERR[CTRL] is set). An alarm is also triggered in case more than one GO bit is set for the PKC_CTRL SFR write access (PKC_ACCESS_ERR.CTRL is set).
2 GOD1	Control bit to start direct operation using parameter set 1 If GOD1 is set, PKC will start a direct / layer0 operation using parameter set 1 (PKC_MODE1, PKC_XYPTR1, PKC_ZRPTR1, PKC_LEN1). Section 'Operating the PKC via the Parameter Set interface (layer 0)' describes how to start layer0 operations. GOD1 is a write-only bit and always read '0'. It is only allowed to set GOD1 when PKC_STATUS[GOANY] and PKC_CTRL[RESET] are cleared. Otherwise a security alarm is triggered (PKC_ACCESS_ERR[CTRL] is set). An alarm is also triggered in case more than one GO bit is set for the PKC_CTRL SFR write access (PKC_ACCESS_ERR[CTRL] is set).
1 STOP	Freeze PKC calculation STOP=1 freezes all PKC activity incl. RAM accesses and reduces the PKC power consumption to its minimum. The difference compared to the reset of the PKC is that a stopped calculation can be continued when STOP is released (reset to '0') again. The status flags are not affected by the STOP control bit. **NOTE** When writing a new value to the STOP flag one should poll it afterwards until the new value is read back to ensure the change has been taken over properly internally.
0 RESET	PKC reset control bit RESET=1 enforces the PKC's reset state during which a calculation cannot be started and by which any ongoing calculation process is stopped. RESET can be set/cleared by the CPU in order to switch between PKC reset and calculation enable. RESET=1 is the default state after a chip reset. RESET=1 disables PKC activity and accordingly also provides its power-saving state. When RESET is set to '1' the read-only status flags PKC_STATUS.CARRY and PKC_STATUS[ZERO] are cleared. If required these flags have to be evaluated before RESET is set. The status bits PKC_STATUS[GOANY] and PKC_STATUS[LOCKED] are reset as well in case RESET is set. RESET does not have an impact on the content of any other PKC SFRs. The parameter set and universal pointer SFRs can be read and written also while RESET is set. Software must clear bit RESET to enable the PKC before it can start a calculation via one of the GO bits in PKC_CTRL. Any attempt to start a calculation via a GO bit while RESET=1 will trigger a PKC security alarm (PKC_ACCESS_ERR[CTRL] is set).

Table continues on the next page...

Table continued from the previous page...

Field	Function
	The PKC configuration setting in PKC_CFG can only be updated while RESET=1. A write access to PKC_CFG while RESET=0 will trigger a PKC security alarm (PKC_ACCESS_ERR[CTRL] is set). **NOTE** When writing a new value to the RESET flag one should poll it afterwards until the new value is read back to ensure the change has been taken over properly internally.

16.6.4 Configuration register (PKC_CFG)

Offset

Register	Offset
PKC_CFG	8h

Function

Configuration register.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved					FMULNOI...	ALPNOISE	SBXNOISE	RNDDLY			REDMULN...	CLKRND	RFU2	RFU1	IDLEOP
Reset	0	0	0	0	0	1	1	1	0	0	0	1	1	0	0	1

Fields

Field	Function
31-11 —	Reserved
10 FMULNOISE	Noise feature not available in this version (flag is don't care).
9 ALPNOISE	Noise feature not available in this version (flag is don't care).

Table continues on the next page...

Table continued from the previous page...

Field	Function
8 SBXNOISE	Noise feature not available in this version (flag is don't care).
7-5 RNDDLY	Random delay feature not available in this version (flag is don't care).
4 REDMULNOISE	Noise in reduced multiplier mode feature not available in this version (flag is don't care).
3 CLKRND	Clock randomization feature not available in this version (flag is don't care).
2 RFU2	RFU
1 RFU1	RFU
0 IDLEOP	Idle operation feature not available in this version (flag is don't care).

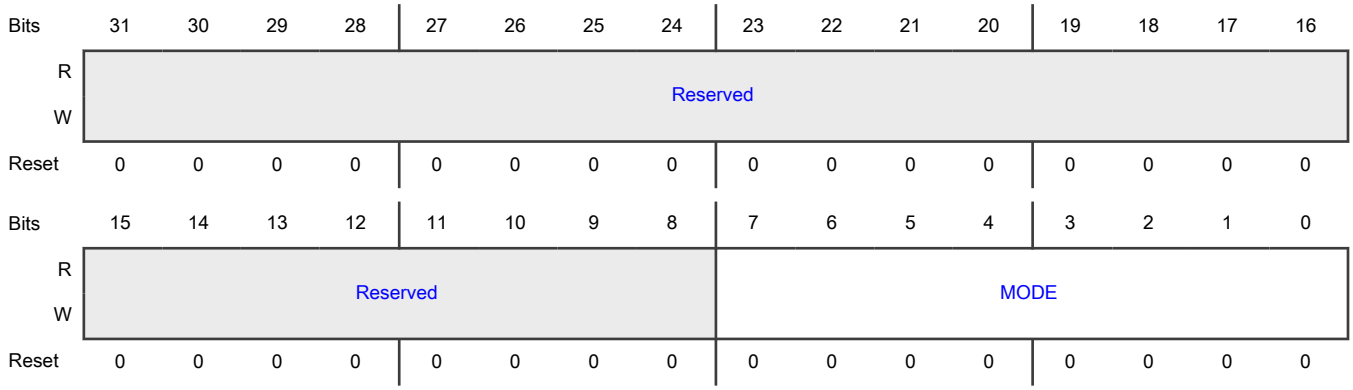
16.6.5 Mode register, parameter set 1 (PKC_MODE1)

Offset

Register	Offset
PKC_MODE1	10h

Function
Mode register, parameter set 1

Diagram



Fields

Field	Function
31-8 —	Reserved
7-0 MODE	Calculation Mode / MC Start address Calculation mode of direct calculation (layer0) are listed in a table in Section 'PKC arithmetic unit (layer 0)'. In case of a not supported calculation mode for layer0 calculation a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set). MC start addresses for hard coded MC pattern are listed in a table in Section 'Fixed Layer 1 MC Patterns'. The start address for the flexible MC pattern depends on the customized MC code previously stored using SFR PKC_MCDATA (data) and PKC_MODE1 (address). **NOTE** This IP version does not support flexible MC patterns (only hard coded ones). Accessing SFR PKC_MCDATA will trigger an error.

16.6.6 X+Y pointer register, parameter set 1 (PKC_XYPTR1)

Offset

Register	Offset
PKC_XYPTR1	14h

Function

X+Y pointer register, parameter set 1

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	YPTR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	XPTR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-16 YPTR	Start address of Y operand in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.
15-0 XPTR	Start address of X operand in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.

16.6.7 Z+R pointer register, parameter set 1 (PKC_ZRPTR1)

Offset

Register	Offset
PKC_ZRPTR1	18h

Function

Z+R pointer register, parameter set 1

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	RPTR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ZPTR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-16 RPTR	Start address of R result in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.
15-0 ZPTR	Start address of Z operand in PKCRAM with byte granularity or constant for calculation modes using CONST If ZPTR is used as address pointer the least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. If ZPTR is used as CONST operand the high byte is ignored and only ZPTR[7:0] is used for the calculation. For shift/rotate operation further most significant bits are ignored depending on PKC_CTRL[REDMUL].

16.6.8 Length register, parameter set 1 (PKC_LEN1)

Offset

Register	Offset
PKC_LEN1	1Ch

Function

Length register, parameter set 1

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	MCLEN															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	LEN															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-16 MCLEN	Loop counter for microcode pattern Defines the length of the loop counter that can be used in layer1 calculation mode, e.g. in MC opcode DecrTBNZ. For the hard coded MC patterns Modular Multiplication (MC start address 0x00), Plain

Table continues on the next page...

Table continued from the previous page...

Field	Function
	Multiplication (0x13), Plain Multiplication with Addition (0x1D) and Modular Reduction (0x33) MCLEN defines the length of the X operand in bytes. The least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. In case MCLEN is too short such that a DecrTBNZ MC opcode is executed on a zero loop counter value a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set).
15-0 LEN	Operand length Defines the length of the operands and the result in bytes. The length of Y, Z and R depend furthermore on the selected calculation mode. The least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. In case LEN is too short such that the resulting length depending on PKC_CTRL[REDMUL] is zero a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set).

16.6.9 Mode register, parameter set 2 (PKC_MODE2)

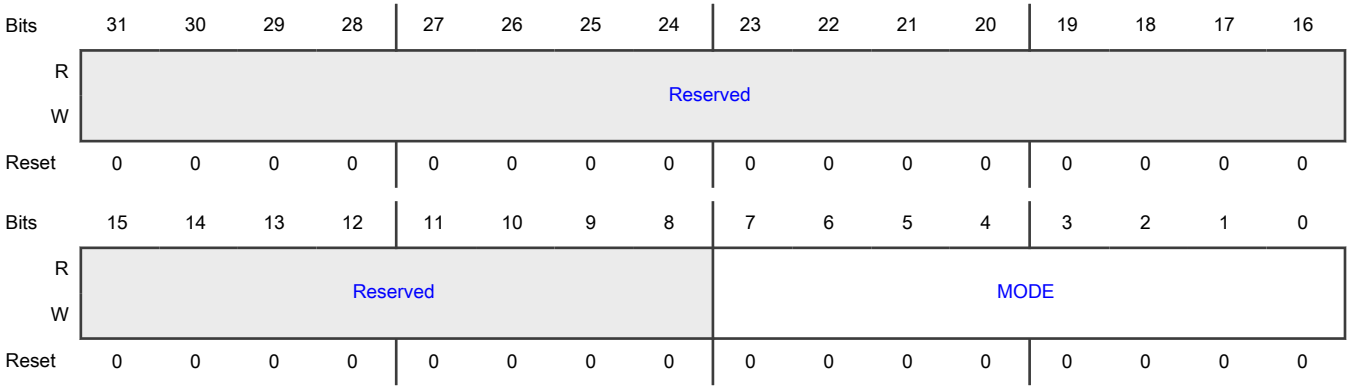
Offset

Register	Offset
PKC_MODE2	20h

Function

Mode register, parameter set 2

Diagram



Fields

Field	Function
31-8 —	Reserved
7-0 MODE	Calculation Mode / MC Start address Calculation mode of direct calculation (layer0) are listed in a table in Section 'PKC arithmetic unit (layer 0)'. In case of a not supported calculation mode for layer0 calculation a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set). MC start addresses for hard coded MC pattern are listed in a table in Section 'Fixed Layer 1 MC Patterns'. The start address for the flexible MC pattern depends on the customized MC code previously stored using SFR PKC_MCDATA (data) and PKC_MODE1 (address). **NOTE** This IP version does not support flexible MC patterns (only hard coded ones). Accessing SFR PKC_MCDATA will trigger an error.

16.6.10 X+Y pointer register, parameter set 2 (PKC_XYPTR2)

Offset

Register	Offset
PKC_XYPTR2	24h

Function
X+Y pointer register, parameter set 2

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	YPTR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	XPTR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-16	Start address of Y operand in PKCRAM with byte granularity

Table continues on the next page...

Table continued from the previous page...

Field	Function
YPTR	Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.
15-0 XPTR	Start address of X operand in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.

16.6.11 Z+R pointer register, parameter set 2 (PKC_ZRPTR2)

Offset

Register	Offset
PKC_ZRPTR2	28h

Function

Z+R pointer register, parameter set 2

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	RPTR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ZPTR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-16 RPTR	Start address of R result in PKCRAM with byte granularity Least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size.
15-0 ZPTR	Start address of Z operand in PKCRAM with byte granularity or constant for calculation modes using CONST If ZPTR is used as address pointer the least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. If ZPTR is used as CONST

Table continues on the next page...

Table continued from the previous page...

Field	Function
	operand the high byte is ignored and only bits 7:0 of ZPTR are used for the calculation. For shift/rotate operation further most significant bits are ignored depending on PKC_CTRL[REDMUL].

16.6.12 Length register, parameter set 2 (PKC_LEN2)

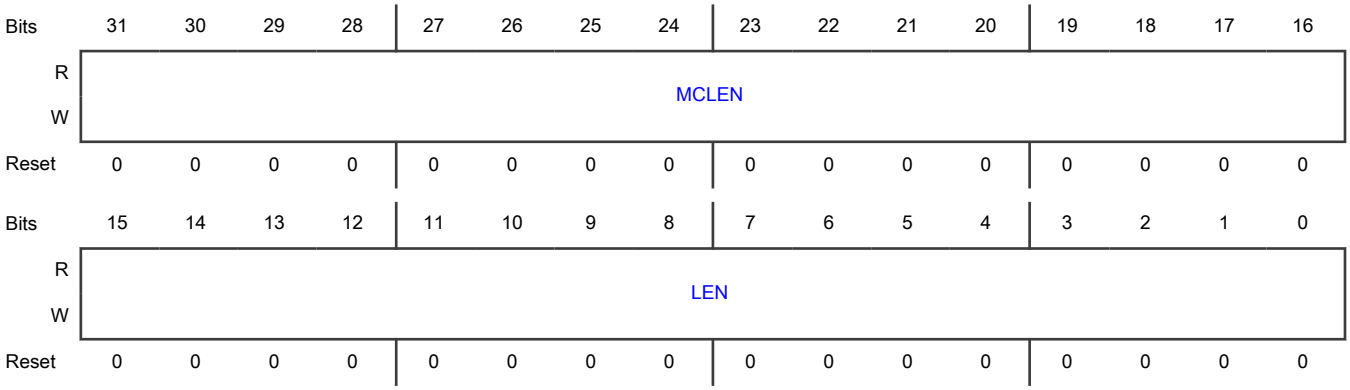
Offset

Register	Offset
PKC_LEN2	2Ch

Function

Length register, parameter set 2

Diagram



Fields

Field	Function
31-16 MCLEN	Loop counter for microcode pattern MCLEN defines the length of the loop counter that can be used in layer1 calculation mode, e.g. in MC opcode DecrTBNZ. For the hard coded MC patterns Modular Multiplication (MC start address 0x00). The least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. In case MCLEN is too short such that a DecrTBNZ MC opcode is executed on a zero loop counter value a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set).
15-0 LEN	Operand length LEN defines the length of the operands and the result in bytes. The length of Y, Z and R depend furthermore on the selected calculation mode.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	The least significant bits are ignored depending on PKC_CTRL[REDMUL] setting. Most significant bits are ignored depending on available PKCRAM size. In case LEN is too short such that the resulting length depending on PKC_CTRL[REDMUL] is zero a PKC security alarm is triggered (PKC_ACCESS_ERR[PKCC] is set).

16.6.13 Universal pointer FUP program (PKC_UPTR)

Offset

Register	Offset
PKC_UPTR	40h

Function

Universal pointer FUP program

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PTR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PTR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 PTR	Pointer to start address of PKC FUP program PKC_UPTR needs to be defined before starting a universal pointer PKC calculation (layer2) via PKC_CTRL[GOU]. The pointer address needs to be valid and the memory space the pointer addresses needs to be enabled for PKC access by the system. Otherwise a security alarm is triggered (PKC_ACCESS_ERR[AHB] is set). TPKC_UPTR must not be updated during a running layer2 calculation while PKC_STATUS[GOANY] is set. Update of this SFR during a universal pointer fetch calculation may result in wrong calculation results and/or security alarms. PKC_UPTR is updated by HW during a layer2 calculation, resp. incremented by 6 for each processed FUP program entry. The least significant bit of PKC_UPTR is ignored as the FUP program has to be stored at even addresses.

Table continues on the next page...

Field	Function
	NOTE After modifying the FUP program it is recommended to update the PKC_UPTR register (via an SFR write access) prior to starting a new layer2 calculation to clear the internal code-fetch buffer.

16.6.14 Universal pointer FUP table (PKC_UPTRT)

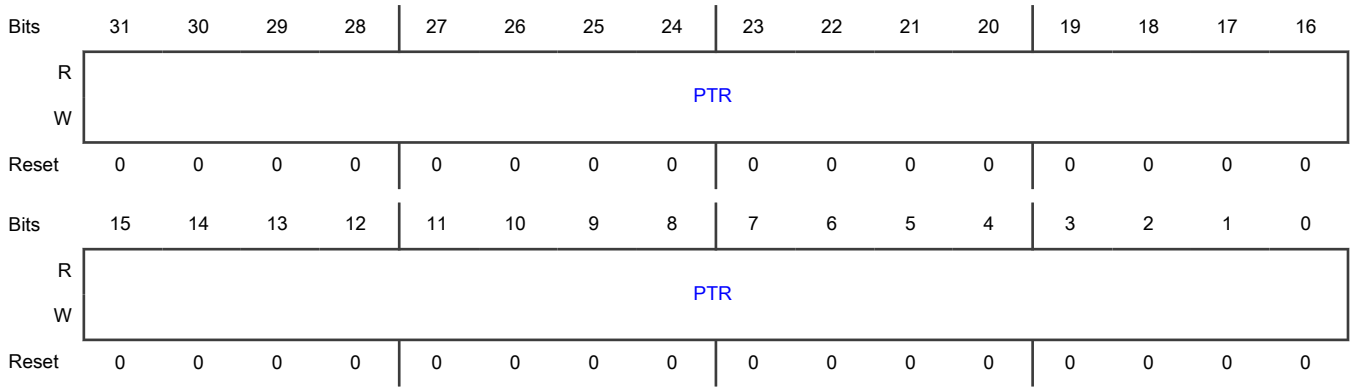
Offset

Register	Offset
PKC_UPTRT	44h

Function

Universal pointer FUP table

Diagram



Fields

Field	Function
31-0 PTR	Pointer to start address of PKC FUP table PKC_UPTRT needs to be defined before starting a universal pointer PKC calculation (layer2) via PKC_CTRL[GOU]. The pointer address needs to be valid and the memory space the pointer addresses needs to be enabled for PKC access by the system. Otherwise a security alarm is triggered (PKC_ACCESS_ERR[AHB] is set). PKC_UPTRT must not be updated during a running layer2 calculation while PKC_STATUS.GOANY is set. Update of this SFR during a universal pointer fetch calculation may result in wrong calculation results and/or security alarms. PKC_UPTRT is not updated by HW during a layer2 calculation. The least significant bit of PKC_UPTRT is ignored as the FUP table has to be stored at even addresses. Any SFR write access to PKC_UPTRT triggers the invalidation of the universal pointer cache, similar to PKC_CTRL[CLRCACHE].

Table continues on the next page...

Field	Function
	NOTE After modifying the FUP table it is recommended to invalidate the universal pointer cache (if it is enabled) prior to starting a new layer2 calculation.

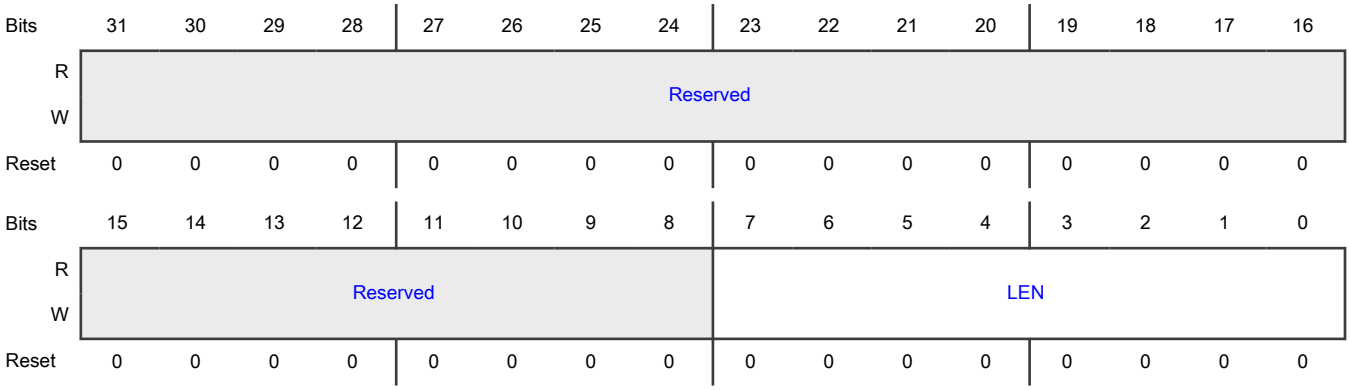
16.6.15 Universal pointer length (PKC_ULEN)

Offset

Register	Offset
PKC_ULEN	48h

Function
Universal pointer length

Diagram



Fields

Field	Function
31-8 —	Reserved
7-0 LEN	Length of universal pointer calculation PKC_ULEN defines how many FUP program entries shall be processed for one layer2 calculation started via PKC_CTRL[GOU]. The FUP program entries include layer0 calculations, layer1 calculations and CRC entries for FUP program integrity protection. PKC_ULEN must not be updated during a running layer2 calculation while PKC_STATUS[GOANY] is set. Update of this SFR during a universal pointer fetch calculation may result in wrong calculation results and/or security alarms. Starting a universal pointer fetch calculation via PKC_CTRL[GOU] while PKC_ULEN is zero triggers a security alarm (PKC_ACCESS_ERR[CTRL] is set). PKC_ULEN is updated by HW during a

Table continues on the next page...

Table continued from the previous page...

Field	Function
	layer2 calculation, resp. decremented for each processed FUP program entry. When PKC_ULEN has been decremented to zero PKC_STATUS[GOANY] is cleared to indicate the start of the last calculation. When the PKC has finalized its last pipe calculation PKC_STATUS[ACTIV] is cleared and the interrupt status bit PKC_INT_STATUS[INT_PDONE] is set.

16.6.16 MC pattern data interface (PKC_MCDATA)

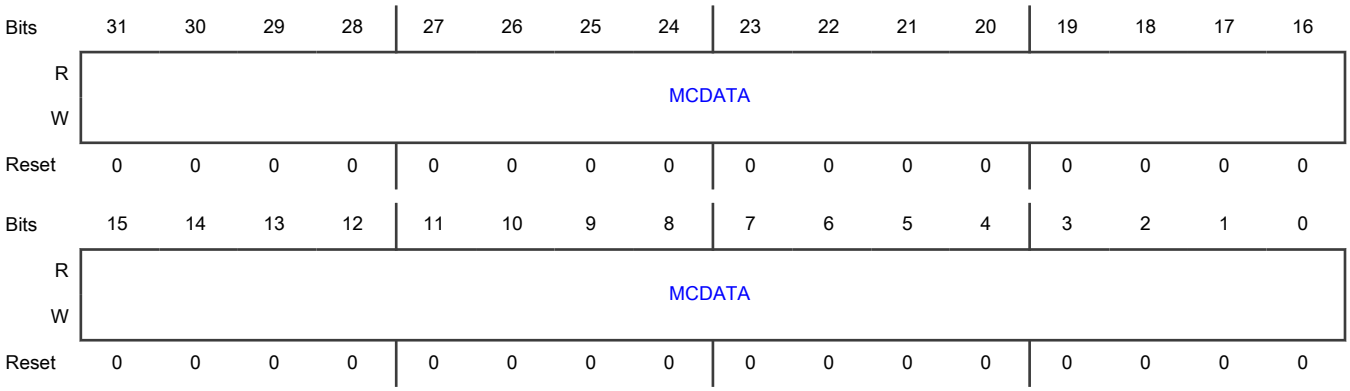
Offset

Register	Offset
PKC_MCDATA	50h

Function

MC pattern data interface

Diagram



Fields

Field	Function
31-0 MCDATA	Microcode read/write data This IP version does not support flexible MC patterns (only hard coded ones). Any read or write access to PKC_MCDATA triggers a security alarm (PKC_ACCESS_ERR[CTRL] is set). **NOTE** When trying to read from PKC_MCDATA a data value of 0 is returned. Any SFR access to PKC_MCDATA increments the PKC_MODE1 SFR by 4.

16.6.17 PKC version register (PKC_VERSION)

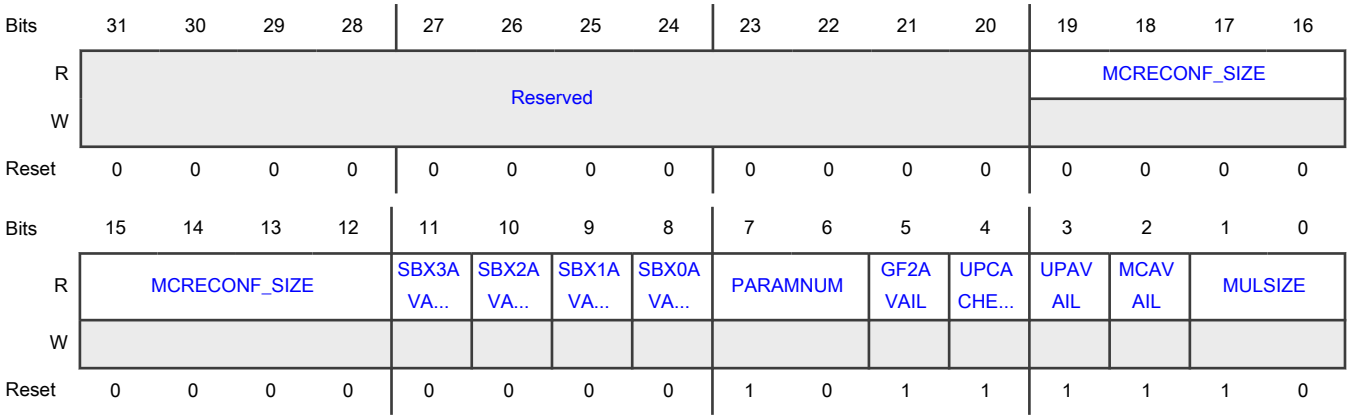
Offset

Register	Offset
PKC_VERSION	60h

Function

PKC version register

Diagram



Fields

Field	Function
31-20 —	Reserved
19-12 MCRECONF_SIZE	Size of reconfigurable MC table in bytes.
11 SBX3AVAIL	SBX3 operation is available
10 SBX2AVAIL	SBX2 operation is available
9 SBX1AVAIL	SBX1 operation is available
8 SBX0AVAIL	SBX0 operation is available

Table continues on the next page...

Table continued from the previous page...

Field	Function
7-6 PARAMNUM	Number of parameter sets for real calculation
5 GF2AVAIL	GF2 calculation modes are available
4 UPCACHEAVAIL	UP cache is available
3 UPAVAIL	UP feature (layer2 calculation) is available
2 MCAVAIL	MC feature (layer1 calculation) is available
1-0 MULSIZE	Native multiplier size and operand granularity 01b - 64-bit multiplier 10b - 128-bit multiplier 11b - 128-bit multiplier

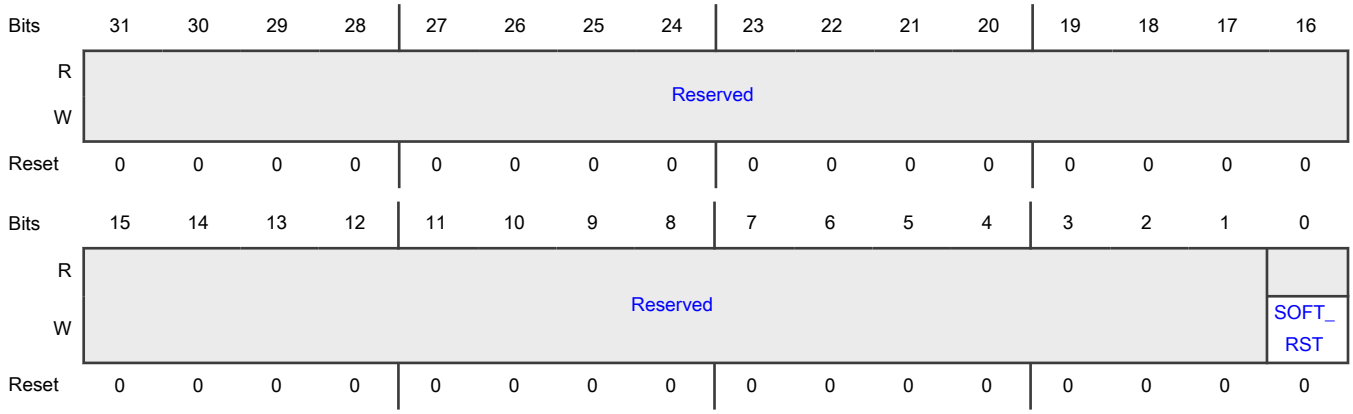
16.6.18 Software reset (PKC_SOFT_RST)

Offset

Register	Offset
PKC_SOFT_RST	FB0h

Function
Software reset

Diagram



Fields

Field	Function
31-1 —	Reserved
0 SOFT_RST	Write 1 to reset module (0 has no effect). All running and pending PKC calculation are stopped. All PKC SFRs are reset except PKC_ACCESS_ERR.

16.6.19 Access Error (PKC_ACCESS_ERR)

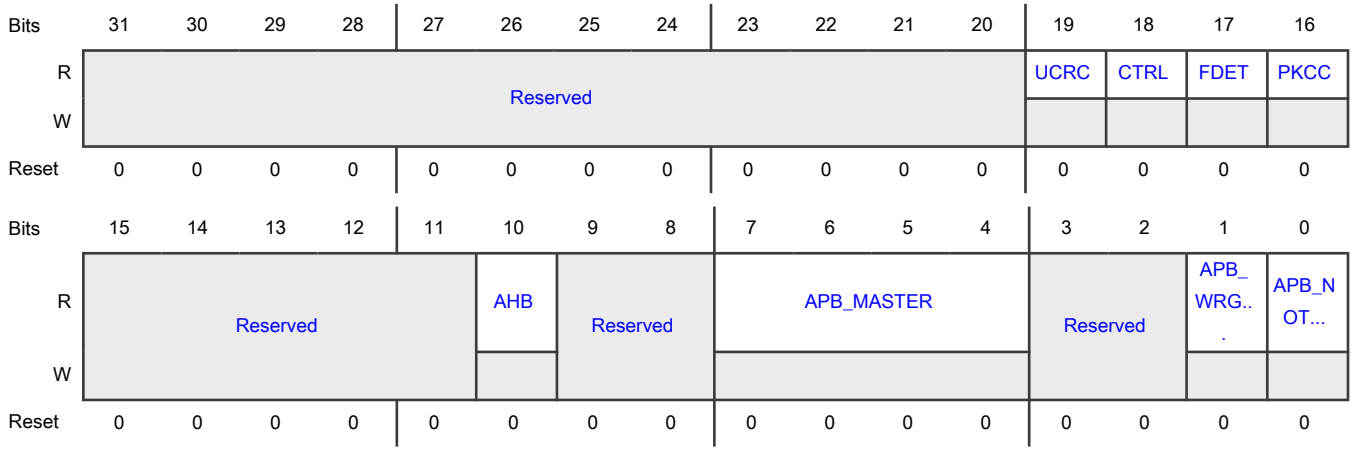
Offset

Register	Offset
PKC_ACCESS_ERR	FC0h

Function

Access Error

Diagram



Fields

Field	Function
31-20 —	Reserved
19 UCRC	Error in layer2 CRC check.
18 CTRL	Error in PKC software control
17 FDET	Error due to error detection circuitry
16 PKCC	Error in PKC coprocessor kernel
15-11 —	Reserved
10 AHB	AHB Error Invalid AHB access Layer2 Only
9-8 —	Reserved
7-4 APB_MASTER	APB Master that triggered first APB error (APB_WRGMD or APB_NOTAV)
3-2 —	Reserved
1 APB_WRGMD	APB Error Wrong access mode
0 APB_NOTAV	APB Error Address not available

16.6.20 Clear Access Error (PKC_ACCESS_ERR_CLR)

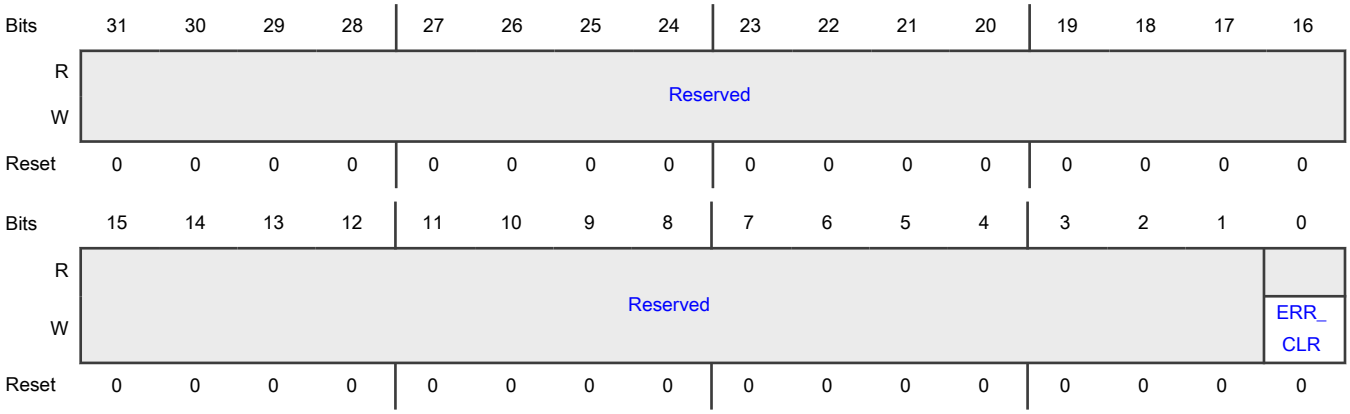
Offset

Register	Offset
PKC_ACCESS_ERR_CLR	FC4h

Function

Clear Access Error

Diagram



Fields

Field	Function
31-1 —	Reserved
0 ERR_CLR	Write 1 to reset PKC_ACCESS_ERR SFR.

16.6.21 Interrupt enable clear (PKC_INT_CLR_ENABLE)

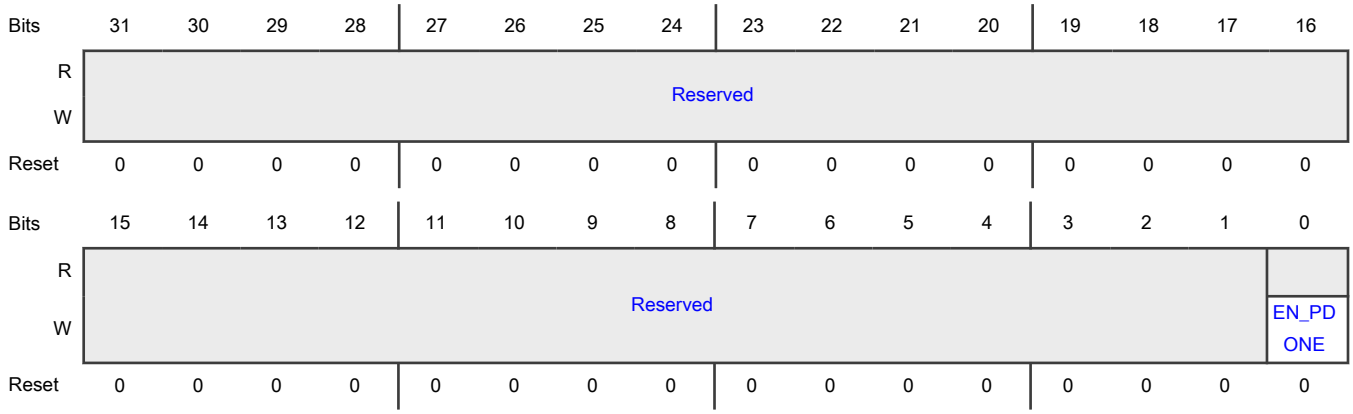
Offset

Register	Offset
PKC_INT_CLR_ENABLE	FD8h

Function

Interrupt enable clear

Diagram



Fields

Field	Function
31-1 —	Reserved
0 EN_PDONE	Write to clear PDONE interrupt enable flag (PKC_INT_ENABLE[EN_PDONE]=0).

16.6.22 Interrupt enable set (PKC_INT_SET_ENABLE)

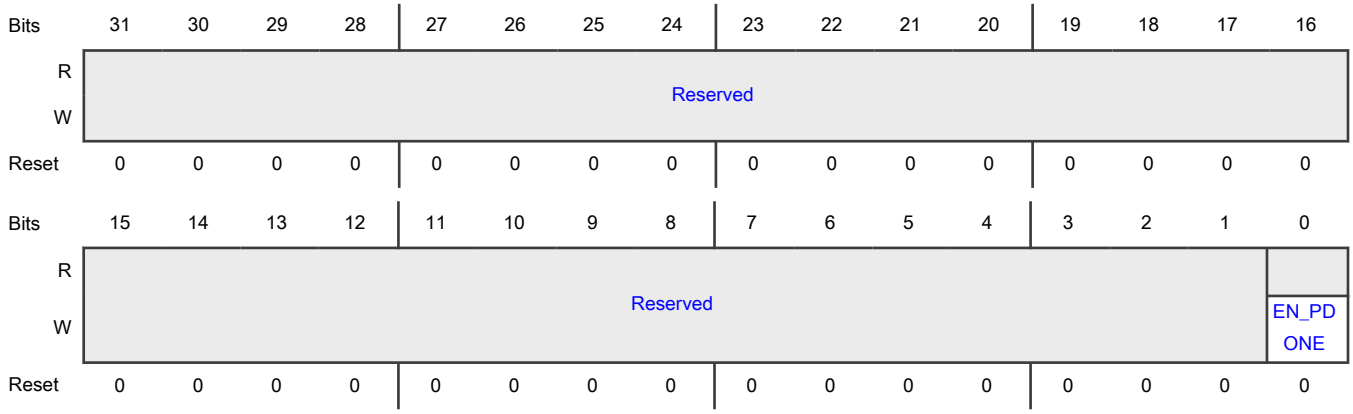
Offset

Register	Offset
PKC_INT_SET_ENABLE	FDCh

Function

Interrupt enable set

Diagram



Fields

Field	Function
31-1 —	Reserved
0 EN_PDONE	Write to set PDONE interrupt enable flag (PKC_INT_ENABLE[EN_PDONE]=1).

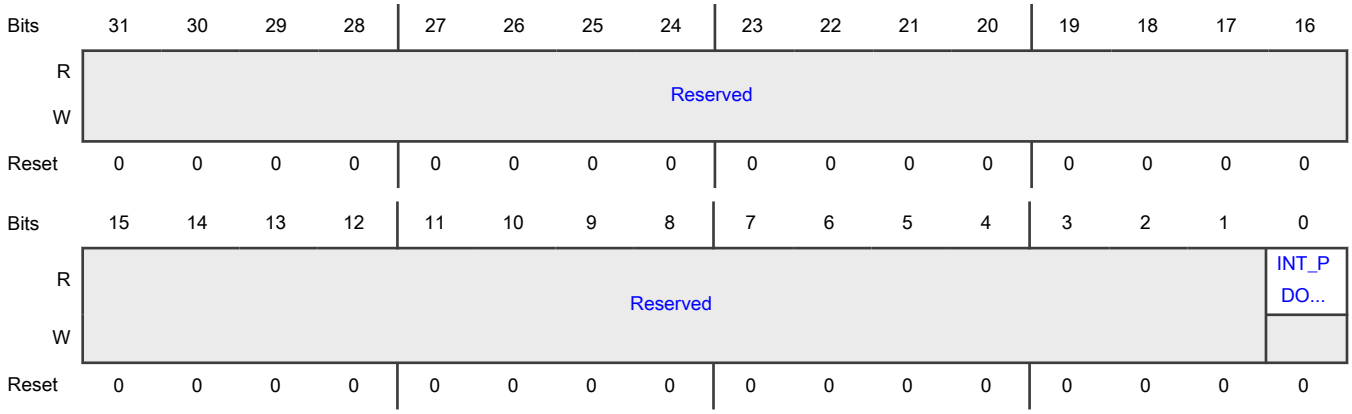
16.6.23 Interrupt status (PKC_INT_STATUS)

Offset

Register	Offset
PKC_INT_STATUS	FE0h

Function
Interrupt status

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_PDONE	End-of-computation status flag INT_PDONE is set after EACH single PKC layer0 or layer1 calculation. In case of a universal pointer calculation (layer2) INT_PDONE is set at the end of the pipe calculation when PKC_ULEN has been decremented to zero and the final PKC calculation has completed.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	INT_PDONE is set independently from the interrupt enable PKC_INT_ENABLE[EN_PDONE]. In case PKC_INT_ENABLE[EN_PDONE]=1 an interrupt towards the CPU is triggered when INT_PDONE is set (level triggered). INT_PDONE is not cleared by PKC hardware but has to be cleared by software, except in case of a reset (chip/block reset, PKC_SOFT_RST, PKC security alarm).

16.6.24 Interrupt enable (PKC_INT_ENABLE)

Offset

Register	Offset
PKC_INT_ENABLE	FE4h

Function

Interrupt enable

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved															EN_PDONE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-1 —	Reserved
0 EN_PDONE	PDONE interrupt enable flag If EN_PDONE=1, an interrupt is triggered every time PKC_INT_STATUS[INT_PDONE] is set. Otherwise the interrupt generation is suppressed.

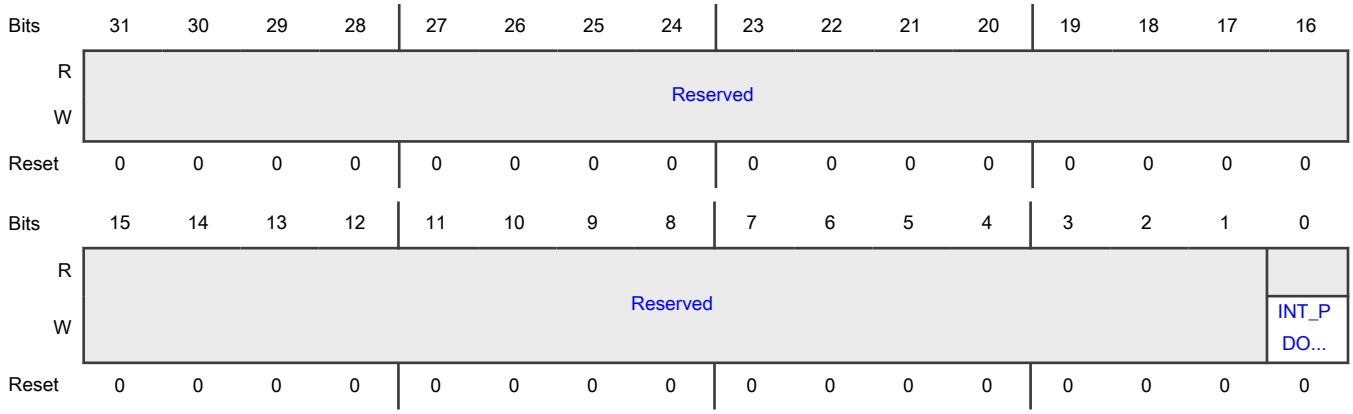
16.6.25 Interrupt status clear (PKC_INT_CLR_STATUS)

Offset

Register	Offset
PKC_INT_CLR_STATUS	FE8h

Function
Interrupt status clear

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_PDONE	Write to clear End-of-computation status flag (PKC_INT_STATUS[INT_PDONE]=0).

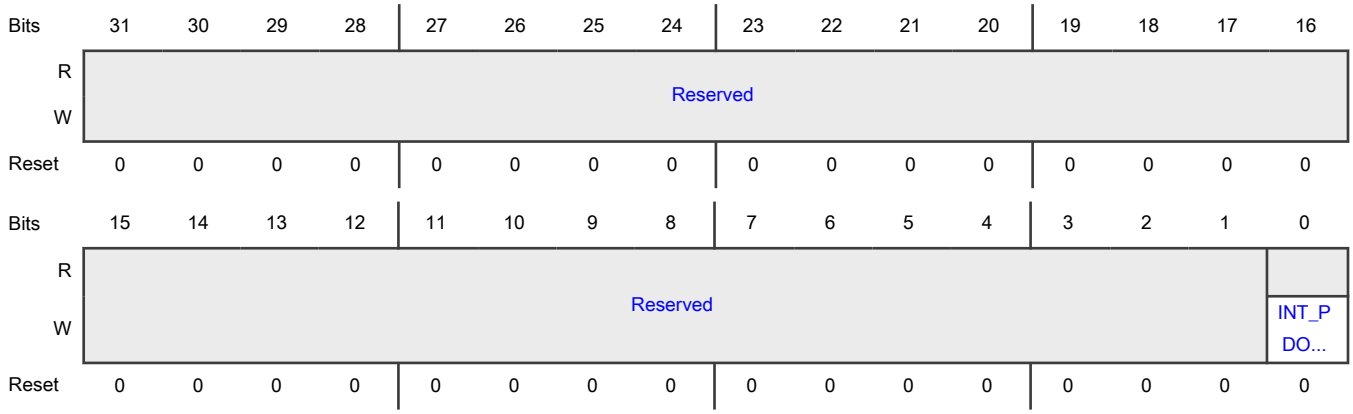
16.6.26 Interrupt status set (PKC_INT_SET_STATUS)

Offset

Register	Offset
PKC_INT_SET_STATUS	FECh

Function
Interrupt status set

Diagram



Fields

Field	Function
31-1 —	Reserved
0 INT_PDONE	Write to set End-of-computation status flag (PKC_INT_STATUS[INT_PDONE]=1) to trigger a PKC interrupt via software, e.g. for debug purposes.

16.6.27 Module ID (PKC_MODULE_ID)

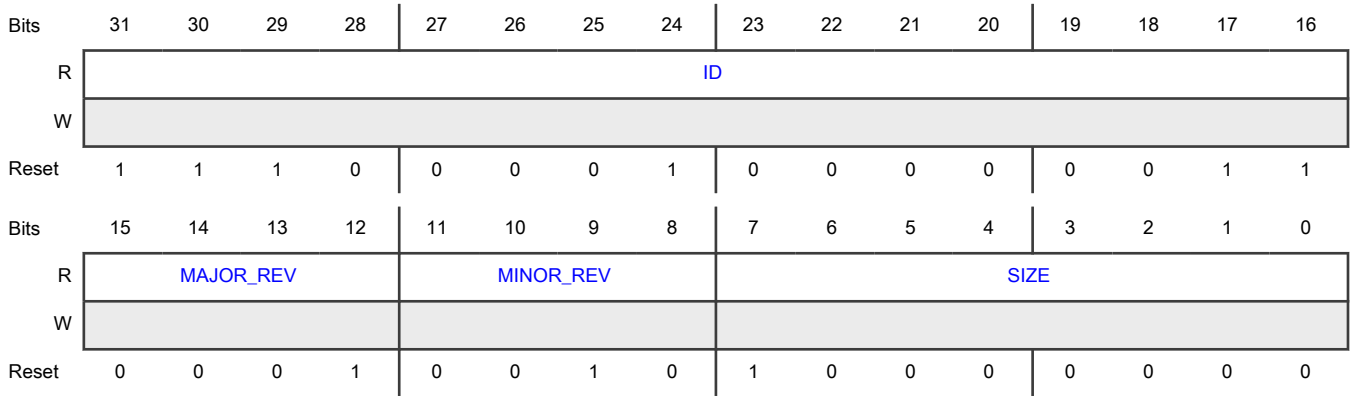
Offset

Register	Offset
PKC_MODULE_ID	FFCh

Function

Module ID

Diagram



Fields

Field	Function
31-16 ID	Module ID
15-12 MAJOR_REV	Major revision
11-8 MINOR_REV	Minor revision
7-0 SIZE	Address space of the IP

Chapter 17
OTP Controller (OTPC)

17.1 Chip-specific OTPC information

Table 393. Reference links to related information[1]

Topic	Related module	Reference
Full description	OTPC	OTPC
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Power management		See the chapter "Power management"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

NOTE

The PDF contains OTP Fuse Map added as an attachment.

NOTE

The reset values of RWC, TIMING1, and TIMING2 registers may change depending on the device's boot settings.

17.1.1 Module instances

This device has one instance of the OTPC module.

NOTE

Clock for the OTPC module is SYSTEM_CLK, which has the same frequency and synchronizes with CPU_CLK and AHB_CLK.

17.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

17.2 Overview

OTPC provides access to the on-chip EFUSE block. Software can read, write and reload EFUSE content into OTPC shadow registers. Configuration options and memory locking mechanisms are also provided.

17.2.1 Block diagram

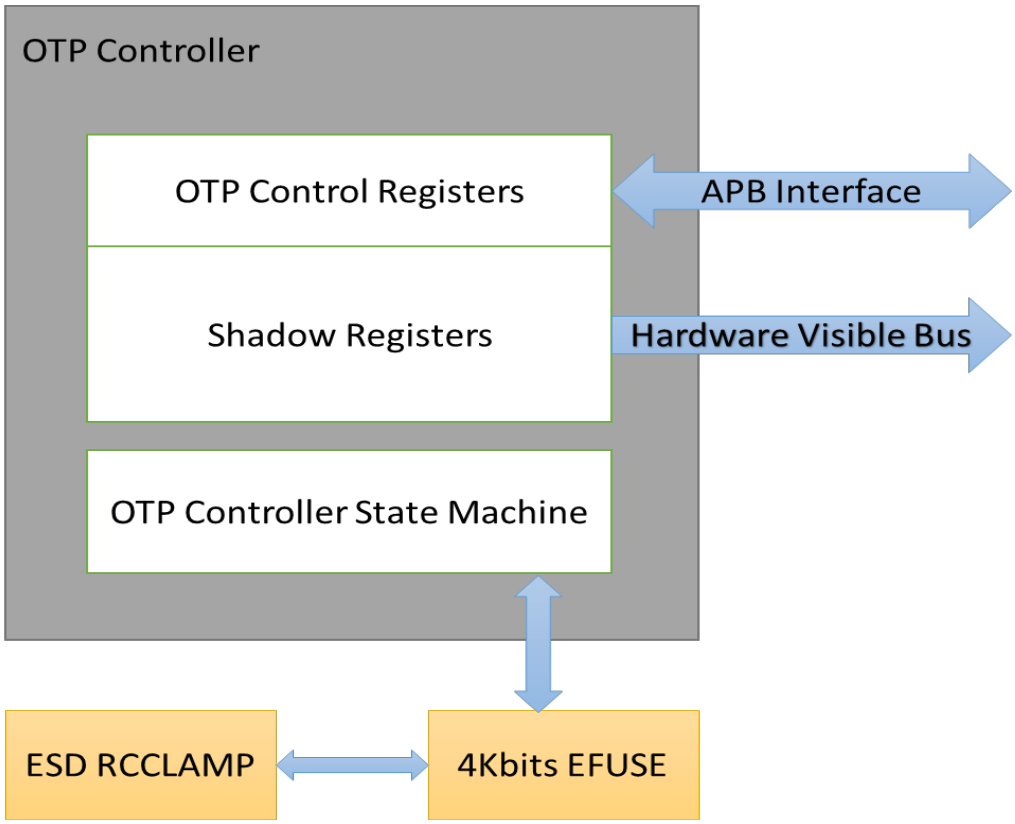


Figure 75. OTPC block diagram

17.2.2 Features

OTPC features include:

- Allows 32-bit word restricted program and read to 4Kbit of the EFUSE block
- Provides for the loading and storing of EFUSE content in shadow registers
- Supports APB access to memory-mapped (restricted) access to 4Kbit of shadow registers
- Supplies a dedicated bus connecting on-chip hardware to shadow register values
- Provides program-protect and read-protect EFUSE
- Allows overriding and read protection of shadow registers
- Provides adjustable timing for EFUSE program and read transactions

17.3 Functional description

OTPC supports these operations:

- Shadow Register Reload
- EFUSE and Shadow Register Read
- EFUSE and Shadow Register Writes

Data Structures in EFUSE describes the supported data formats.

17.3.1 Shadow Register Reload

Not all EFUSE words are shadowed. Only 16'd64 bytes of EFUSE are loaded to shadow registers. EFUSE information is therefore available through memory-mapped shadow registers. The non-shadowed EFUSE are accessible using the EFUSE read operation. See Read procedure.

If EFUSE locations are subsequently programmed, the shadow registers must be reloaded to keep them coherent with the EFUSE bank arrays. The reload operation allows software to force a reload of the shadow registers (including LOCK) without having to reset the device.

17.3.1.1 Reload procedure

To force a reload of the shadow registers, complete the following steps:

1. Configure TIMING1 with timing values to match the current clock frequency.
2. Check that SR[BUSY] and the SR error bits are cleared. OTPC does not support overlapped accesses. Any pending write, read or reload must be completed before requesting a new access.
3. Set RLC[RELOAD_SHADOWS]. OTPC then reads all the EFUSE words one by one and puts them into their corresponding shadow registers.
4. Wait for SR[BUSY] and RLC[RELOAD_SHADOWS] to be cleared by OTPC.

OTPC automatically clears the SR[BUSY] and RLC[RELOAD_SHADOWS] bits after successful completion of the operation.

17.3.2 EFUSE and Shadow Register Read

All shadow registers are always readable through the APB bus except for some secret keys regions. When their corresponding EFUSE lock bits are set, the shadow registers also become read-locked. After read locking, reading from these registers returns 0xBADDBADD. In addition, SR[ERROR] is set. Software must clear SR[ERROR] before requesting any new write, read or reload access. Subsequent reads to unlocked shadow locations still work successfully however.

17.3.2.1 Read procedure

Complete the following steps to read a fuse word directly from the EFUSE block:

1. Configure TIMING1 with timing values to match the current clock frequency.
2. Check that SR[BUSY] and the SR error bits are cleared. OTPC does not support overlapped accesses. Any pending write, read or reload must be completed before requesting a read access.
3. Write the requested word address to RWC[ADDR] and set RWC[READ_EFUSE] at the same time. This automatically sets SR[BUSY]. OTPC reads 4 bytes from the requested EFUSE address one by one and then stores the read value in RDATA.
4. Wait for SR[BUSY] to be cleared by OTPC. A read request to a protected or locked region results in no OTPC access and no setting of SR[BUSY]. In addition, SR[ERROR] is set. Software must clear SR[ERROR] before requesting any new access.
5. Read RDATA to get the EFUSE word value. When SR[ERROR] is set, RDATA is 0xBADDBADD.

17.3.3 EFUSE and Shadow Register Writes

Software can override shadow register bits, by writing directly to the shadow registers, unless the corresponding EFUSE lock bit for the region is set. When the lock shadow bit is set, the shadow registers for that lock region become write-locked. The LOCK shadow register has no shadow or EFUSE lock bits and is always read-only.

17.3.4 Data Structures in EFUSE

OTPC provides three data structures in EFUSE.

17.3.4.1 Triple voting data structure

In the triple-voting data structure, three EFUSE bytes generate one valid data byte. This structure supports incremental programming to EFUSE bits.

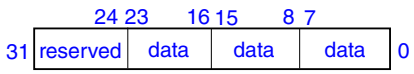


Figure 76. Data Redundancy (Triple Voting) Addressing

17.3.4.2 Raw data structure

The raw data structure simply keeps in-coming and out-going data as-is without modifying its structure.

17.3.4.3 ECC data structure

The ECC data structure uses Hamming code to correct 1-bit errors and detect 2-bit errors. This structure uses 32 data bits and 8 parity bits for a total of 40 bits.

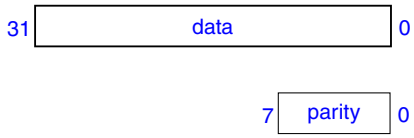


Figure 77. ECC Data and Parity Addressing

If programming ECC-protected regions, software must write the data to WDATA before programming. OTPC automatically generates the associated 8 parity bits.

If reading from ECC-protected regions:

- RDATA returns the original data bits if no ECC error occurred.
- RDATA returns the corrected data bits if 1 ECC error occurred. SR[ECC_SF] is set.
- RDATA returns the original data bits if 2 ECC errors occurred. SR[ECC_DF] is set.

NOTE

ECC fuse words do not support incremental programming. As the ECC is generated for the whole word, the whole word must be setup at one time and individual bits cannot be modified independently.

17.3.5 Clocking

See the chip-specific OTPC information for the required clocking.

17.3.6 Interrupts

OTPC has no interrupts.

17.4 External signals

OTPC has no external signals.

17.5 Initialization

After obtaining the required clocking, OTPC is ready for use.

17.6 Application information

See the following procedures for OTPC applications:

- Reload procedure
- Read procedure

17.7 Memory Map and register definition

This section includes the OTP Controller module memory map and detailed descriptions of all registers.

17.7.1 OTPC register descriptions

17.7.1.1 OTPC memory map

OTPC0 base address: 400C_9000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Version ID (VERID)	32	R	0100_0000h
4h	Parameters (PARAM)	32	R	0000_0040h
8h	Status (SR)	32	RW	0000_0000h
10h	Read and Write Control (RWC)	32	RW	0000_0000h
14h	Reload Control (RLC)	32	RW	0000_0000h
18h	Power Control (PCR)	32	RW	0000_0000h
20h	Write Data (WDATA)	32	RW	0000_0000h
24h	Read Data (RDATA)	32	R	0000_0000h
30h	Timing1 (TIMING1)	32	RW	480C_0711h
34h	Timing2 (TIMING2)	32	RW	0000_0600h
200h	Lock (LOCK)	32	RW	0000_0000h
204h	Secure (SECURE)	32	R	See section
208h	Inverted Secure (SECURE_INV)	32	R	See section
20Ch	Debug and Key (DBG_KEY)	32	R	0000_0000h
210h	MISC Config (MISC_CFG)	32	RW	0000_0000h
214h	PHANTOM Config (PHANTOM_CFG)	32	RW	0000_0000h
218h	Flexible Config 0 (FLEX_CFG0)	32	RW	0000_0000h
21Ch	Flexible Config 1 (FLEX_CFG1)	32	RW	0000_0000h

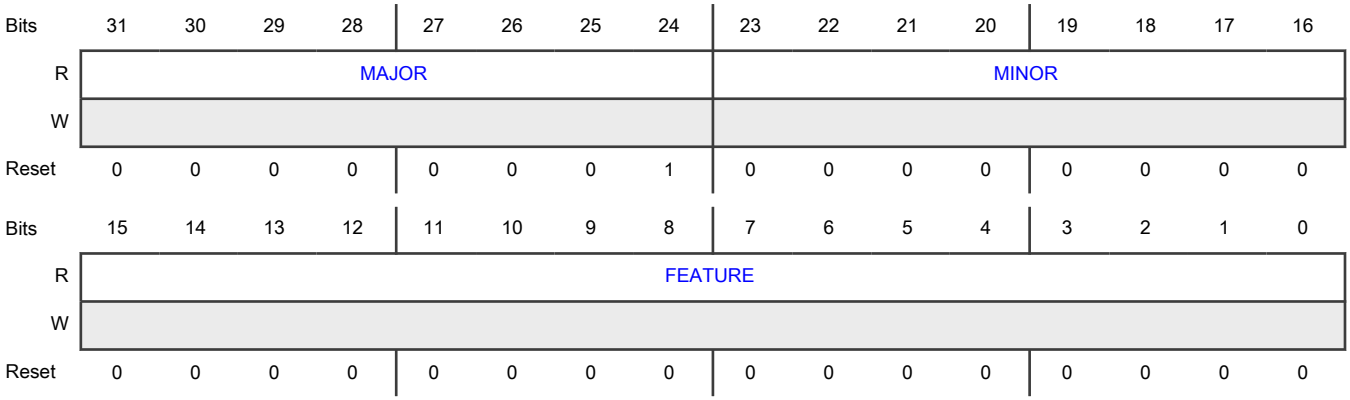
17.7.1.2 Version ID (VERID)

Offset

Register	Offset
VERID	0h

Function
Contains version numbers for the module design and feature set.

Diagram



Fields

Field	Function
31-24 MAJOR	Major Version Number Returns the major version number for the specification.
23-16 MINOR	Minor Version Number Returns the minor version number for the specification.
15-0 FEATURE	Feature Specification Number Returns the feature set number. 0000_0000_0000_0000b - Standard feature set

17.7.1.3 Parameters (PARAM)

Offset

Register	Offset
PARAM	4h

Function

Contains parameter values that were implemented in the module.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	NUM_FUSE															
W																
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Fields

Field	Function
31-16 —	Reserved
15-0 NUM_FUSE	Number of fuse bytes Contains the number of bytes loaded from EFUSE to the shadow registers. **NOTE** This bit does not represent the total number of fuses.

17.7.1.4 Status (SR)

Offset

Register	Offset
SR	8h

Function

Contains operational error and fault flags, as well as the OTPC busy status indicator.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved											FSC	IRC	ADC	FLC	FSM
W												W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		WR_POWE...	WR_REG_...	WR_REG_...	RD_REG_...	WR_FUSE...	RD_FUSE...	Reserved			TRI_F	ECC_DF	ECC_SF	ERROR	BUSY
W												W1C	W1C	W1C	W1C	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-21 —	Reserved
20 FSC	Fuse and shadow register compare error Indicates the values of EFUSE and the shadow registers do not match during initial and reload phases. 0b - No error 1b - Error
19 IRC	Inverted register compare error Indicates the SECURE_INV register has not maintained the inverted bit values of the SECURE register. 0b - No error 1b - Error
18 ADC	Address and data compare error Indicates the address compare or data compare does not match during a fuse data read. OTPC stores backup values of the address and data when reading from EFUSE. This comparison checks if the address and data values are identical to their original values. 0b - No error 1b - Error
17 FLC	Fuse load counter error Indicates the fuse load counter does not match the expected value during initialization and reload phases. 0b - No error 1b - Error
16	Finite-state machine error

Table continues on the next page...

Table continued from the previous page...

Field	Function
FSM	Indicates the OTPC finite-state machine has entered an unknown state. 0b - No error 1b - Error
15-14 —	Reserved
13 WR_POWER_OFF	Write when power off error Indicates an error when attempting a program with PCR[HVREQ] or PCR[LVREQ] fields not set. Clearing the ERROR bit also clears this bit. 0b - No error 1b - Error
12 WR_REG_BUSY	Write register when busy error Indicates an error when attempting a write to a register (except the Status register) while OTPC is busy. Clearing the ERROR bit also clears this bit. 0b - No error 1b - Error
11 WR_REG_LOCK	Write register lock error Indicates an error when attempting a write to a locked shadow register. Clearing the ERROR bit also clears this bit. 0b - No error 1b - Error
10 RD_REG_LOCK	Read register lock error Indicates an error when attempting a read from a locked shadow register. Clearing the ERROR bit also clears this bit. 0b - No error 1b - Error
9 WR_FUSE_LOCK	Write fuse lock error Indicates an error when attempting a program to a locked EFUSE word. Clearing the ERROR bit also clears this bit. 0b - No error 1b - Error
8 RD_FUSE_LOCK	Read fuse lock error Indicates an error when attempting a read from a locked EFUSE word. Clearing the ERROR bit also clears this bit.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - No error 1b - Error
7-5 —	Reserved
4 TRI_F	Triple voting fault Indicates a triple voting fault occurred during an EFUSE read. 0b - No fault 1b - Fault
3 ECC_DF	ECC double fault Indicates an ECC double fault occurred during an EFUSE read. 0b - No fault 1b - Fault
2 ECC_SF	ECC single fault Indicates an ECC single fault occurred during an EFUSE read. 0b - No fault 1b - Fault
1 ERROR	Error flag Indicates the result of the last operation. When set, indicates one of following illegal operations: • A program is attempted with HVREQ or LVREQ not set. • A write is attempted to a register (except the Status register) while OTPC is busy. • A write is attempted to a locked shadow register. • A read is attempted from a locked shadow register. • A program is attempted to a locked EFUSE word. • A read is attempted from a locked EFUSE word. 0b - No error 1b - Error
0 BUSY	Busy status Indicates the OTPC busy status. BUSY is set during write, read, reload and auto-load sequences. It's auto-cleared when exiting from these sequences. 0b - Not busy (transaction complete) 1b - Busy

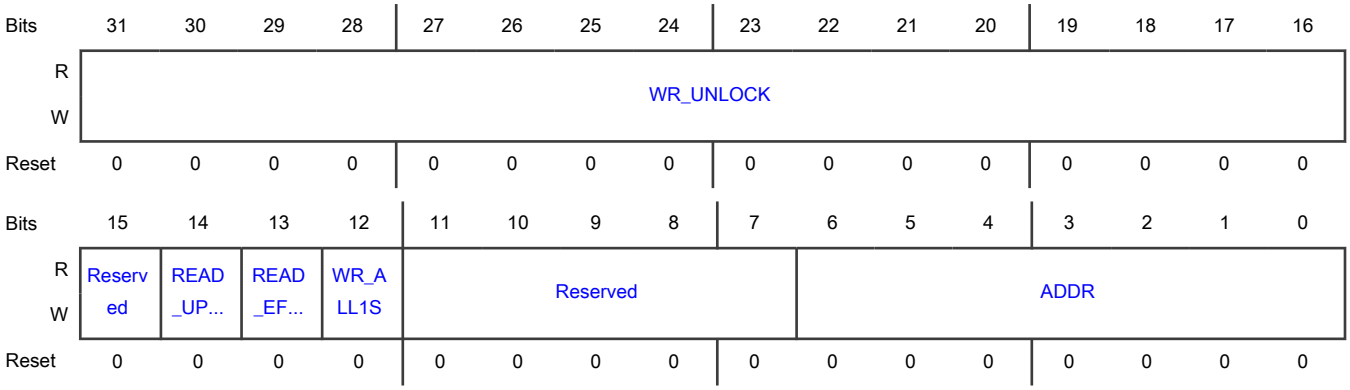
17.7.1.5 Read and Write Control (RWC)

Offset

Register	Offset
RWC	10h

Function
Controls the EFUSE program and read operations.

Diagram



Fields

Field	Function
31-16 WR_UNLOCK	Write Unlock Writing the value 0x9527 unlocks write access and enables EFUSE program operation. Always returns 0 when reading.
15 —	Reserved
14 READ_UPDATE	Read update Updates the related shadow register when reading EFUSE content. 0b - Shadow register does not update 1b - Shadow register updates
13 READ_EFUSE	Read EFUSE Initiates EFUSE read or program operations. 0b - Starts program operation when the WR_UNLOCK value is 0x9527; otherwise, takes no action. 1b - Starts read operation

Table continues on the next page...

Table continued from the previous page...

Field	Function
12 WR_ALL1S	Write all 1s Bypasses the value of WDATA and writes all 1s to the assigned EFUSE address, regardless of the program lock bit. **NOTE** This option is for ECC-protected regions only and should be used with caution. 0b - Uses the WDATA value 1b - Writes all 1s
11-7 —	Reserved
6-0 ADDR	EFUSE address Contains the EFUSE word address to be used for a program or read operation.

17.7.1.6 Reload Control (RLC)

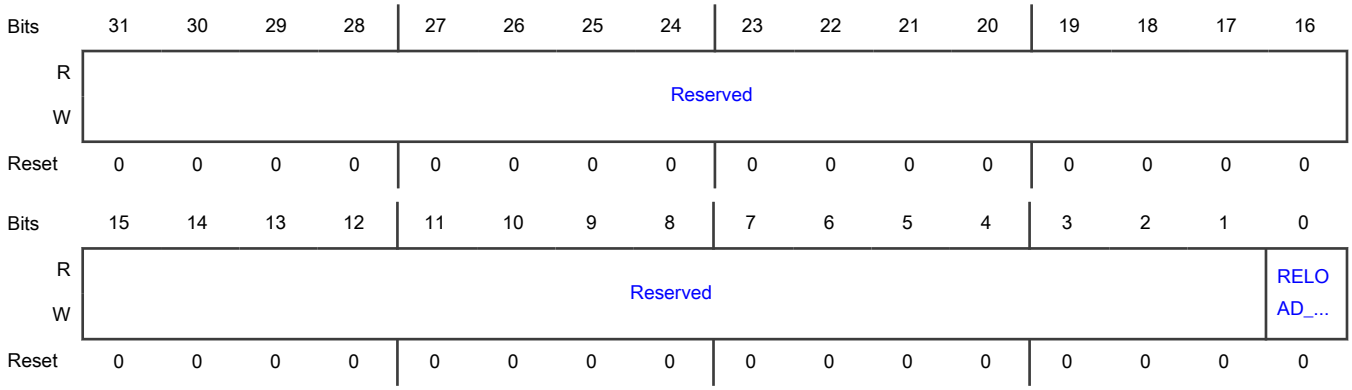
Offset

Register	Offset
RLC	14h

Function

Controls the reload operation, moving EFUSE content to the shadow registers.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 RELOAD_SHADOWS	Reload shadow registers Initiates the reload operation to move all EFUSE content to the shadow registers. Auto-cleared when exiting the reload sequence. 0b - No action (when writing) or reload complete (when reading) 1b - Reload

17.7.1.7 Power Control (PCR)

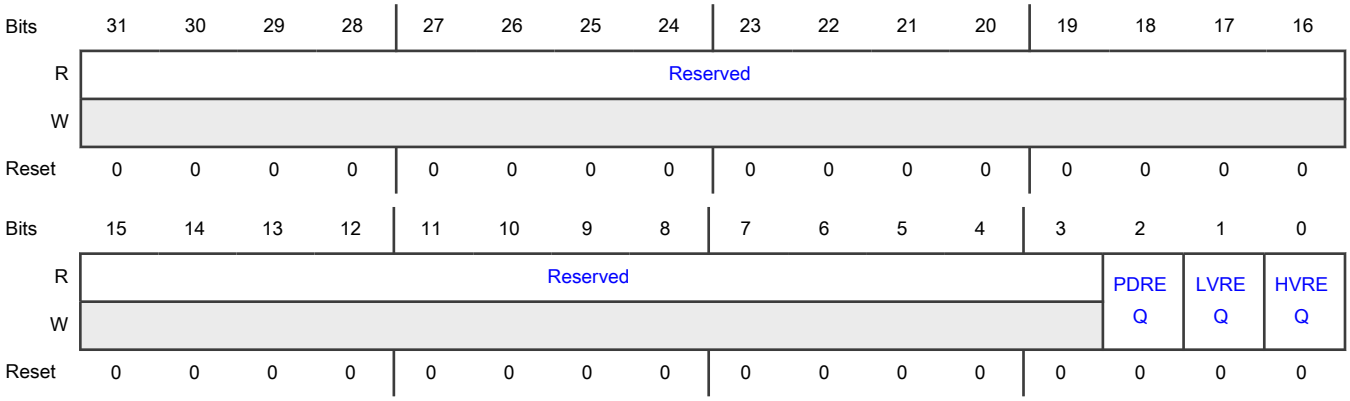
Offset

Register	Offset
PCR	18h

Function

Controls the voltage configuration for programming the EFUSE and includes the option to power down.

Diagram



Fields

Field	Function
31-3 —	Reserved
2	Power down request

Table continues on the next page...

Table continued from the previous page...

Field	Function
PDREQ	Requests to power down the EFUSE block. 0b - PD pin is set to low when OTPC is in idle state. It means EFUSE hardmacro is in standby mode. Idle state means OTPC is not in read and program modes. 1b - PD pin is set to high when OTPC is in idle state. It means EFUSE hardmacro is in power down mode.
1 LVREQ	Weak switch request Requests to turn on or off the weak switch for the EFUSE programming voltage. 0b - Turn off 1b - Turn on
0 HVREQ	Strong switch request Requests to turn on or off the strong switch for the EFUSE programming voltage. 0b - Turn off 1b - Turn on

17.7.1.8 Write Data (WDATA)

Offset

Register	Offset
WDATA	20h

Function

Contains the data used for an EFUSE program operation.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DAT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	DAT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 DAT	Write data Contains the data used for EFUSE programming.

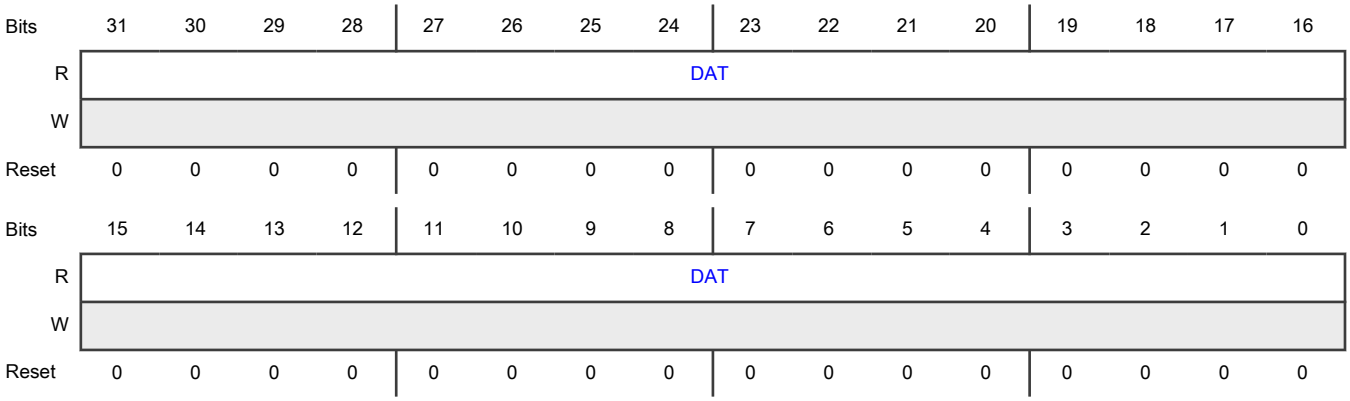
17.7.1.9 Read Data (RDATA)

Offset

Register	Offset
RDATA	24h

Function
Contains the data received from an EFUSE read operation.

Diagram



Fields

Field	Function
31-0 DAT	Read data Contains the data read from EFUSE.

17.7.1.10 Timing1 (TIMING1)

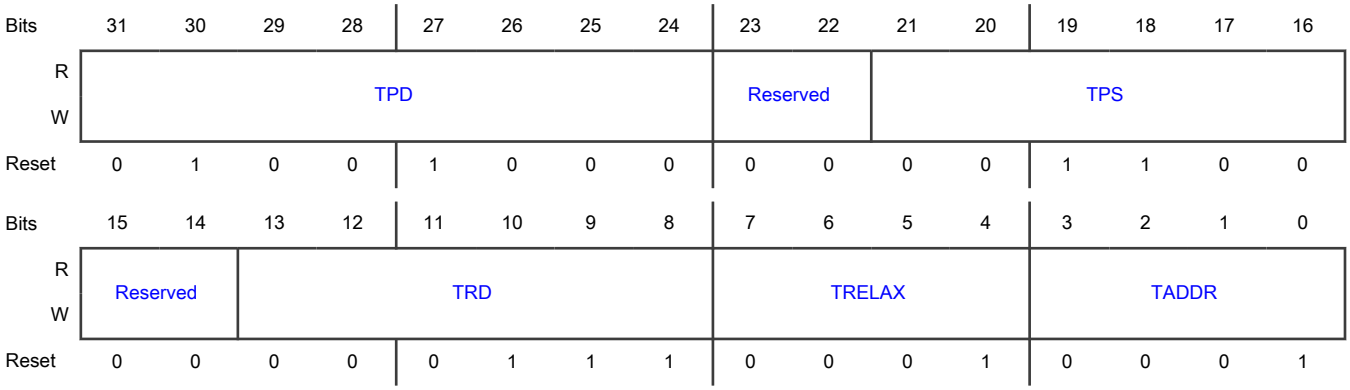
Offset

Register	Offset
TIMING1	30h

Function

Contains transaction timing parameters for accessing the EFUSE block.

Diagram



Fields

Field	Function
31-24 TPD	PD to CSB setup time between power down signal deassertion and chip select signal assertion {Tpd * clock_cycle} >= EFUSE block timing parameter TSUR_PD_CS
23-22 —	Reserved
21-16 TPS	PS to CSB setup and hold time between power switch and chip select assertion {(Tps+1) * clock_cycle} >= max value of EFUSE block timing parameters TSUP_PS_CS and THP_PS_CS
15-14 —	Reserved
13-8 TRD	Read strobe pulse width time {(Trd+1) * clock_cycle} >= EFUSE block timing parameter TRD
7-4 TRELAX	CSB, PGENB and LOAD to STROBE setup and hold time {(Trelax+Taddr+2) * clock_cycle} >= max value of EFUSE block timing parameters TSUR_CS, THR_CS, TSUR_PG, THR_PG, TSUR_LD, THR_LD, TSUP_CS, THP_CS, TSUP_PG, THP_PG, TSUP_LD and THP_LD
3-0 TADDR	Address to STROBE setup and hold time {(Taddr+1) * clock_cycle} >= max value of EFUSE block timing parameters TSUR_A, THR_A, TSUP_A and THP_A

17.7.1.11 Timing2 (TIMING2)

Offset

Register	Offset
TIMING2	34h

Function
Contains the program strobe timing parameter for accessing the EFUSE block.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved				TPGM											
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-12 —	Reserved
11-0 TPGM	Typical program strobe pulse width time {(Tpgm+1) * clock_cycle} = EFUSE block timing parameter TPGM

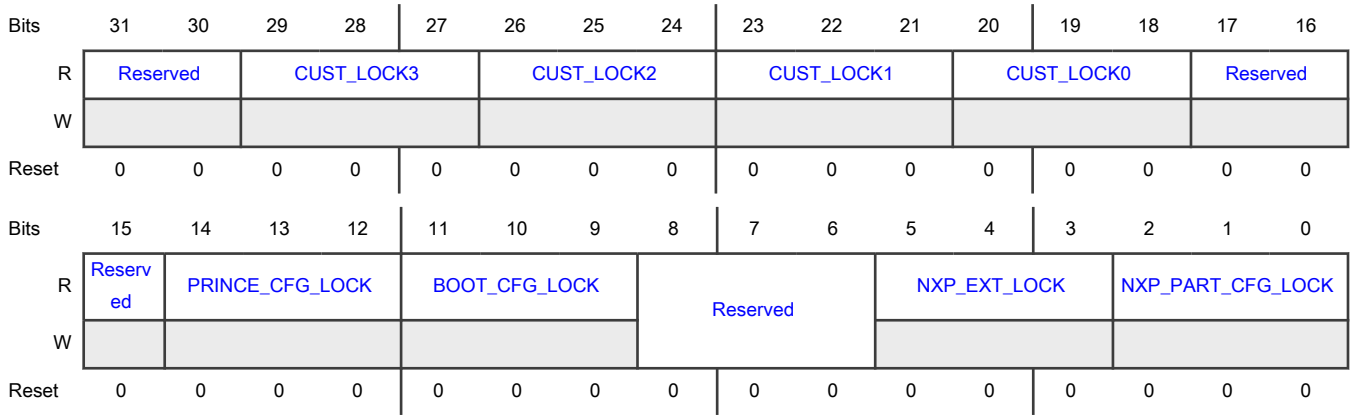
17.7.1.12 Lock (LOCK)

Offset

Register	Offset
LOCK	200h

Function
Controls lock options for different address ranges within the EFUSE. It also allows configuration shadow registers to be locked.

Diagram



Fields

Field	Function
31-30 —	Reserved
29-27 CUST_LOCK3	CUST Lock 3 Configures the locking of EFUSE bits in the address range 2464 to 2719. • bit 0: When set, blocks programming. • bit 1: Reserved • bit 2: When set, blocks reading.
26-24 CUST_LOCK2	CUST Lock 2 Configures the locking of EFUSE bits in the address range 2336 to 2463. • bit 0: When set, blocks programming. • bit 1: Reserved • bit 2: When set, blocks reading.
23-21 CUST_LOCK1	CUST Lock 1 Configures the locking of EFUSE bits in the address range 2208 to 2335. • bit 0: When set, blocks programming. • bit 1: Reserved • bit 2: When set, blocks reading.
20-18 CUST_LOCK0	CUST Lock 0 Configures the locking of EFUSE bits in the address range 2080 to 2207. • bit 0: When set, blocks programming. • bit 1: Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
	• bit 2: When set, blocks reading.
17-15 —	Reserved
14-12 PRINCE_CFG_LOCK	Prince Config Lock Configures the locking of EFUSE bits in the address range 1440 to 1951. • bit 0: When set, blocks programming. • bit 1: Reserved • bit 2: When set, blocks reading.
11-9 BOOT_CFG_LOCK	Boot config Lock Configures the locking of EFUSE bits in the address range 800 to 1439. • bit 0: When set, blocks programming. • bit 1: Reserved • bit 2: When set, blocks reading.
8-6 —	Reserved
5-3 NXP_EXT_LOCK	NXP EXT Lock Configures the locking of EFUSE bits in the address range 576 to 639. • bit 0: When set, blocks programming. • bit 1: Reserved • bit 2: When set, blocks reading.
2-0 NXP_PART_CFG_LOCK	NXP Part Config Lock Configures the locking of the MISC_CFG, PHANTOM_CFG, FLEX_CFG0 and FLEX_CFG1 shadow registers and the locking of EFUSE bits in the address range 384 to 575. • bit 0: When set, blocks EFUSE programming. • bit 1: When set, blocks writing shadow registers. • bit 2: When set, blocks reading shadow registers and EFUSE.

17.7.1.13 Secure (SECURE)

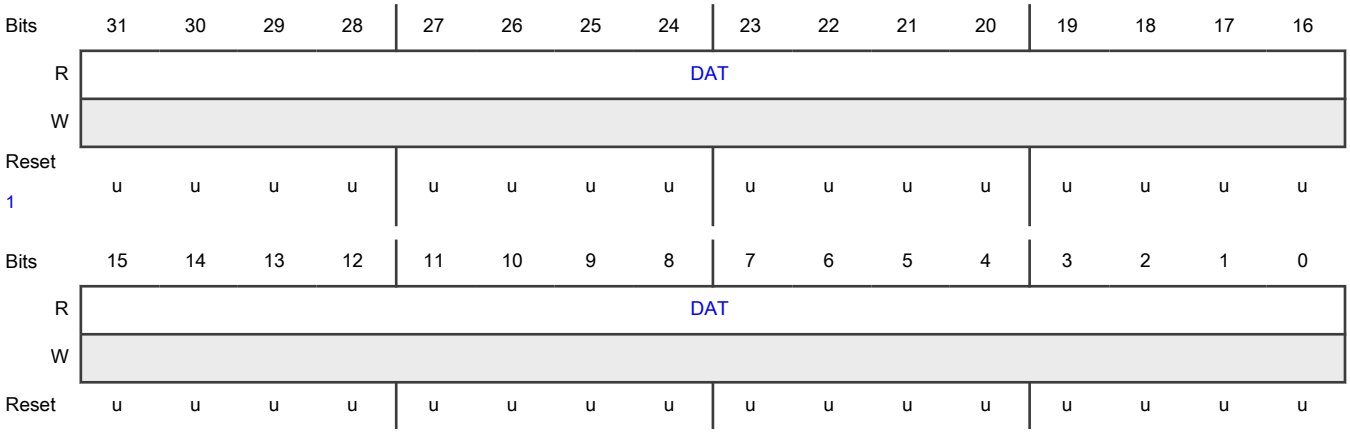
Offset

Register	Offset
SECURE	204h

Function

This shadow register is for secure data.

Diagram



1. When the chip resets, this register initially resets to 0xFFFF_FFFF for better security. During boot, OTPC loads it with the value from EFUSE.

Fields

Field	Function
31-0 DAT	Data Contains the secure data.

17.7.1.14 Inverted Secure (SECURE_INV)

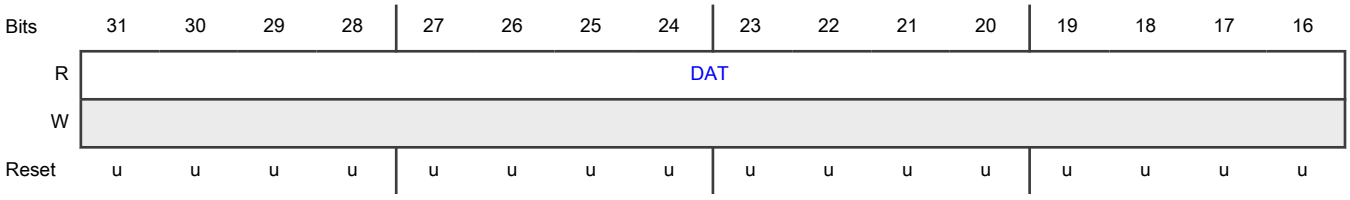
Offset

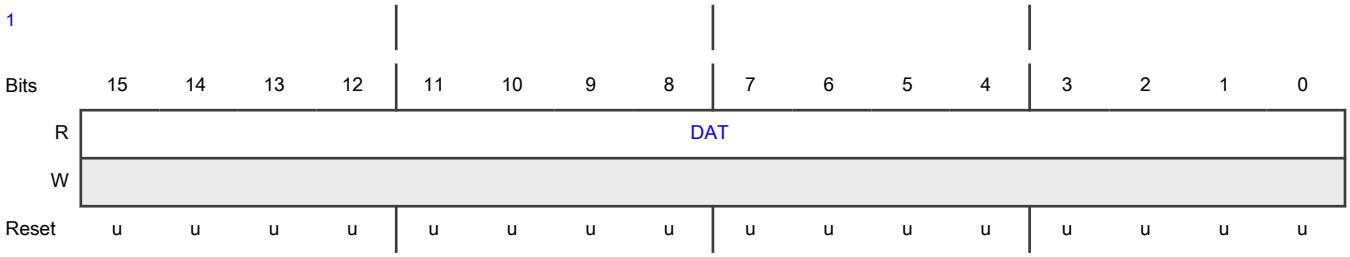
Register	Offset
SECURE_INV	208h

Function

Holds the inverted value of the Secure register as an error check.

Diagram





1. When the chip resets, this register initially resets to 0xFFFF_FFFF for better security. During boot, OTPC loads it with the inverted value of SECURE.

Fields

Field	Function
31-0 DAT	Data Contains the inverted value of SECURE[DAT].

17.7.1.15 Debug and Key (DBG_KEY)

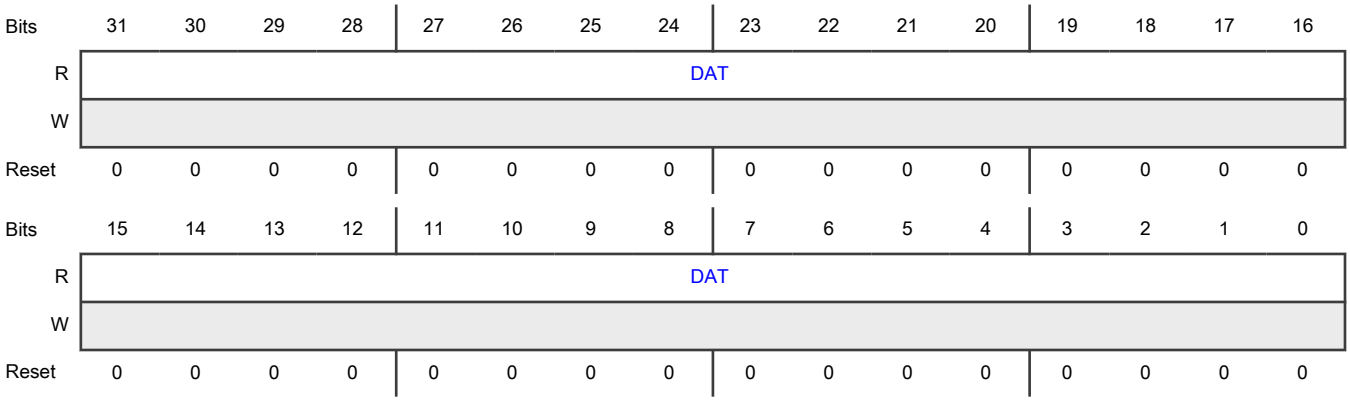
Offset

Register	Offset
DBG_KEY	20Ch

Function

This shadow register is for debug and key.

Diagram



Fields

Field	Function
31-0 DAT	Data Contains the debug and key data.

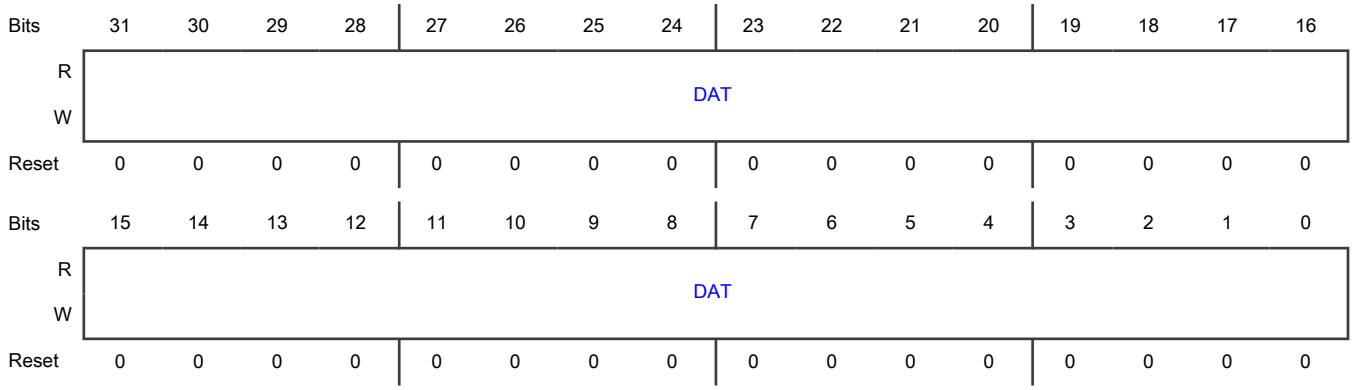
17.7.1.16 MISC Config (MISC_CFG)

Offset

Register	Offset
MISC_CFG	210h

Function
This shadow register is for miscellaneous configuration.

Diagram



Fields

Field	Function
31-0 DAT	Data Contains configuation data.

17.7.1.17 PHANTOM Config (PHANTOM_CFG)

Offset

Register	Offset
PHANTOM_CFG	214h

Function
This shadow register is for phantom configuration.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DAT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	DAT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 DAT	Data Contains configuation data.

17.7.1.18 Flexible Config 0 (FLEX_CFG0)

Offset

Register	Offset
FLEX_CFG0	218h

Function

This shadow register is for flexible configuration.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DAT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	DAT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 DAT	Data Contains configuation data.

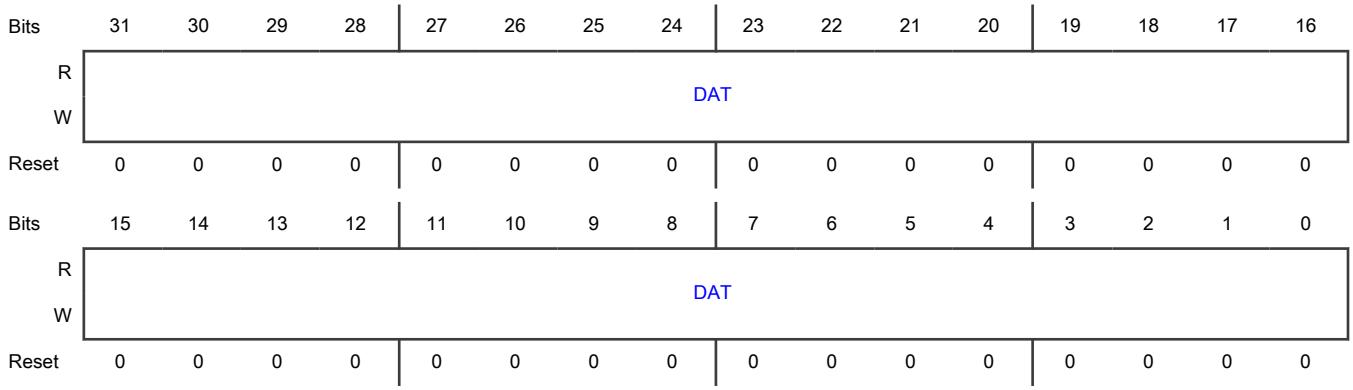
17.7.1.19 Flexible Config 1 (FLEX_CFG1)

Offset

Register	Offset
FLEX_CFG1	21Ch

Function
This shadow register is for flexible configuration.

Diagram



Fields

Field	Function
31-0 DAT	Data Contains configuation data.

Chapter 18
Code Watchdog Timer (CDOG)

18.1 Chip-specific CDOG information

Table 394. Reference links to related information[1]

Topic	Related module	Reference
Full description	CDOG	CDOG
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Power management		See the chapter "Power management"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

NOTE

The reset values of CONTROL, INSTRUCTION_TIMER, and FLAGS registers may change depending on the device's boot settings.

18.1.1 Module instances

This device has two instances of the CDOG module, CDOG0 and CDOG1.

NOTE

Both the CDOG modules are interfaced to the main CM33 core (CPU0). Clock for both the CDOG modules are the same as CPU0 clock and are gated when CPU0 enters sleep mode. The interrupt pause feature (CONTROL[IRQ_PAUSE]) is also connected to CPU0 for both of the CDOG instantiations.

18.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

18.2 Overview

CDOG helps protect the integrity of software by detecting unexpected changes (faults) in code execution flow. This module can be configured to reset or interrupt the processor core when it detects a fault.

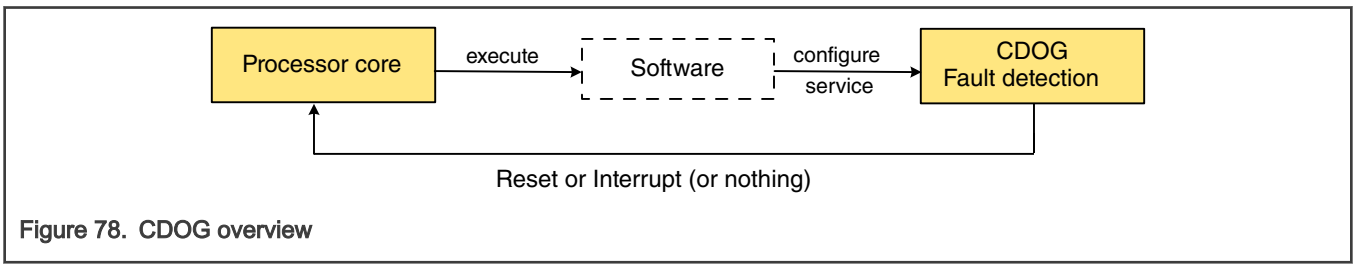


Figure 78. CDOG overview

18.2.1 Block diagram

The following block diagram shows the components of CDOG.

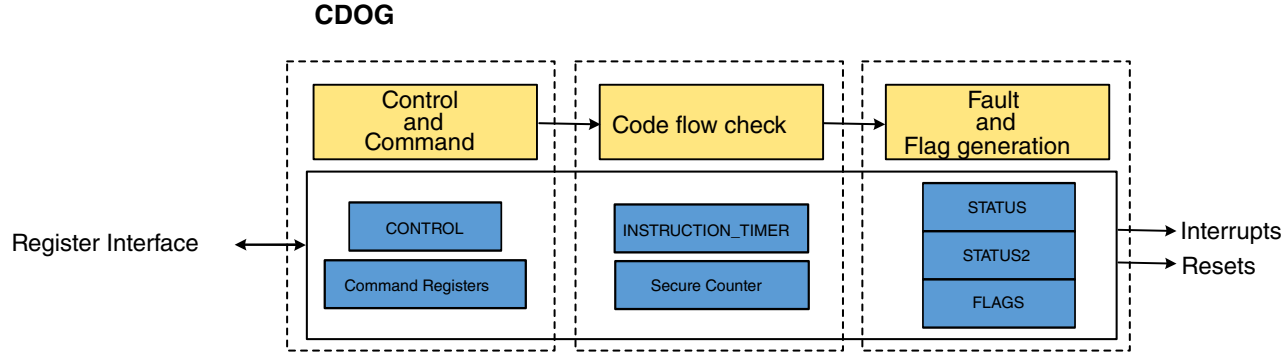


Figure 79. Block diagram of CDOG

18.2.2 Features

CDOG includes the following code flow and data integrity checking:

- Faults and flags configurable to generate a system reset, interrupts, or nothing
- Counters for statistics on code behavior patterns for fault types

18.3 Functional description

The following sections describe functional details of this module.

18.3.1 Code flow checking

CDOG provides two primary mechanisms for detecting low-cost fault attacks and the execution of unexpected instruction sequences:

- Secure Counter
- Instruction Timer (INSTRUCTION_TIMER)

18.3.1.1 Secure counter

The Secure Counter is a 32-bit accumulator that holds a dynamically changing value that can periodically be evaluated to determine if a program is executing as expected. If a mismatch is detected, a fault is generated.

- Secure Counter is an accumulator that when loaded with an initial value lets runtime software issue ADD & SUB commands to increment/decrement the counter.
- Periodically, a Secure Counter value check is initiated by writing the STOP, RESTART, or ASSERT16 command register. The value written to STOP, RESTART, or ASSERT16 is compared with the current value of the Secure Counter.
- If a mismatch is detected between the Secure Counter and the value written to STOP, RESTART, or ASSERT16 command register, the execution flow has potentially been altered by a fault attack or some other suspicious activity.

18.3.1.2 Instruction timer

The Instruction Timer is a 32-bit count-down timer that an application uses to set the number of instructions that software expects to execute. The Instruction Timer counts off the instructions as they are executed (clocks) and if the number is exhausted before the CDOG is serviced, a fault is generated.

The application programmer pre-loads the Instruction Timer with slightly more than the number of instructions in the next execution sequence. The CDOG detects cases where the counter is reset to 0 before all instructions execute, which may indicate that unauthorized instructions are being executed.

- Instruction Timer places a hard upper-limit on the interval between checks of the Secure Counter.

- The Instruction Timer can be programmed to either pause, or keep running while the interrupt service routines are executing.
- The START command loads the internal decremental counter. Before the counter generates an underflow (reaches 0), a STOP or RESTART command must be executed to force a Secure Counter check.

18.3.2 Modes of operation (STATE)

The CDOG has two legal states: IDLE and ACTIVE.

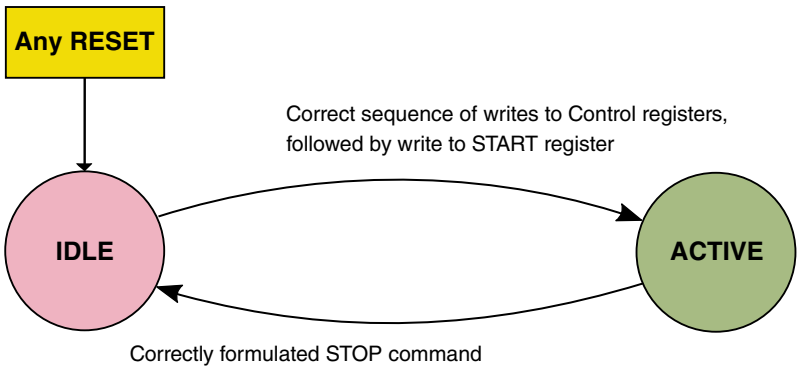


Figure 80. CDOG state diagram

- The two states are encoded in the read-only STATUS[CURST] field.
- After any reset (including a reset generated by the CDOG itself), the module will be in IDLE state.
- To change the module state to ACTIVE, software must execute a correct sequence of writes to the CONTROL group, followed by a START command. See Example use cases.
- Once ACTIVE, a correctly formulated STOP command will change the state back to IDLE.

18.3.3 Faults, flags, and counters

18.3.3.1 Fault types

All fault types (except CONTROL) can be individually controlled to generate either a system reset, an interrupt, or nothing.

Table 395. Fault types

This fault...	Occurs when...
TIMEOUT	The Instruction Timer reaches '0'.
MISCOMPARE	Either a STOP or a RESTART command is issued, and the value passed in the instruction does not match the content of the Secure Counter. ASSERT16 command is issued, and the lower 16-bit value passed in the instruction does not match the lower 16-bit value of the Secure Counter.
SEQUENCE	The prescribed sequence of interactions between software and CDOG is violated.
CONTROL	Any of the CONTROL register's fields contain an illegal value.
STATE	The internal state machine contains any value other than 5h or Ah.
ADDRESS	When an undefined address in the module's register space is accessed.

Table continues on the next page...

Table 395. Fault types (continued)

This fault...	Occurs when...
	NOTE Address 0xC is reserved, but no ADDRESS fault will be generated when 0xC is accessed.

For CONTROL fault, a system reset is requested when FLAGS[CNT_FLAG] = 1 and CONTROL[LOCK_CTRL] != 10b.

Lock the CONTROL register by writing CONTROL[LOCK_CTRL] == 01b at the same time that the desired, valid configuration is written. This blocks additional writes, and prevents CONTROL faults.

Interrupts are available as an alternative to system reset generation, for all fault types (except CONTROL), primarily to facilitate code development and debug operations. In the final application, interrupts can be enabled for some faults, and reset can be enabled for other faults (or disabled completely), at the risk of reduced security.

18.3.3.2 Fault counters

Each fault type has an associated counter that increments when the fault is detected. The counters are cleared by POR, but not by a system reset. Thus, statistics can be built up over many code watchdog resets to reveal the behavior patterns of a specific type of attack.

18.3.3.3 Flags

Each fault type has an associated flag accessible through the FLAGS register. A flag sets whenever its associated fault is detected, and can be cleared by software. The flags themselves (when enabled) generate the module's system reset or interrupt outputs.

The flags retain their value through a system reset (including one generated by the CDOG), but are reset by a POR. Using this mechanism, software can easily answer the 'How did I get here?' question by reading the FLAGS register after any reset.

To facilitate testing and code development, write to the flags directly when the module is not locked (CONTROL[LOCK_CTRL] = 10b). When the module is locked (CONTROL[LOCK_CTRL] = 01b), the flags can be cleared by writing '1' to their bit positions.

Table 396. FLAGS summary

Name	POR	ADDR	STATE	CONTROL	SEQUENCE	MISCOMPARE	TIMEOUT
Bit	16	5	4	3	2	1	0
Reset value after POR?	1	0	0	0	0	0	0
Writeable when unlocked?	Y	Y	Y	Y	Y	Y	Y
W1C when locked?	Y	Y	Y	Y	Y	Y	Y

18.3.3.4 Fault generation

Some faults are related to when and how registers can be accessed. See Figure 81.

The CDOG supports the following faults:

Table 397. Fault generation

This fault...	Is generated when...
SEQUENCE	• Software does not write to RELOAD before the START command is issued. • Software writes to RELOAD in ACTIVE state. Only write to RELOAD in IDLE state. • Software attempts to write to START in ACTIVE state. Only write to START in IDLE state. • In IDLE state, software attempts to write to a COMMAND group register other than START. • In ACTIVE state, software attempts to write to a CONTROL group register.
ADDRESS	• A read access (within the CDOG address space) to any address outside the CONTROL register group. • A write access (within the CDOG address space) to any address outside both the CONTROL and COMMAND register groups. **NOTE** Address 0xC is reserved in the address space of the CONTROL register group.
TIMEOUT	• The countdown is one clock before the Instruction Timer reaches '0'.
MISCOMPARE	• The STOP register is written and the current Secure Counter value does not match the write data. • The RESTART register is written and the current Secure Counter value does not match the write data. • The ASSERT16 register is written and the current Secure Counter lower 16-bit value does not match the 16-bit write data.
CONTROL	• The CONTROL[LOCK_CTRL] field is any value other than 01b or 10b. • The CONTROL bit fields: TIMEOUT_CTRL, MISCOMPARE_CTRL, SEQUENCE_CTRL, STATE_CTRL, or ADDRESS_CTRL contain any value other than 001b, 010b, or 100b. • The CONTROL[DEBUG_HALT_CTRL] field contains any value other than 01b or 10b. • The CONTROL[IRQ_PAUSE] field contains any value other than 01b or 10b.
STATE	• The STATE machine contains any value other than 5h or Ah. See STATUS[CURST].

18.3.4 Clocking

CDOG has only one clock input. See the chip-specific CDOG section for information on available clock inputs and the default option for a specific device.

18.3.5 Reset

CDOG has two hardware reset input sources: power-on reset (POR) and system reset.

CDOG can generate a system reset using the CONTROL register. See Faults, flags, and counters for details on how CDOG generates a system reset.

18.3.5.1 Power on reset (POR)

The logic and all registers of this module are reset to their default states on a POR.

18.3.5.2 System reset

The logic and most of the registers of this module are reset to their default states on a system reset.

The following registers cannot be reset by a system reset:

- Persistent Data Storage (PERSISTENT)
- Flags (FLAGS)
- Status 1 (STATUS), except the STATUS[CURST] field
- Status 2 (STATUS2)

18.3.6 Interrupts

CDOG has only one interrupt output. The interrupt signal output by CDOG can be connected to the system's interrupt controller. The following table lists CDOG's interrupt sources:

Table 398. Interrupt Summary

Interrupt flag	Interrupt enable	Description
FLAGS[TO_FLAG]	CONTROL[TIMEOUT_CTRL] = 3'b010	TIMEOUT fault interrupt
FLAGS[MISCOM_FLAG]	CONTROL[MISCOMPARE_CTRL] = 3'b010	MISCOMPARE fault interrupt
FLAGS[SEQ_FLAG]	CONTROL[SEQUENCE_CTRL] = 3'b010	SEQUENCE fault interrupt
FLAGS[STATE_FLAG]	CONTROL[STATE_CTRL] = 3'b010	STATE fault interrupt
FLAGS[ADDR_FLAG]	CONTROL[ADDRESS_CTRL] = 3'b010	ADDRESS fault interrupt

18.4 Application information

This section gives examples on how to program CDOG to monitor code execution flow.

18.4.1 Example use cases

There is generally a strict sequence to configure the CDOG, activated and serviced as follows:

1. Write an Instruction Timer reload value, corresponding to the current code section, to the Instruction Timer Reload Register (RELOAD) register.
2. Write a control word to the Control Register (CONTROL) register, where each fault type is configured to generate a reset, interrupt, or neither. Then lock the Control register using the CONTROL[LOCK_CTRL] field. Instruction Timer Register (INSTRUCTION_TIMER) may be always kept running, or run only during non-IRQ execution (paused during interrupt processing) based on the value in the CONTROL[IRQ_PAUSE] field.
3. Activate the module by writing to the START Command Register (START) register. The initial value for the Secure Counter is the value written. Instruction Timer Register (INSTRUCTION_TIMER) immediately starts decrementing from the RELOAD value on every clock cycle.
4. At strategically chosen way points in the code flow of the application, update the Secure Counter by issuing ADD or SUB commands, or assert the lower 16-bit value of the Secure Counter by the ASSERT16 command.

5. When the way point that corresponds to the number of executed instructions represented by the RELOAD value (the end of the current code section) is reached, write the expected value of the Secure Counter to the STOP command register. Assuming the written value compares exactly to the contents of the Secure Counter, the instruction timer stops. Then the process can begin again for the next code section, by repeating steps 1 through 5.

Another example of a configuration and start procedure is as follows:

1. Write the Instruction Timer Register (INSTRUCTION_TIMER) reload value to the Instruction Timer Reload Register (RELOAD) register. Write a very large number to provide a buffer for large values.
2. Write a control word to the Control Register (CONTROL) register, where each fault type is configured to generate a reset, interrupt, or neither. Then lock the Control register using the CONTROL[LOCK_CTRL] field.
3. Activate the module by writing to the START Command Register (START) register. The initial value for the Secure Counter is the value written. INSTRUCTION_TIMER immediately starts decrementing from the RELOAD value on every clock.
4. At strategically chosen way points in the code flow of the application, update the Secure Counter by issuing ADD or SUB commands, or assert the lower 16-bit value of the Secure Counter by the ASSERT16 command.
5. During the execution flow, the application must always be aware of the INSTRUCTION_TIMER register value approach toward 0.
    a. Read the INSTRUCTION_TIMER value regularly to determine the expended time.
    b. Before the INSTRUCTION_TIMER value reaches 0, software must service the CDOG by writing the Secure Counter expected value to the RESTART command register.
    c. Assuming the value written compares exactly to the contents of the Secure Counter, INSTRUCTION_TIMER reloads with the value in the RELOAD register as it decrements toward 0.

18.4.2 Using CDOG during code development debug

Following are suggestions for facilitating the code development and debug cycle:

- Maintain control while adjusting the timing and fine-tuning of the counting values by using interrupt-on-fault (instead of reset).
- Trigger faults by direct-writing the bits in the FLAGS register with the same result as hardware-triggered faults.
- Pause the instruction timer during a pause of a debug session by using the CONTROL[DEBUG_HALT_CTRL] field.

18.5 Memory map and register definition

This section includes the memory map and detailed descriptions of all registers of this module.

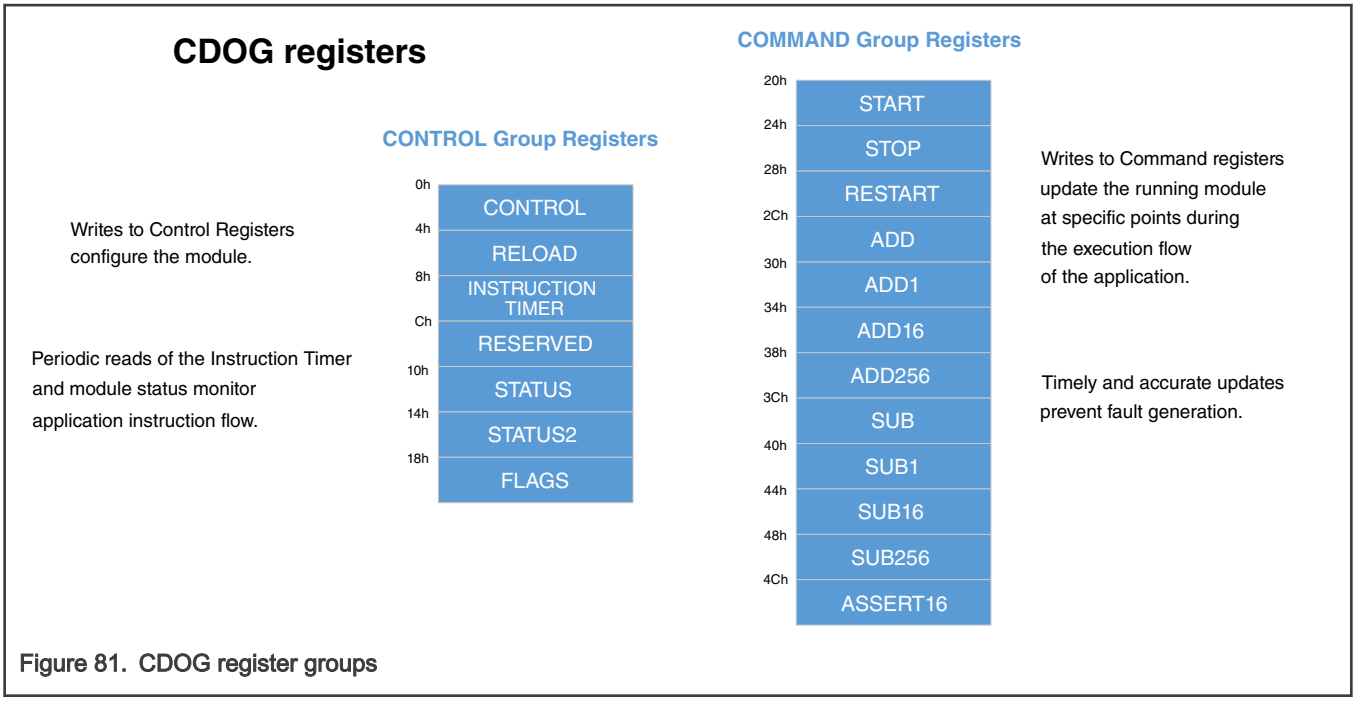


Figure 81. CDOG register groups

18.5.1 CDOG register descriptions

18.5.1.1 CDOG memory map

CDOG0 base address: 400B_B000h

CDOG1 base address: 400B_C000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Control Register (CONTROL)	32	RW	5009_2492h
4h	Instruction Timer Reload Register (RELOAD)	32	RW	FFFF_FFFFh
8h	Instruction Timer Register (INSTRUCTION_TIMER)	32	R	FFFF_FFFFh
10h	Status 1 Register (STATUS)	32	R	5000_0000h
14h	Status 2 Register (STATUS2)	32	R	0000_0000h
18h	Flags Register (FLAGS)	32	RW	0001_0000h
1Ch	Persistent Data Storage Register (PERSISTENT)	32	RW	0000_0000h
20h	START Command Register (START)	32	W	0000_0000h
24h	STOP Command Register (STOP)	32	W	0000_0000h
28h	RESTART Command Register (RESTART)	32	W	0000_0000h
2Ch	ADD Command Register (ADD)	32	W	0000_0000h
30h	ADD1 Command Register (ADD1)	32	W	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
34h	ADD16 Command Register (ADD16)	32	W	0000_0000h
38h	ADD256 Command Register (ADD256)	32	W	0000_0000h
3Ch	SUB Command Register (SUB)	32	W	0000_0000h
40h	SUB1 Command Register (SUB1)	32	W	0000_0000h
44h	SUB16 Command Register (SUB16)	32	W	0000_0000h
48h	SUB256 Command Register (SUB256)	32	W	0000_0000h
4Ch	ASSERT16 Command Register (ASSERT16)	32	W	0000_0000h

18.5.1.2 Control Register (CONTROL)

Offset

Register	Offset
CONTROL	0h

Function

The Control register (CONTROL) contains all the controllable attributes of the module, such as how the CDOG module responds when detecting a fault.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DEBUG_HALT_CTRL		IRQ_PAUSE		0								ADDRESS_CTRL			STATE_C...
W																
Reset	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	STATE_CTRL		Reserved		SEQUENCE_CTRL				MISCOMPARE_CTRL				TIMEOUT_CTRL			LOCK_CTRL
W																
Reset	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0

Fields

Field	Function
31-30	DEBUG_HALT control Controls whether the Instruction Timer runs or stops when the CPU asserts DEBUG_HALT.

Table continues on the next page...

Table continued from the previous page...

Field	Function
DEBUG_HALT_CTRL	See chip-specific CDOG section for information on DEBUG_HALT connections for a specific device. 01b - Keep the timer running 10b - Stop the timer
29-28 IRQ_PAUSE	IRQ pause control Controls whether the Instruction Timer runs or stops when the CPU asserts IRQ_PAUSE. See chip-specific CDOG section for information on IRQ_PAUSE connections for a specific device. 01b - Keep the timer running 10b - Stop the timer
27-20 —	Reserved
19-17 ADDRESS_CTRL	ADDRESS fault control Controls how the CDOG module responds when detecting an ADDRESS fault (FLAGS[ADDR_FLAG]). 001b - Enable reset 010b - Enable interrupt 100b - Disable both reset and interrupt
16-14 STATE_CTRL	STATE fault control Controls how the CDOG module responds when detecting a STATE fault (FLAGS[STATE_FLAG]). 001b - Enable reset 010b - Enable interrupt 100b - Disable both reset and interrupt
13-11 —	Reserved Do not change its value.
10-8 SEQUENCE_CTRL	SEQUENCE fault control Controls how the CDOG module responds when detecting a SEQUENCE fault (FLAGS[SEQUENCE_FLAG]). 001b - Enable reset 010b - Enable interrupt 100b - Disable both reset and interrupt
7-5 MISCOMPARE_CTRL	MISCOMPARE fault control Controls how the CDOG module responds when detecting a MISCOMPARE fault (FLAGS[MISCOMPARE_FLAG]).

Table continues on the next page...

Table continued from the previous page...

Field	Function
	001b - Enable reset 010b - Enable interrupt 100b - Disable both reset and interrupt
4-2 TIMEOUT_CTRL	TIMEOUT fault control Controls how the CDOG module responds when detecting a TIMEOUT fault (FLAGS[TIMEOUT_FLAG]). 001b - Enable reset 010b - Enable interrupt 100b - Disable both reset and interrupt
1-0 LOCK_CTRL	Lock control Resets to unlocked. Once locked, LOCK_CTRL remains locked until the next system reset. The Control register is writable and readable only when unlocked, LOCK_CTRL = 10. Once locked, the written value to the Control register is ignored, and reading the Control register returns 0. 01b - Locked 10b - Unlocked

18.5.1.3 Instruction Timer Reload Register (RELOAD)

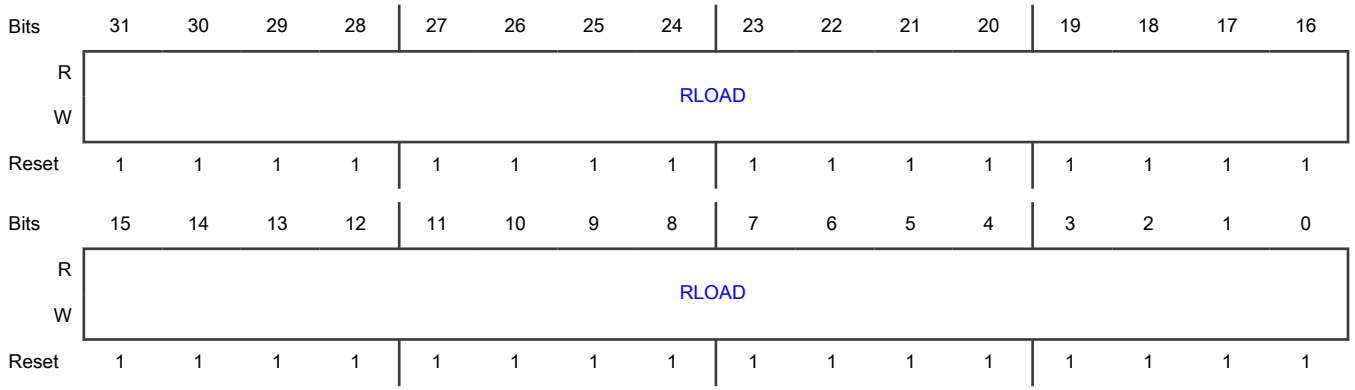
Offset

Register	Offset
RELOAD	4h

Function

The Instruction Timer RELOAD register contains the value from which the Instruction Timer counts down when a valid START or RESTART command is executed. Write to RELOAD only when the module is in the IDLE state. A write to RELOAD must occur before issuing a START command.

Diagram



Fields

Field	Function
31-0 RLOAD	Instruction Timer reload value Contains the starting countdown value used by the Instruction Timer whenever a START or RESTART command is executed.

18.5.1.4 Instruction Timer Register (INSTRUCTION_TIMER)

Offset

Register	Offset
INSTRUCTION_TIMER	8h

Function

The Instruction Timer register (INSTRUCTION_TIMER) contains the current count value of the Instruction Timer.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	INSTIM															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INSTIM															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-0 INSTIM	Current value of the Instruction Timer The current value of the timer can be read from this address. The Software cannot write to INSTRUCTION_TIMER.

18.5.1.5 Status 1 Register (STATUS)

Offset

Register	Offset
STATUS	10h

Function

The Status 1 register (STATUS) holds the current module status and fault counts. The fields (except CURST) in STATUS are reset only by POR.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CURST				0				NUMILSEQF							
W																
Reset	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	NUMMISCOMPF								NUMTOF							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-28 CURST	Current State Indicates the current state of the module. The only legal states are: • 0x5 IDLE • 0xA ACTIVE Any other value generates a STATE fault (FLAGS[STATE_FLAG]). Resets to 0x5 = IDLE on any reset (unlike other bits in this register that only reset on POR).
27-24 —	Reserved
23-16 NUMILSEQF	Number of SEQUENCE faults (FLAGS[SEQUENCE_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.
15-8 NUMMISCOMPF	Number of MISCOMPARE faults (FLAGS[MISCOMPARE_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.
7-0 NUMTOF	Number of TIMEOUT faults (FLAGS[TIMEOUT_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.

18.5.1.6 Status 2 Register (STATUS2)

Offset

Register	Offset
STATUS2	14h

Function

Status 2 register (STATUS2) holds additional fault counts. The fields in STATUS2 are reset only by POR.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0								NUMILLA							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	NUMILLSTF								NUMCNTF							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-24 —	Reserved
23-16 NUMILLA	Number of ADDRESS faults (FLAGS[ADDR_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.
15-8 NUMILLSTF	Number of STATE faults (FLAGS[STATE_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.
7-0 NUMCNTF	Number of CONTROL faults (FLAGS[CONTROL_FLAG]) since the last POR Resets to 0. Will not increment past 0xFF.

18.5.1.7 Flags Register (FLAGS)

Offset

Register	Offset
FLAGS	18h

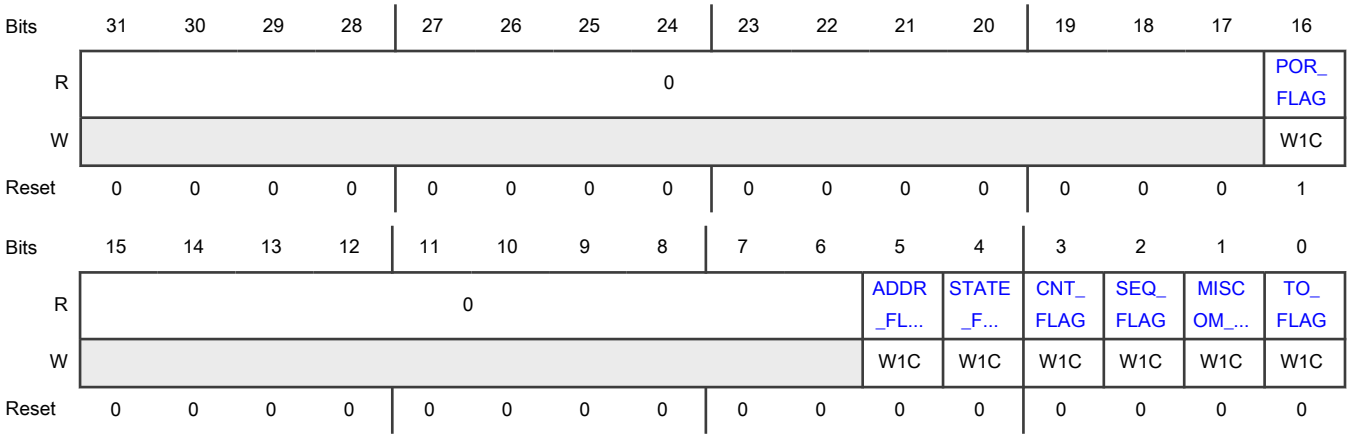
Function

The Flags register (FLAGS) contains the fault detection flags, as well as a POR event flag. The fault detection flags may generate a reset or an interrupt as configured in the CONTROL register.

The FLAGS fields retain their value through a system reset, including one generated by the CDOG. However, FLAGS is reset by a POR.

Writability of the fields depends on the module state. See Flags.

Diagram



Fields

Field	Function
31-17 —	Reserved
16 POR_FLAG	Power-on reset flag POR_FLAG is set by a Power-on reset event. 0b - A Power-on reset event has not occurred 1b - A Power-on reset event has occurred
15-6 —	Reserved
5 ADDR_FLAG	ADDRESS fault flag Hardware sets ADDR_FLAG when CDOG detects an ADDRESS fault. 0b - An ADDRESS fault has not occurred 1b - An ADDRESS fault has occurred
4 STATE_FLAG	STATE fault flag Hardware sets STATE_FLAG when CDOG detects a STATE fault. 0b - A STATE fault has not occurred

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - A STATE fault has occurred
3 CNT_FLAG	CONTROL fault flag Hardware sets CNT_FLAG when CDOG detects a CONTROL fault. 0b - A CONTROL fault has not occurred 1b - A CONTROL fault has occurred
2 SEQ_FLAG	SEQUENCE fault flag Hardware sets the SEQ_FLAG when CDOG detects a SEQUENCE fault. 0b - A SEQUENCE fault has not occurred 1b - A SEQUENCE fault has occurred
1 MISCOM_FLAG	MISCOMPARE fault flag Hardware sets MISCOM_FLAG when CDOG detects a MISCOMPARE fault. 0b - A MISCOMPARE fault has not occurred 1b - A MISCOMPARE fault has occurred
0 TO_FLAG	TIMEOUT fault flag Hardware sets TO_FLAG when CDOG detects a TIMEOUT fault. 0b - A TIMEOUT fault has not occurred 1b - A TIMEOUT fault has occurred

18.5.1.8 Persistent Data Storage Register (PERSISTENT)

Offset

Register	Offset
PERSISTENT	1Ch

Function

The Persistent Data Storage register (PERSISTENT) provides an application with 32 bits of scratchpad storage for information that needs to persist through resets other than a Power-On Reset (POR).

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PERSIS															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PERSIS															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 PERSIS	Persistent Storage A write value to PERSIS remains unchanged after any reset other than a POR.

18.5.1.9 START Command Register (START)

Offset

Register	Offset
START	20h

Function

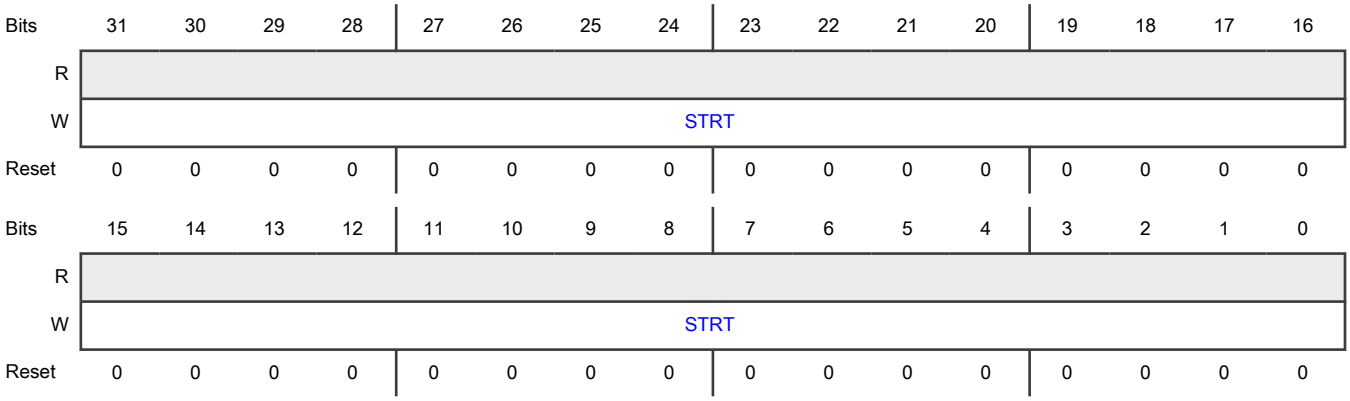
Writing to the Start Command register (START) issues a Start command:

- The value written to STRT is loaded into the Secure Counter as a start value.
- The RELOAD value is loaded into the Instruction Timer.
- The module enters the ACTIVE state in which the Instruction Timer counts down on every clock.

NOTE

Write to START only during the IDLE state, and only after writing to the RELOAD register. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 STRT	Start command The value written to STRT is loaded into the Secure Counter as a start value.

18.5.1.10 STOP Command Register (STOP)

Offset

Register	Offset
STOP	24h

Function

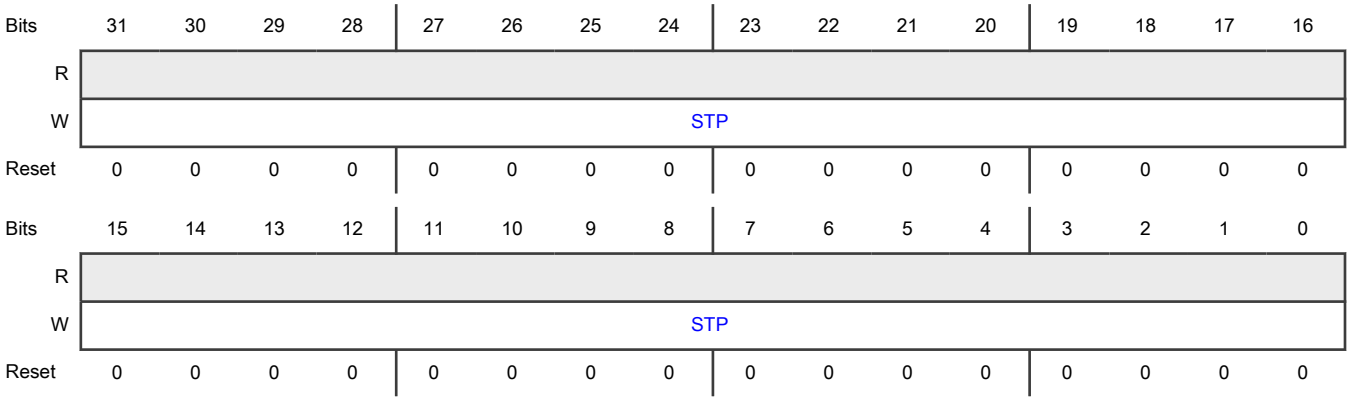
Writing to the Stop Command register (STOP) issues a Stop command:

- The Instruction Timer is stopped.
- The value written to STP is compared with the current value of the Secure Counter.

NOTE

Write to STOP only during the ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 STP	Stop command The value written to STP is compared with the current value of the Secure Counter.

18.5.1.11 RESTART Command Register (RESTART)

Offset

Register	Offset
RESTART	28h

Function

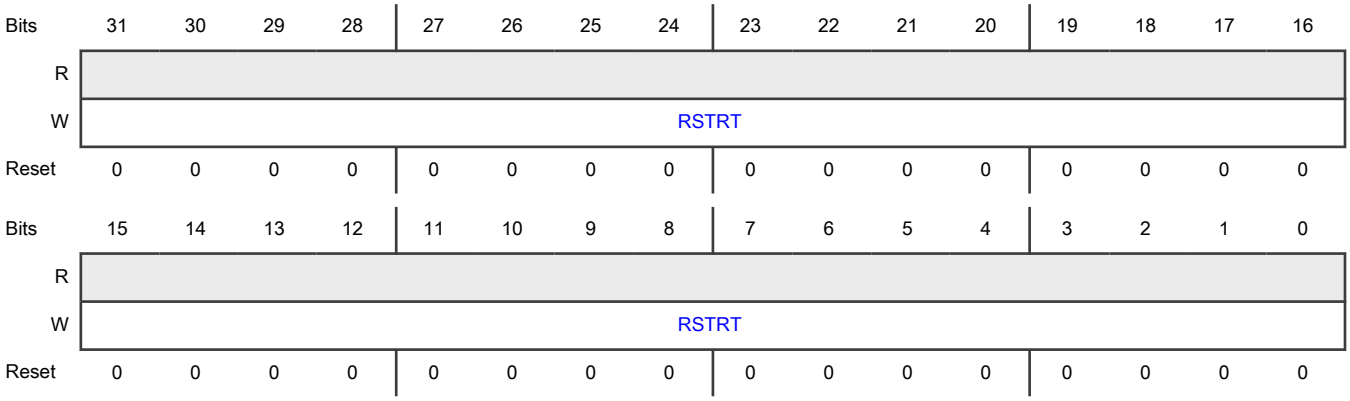
Writing to the Restart Command register (RESTART) issues a Restart command:

- The value written to RSTRT is compared with the current value of the Secure Counter.
- If the values match, the Instruction Timer is reloaded (with the RELOAD value) and starts counting down again.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 RSTRT	Restart command The value written to RSTRT is compared with the current value of the Secure Counter.

18.5.1.12 ADD Command Register (ADD)

Offset

Register	Offset
ADD	2Ch

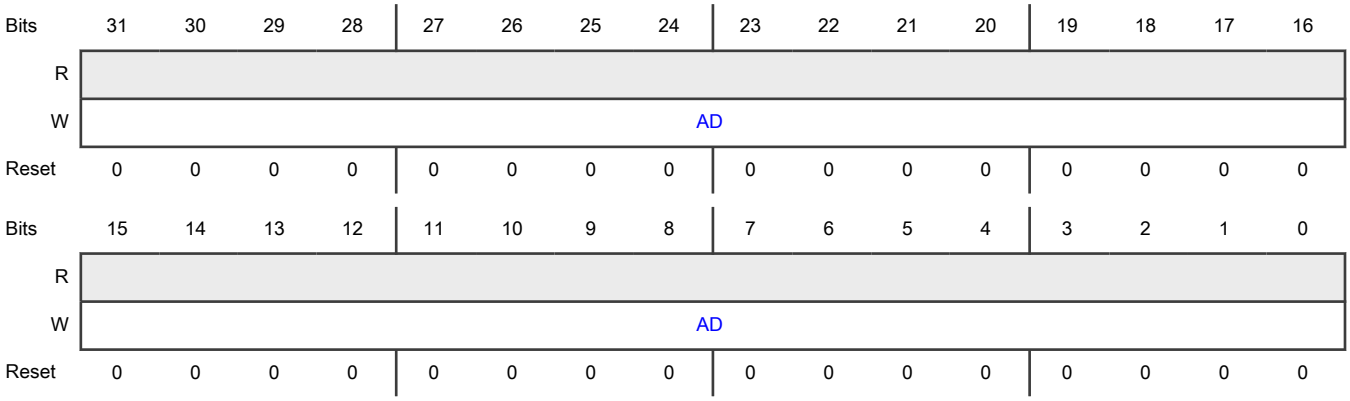
Function

Writing to the Add Command register (ADD) issues an Add command: The value written to AD is added to the current value of the Secure Counter

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 AD	ADD Write Value The value written to AD is added to the current value of the Secure Counter.

18.5.1.13 ADD1 Command Register (ADD1)

Offset

Register	Offset
ADD1	30h

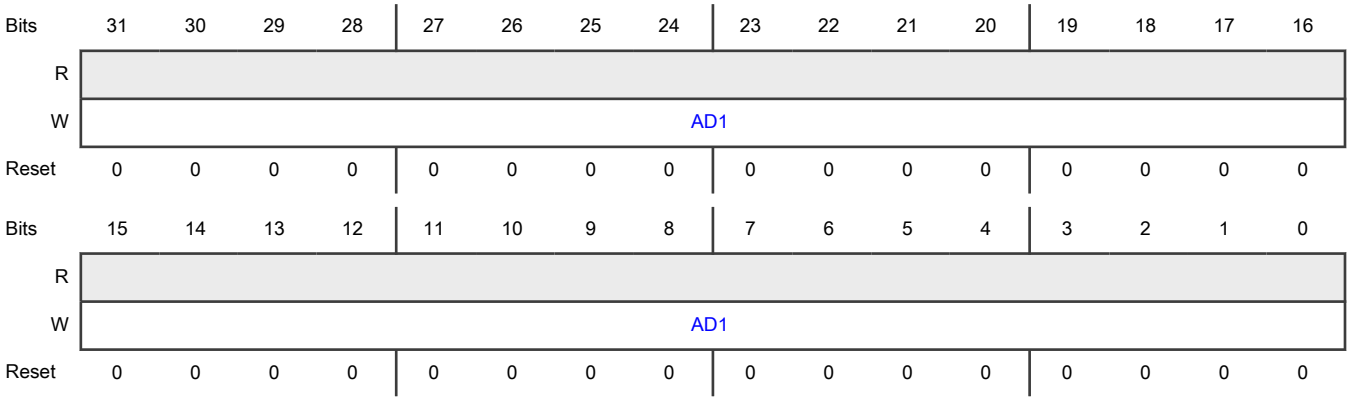
Function

Writing to the ADD1 Command register (ADD1) issues an ADD1 command: The value 1 is added to the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 AD1	ADD 1 Secure Counter is always incremented by 1. The value written to AD1 is ignored.

18.5.1.14 ADD16 Command Register (ADD16)

Offset

Register	Offset
ADD16	34h

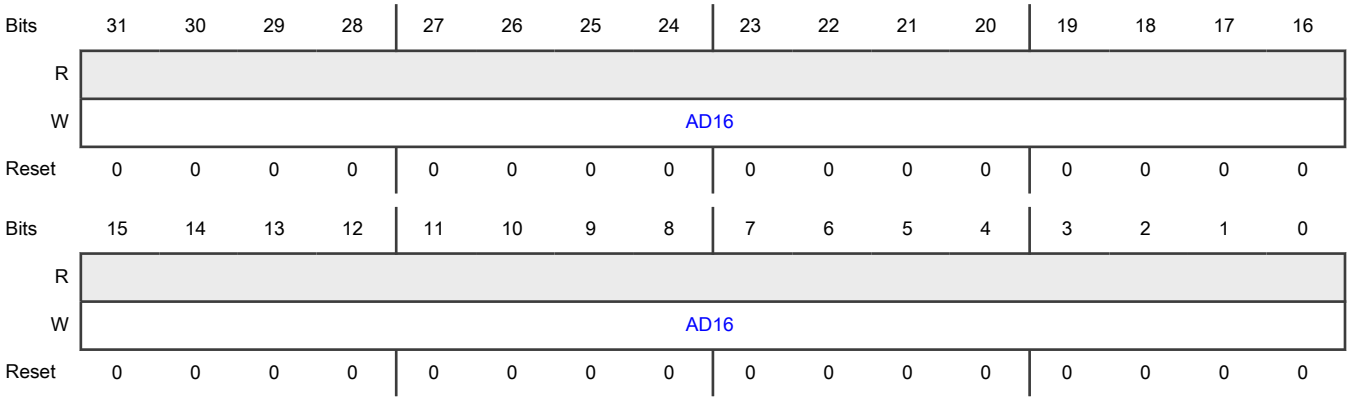
Function

Writing to the ADD16 Command register (ADD16) issues an ADD16 command: The value 16 is added to the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 AD16	ADD 16 Secure Counter is always incremented by 16. The value written to AD16 is ignored.

18.5.1.15 ADD256 Command Register (ADD256)

Offset

Register	Offset
ADD256	38h

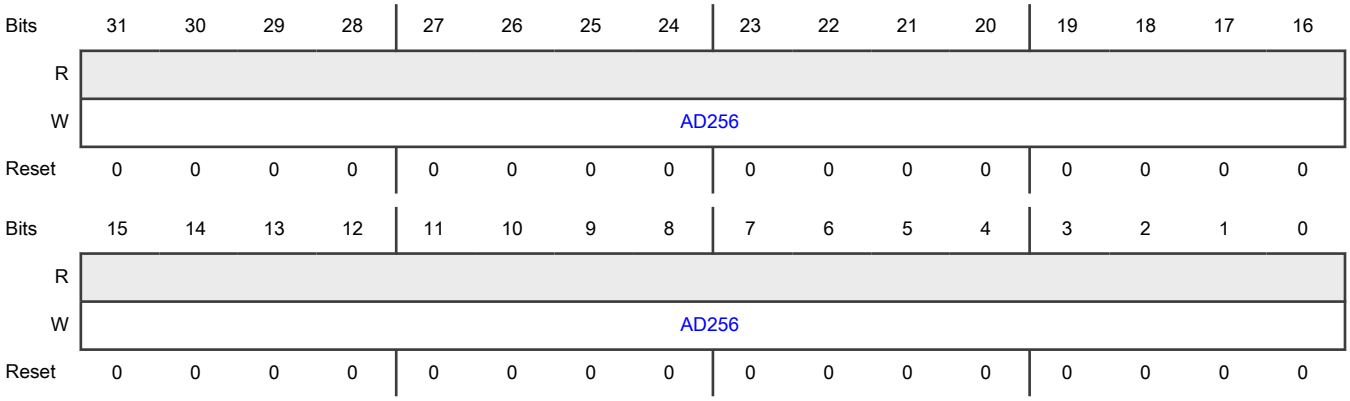
Function

Writing to the ADD256 Command register (ADD256) issues an ADD256 command: The value 256 is added to the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 AD256	ADD 256 Secure Counter is always incremented by 256. The value written to AD256 is ignored.

18.5.1.16 SUB Command Register (SUB)

Offset

Register	Offset
SUB	3Ch

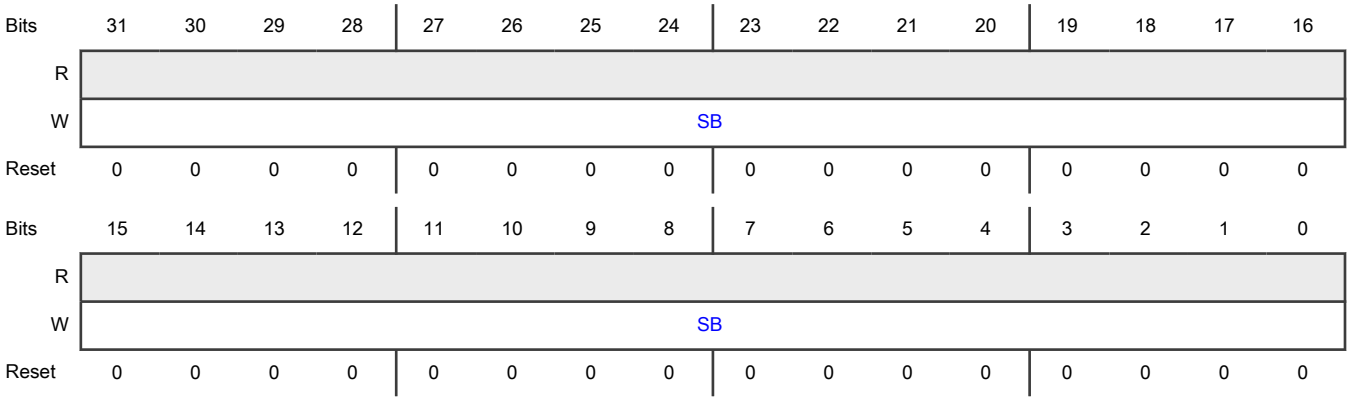
Function

Writing to the SUB Command register (SUB) issues a SUB command: The value written to SB is subtracted from the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 SB	Subtract Write Value The value written to SB is subtracted from the current value of the Secure Counter.

18.5.1.17 SUB1 Command Register (SUB1)

Offset

Register	Offset
SUB1	40h

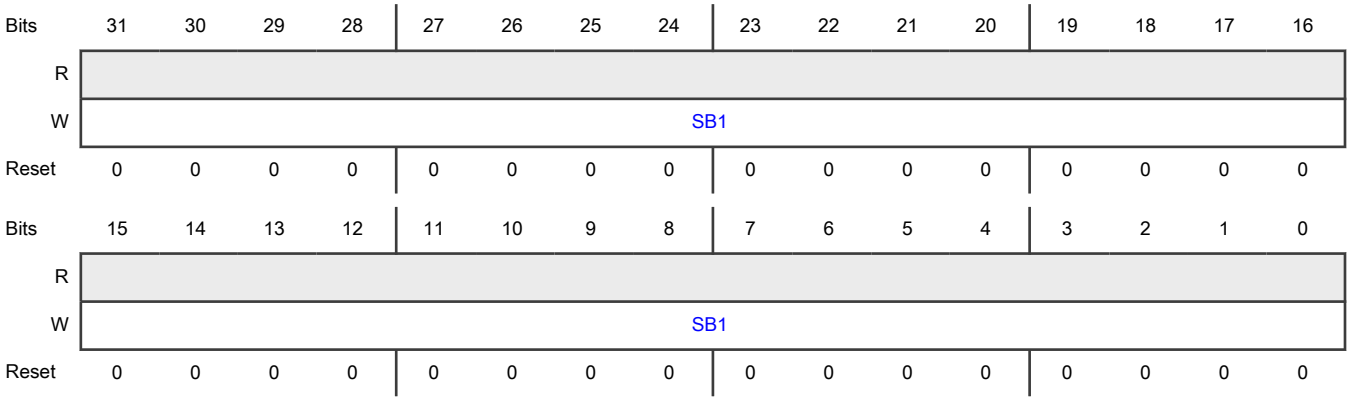
Function

Writing to the SUB1 Command register (SUB1) issues an SUB1 command: The value 1 is subtracted from the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 SB1	Subtract 1 Secure Counter is always decremented by 1. The value written to SB1 is ignored.

18.5.1.18 SUB16 Command Register (SUB16)

Offset

Register	Offset
SUB16	44h

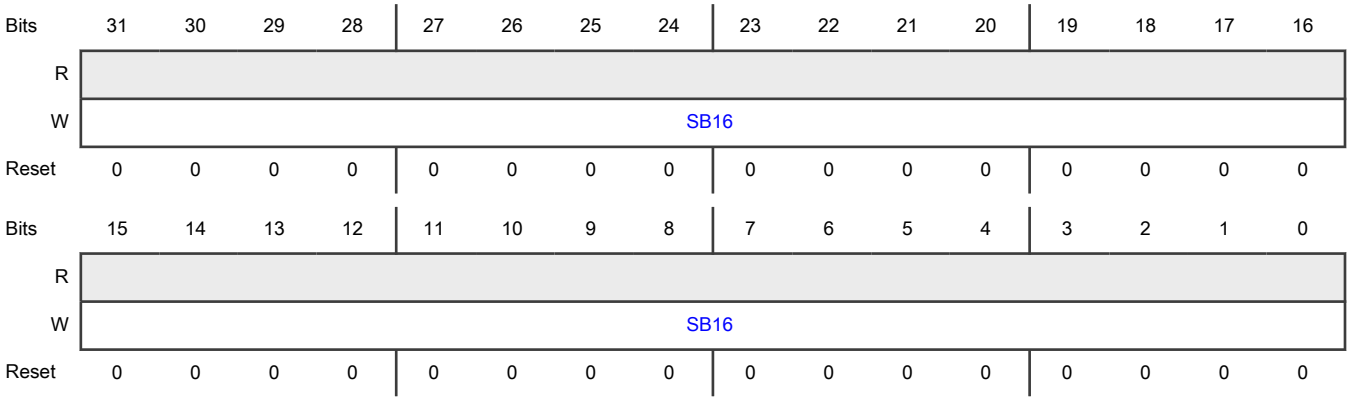
Function

Writing to the SUB16 Command register (SUB16) issues an SUB16 command: The value 16 is subtracted from the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 SB16	Subtract 16 Secure Counter is always decremented by 16. The value written to SB16 is ignored.

18.5.1.19 SUB256 Command Register (SUB256)

Offset

Register	Offset
SUB256	48h

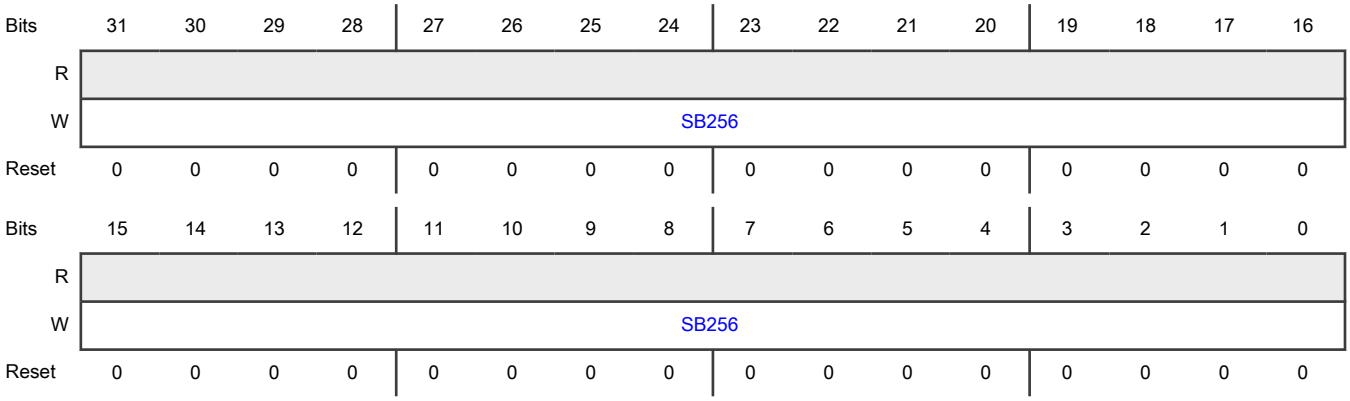
Function

Writing to the SUB256 Command register (SUB256) issues an SUB256 command: The value 256 is subtracted from the current value of the Secure Counter.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 SB256	Subtract 256 Secure Counter is always decremented by 256. The value written to SB256 is ignored.

18.5.1.20 ASSERT16 Command Register (ASSERT16)

Offset

Register	Offset
ASSERT16	4Ch

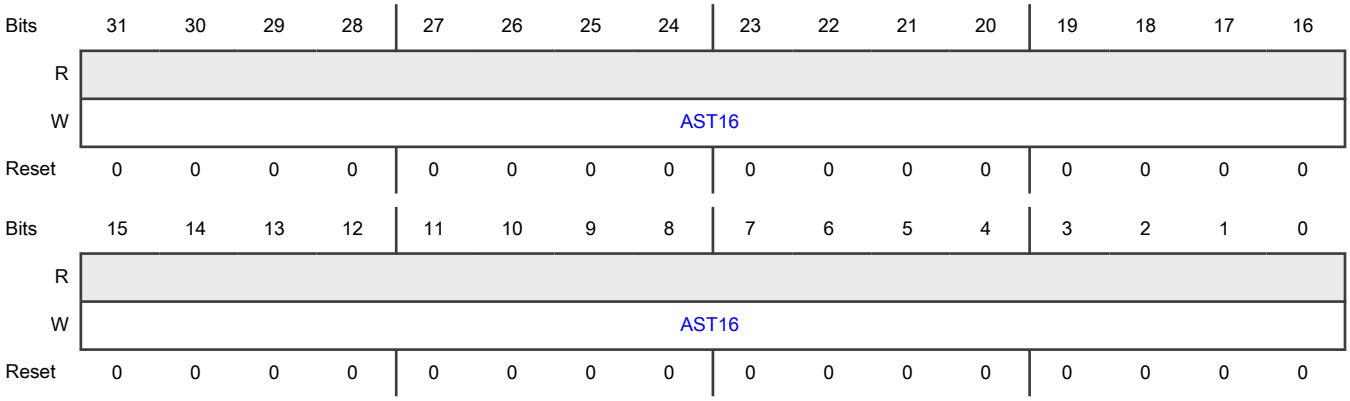
Function

Writing to the ASSERT16 Command register (ASSERT16) issues an ASSERT16 command: The lower 16-bit value written to AST16 is compared with the current lower 16-bit value of the Secure Counter. This command does not affect the Instruction Timer.

NOTE

Can only be written during ACTIVE state. A SEQUENCE fault (FLAGS[SEQUENCE_FLAG]) generates if this specified sequence is not followed.

Diagram



Fields

Field	Function
31-0 AST16	ASSERT16 Command The lower 16-bit value written to AST16 is compared with the current lower 16-bit value of the Secure Counter.

Chapter 19
Glitch Detect (GDET)

19.1 Chip-specific GDET information

Table 399. Reference links to related information[1]

Topic	Related module	Reference
Full description	GDET	GDET
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

NOTE

The device has analog voltage glitch detector (aGDET) and two digital voltage glitch detectors (GDET) which enhance protection against voltage manipulation. The NXP boot ROM configures aGDET and GDET during device initialization to reset the device when voltage manipulation is detected. The aGDET and GDET were validated on silicon across multiple process, voltage, and temperature corners with maximum noise profile generated by activating a large number of possible boot paths. To avoid false triggering due to unforeseen environment or application behavior, NXP recommends disabling aGDET and GDET in early start-up code. aGDET and/or GDET can be used by the application when security-critical operations are taking place, e.g., cryptographic operation using private or symmetric keys (digital signature, encryption, decryption, etc.), or sensitive decision-making process. However, characterization of the system noise profile and interactions with the sensors are required.

19.1.1 Module instances

This device has two instances of the GDET module, GDET0 and GDET1.

Each GDET protects one VDD_CORE pin. Both GDETs should be initialized by ROM during boot.

19.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

19.1.3 GDET configuration

GDET is an on-chip sensor that enables protection of the chip from glitch attacks. The Boot ROM configures and enables the GDET during device initialization.

Glitch detection is based on the expected core voltage (CORE_VDD), so the GDET module must be reconfigured if the core voltage is changed. Refer to GDET voltage mode change for the steps to configure the GDET voltage selection. For this device, the mapping between the GDET_DLY_CTRL[VOL_SEL] values and the core voltage are mentioned in the table below.

NOTE

Other than using the procedure in GDET voltage mode change when changing CORE_VDD, do not change the configuration of the GDET registers.

Table 400. GDET_DLY_CTRL[VOL_SEL] value and core voltage mapping

GDET_DLY_CTRL[VOL_SEL] value	Core voltage
2'b00	Mid Voltage (1.0 V)
2'b01	Normal Voltage (1.1 V)
2'b10	Over Drive Voltage (1.2 V)
2'b11	Reserved

This chip provides various registers in different modules to configure the GDET module. See the list below for the references to such registers:

- The following SYSCON register can be used to control the GDET reference clock:
  - REF_CLK_CTRL[GDET_REFCLK_EN]
  - REF_CLK_CTRL[GDET_REFCLK_EN_SET]
  - REF_CLK_CTRL[GDET_REFCLK_EN_CLR]
- The following SYSCON registers contain both control and status bits for GDET:
  - GDET0_CTRL
  - GDET1_CTRL

19.2 Overview

Glitch attacks are easy to apply as an attacker does not need to open the package of a chip. The required attacker equipment is cheap but extremely powerful. By glitch attacks, a hacker can cause the CPU to “skip” commands and cause to ignore certain checks in the software.

To protect against them, a detector connected to the supply lines is an appropriate way.

19.2.1 Features

- Glitch protection
- System level noise resistance
- Standard APB slave register interface for programming
- Auto trim support
- Error Management
  - Dedicated interrupt / error line for positive and negative events
  - Gray-coded based event counter for error counting
- Support for multiple voltage and reference clock modes

19.3 Architecture Description

The GDET is composed of a register bank accessible through a standard APB interface and a core block.

In the core block, the glitch detectors are instantiated.

19.3.1 Functional Description

The GDET core monitors the propagation delay of the logic elements in the design. Variations of the propagation delay beyond the defined window are flagged as an error.

19.3.1.1 GDET Startup

The GDET must be configured before been able to start monitoring glitch events.

To configure the GDET the host shoud read the GDET configuration from the OTP memory of the SoC and write into the GDET memory map.

A GDET configuration is formed of the setting for the 7 registers below:

- GDET_CONF_0
- GDET_CONF_1
- GDET_CONF_2
- GDET_CONF_3
- GDET_CONF_4
- GDET_CONF_5
- GDET_ENABLE1

NOTE

Please remember to write the GDET_ENABLE1 register only at the end of the configuration sequence to avoid the start of the GDET with a partial configuration

19.3.1.2 GDET voltage mode change

When the host requires to change the drive mode, the GDET must be informed accessing the GDET_DLY_CTRL register.

The sequence of steps that the host should use is shown below:

1. Set the SoC register[1] that is controlling the GDET isolate_on to be active.
2. Change the SoC voltage drive mode to the new level in the power control unit.
3. Change the GDET_DLY_CTRL to select the new drive mode (and set high also GDET_DLY_CTRL[SW_VOL_CTRL] for a SW control).
4. Write high the GDET_RESET[SFT_RST] to issue a fast update of the detector to the new voltage level.
5. Reset the SoC register that is controlling the GDET isolate_on.

19.3.1.3 Memory Map

The block has only SFR space that can be accessed directly from the APB slave.

NOTE

The GDET Memory Map is shown to provide details of all the SFRs that are implemented in the GDET memory slot. Nevertheless the user should NOT access any SFR field that has not been specifically mentioned to be used in the GDET startup or GDET voltage mode switch procedure.

19.3.1.3.1 APB: GDET SFR register descriptions

19.3.1.3.1.1 APB: GDET SFR memory map

GDET0.APB base address: 8004_8000h

GDET1.APB base address: 8004_A000h

[1] See chip-specific GDET information for details on the SoC register(s) used to control GDET.

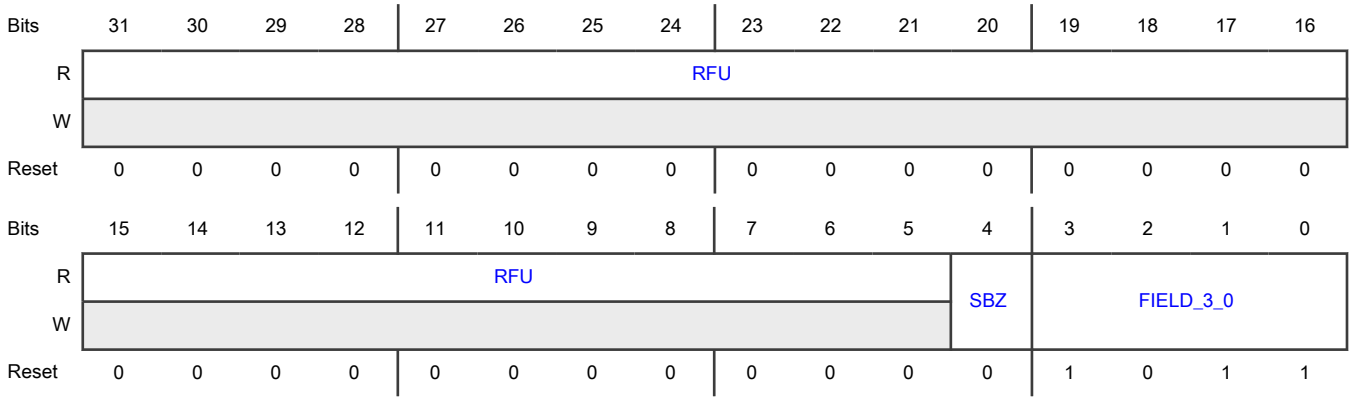
Offset	Register	Width (In bits)	Access	Reset value
0h	GDET Configuration 0 Register (GDET_CONF_0)	32	RW	0000_000Bh
4h	GDET Configuration 1 Register (GDET_CONF_1)	32	RW	0000_000Eh
8h	GDET Enable Register (GDET_ENABLE1)	32	RW	0000_0000h
Ch	GDET Configuration 2 Register (GDET_CONF_2)	32	RW	3F00_0004h
10h	GDET Configuration 3 Register (GDET_CONF_3)	32	RW	0000_0004h
14h	GDET Configuration 4 Register (GDET_CONF_4)	32	RW	0000_0004h
18h	GDET Configuration 5 Register (GDET_CONF_5)	32	RW	0000_0618h
FC0h	GDET Reset Register (GDET_RESET)	32	RW	0000_0000h
FC4h	GDET Test Register (GDET_TEST)	32	RW	0000_0000h
FCCh	GDET Delay Control Register (GDET_DLY_CTRL)	32	RW	0000_0001h

19.3.1.3.1.2 GDET Configuration 0 Register (GDET_CONF_0)

Offset

Register	Offset
GDET_CONF_0	0h

Diagram



Fields

Field	Function
31-5 RFU	Reserved for Future Use

Table continues on the next page...

Table continued from the previous page...

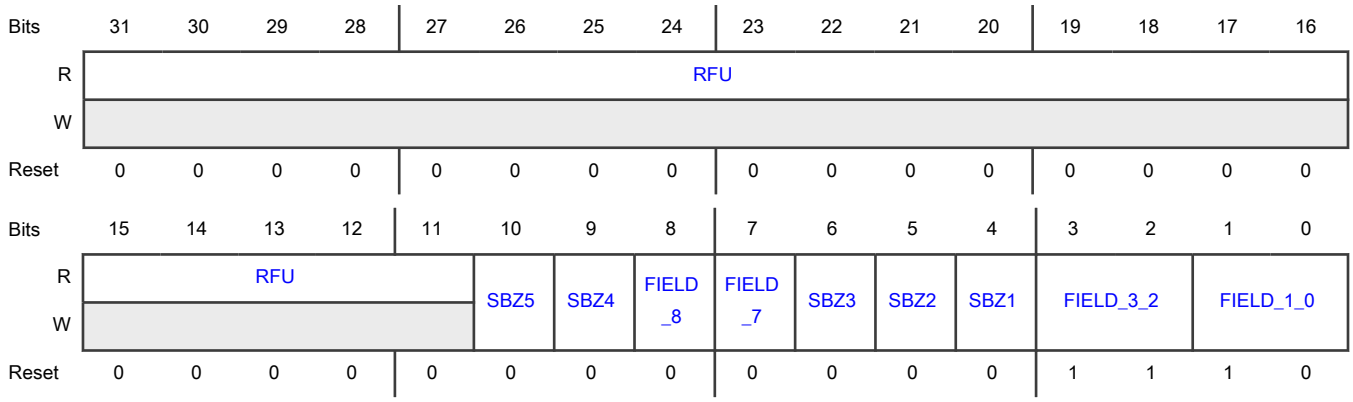
Field	Function
4 SBZ	Should Be Left to Zero
3-0 FIELD_3_0	GDET Configuration 0 Field 3_0

19.3.1.3.1.3 GDET Configuration 1 Register (GDET_CONF_1)

Offset

Register	Offset
GDET_CONF_1	4h

Diagram



Fields

Field	Function
31-11 RFU	Reserved for Future Use
10 SBZ5	Should Be Left to Zero
9 SBZ4	Should Be Left to Zero
8 FIELD_8	GDET Configuration 1 Field 8

Table continues on the next page...

Table continued from the previous page...

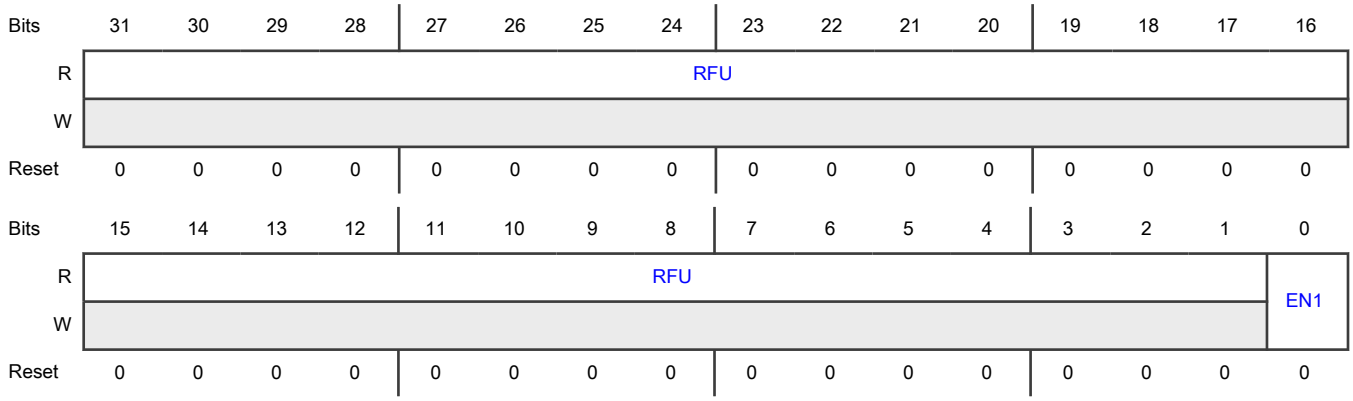
Field	Function
7 FIELD_7	GDET Configuration 1 Field 7
6 SBZ3	Should Be Left to Zero
5 SBZ2	Should Be Left to Zero
4 SBZ1	Should Be Left to Zero
3-2 FIELD_3_2	GDET Configuration 1 Field 3_2
1-0 FIELD_1_0	GDET Configuration 1 Field 1_0

19.3.1.3.1.4 GDET Enable Register (GDET_ENABLE1)

Offset

Register	Offset
GDET_ENABLE1	8h

Diagram



Fields

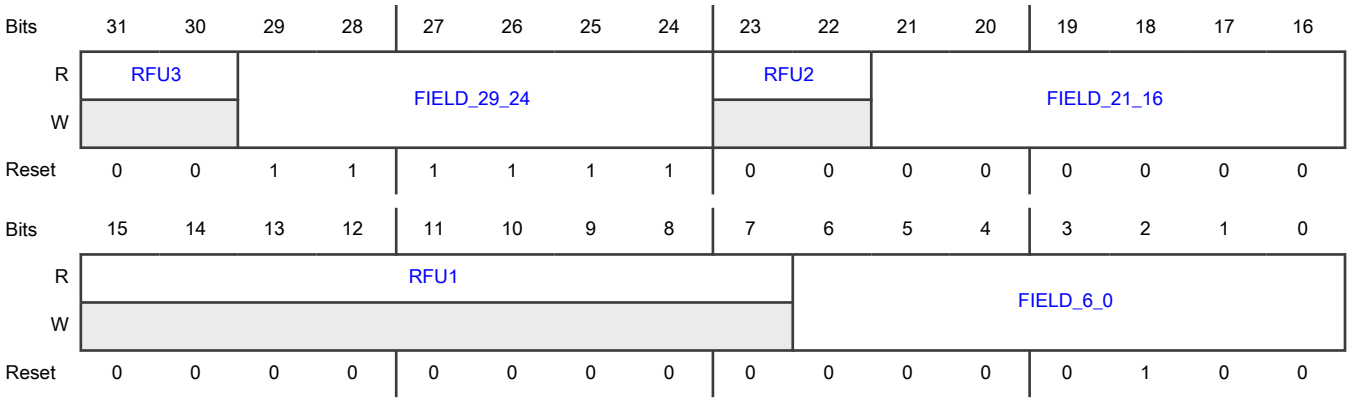
Field	Function
31-1 RFU	Reserved for Future Use
0 EN1	If set, the detector will be clock gated

19.3.1.3.1.5 GDET Configuration 2 Register (GDET_CONF_2)

Offset

Register	Offset
GDET_CONF_2	Ch

Diagram



Fields

Field	Function
31-30 RFU3	Reserved for Future Use
29-24 FIELD_29_24	GDET Configuration 2 Field 29_24
23-22 RFU2	Reserved for Future Use
21-16 FIELD_21_16	GDET Configuration 2 Field 21_16

Table continues on the next page...

Table continued from the previous page...

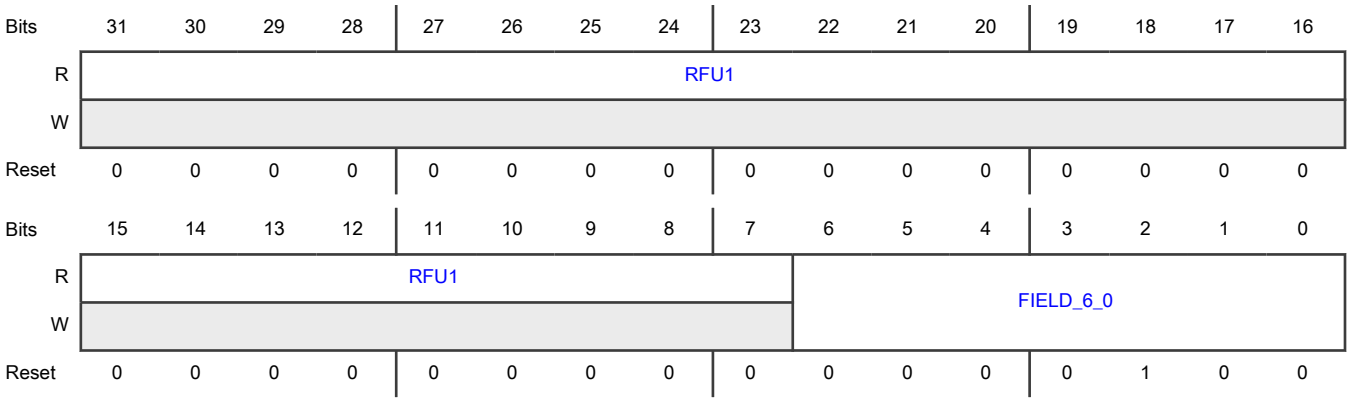
Field	Function
15-7 RFU1	Reserved for Future Use
6-0 FIELD_6_0	GDET Configuration 2 Field 6_0

19.3.1.3.1.6 GDET Configuration 3 Register (GDET_CONF_3)

Offset

Register	Offset
GDET_CONF_3	10h

Diagram



Fields

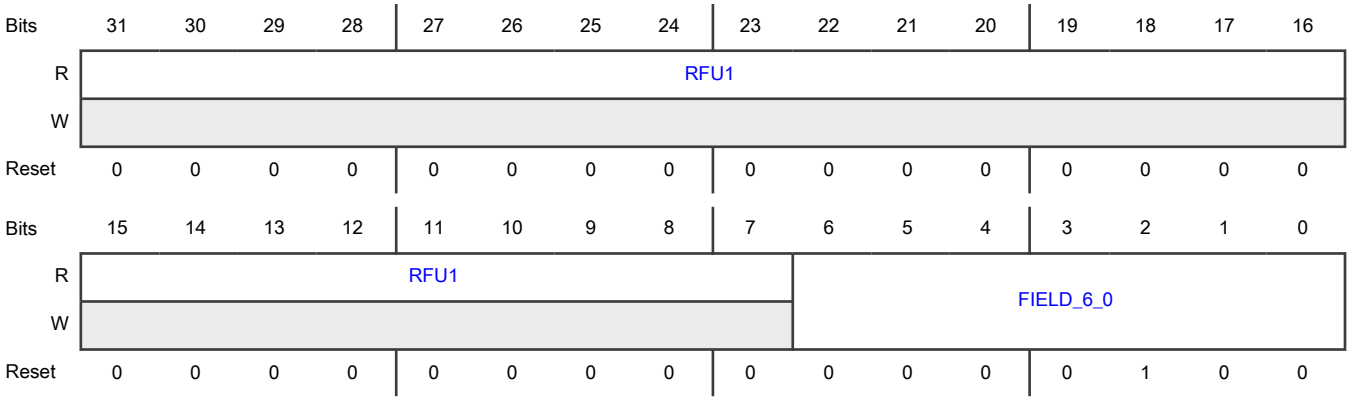
Field	Function
31-7 RFU1	Reserved for Future Use
6-0 FIELD_6_0	GDET Configuration 3 Field 6_0

19.3.1.3.1.7 GDET Configuration 4 Register (GDET_CONF_4)

Offset

Register	Offset
GDET_CONF_4	14h

Diagram



Fields

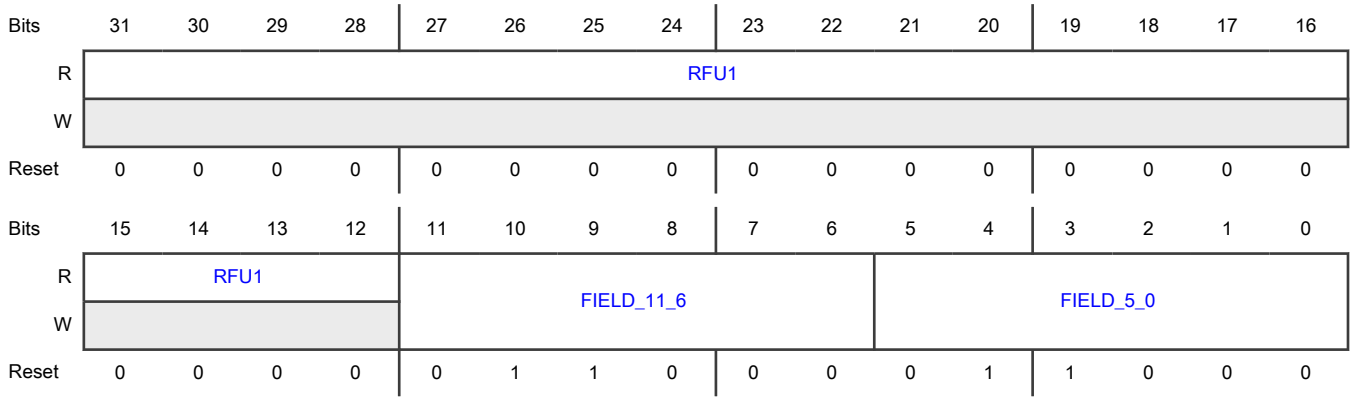
Field	Function
31-7 RFU1	Reserved for Future Use
6-0 FIELD_6_0	GDET Configuration 4 Field 6_0

19.3.1.3.1.8 GDET Configuration 5 Register (GDET_CONF_5)

Offset

Register	Offset
GDET_CONF_5	18h

Diagram



Fields

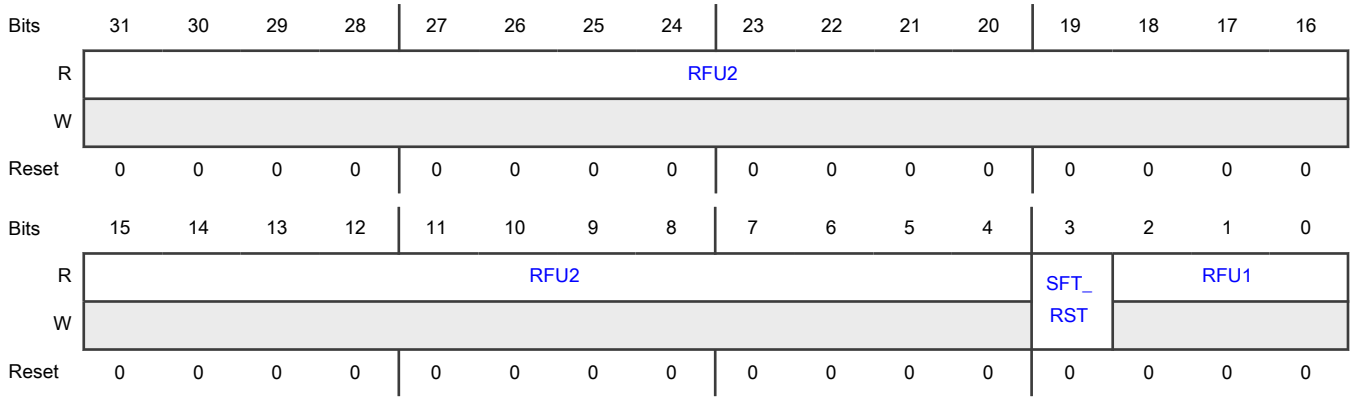
Field	Function
31-12 RFU1	Reserved for Future Use
11-6 FIELD_11_6	GDET Configuration 5 Field 11_6
5-0 FIELD_5_0	GDET Configuration 5 Field 5_0

19.3.1.3.1.9 GDET Reset Register (GDET_RESET)

Offset

Register	Offset
GDET_RESET	FC0h

Diagram



Fields

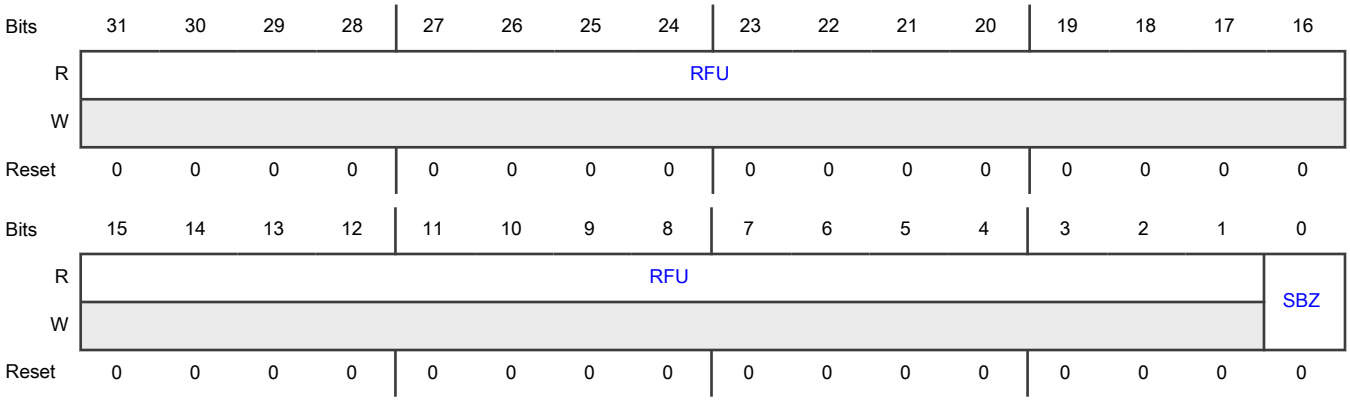
Field	Function
31-4 RFU2	Reserved for Future Use
3 SFT_RST	Soft Reset for the Core Reset
2-0 RFU1	Reserved for Future Use

19.3.1.3.1.10 GDET Test Register (GDET_TEST)

Offset

Register	Offset
GDET_TEST	FC4h

Diagram



Fields

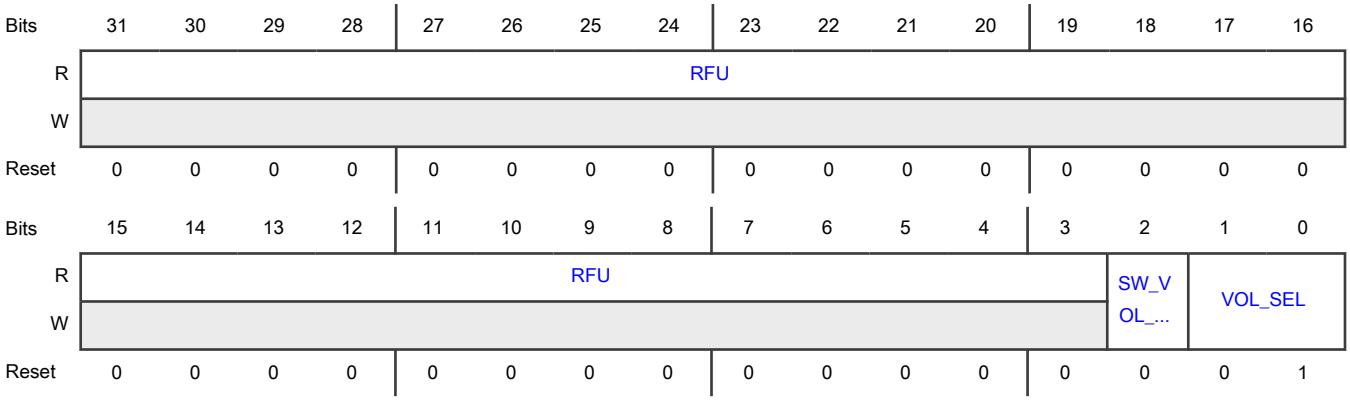
Field	Function
31-1 RFU	Reserved for Future Use
0 SBZ	Should Be Left to Zero

19.3.1.3.1.11 GDET Delay Control Register (GDET_DLY_CTRL)

Offset

Register	Offset
GDET_DLY_CTRL	FCCh

Diagram



Fields

Field	Function
31-3 RFU	Reserved for Future Use
2 SW_VOL_CTRL	Select the Control of the Trim Code to the Delay Line
1-0 VOL_SEL	GDET Delay Control of the Voltage Mode

19.4 Terms and Definition

Acronym	Explanation
AON	Always ON
ATE	Automatic Test Equipment
CDC	Clock Domain Crossing
DVS	Dynamic Voltage Scaling
FSM	Finite State Machine

Table continues on the next page...

Table continued from the previous page...

Acronym	Explanation
GDET	Glitch Detector
ICG	Internal Clock Gating
NVM	Non Volatile Memory
OTP	One Time Programmable (memory)
RTL	Register Transfer Level
SFR	Special Function Register
SoC	System on Chip
SVS	Static Voltage Scaling

Chapter 20
Intrusion and Tamper Response Controller (ITRC)

20.1 Chip-specific ITRC information

Table 401. Reference links to related information[1]

Topic	Related module	Reference
Full description	ITRC	ITRC
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

NOTE

The reset value of OUT3_SEL0, OUT4_SEL0, STATUS, and STATUS1 registers may change depending on the device's boot settings.

20.1.1 Module instances

This device has one instance of the ITRC module.

20.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

20.2 Overview

Intrusion and Tamper Response Controller (ITRC) provides a mechanism to configure the response action for an intrusion event detected by on chip security sensors. Intrusion Response is the action a device performs in order to prevent misuse of the device or disclosure of critical assets (cryptographic keys, personal data) that are generated or stored within the device. The response mechanism is typically triggered by either a signal from an on-chip sensor designed to detect that the device is in a threat condition or by an explicit command provided by the software.

20.2.1 Block diagram

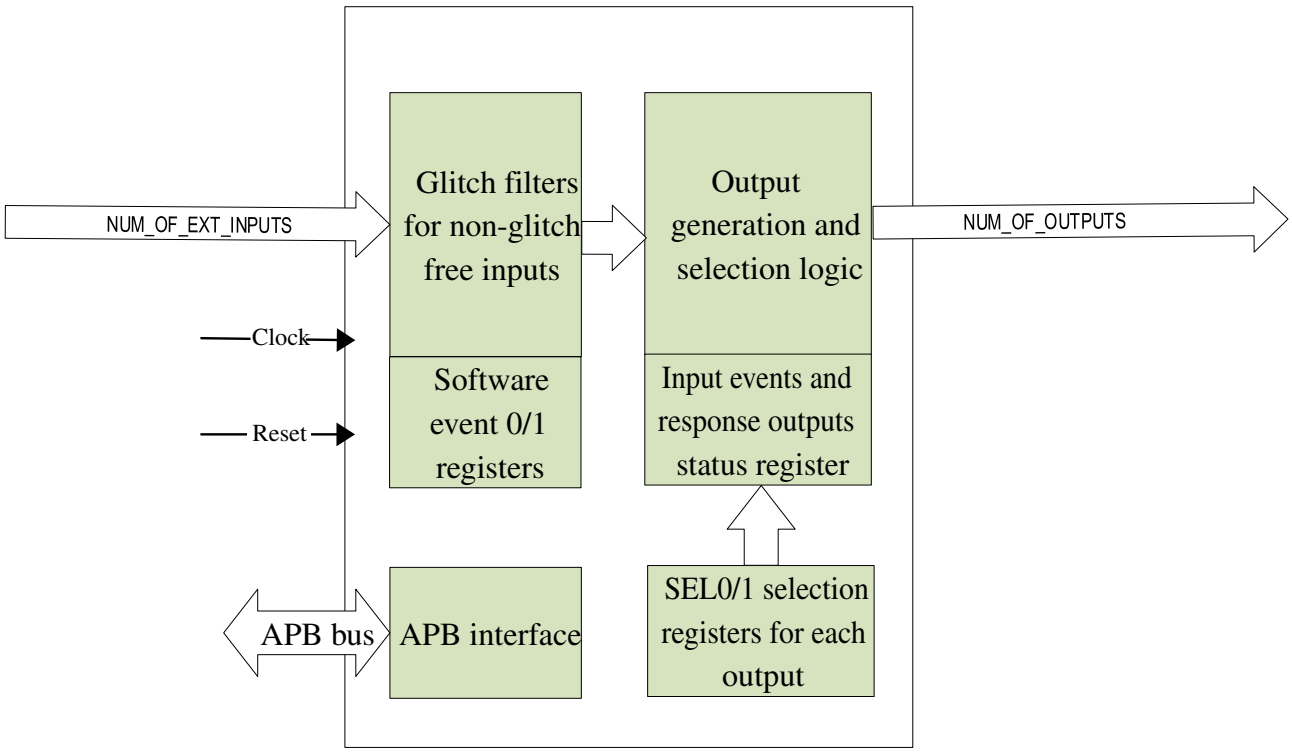


Figure 82. ITRC block diagram

20.3 Functional description

The objective of the intrusion response is to zeroize (erase) all memory locations that contain cryptographic keys, passwords, or other critical assets that need to be protected from disclosure to hostile entities. To be effective the response also needs to prevent the device from being misused while in the threat condition, by inhibiting authentication, key management, and cryptographic services from being initiated. The response action needs to be completed quickly enough to prevent the threat from compromising the integrity of the device.

On this device, the following blocks store critical assets, which need to be zeroized:

- CSS key store
- PUF key store
- PKC block is used for accelerating asymmetric cryptographic operations hence intermittent data stored in PKC RAM should also be zeroized to avoid key leakage.
- RAM A memory bank is retained using VBAT in low power modes. Hence it is also referred as long-term memory where device critical configuration data could be stored.

20.3.1 Clocks

This module is clocked by the AHB clock.

20.3.2 Reset

All control registers reset to default value on PoR and ITRC CHIP_RESET.

20.3.3 Power domain

This block is in VDD_CORE_WAKE domain which survive during power-down mode. But during deep power-down mode the state will be lost and ROM reinitializes the block.

20.4 Signals

20.4.1 Input Signals

Following are the explicit and implicit intrusion event detectors, which generate level and latched signal towards the interrupt controller.

Table 402. Input signals

Signal Name	Description
IN0	Glitch detector interrupts (GDET0 and GDET1)
IN1	TDET tamper output TDET_SR[DTF]
IN2	Code Watchdog 0 interrupt (CDOG0_FLAGS)
IN3	VDD_MAIN volt tamper output (VDD_MAIN_STATUSA[VOLT_DET])
IN4	SPC VDD_CORE low voltage detect (SPC_VD_STAT[COREVDD_LVDF])
IN5	Watchdog 0 timer event (WWDT0_MOD[WDTOF])
IN6	Flash ECC error (FMU_FSTAT[DFDIF])
IN7	Secure violation interrupt (Memory Block Checker (MBC) interrupt or secure AHB (AHBSC) matrix violation)
IN8	ELS reported error event (ELS_STATUS[ELS_ERR])
IN9	SPC VDD_CORE glitch detect (SPC_VDD_CORE_GLITCH_DETECT_SC[GLITCH_DETECT_FLAG])
IN10	Error flags from PKC module (PKC_ACCESS_ERR register)
IN11	Code Watchdog 1 interrupt (CDOG1_FLAGS)
IN12	Watchdog 1 timer event (WWDT1_MOD[WDTOF])
IN13	FREQME out of range status output (FREQME_CTRLSTAT[GT_MAX_STAT, LT_MIN_STAT])
IN14	ITRC Software event 0 - this event is set by SW writing (any random value) to SW_EVENT0 register.
IN15	ITRC Software event 1 - this event is set by SW writing (any random value) to SW_EVENT1 register.
IN16	SPC VDD_SYS low voltage detect (SPC_VD_STAT[SYSVDD_LVDF])
IN17	SPC VDD_IO low voltage detect (SPC_VD_STAT[IOVDD_LVDF])

Table continues on the next page...

Table 402. Input signals (continued)

Signal Name	Description
IN18	VDD_MAIN light tamper output (VBAT STATUS1[LIGHT_DET])
IN19	VDD_MAIN temperature tamper output (VBAT_STATUSA[TEMP_DET])
IN20	VDD_MAIN clock tamper output (VBAT_STATUSA[CLOCK_DET])
IN21 - IN24	INTM interrupt monitor error 0-3 (INTM_STATUSn[STATUS])
IN25 - IN32	At power-up SoC specific trim data is loaded to analog blocks from special flash regions by HW state machine. These signals indicate ECC error during SoC trim loading operation.
IN33	GDET0 & GDET1 SFR error that occurs when invalid address has been accessed in GDET0 and GDET1 block.
IN34	SPC VDD_CORE high voltage detect (SPC_VD_STAT[COREVDD_HVDF])
IN35	SPC VDD_SYS_HVD high voltage detect (SPC_VD_STAT[SYSVDD_HVDF])
IN36	SPC VDD_IO high voltage detect (SPC_VD_STAT[IOVDD_HVDF])
IN37	FLEXSPI GCM error (FLEXSPI_INTR[AHBGCMERR])
IN38 - IN47	Reserved

20.4.2 Output signals

Intrusion response output signals:

Table 403. Output signals

Signal Name	Alias	Description
OUT0	ITRC_IRQ	Generates ITRC interrupt.
OUT1	ELS_RESET	This will destroy the key store inside ELS S50 quickly protecting the keys.
OUT2	PUF_ZEROIZE	This will disable PUF from outputting keys.
OUT3	RAM_ZEROIZE	Sets internal "RAM_ZEROIZE" signal, which is checked by ROM during the device RESET. If the signal is set, ROM zeroizes PKC and RAMA based on user settings ITRC_ZEROIZE in CMPA. This signal is cleared on power-cycle or after ROM zeroizes PKC_RAM and RAMA memory banks.

Table continues on the next page...

Table 403. Output signals (continued)

Signal Name	Alias	Description
		Hence this output should be selected in-conjunction with "CHIP_RESET". The current status of this signal is reflected in VBAT_STATA[SEC0_DET] register field.
OUT4	CHIP_RESET	Generate a reset request to CMC to reset the chip when asserted.
OUT5	TMPR_OUT0	This signal is connected following input multiplexers present on chip: • DMAMUX0/DMAMUX1 • SCT0 INPUTMUX • CTIMERs INPUTMUX • EZH INPUTMUX • External pin INPUTMUX
OUT6	TMPR_OUT1	Connected to the inputs of the following PINMUXes: • DMAMUX0/DMAMUX1 • SCT0 INPUTMUX • CTIMERs INPUTMUX • EZH INPUTMUX • External Pin INPUTMUX

20.5 Memory map and register definition

This section includes the ITRC module memory map and detailed descriptions of all registers.

20.5.1 INx_SELy register behavior

To counter glitch attacks all trigger selection registers are defined as per secure coding techniques. Which requires a control signal be driven by a multi-bit field distributed among multiple registers located at different address.

- For each intrusion response output signal a OUTx_SEL0 and OUTx_SEL1 registers are allocated.
- Each input trigger event is allocated two bits in OUTx_SEL0 and OUTx_SEL1 register each.
- All four bits are combined to decide whether:
  - The input signal is selected as trigger or not.
  - The input signal field is writable or not. Once the field is locked then it can't be changed until any reset to ITRC is asserted. See Reset.
  - Below table describes the combination logic:

Table 404. Event select table

INx_SEL0	INx_SEL1	Signal selected?	Writable field?
10	10	No	Yes (default after reset)
01	10	Yes	Yes
don't care	!="10"	Yes	No
00 or 11	don't care	Yes	No

20.5.1 Intrusion and Tamper Response Controller register descriptions

20.5.1.1 ITRC memory map

ITRC0 base address: 4002_6000h

Offset	Register	Width (In bits)	Access	Reset value
0h	ITRC outputs and IN0 to IN15 Status (STATUS)	32	RW	0000_000Ah
4h	ITRC IN16 to IN47 Status (STATUS1)	32	RW	0000_001Ch
8h	Trigger Source IN0 to IN15 selector (OUT0_SEL0)	32	RW	AAAA_AAAAh
Ch	Trigger Source IN0 to IN15 selector (OUT0_SEL1)	32	RW	AAAA_AAAAh
10h	Trigger Source IN0 to IN15 selector (OUT1_SEL0)	32	RW	AAAA_AAAAh
14h	Trigger Source IN0 to IN15 selector (OUT1_SEL1)	32	RW	AAAA_AAAAh
18h	Trigger Source IN0 to IN15 selector (OUT2_SEL0)	32	RW	AAAA_AAAAh
1Ch	Trigger Source IN0 to IN15 selector (OUT2_SEL1)	32	RW	AAAA_AAAAh
20h	Trigger Source IN0 to IN15 selector (OUT3_SEL0)	32	RW	AAAA_AAAAh
24h	Trigger Source IN0 to IN15 selector (OUT3_SEL1)	32	RW	AAAA_AAAAh
28h	Trigger Source IN0 to IN15 selector (OUT4_SEL0)	32	RW	AAAA_AAAAh
2Ch	Trigger Source IN0 to IN15 selector (OUT4_SEL1)	32	RW	AAAA_AAAAh
30h	Trigger Source IN0 to IN15 selector (OUT5_SEL0)	32	RW	AAAA_AAAAh
34h	Trigger Source IN0 to IN15 selector (OUT5_SEL1)	32	RW	AAAA_AAAAh
38h	Trigger Source IN0 to IN15 selector (OUT6_SEL0)	32	RW	AAAA_AAAAh
3Ch	Trigger Source IN0 to IN15 selector (OUT6_SEL1)	32	RW	AAAA_AAAAh
48h	Trigger Source IN16 to IN31 selector (OUT0_SEL0_1)	32	RW	AAAA_AAAAh
4Ch	Trigger Source IN16 to IN31 selector (OUT0_SEL1_1)	32	RW	AAAA_AAAAh
50h	Trigger Source IN16 to IN31 selector (OUT1_SEL0_1)	32	RW	AAAA_AAAAh
54h	Trigger Source IN16 to IN31 selector (OUT1_SEL1_1)	32	RW	AAAA_AAAAh
58h	Trigger Source IN16 to IN31 selector (OUT2_SEL0_1)	32	RW	AAAA_AAAAh

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
5Ch	Trigger Source IN16 to IN31 selector (OUT2_SEL1_1)	32	RW	AAAA_AAAAh
60h	Trigger Source IN16 to IN31 selector (OUT3_SEL0_1)	32	RW	AAAA_AAAAh
64h	Trigger Source IN16 to IN31 selector (OUT3_SEL1_1)	32	RW	AAAA_AAAAh
68h	Trigger Source IN16 to IN31 selector (OUT4_SEL0_1)	32	RW	AAAA_AAAAh
6Ch	Trigger Source IN16 to IN31 selector (OUT4_SEL1_1)	32	RW	AAAA_AAAAh
70h	Trigger Source IN16 to IN31 selector (OUT5_SEL0_1)	32	RW	AAAA_AAAAh
74h	Trigger Source IN16 to IN31 selector (OUT5_SEL1_1)	32	RW	AAAA_AAAAh
78h	Trigger Source IN16 to IN31 selector (OUT6_SEL0_1)	32	RW	AAAA_AAAAh
7Ch	Trigger Source IN16 to IN31 selector (OUT6_SEL1_1)	32	RW	AAAA_AAAAh
88h	Trigger source IN32 to IN47 selector (OUT0_SEL0_2)	32	RW	A000_2AAAh
8Ch	Trigger source IN32 to IN47 selector (OUT0_SEL1_2)	32	RW	A000_2AAAh
90h	Trigger source IN32 to IN47 selector (OUT1_SEL0_2)	32	RW	A000_2AAAh
94h	Trigger source IN32 to IN47 selector (OUT1_SEL1_2)	32	RW	A000_2AAAh
98h	Trigger source IN32 to IN47 selector (OUT2_SEL0_2)	32	RW	A000_2AAAh
9Ch	Trigger source IN32 to IN47 selector (OUT2_SEL1_2)	32	RW	A000_2AAAh
A0h	Trigger source IN32 to IN47 selector (OUT3_SEL0_2)	32	RW	A000_2AAAh
A4h	Trigger source IN32 to IN47 selector (OUT3_SEL1_2)	32	RW	A000_2AAAh
A8h	Trigger source IN32 to IN47 selector (OUT4_SEL0_2)	32	RW	A000_2AAAh
ACh	Trigger source IN32 to IN47 selector (OUT4_SEL1_2)	32	RW	A000_2AAAh
B0h	Trigger source IN32 to IN47 selector (OUT5_SEL0_2)	32	RW	A000_2AAAh
B4h	Trigger source IN32 to IN47 selector (OUT5_SEL1_2)	32	RW	A000_2AAAh
B8h	Trigger source IN32 to IN47 selector (OUT6_SEL0_2)	32	RW	A000_2AAAh
BCh	Trigger source IN32 to IN47 selector (OUT6_SEL1_2)	32	RW	A000_2AAAh
F0h	Software event 0 (SW_EVENT0)	32	W	0000_0000h
F4h	Software event 1 (SW_EVENT1)	32	W	0000_0000h

20.5.1.2 ITRC outputs and IN0 to IN15 Status (STATUS)

Offset

Register	Offset
STATUS	0h

Function

Status register showing the state of intrusion event input signals and output action signals, input signals from IN0 - IN15. This register is cleared on PoR reset or by writing 1 to corresponding bit by software.

NOTE

Input source must be cleared at the originating module first, then clear the input status bit that are selected for that output in STATUS register.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved									OUT6_ST...	OUT5_ST...	OUT4_ST...	OUT3_ST...	OUT2_ST...	OUT1_ST...	OUT0_ST...
W										W1C	W1C	W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IN15_ST...	IN14_ST...	IN113_S...	IN112_S...	IN11_ST...	IN10_ST...	IN9_STA...	IN8_STA...	IN7_STA...	IN6_STA...	IN5_STA...	IN4_STA...	IN3_STA...	IN2_STA...	IN1_STA...	IN0_STA...
W	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

Fields

Field	Function
31-23 —	Reserved
22 OUT6_STATUS	ITRC triggered TMPR_OUT1 internal signal connected to various on-chip multiplexers. 0b - Output not triggered. 1b - Output has been triggered.
21 OUT5_STATUS	ITRC triggered TMPR_OUT0 internal signal connected to various on-chip multiplexers. 0b - Output not triggered. 1b - Output has been triggered.
20 OUT4_STATUS	ITRC triggered CHIP_RESET to reset the chip after all other response process finished. 0b - Output not triggered. 1b - Output has been triggered.
19 OUT3_STATUS	ITRC triggered RAM_ZEROIZE. 0b - Output not triggered. 1b - Output has been triggered.
18	ITRC triggered PUF_ZEROIZE to clear PUF key store and RAM.

Table continues on the next page...

Table continued from the previous page...

Field	Function
OUT2_STATUS	0b - Output not triggered. 1b - Output has been triggered.
17 OUT1_STATUS	ITRC triggered ELS_RESET to clear ELS key store. 0b - Output not triggered. 1b - Output has been triggered.
16 OUT0_STATUS	ITRC triggered ITRC_IRQ output. 0b - Output not triggered. 1b - Output has been triggered.
15 IN15_STATUS	Software event 1 occurred. 0b - Output not triggered. 1b - Output has been triggered.
14 IN14_STATUS	Software event 0 occurred. 0b - Output not triggered. 1b - Output has been triggered.
13 IN113_STATUS	FREQME out of range status output. 0b - Output not triggered. 1b - Output has been triggered.
12 IN112_STATUS	Watchdog 1 timer event interrupt. 0b - Output not triggered. 1b - Output has been triggered.
11 IN11_STATUS	Code Watchdog 1 interrupt. 0b - Output not triggered. 1b - Output has been triggered.
10 IN10_STATUS	PKC module detected an error event. 0b - Output not triggered. 1b - Output has been triggered.
9 IN9_STATUS	SPC VDD_CORE glitch detect event occurred. 0b - Output not triggered. 1b - Output has been triggered.
8 IN8_STATUS	ELS error event occurred. 0b - Output not triggered. 1b - Output has been triggered.

Table continues on the next page...

Table continued from the previous page...

Field	Function
7 IN7_STATUS	AHB secure bus checkers detected illegal access. 0b - Output not triggered. 1b - Output has been triggered.
6 IN6_STATUS	Flash ECC mismatch event occurred. 0b - Output not triggered. 1b - Output has been triggered.
5 IN5_STATUS	Watch Dog timer event occurred. 0b - Output not triggered. 1b - Output has been triggered.
4 IN4_STATUS	SPC VDD_CORE_LVD detect. 0b - Output not triggered. 1b - Output has been triggered.
3 IN3_STATUS	VDD_MAIN volt tamper output. 0b - Output not triggered. 1b - Output has been triggered.
2 IN2_STATUS	Code Watchdog 0 interrupt. 0b - Output not triggered. 1b - Output has been triggered.
1 IN1_STATUS	TDET tamper output. 0b - Output not triggered. 1b - Output has been triggered.
0 IN0_STATUS	GDET0 & 1 interrupt. 0b - Output not triggered. 1b - Output has been triggered.

20.5.1.3 ITRC IN16 to IN47 Status (STATUS1)

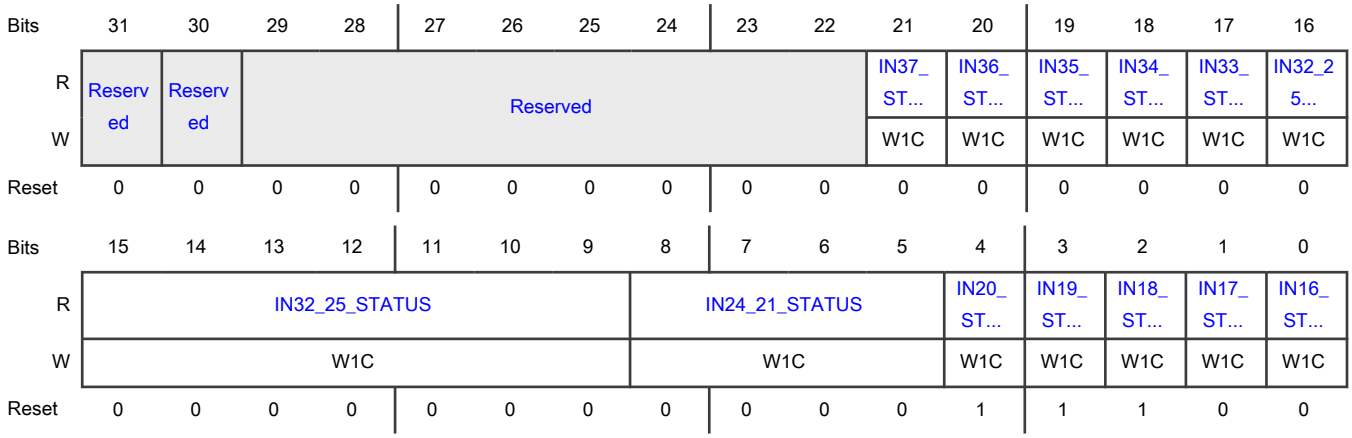
Offset

Register	Offset
STATUS1	4h

Function

This is the status register for signals IN16 - IN47.

Diagram



Fields

Field	Function
31 —	Reserved
30 —	Reserved
29-22 —	Reserved
21 IN37_STATUS	FLEXSPI GCM error event occurred. 0b - Output not triggered. 1b - Output has been triggered.
20 IN36_STATUS	SPC VDD_IO high voltage detect event occurred. 0b - Output not triggered. 1b - Output has been triggered.
19 IN35_STATUS	SPC VDD_SYS_HVD high voltage detect event occurred. 0b - Output not triggered. 1b - Output has been triggered.
18 IN34_STATUS	SPC VDD_CORE high voltage detect event occurred. 0b - Output not triggered. 1b - Output has been triggered.
17 IN33_STATUS	GDET0/1 SFR error event occurred. 0b - Output not triggered. 1b - Output has been triggered.

Table continues on the next page...

Table continued from the previous page...

Field	Function
16-9 IN32_25_STATUS	MSF SOCTRIM 7~0 ECC error event occurred. 0000_0000b - Output not triggered. 0000_0001b - Output has been triggered.
8-5 IN24_21_STATUS	INTM interrupt monitor error 3~0 event occurred. 0000b - Output not triggered. 0001b - Output has been triggered.
4 IN20_STATUS	VDD_MAIN clock tamper output event occurred. 0b - Output not triggered. 1b - Output has been triggered.
3 IN19_STATUS	Reserved 0b - Output not triggered. 1b - Output has been triggered.
2 IN18_STATUS	Reserved 0b - Output not triggered. 1b - Output has been triggered.
1 IN17_STATUS	SPC VDD_IO_LVD detect event occurred. 0b - Output not triggered. 1b - Output has been triggered.
0 IN16_STATUS	SSPC VDD_SYS_LVD detect event occurred. 0b - Output not triggered. 1b - Output has been triggered.

20.5.1.4 Trigger Source IN0 to IN15 selector (OUT0_SEL0 - OUT6_SEL0)

Offset

Register	Offset
OUT0_SEL0	8h
OUT1_SEL0	10h
OUT2_SEL0	18h
OUT3_SEL0	20h
OUT4_SEL0	28h
OUT5_SEL0	30h
OUT6_SEL0	38h

Function

OUTx_SEL0/1 register is used together with OUTx_SEL1/0 register to select IN0 to IN15 to output triggers. See Output signals for the output triggers in ITRC.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IN15_SELn		IN14_SELn		IN13_SELn		IN12_SELn		IN11_SELn		IN10_SELn		IN9_SELn		IN8_SELn	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IN7_SELn		IN6_SELn		IN5_SELn		IN4_SELn		IN3_SELn		IN2_SELn		IN1_SELn		IN0_SELn	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Fields

Field	Function
31-30 IN15_SELn	Selects software event 1 as a trigger source. See Table 404 for bits combination logic.
29-28 IN14_SELn	Selects software event 0 as a trigger source. See Table 404 for bits combination logic.
27-26 IN13_SELn	Selects FREQME out of range status output as a trigger source. See Table 404 for bits combination logic.
25-24 IN12_SELn	Selects Watchdog 1 timer event as a trigger source. See Table 404 for bits combination logic.
23-22 IN11_SELn	Selects Code Watchdog 1 event as a trigger source. See Table 404 for bits combination logic.
21-20 IN10_SELn	Selects PKC error event as a trigger source. See Table 404 for bits combination logic.
19-18 IN9_SELn	Selects SPC VDD_CORE glitch detector as a trigger source. See Table 404 for bits combination logic.
17-16 IN8_SELn	Selects ELS error event as a trigger source. See Table 404 for bits combination logic.
15-14 IN7_SELn	Selects AHB secure bus or MBC bus illegal access event as a trigger source. See Table 404 for bits combination logic.

Table continues on the next page...

Table continued from the previous page...

Field	Function
13-12 IN6_SELn	Selects Flash ECC mismatch event as a trigger source. See Table 404 for bits combination logic.
11-10 IN5_SELn	Selects Watchdog 0 timer event as a trigger source. See Table 404 for bits combination logic.
9-8 IN4_SELn	Selects low-voltage event on VDD_CORE rail as a trigger source. See Table 404 for bits combination logic.
7-6 IN3_SELn	Selects VDD_MAIN voltage tamper event as a trigger source. See Table 404 for bits combination logic.
5-4 IN2_SELn	Selects Code Watchdog 0 event as a trigger source. See Table 404 for bits combination logic.
3-2 IN1_SELn	Selects TDET event as a trigger source. See Table 404 for bits combination logic.
1-0 IN0_SELn	Selects digital glitch detector as a trigger source. See Table 404 for bits combination logic.

20.5.1.5 Trigger Source IN0 to IN15 selector (OUT0_SEL1 - OUT6_SEL1)

Offset

Register	Offset
OUT0_SEL1	Ch
OUT1_SEL1	14h
OUT2_SEL1	1Ch
OUT3_SEL1	24h
OUT4_SEL1	2Ch
OUT5_SEL1	34h
OUT6_SEL1	3Ch

Function

OUTx_SEL0/1 register is used together with OUTx_SEL1/0 register to select IN0 to IN15 to output triggers. See Output signals for the output triggers in ITRC.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IN15_SELn		IN14_SELn		IN13_SELn		IN12_SELn		IN11_SELn		IN10_SELn		IN9_SELn		IN8_SELn	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IN7_SELn		IN6_SELn		IN5_SELn		IN4_SELn		IN3_SELn		IN2_SELn		IN1_SELn		IN0_SELn	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Fields

Field	Function
31-30 IN15_SELn	Selects software event 1 as a trigger source. See Table 404 for bits combination logic.
29-28 IN14_SELn	Selects software event 0 as a trigger source. See Table 404 for bits combination logic.
27-26 IN13_SELn	Selects FREQME out of range status output as a trigger source. See Table 404 for bits combination logic.
25-24 IN12_SELn	Selects Watchdog 1 timer event as a trigger source. See Table 404 for bits combination logic.
23-22 IN11_SELn	Selects Code Watchdog 1 event as a trigger source. See Table 404 for bits combination logic.
21-20 IN10_SELn	Selects PKC error event as a trigger source. See Table 404 for bits combination logic.
19-18 IN9_SELn	Selects SPC VDD_CORE glitch detector as a trigger source. See Table 404 for bits combination logic.
17-16 IN8_SELn	Selects ELS error event as a trigger source. See Table 404 for bits combination logic.
15-14 IN7_SELn	Selects AHB secure bus or MBC bus illegal access event as a trigger source. See Table 404 for bits combination logic.
13-12 IN6_SELn	Selects Flash ECC mismatch event as a trigger source. See Table 404 for bits combination logic.

Table continues on the next page...

Table continued from the previous page...

Field	Function
11-10 IN5_SELn	Selects Watchdog 0 timer event as a trigger source. See Table 404 for bits combination logic.
9-8 IN4_SELn	Selects low-voltage event on VDD_CORE rail as a trigger source. See Table 404 for bits combination logic.
7-6 IN3_SELn	Selects VDD_MAIN voltage tamper event as a trigger source. See Table 404 for bits combination logic.
5-4 IN2_SELn	Selects Code Watchdog 0 event as a trigger source. See Table 404 for bits combination logic.
3-2 IN1_SELn	Selects TDET event as a trigger source. See Table 404 for bits combination logic.
1-0 IN0_SELn	Selects digital glitch detector as a trigger source. See Table 404 for bits combination logic.

20.5.1.6 Trigger Source IN16 to IN31 selector (OUT0_SEL0_1 - OUT6_SEL0_1)

Offset

Register	Offset
OUT0_SEL0_1	48h
OUT1_SEL0_1	50h
OUT2_SEL0_1	58h
OUT3_SEL0_1	60h
OUT4_SEL0_1	68h
OUT5_SEL0_1	70h
OUT6_SEL0_1	78h

Function

OUTx_SEL0/1_1 register is used together with OUTx_SEL1/0_1 register to select IN16 to IN31 to trigger. See Output signals for the output triggers in ITRC.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IN31_SELn		IN30_SELn		IN29_SELn		IN28_SELn		IN27_SELn		IN26_SELn		IN25_SELn		IN24_SELn	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IN23_SELn		IN22_SELn		IN21_SELn		IN20_SELn		IN19_SELn		IN18_SELn		IN17_SELn		IN16_SELn	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Fields

Field	Function
31-30 IN31_SELn	Selects MSF SOCTRIM 6 ECC error event as a trigger source. See Table 404 for bits combination logic.
29-28 IN30_SELn	Selects MSF SOCTRIM 5 ECC error event as a trigger source. See Table 404 for bits combination logic.
27-26 IN29_SELn	Selects MSF SOCTRIM 4 ECC error event as a trigger source. See Table 404 for bits combination logic.
25-24 IN28_SELn	Selects MSF SOCTRIM 3 ECC error event as a trigger source. See Table 404 for bits combination logic.
23-22 IN27_SELn	Selects MSF SOCTRIM 2 ECC error event as a trigger source. See Table 404 for bits combination logic.
21-20 IN26_SELn	Selects MSF SOCTRIM 1 ECC error event as a trigger source. See Table 404 for bits combination logic.
19-18 IN25_SELn	Selects MSF SOCTRIM 0 ECC error event as a trigger source. See Table 404 for bits combination logic.
17-16 IN24_SELn	Selects INTM interrupt monitor error 3 event as a trigger source. See Table 404 for bits combination logic.
15-14 IN23_SELn	Selects INTM interrupt monitor error 2 event as a trigger source. See Table 404 for bits combination logic.
13-12 IN22_SELn	Selects INTM interrupt monitor error 1 event as a trigger source. See Table 404 for bits combination logic.

Table continues on the next page...

Table continued from the previous page...

Field	Function
11-10 IN21_SELn	Selects INTM interrupt monitor error 0 event as a trigger source. See Table 404 for bits combination logic.
9-8 IN20_SELn	Selects VDD_MAIN clock tamper output event as a trigger source. See Table 404 for bits combination logic.
7-6 IN19_SELn	Selects VDD_MAIN temperature tamper output event as a trigger source. See Table 404 for bits combination logic.
5-4 IN18_SELn	Reserved See Table 404 for bits combination logic.
3-2 IN17_SELn	Selects SPC VDD_IO_LVD detect as a trigger source. See Table 404 for bits combination logic.
1-0 IN16_SELn	Selects SPC VDD_SYS_LVD detect as a trigger source. See Table 404 for bits combination logic.

20.5.1.7 Trigger Source IN16 to IN31 selector (OUT0_SEL1_1 - OUT6_SEL1_1)

Offset

Register	Offset
OUT0_SEL1_1	4Ch
OUT1_SEL1_1	54h
OUT2_SEL1_1	5Ch
OUT3_SEL1_1	64h
OUT4_SEL1_1	6Ch
OUT5_SEL1_1	74h
OUT6_SEL1_1	7Ch

Function

OUTx_SEL0/1_1 register is used together with OUTx_SEL1/0_1 register to select IN16 to IN31 to trigger. See Output signals for the output triggers in ITRC.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IN31_SELn		IN30_SELn		IN29_SELn		IN28_SELn		IN27_SELn		IN26_SELn		IN25_SELn		IN24_SELn	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IN23_SELn		IN22_SELn		IN21_SELn		IN20_SELn		IN19_SELn		IN18_SELn		IN17_SELn		IN16_SELn	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Fields

Field	Function
31-30 IN31_SELn	Selects MSF SOCTRIM 6 ECC error event as a trigger source. See Table 404 for bits combination logic.
29-28 IN30_SELn	Selects MSF SOCTRIM 5 ECC error event as a trigger source. See Table 404 for bits combination logic.
27-26 IN29_SELn	Selects MSF SOCTRIM 4 ECC error event as a trigger source. See Table 404 for bits combination logic.
25-24 IN28_SELn	Selects MSF SOCTRIM 3 ECC error event as a trigger source. See Table 404 for bits combination logic.
23-22 IN27_SELn	Selects MSF SOCTRIM 2 ECC error event as a trigger source. See Table 404 for bits combination logic.
21-20 IN26_SELn	Selects MSF SOCTRIM 1 ECC error event as a trigger source. See Table 404 for bits combination logic.
19-18 IN25_SELn	Selects MSF SOCTRIM 0 ECC error event as a trigger source. See Table 404 for bits combination logic.
17-16 IN24_SELn	Selects INTM interrupt monitor error 3 event as a trigger source. See Table 404 for bits combination logic.
15-14 IN23_SELn	Selects INTM interrupt monitor error 2 event as a trigger source. See Table 404 for bits combination logic.
13-12 IN22_SELn	Selects INTM interrupt monitor error 1 event as a trigger source. See Table 404 for bits combination logic.

Table continues on the next page...

Table continued from the previous page...

Field	Function
11-10 IN21_SELn	Selects INTM interrupt monitor error 0 event as a trigger source. See Table 404 for bits combination logic.
9-8 IN20_SELn	Selects VDD_MAIN clock tamper output event as a trigger source. See Table 404 for bits combination logic.
7-6 IN19_SELn	Selects VDD_MAIN temperature tamper output event as a trigger source. See Table 404 for bits combination logic.
5-4 IN18_SELn	Reserved See Table 404 for bits combination logic.
3-2 IN17_SELn	Selects SPC VDD_IO_LVD detect as a trigger source. See Table 404 for bits combination logic.
1-0 IN16_SELn	Selects SPC VDD_SYS_LVD detect as a trigger source. See Table 404 for bits combination logic.

20.5.1.8 Trigger source IN32 to IN47 selector (OUT0_SEL0_2 - OUT6_SEL0_2)

Offset

Register	Offset
OUT0_SEL0_2	88h
OUT1_SEL0_2	90h
OUT2_SEL0_2	98h
OUT3_SEL0_2	A0h
OUT4_SEL0_2	A8h
OUT5_SEL0_2	B0h
OUT6_SEL0_2	B8h

Function

OUTx_SEL0/1_2 register is used together with OUTx_SEL1/0_2 register to select IN32 to IN47 to trigger. See Output signals for the output triggers in ITRC.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved		Reserved		Reserved											
Reset	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved		Reserved		IN37_SELn		IN36_SELn		IN35_SELn		IN34_SELn		IN33_SELn		IN32_SELn	
Reset	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Fields

Field	Function
31-30 —	Reserved
29-28 —	Reserved
27-14 —	Reserved
13-12 —	Reserved
11-10 IN37_SELn	Selects FLEXSPI GCM error as a trigger source. See Table 404 for bits combination logic.
9-8 IN36_SELn	Selects VDD_IO_HVD as a trigger source. See Table 404 for bits combination logic.
7-6 IN35_SELn	Selects VDD_SYS_HVD as a trigger source. See Table 404 for bits combination logic.
5-4 IN34_SELn	Selects SPC VDD_CORE_HVD as a trigger source. See Table 404 for bits combination logic.
3-2 IN33_SELn	Selects GDET0 & 1 SFR error detect as a trigger source. See Table 404 for bits combination logic.
1-0 IN32_SELn	Selects MSF SOCTRIM 7 ECC error event as a trigger source. See Table 404 for bits combination logic.

20.5.1.9 Trigger source IN32 to IN47 selector (OUT0_SEL1_2 - OUT6_SEL1_2)

Offset

Register	Offset
OUT0_SEL1_2	8Ch
OUT1_SEL1_2	94h
OUT2_SEL1_2	9Ch
OUT3_SEL1_2	A4h
OUT4_SEL1_2	ACh
OUT5_SEL1_2	B4h
OUT6_SEL1_2	BCh

Function

OUTx_SEL0/1_2 register is used together with OUTx_SEL1/0_2 register to select IN32 to IN47 to trigger. See Output signals for the output triggers in ITRC.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved		Reserved		Reserved											
Reset	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved		Reserved		IN37_SELn		IN36_SELn		IN35_SELn		IN34_SELn		IN33_SELn		IN32_SELn	
Reset	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Fields

Field	Function
31-30 —	Reserved
29-28 —	Reserved
27-14 —	Reserved
13-12	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
11-10 IN37_SELn	Selects FLEXSPI GCM error as a trigger source. See Table 404 for bits combination logic.
9-8 IN36_SELn	Selects VDD_IO_HVD as a trigger source. See Table 404 for bits combination logic.
7-6 IN35_SELn	Selects VDD_SYS_HVD as a trigger source. See Table 404 for bits combination logic.
5-4 IN34_SELn	Selects SPC VDD_CORE_HVD as a trigger source. See Table 404 for bits combination logic.
3-2 IN33_SELn	Selects GDET0 & 1 SFR error detect as a trigger source. See Table 404 for bits combination logic.
1-0 IN32_SELn	Selects MSF SOCTRIM 7 ECC error event as a trigger source. See Table 404 for bits combination logic.

20.5.1.10 Software event 0 (SW_EVENT0)

Offset

Register	Offset
SW_EVENT0	F0h

Function

A write operation to this register sets the software event 0 signal active. The software event is self-cleared.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	TRIGGER_SW_EVENT_0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	TRIGGER_SW_EVENT_0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 TRIGGER_SW_EVENT_0	Trigger software event 0.

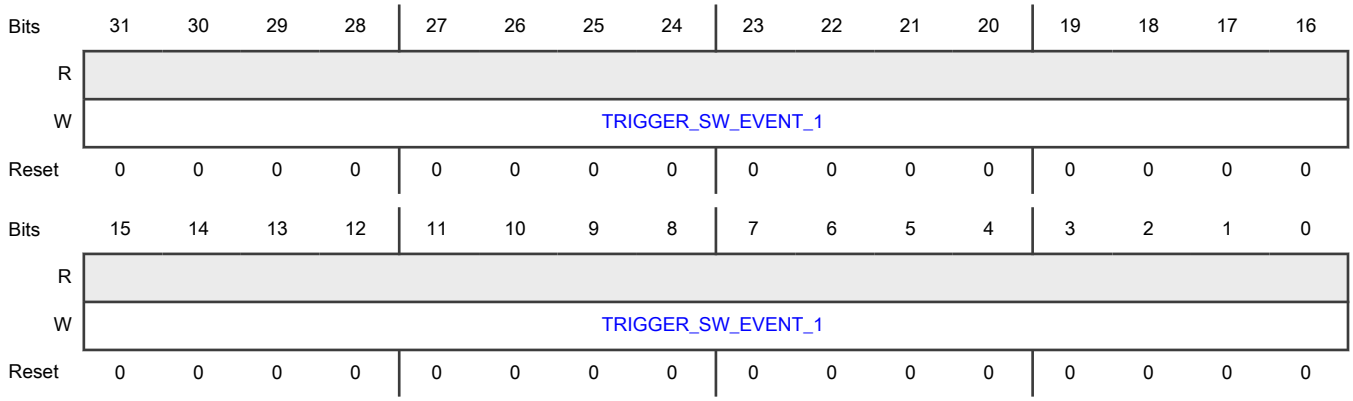
20.5.1.11 Software event 1 (SW_EVENT1)

Offset

Register	Offset
SW_EVENT1	F4h

Function
A write operation to this register sets the software event 1 signal active. The software event is self-cleared.

Diagram



Fields

Field	Function
31-0 TRIGGER_SW_EVENT_1	Trigger software event 1.

Chapter 21
Memory Block Checker (MBC)

21.1 Chip-specific MBC information

Table 405. Reference links to related information[1]

Topic	Related module	Reference
Full description	MBC	MBC
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

NOTE

The MBC is integrated within the device to see all transactions as secure-privileged. The MBC is only intended to configure read, write and execute privileges according to the MBC0_MEMN_GLBACn[SPR, SPW, and SPX] values. The AHBSC should be used to configure secure/non-secure and privileged/non-privileged access.

NOTE

The reset values of MBC0_DOM0_MEM1_BLK_CFG_W0, MBC0_DOM0_MEM2_BLK_CFG_W0 and MBC0_MEMN_GLBAC0 registers may change depending on the device's boot settings.

NOTE

The lock field (LK) in MBC0_MEMN_GLBAC1, MBC0_MEMN_GLBAC2, MBC0_MEMN_GLBAC3, MBC0_MEMN_GLBAC6 and MBC0_MEMN_GLBAC7 registers are set to lock state (LK= 1) at reset. Hence these registers cannot be changed and blocks that are assigned to one of these global access control indexes cannot change the access policy index until next reset.

21.1.1 Module instances

This device has one instance of the MBC module.

The following table lists the memory blocks of MBC.

Table 406. Memory blocks of MBC

MBC0_MEMn	Memory	Block size	No. of blocks
MEM0	Main flash	32 KB	64
MEM1	IFR0	8 KB	8
MEM2	IFR1	4 KB	4
MEM3	Not used	Not used	Not used

21.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

21.2 Overview

The Memory Block Checker (MBC) implements access controls for on-chip internal memories based on a fixed-sized block format. MBC provides hardware access control for system bus references targeted at on-chip memory spaces.

Using the pre-programmed region descriptors with fixed-sized block format and their associated access rights, the MBC concurrently monitors system bus transactions and evaluates the appropriateness of each transfer. Memory references with sufficient access control rights are completed, while the references that are not mapped to any region descriptor or have insufficient rights are terminated with an access error response.

21.2.1 Block diagram

The Memory Block Checker (MBC) implements access controls for on-chip internal memories and slave peripherals based on a fixed-sized block format.

See Figure 83 for MBC block diagram focusing on topology and connections.

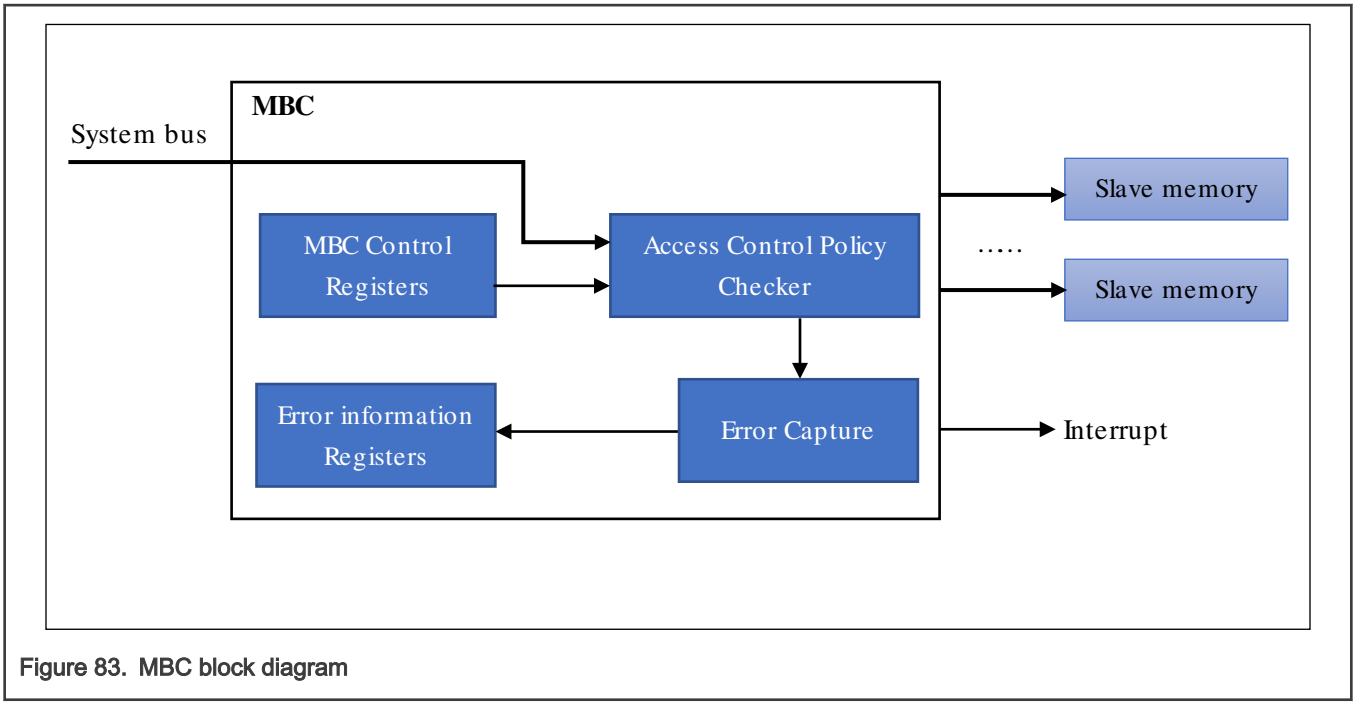


Figure 83. MBC block diagram

21.2.2 Features

Defines access rights to slave targets defined in MBCs for on-chip memories.

21.3 Functional description

This section provides more details on the operation and implementation of MBC.

21.3.1 Memory Block Checker (MBC)

The Memory Block Checker provides access control for system bus references targeted to on-chip internal memories . Using programmed block configuration registers which define the access rights per block, the MBC concurrently monitors system bus transactions and evaluates the appropriateness of each transfer. Memory references that have sufficient access rights are allowed to complete, while references that have insufficient rights are terminated with an access error response.

Refer to the chip-specific section to determine the number and size of blocks for each sub memory supported on this device.

Each MBCm_DOMd_BLK_CFG_W contains eight access control structures. Each structure is comprised of a 3-bit MBACSEL (Memory Block Access Control Select). The MBACSEL field selects a MBCm_MEMN_GLBACr register, which controls the read/write/execute/lock access to the corresponding block. Below table provides the eigth possible access control combinations. In case you are looking for different access combination, such as write and execute allowed but read blocked, you can modify one of the unlocked configurations registers (MBC0_MEMN_GLBAC0 or MBC0_MEMN_GLBAC4 or MBC0_MEMN_GLBAC5) to 0x0000_3300.

Table below depicts eigth possible access control combinations.

Table 407. Access control combinations

GAC Index	Register name	Read	Write	Execute	Lock	Description
0	MBC0_MEMN_GLBAC0 (0x0000_6600)	1	1	0	0	**Data flash and unlocked** (RW__). Default flash behavior with read and write access only. Instruction fetches (execute access) are blocked and generates security violation. Blocks assigned with this access control index can be changed to another index by software. This configuration register is unlocked and updatable by application.
1	MBC0_MEMN_GLBAC1 (0x8000_6600)	1	1	0	1	**Data flash and locked** (RW_L). Read and write operations are allowed but execution is blocked. Blocks assigned with this access control index cannot be changed to another index until next reset.
2	MBC0_MEMN_GLBAC2 (0x8000_5500)	1	0	1	1	**Read only Memory (ROM) and locked** (R_XL) Read & execute operations are allowed but write is blocked. Blocks assigned with this access control index cannot be changed to another index until next reset.
3	MBC0_MEMN_GLBAC3 (0x8000_4400)	1	0	0	1	**Data Read only memory (DROM) and locked** (R__L) Read operations is allowed but write & execute are blocked. Blocks assigned with this access control index cannot be changed to another index until next reset.
4	MBC0_MEMN_GLBAC4 (0x0000_5500)	1	0	1	0	**Read-Only-Memory (ROM) and unlocked** (R_X_) Read & execute operations are allowed but write is blocked.

Table continues on the next page...

Table 407. Access control combinations (continued)

GAC Index	Register name	Read	Write	Execute	Lock	Description
						Blocks assigned with this access control index can be changed to another index by software. This configuration register is unlocked and updatable by application.
5	MBC0_MEMN_GLBAC5 (0x0000_1100)	0	0	1	0	**eXecute-only Memory (XOM) and unlocked** (__X_) Execute operations are allowed but read & write are blocked. Blocks assigned with this access control index can be changed to another index by software. This configuration register is unlocked and updatable by application.
6	MBC0_MEMN_GLBAC6 (0x8000_1100)	0	0	1	1	**eXecute-Only-Memory (XOM) and locked** (__XL) Execute operations are allowed but read & write are blocked. Blocks assigned with this access control index cannot be changed to another index until next reset.
7	MBC0_MEMN_GLBAC7 (0x8000_0000)	0	0	0	1	**Hidden** (___L) Read, Write & execute operations are blocked. Once this access control index is assigned to a block, the memory range associated with block is hidden until next reset.

GLBAC1-7 also have a lock bit MBCm_MEMn_GLBACr[LK]. When LK=1, the GLBACr register is read-only until the next reset. Furthermore, if MBCm_DOMd_BLK_CFG_W[MBACSEL] selects a GLBAC that is locked, the 3-bit MBCACSEL field is also locked until the next reset. GLBAC0 cannot be locked.

21.3.1.1 Memory block hit determination

MBCm_MEMs_GLBCFG[NBLK, SIZE_LOG2] is used to determine which bits of the address correspond to block number.

```
block_n_hit = (addr[MSB:LSB] == n)
where LSB=SIZE_LOG2
MSB=f{NBLKS,LSB}
```

Note that the block hit determination is based only on the address comparison of the first byte being accessed; that is, the MBC does not check the size of the access to make sure it entirely fits within the region.

21.3.1.2 Memory block access evaluation

For a block n hit, the MBC logic evaluates the access rights defined by the MBCm_DOMd_BLK_CFG_Ww registers. The block number hit selects the appropriate MBCm_DOMd_BLK_CFG_Ww register to use in the access evaluation. The {R,W,X}, nonsecure and privilege attributes of the access are compared with the MBC global access control policy defined in MBCm_DOMd_BLK_CFG_W[MBACSEL, NSE] fields.

MBC access evaluation terminates the access with a bus error and reports an access error for only one condition - when the access doesn't have sufficient access rights. By nature, the block checker can only hit in ONE block, and by implementation, there are no "miss" accesses. Accesses outside of the range covered by the MBC are not sent to the MBC. There also aren't any individual block valid bits.

21.3.2 Interrupts

This module outputs an interrupt signal which can be connected to the system's interrupt controller. Please check chip-specific interrupt assignment for details. Interrupt is asserted on detection of access violation by any checker, and it remains asserted until MBCn_MEM3_GLBCFG[CLRE] is cleared. This interrupt is in addition to interrupt generated by master after receiving bus error due to access violation by memory block checker . So, system may receive interrupt through different channels but from the same access violation at checker.

21.4 External signals

This module has no external signals.

21.5 Register descriptions

Unless noted otherwise, the programming model registers can be accessed via 8-, 16- or 32-bit reads and 32-bit write references. Attempted accesses in a different operating mode, using unsupported write data sizes, writes to read-only resources, or access to reserved spaces are terminated with an error unless noted otherwise.

21.5.1 MBC register descriptions

21.5.1.1 MBC memory map

MBC0.MBC base address: 400C_7000h

Offset	Register	Width (In bits)	Access	Reset value
0h	MBC Global Configuration Register (MBC0_MEM0_GLBCFG)	32	R	000F_0040h
4h	MBC Global Configuration Register (MBC0_MEM1_GLBCFG)	32	R	000D_0008h
8h	MBC Global Configuration Register (MBC0_MEM2_GLBCFG)	32	R	000C_0004h
Ch	MBC Global Configuration Register (MBC0_MEM3_GLBCFG)	32	RW	0000_0000h
20h	MBC Global Access Control (MBC0_MEMN_GLBAC0)	32	RW	0000_6600h
24h	MBC Global Access Control (MBC0_MEMN_GLBAC1)	32	RW	8000_6600h
28h	MBC Global Access Control (MBC0_MEMN_GLBAC2)	32	RW	8000_5500h
2Ch	MBC Global Access Control (MBC0_MEMN_GLBAC3)	32	RW	8000_4400h
30h	MBC Global Access Control (MBC0_MEMN_GLBAC4)	32	RW	0000_5500h
34h	MBC Global Access Control (MBC0_MEMN_GLBAC5)	32	RW	0000_1100h
38h	MBC Global Access Control (MBC0_MEMN_GLBAC6)	32	RW	8000_1100h
3Ch	MBC Global Access Control (MBC0_MEMN_GLBAC7)	32	RW	8000_0000h
40h	MBC Memory Block Configuration Word (MBC0_DOM0_MEM0_BLK_CFG_W0)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
44h	MBC Memory Block Configuration Word (MBC0_DOM0_MEM0_BLK_CFG_W1)	32	RW	0000_0000h
48h	MBC Memory Block Configuration Word (MBC0_DOM0_MEM0_BLK_CFG_W2)	32	RW	0000_0000h
4Ch	MBC Memory Block Configuration Word (MBC0_DOM0_MEM0_BLK_CFG_W3)	32	RW	0000_0000h
50h	MBC Memory Block Configuration Word (MBC0_DOM0_MEM0_BLK_CFG_W4)	32	RW	0000_0000h
54h	MBC Memory Block Configuration Word (MBC0_DOM0_MEM0_BLK_CFG_W5)	32	RW	0000_0000h
58h	MBC Memory Block Configuration Word (MBC0_DOM0_MEM0_BLK_CFG_W6)	32	RW	0000_0000h
5Ch	MBC Memory Block Configuration Word (MBC0_DOM0_MEM0_BLK_CFG_W7)	32	RW	0000_0000h
180h	MBC Memory Block Configuration Word (MBC0_DOM0_MEM1_BLK_CFG_W0)	32	RW	0000_0000h
1A8h	MBC Memory Block Configuration Word (MBC0_DOM0_MEM2_BLK_CFG_W0)	32	RW	0000_0000h

21.5.1.2 MBC Global Configuration Register (MBC0_MEM0_GLBCFG - MBC0_MEM3_GLBCFG)

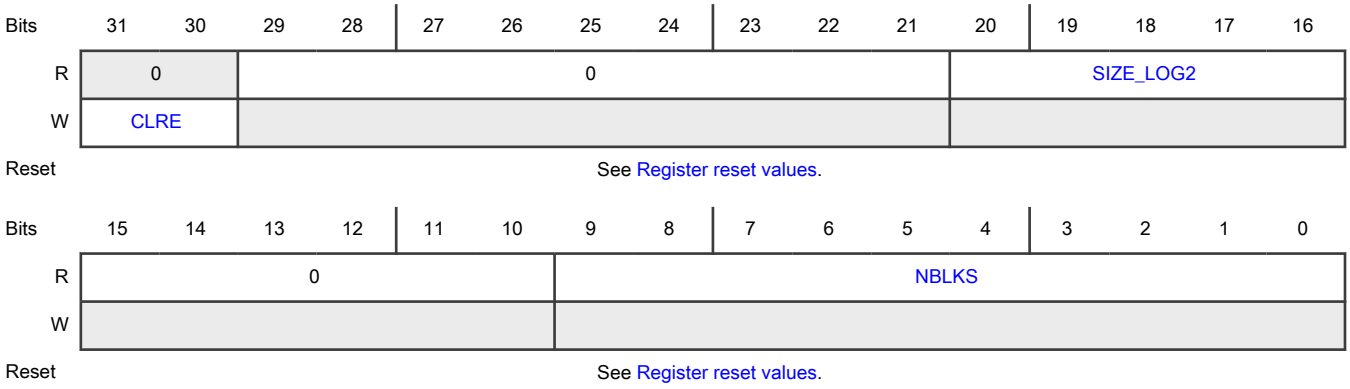
Offset

Register	Offset
MBC0_MEM0_GLBCFG	0h
MBC0_MEM1_GLBCFG	4h
MBC0_MEM2_GLBCFG	8h
MBC0_MEM3_GLBCFG	Ch

Function

These MBC global configuration read-only registers contain information on the MBC's hardware configuration. Specifically, it defines the number of memory blocks and the size of each block in each MBC mem (r).

Diagram



Register reset values

Register	Reset value
MBC0_MEM0_GLBCFG	000F_0040h
MBC0_MEM1_GLBCFG	000D_0008h
MBC0_MEM2_GLBCFG	000C_0004h
MBC0_MEM3_GLBCFG	0000_0000h

Fields

Field	Function
31-30 CLRE	Clear Error This 2-bit, write-only field controls the clearing of an access violation error and any interrupt due to error, on writing 01b to this field. A write of any value other than 01b has no effect. **NOTE** This field is not supported in every instance. The following table includes only supported registers.

Instance	Field supported in	Field not supported in
MBC0.MBC	MBC0_MEM3_GLBCFG	MBC0_MEM0_GLBCFG–MBC0_MEM2_GLBCFG

29-21 —	Reserved
20-16 SIZE_LOG2	Log2 size per block For example SIZE_LOG2=0x0C is 2^12=4 KB blocks.

Table continues on the next page...

Table continued from the previous page...

Field	Function
15-10 —	Reserved
9-0 NBLKS	Number of blocks in this memory

21.5.1.3 MBC Global Access Control (MBC0_MEMN_GLBAC0)

Offset

Register	Offset
MBC0_MEMN_GLBAC0	20h

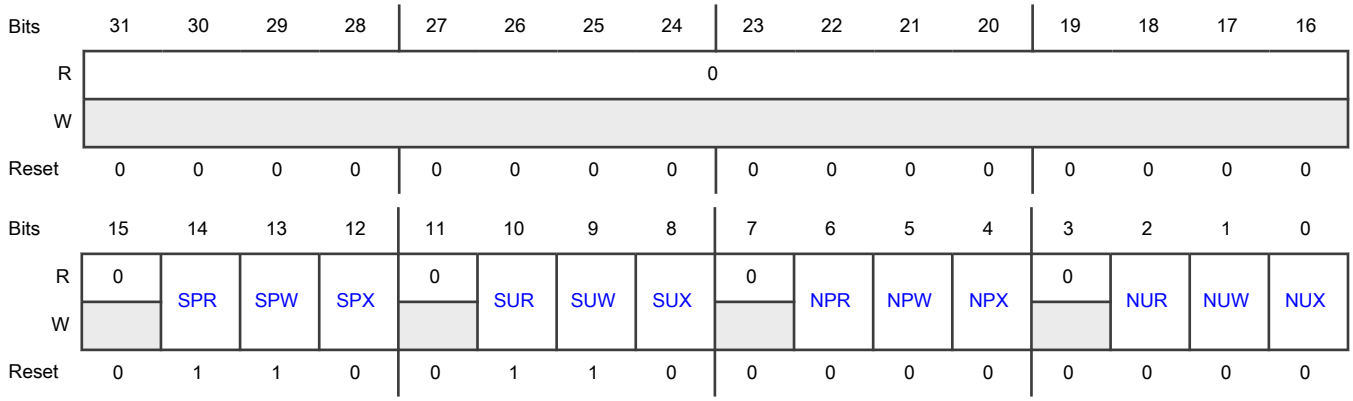
Function

These fully programmable R/W fields define the R/W/X (read/write/execute) access control flags with each of the 4 supported operating modes: SecurePriv, SecureUser, NonsecurePriv, Nonsecure User.

NOTE

The lock field (LK) in MBC0_MEMN_GLBAC1, MBC0_MEMN_GLBAC2, MBC0_MEMN_GLBAC3, MBC0_MEMN_GLBAC6 and MBC0_MEMN_GLBAC7 registers are set to lock state (LK= 1) at reset. Hence these registers cannot be changed and blocks that are assigned to one of these global access control indexes cannot change the access policy index until next reset.

Diagram



Fields

Field	Function
31-15	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
14 SPR	SecurePriv Read SecurePriv read access control flag 0b - Read access is not allowed in Secure Privilege mode. 1b - Read access is allowed in Secure Privilege mode.
13 SPW	SecurePriv Write SecurePriv write access control flag 0b - Write access is not allowed in Secure Privilege mode. 1b - Write access is allowed in Secure Privilege mode.
12 SPX	SecurePriv Execute SecurePriv execute access control flag 0b - Execute access is not allowed in Secure Privilege mode. 1b - Execute access is allowed in Secure Privilege mode.
11 —	Reserved
10 SUR	SecureUser Read SecureUser read access control flag 0b - Read access is not allowed in Secure User mode. 1b - Read access is allowed in Secure User mode.
9 SUW	SecureUser Write SecureUser write access control flag 0b - Write access is not allowed in Secure User mode. 1b - Write access is allowed in Secure User mode.
8 SUX	SecureUser Execute SecureUser execute access control flag 0b - Execute access is not allowed in Secure User mode. 1b - Execute access is allowed in Secure User mode.
7 —	Reserved
6	NonsecurePriv Read NonsecurePriv read access control flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
NPR	0b - Read access is not allowed in Nonsecure Privilege mode. 1b - Read access is allowed in Nonsecure Privilege mode.
5 NPW	NonsecurePriv Write NonsecurePriv write access control flag 0b - Write access is not allowed in Nonsecure Privilege mode. 1b - Write access is allowed in Nonsecure Privilege mode.
4 NPX	NonsecurePriv Execute NonsecurePriv execute access control flag 0b - Execute access is not allowed in Nonsecure Privilege mode. 1b - Execute access is allowed in Nonsecure Privilege mode.
3 —	Reserved
2 NUR	NonsecureUser Read NonsecureUser read access control flag 0b - Read access is not allowed in Nonsecure User mode. 1b - Read access is allowed in Nonsecure User mode.
1 NUW	NonsecureUser Write NonsecureUser write access control flag 0b - Write access is not allowed in Nonsecure User mode. 1b - Write access is allowed in Nonsecure User mode.
0 NUX	NonsecureUser Execute NonsecureUser execute access control flag 0b - Execute access is not allowed in Nonsecure User mode. 1b - Execute access is allowed in Nonsecure User mode.

21.5.1.4 MBC Global Access Control (MBC0_MEMN_GLBAC1)

Offset

Register	Offset
MBC0_MEMN_GLBAC1	24h

Function

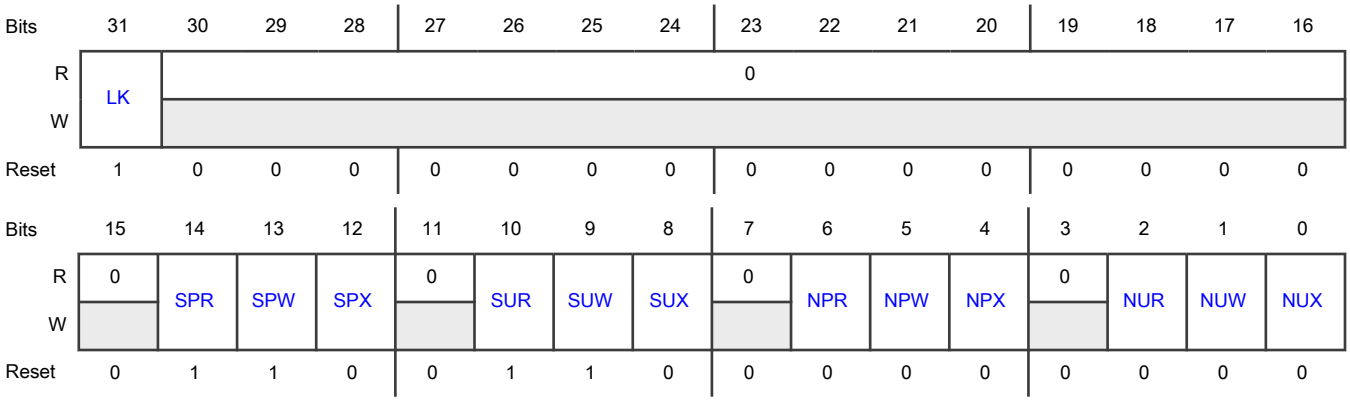
These fully programmable R/W fields define the R/W/X (read/write/execute) access control flags with each of the 4 supported operating modes: SecurePriv, SecureUser, NonsecurePriv, Nonsecure User. The lock bit provides the ability to lock these control access flags and to lock the MBC_BLK_CFG[MBACSEL] field when a locked MBC_MEMN_GLBAC register is selected.

This register is not writable.

NOTE

The lock field (LK) in MBC0_MEMN_GLBAC1, MBC0_MEMN_GLBAC2, MBC0_MEMN_GLBAC3, MBC0_MEMN_GLBAC6 and MBC0_MEMN_GLBAC7 registers are set to lock state (LK= 1) at reset. Hence these registers cannot be changed and blocks that are assigned to one of these global access control indexes cannot change the access policy index until next reset.

Diagram



Fields

Field	Function
31 LK	LOCK This 1-bit field provides a mechanism to limit writes to the this register to protect its contents. Once set, this bit remains asserted until the next reset. 0b - This register is not locked and can be altered. 1b - This register is locked and cannot be altered.
30-15 —	Reserved
14 SPR	SecurePriv Read SecurePriv read access control flag 0b - Read access is not allowed in Secure Privilege mode. 1b - Read access is allowed in Secure Privilege mode.
13 SPW	SecurePriv Write SecurePriv write access control flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Write access is not allowed in Secure Privilege mode. 1b - Write access is allowed in Secure Privilege mode.
12 SPX	SecurePriv Execute SecurePriv execute access control flag 0b - Execute access is not allowed in Secure Privilege mode. 1b - Execute access is allowed in Secure Privilege mode.
11 —	Reserved
10 SUR	SecureUser Read SecureUser read access control flag 0b - Read access is not allowed in Secure User mode. 1b - Read access is allowed in Secure User mode.
9 SUW	SecureUser Write SecureUser write access control flag 0b - Write access is not allowed in Secure User mode. 1b - Write access is allowed in Secure User mode.
8 SUX	SecureUser Execute SecureUser execute access control flag 0b - Execute access is not allowed in Secure User mode. 1b - Execute access is allowed in Secure User mode.
7 —	Reserved
6 NPR	NonsecurePriv Read NonsecurePriv read access control flag 0b - Read access is not allowed in Nonsecure Privilege mode. 1b - Read access is allowed in Nonsecure Privilege mode.
5 NPW	NonsecurePriv Write NonsecurePriv write access control flag 0b - Write access is not allowed in Nonsecure Privilege mode. 1b - Write access is allowed in Nonsecure Privilege mode.
4	NonsecurePriv Execute

Table continues on the next page...

Table continued from the previous page...

Field	Function
NPX	NonsecurePriv execute access control flag 0b - Execute access is not allowed in Nonsecure Privilege mode. 1b - Execute access is allowed in Nonsecure Privilege mode.
3 —	Reserved
2 NUR	NonsecureUser Read NonsecureUser read access control flag 0b - Read access is not allowed in Nonsecure User mode. 1b - Read access is allowed in Nonsecure User mode.
1 NUW	NonsecureUser Write NonsecureUser write access control flag 0b - Write access is not allowed in Nonsecure User mode. 1b - Write access is allowed in Nonsecure User mode.
0 NUX	NonsecureUser Execute NonsecureUser execute access control flag 0b - Execute access is not allowed in Nonsecure User mode. 1b - Execute access is allowed in Nonsecure User mode.

21.5.1.5 MBC Global Access Control (MBC0_MEMN_GLBAC2)

Offset

Register	Offset
MBC0_MEMN_GLBAC2	28h

Function

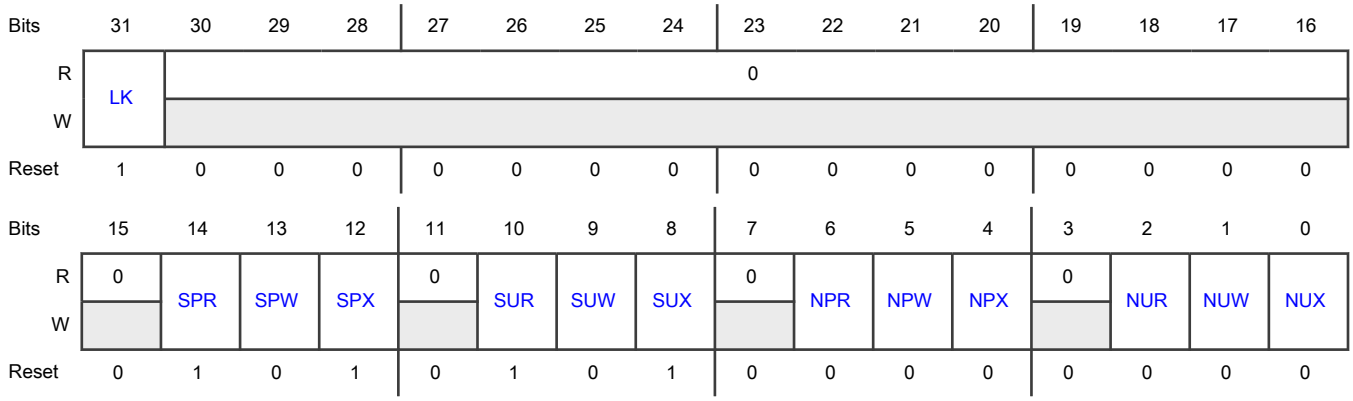
These fully programmable R/W fields define the R/W/X (read/write/execute) access control flags with each of the 4 supported operating modes: SecurePriv, SecureUser, NonsecurePriv, Nonsecure User. The lock bit provides the ability to lock these control access flags and to lock the MBC_BLK_CFG[MBACSEL] field when a locked MBC_MEMN_GLBAC register is selected.

This register is not writable.

NOTE

The lock field (LK) in MBC0_MEMN_GLBAC1, MBC0_MEMN_GLBAC2, MBC0_MEMN_GLBAC3, MBC0_MEMN_GLBAC6 and MBC0_MEMN_GLBAC7 registers are set to lock state (LK= 1) at reset. Hence these registers cannot be changed and blocks that are assigned to one of these global access control indexes cannot change the access policy index until next reset.

Diagram



Fields

Field	Function
31 LK	LOCK This 1-bit field provides a mechanism to limit writes to the this register to protect its contents. Once set, this bit remains asserted until the next reset. 0b - This register is not locked and can be altered. 1b - This register is locked and cannot be altered.
30-15 —	Reserved
14 SPR	SecurePriv Read SecurePriv read access control flag 0b - Read access is not allowed in Secure Privilege mode. 1b - Read access is allowed in Secure Privilege mode.
13 SPW	SecurePriv Write SecurePriv write access control flag 0b - Write access is not allowed in Secure Privilege mode. 1b - Write access is allowed in Secure Privilege mode.
12 SPX	SecurePriv Execute SecurePriv execute access control flag 0b - Execute access is not allowed in Secure Privilege mode. 1b - Execute access is allowed in Secure Privilege mode.
11 —	Reserved
10	SecureUser Read

Table continues on the next page...

Table continued from the previous page...

Field	Function
SUR	SecureUser read access control flag 0b - Read access is not allowed in Secure User mode. 1b - Read access is allowed in Secure User mode.
9 SUW	SecureUser Write SecureUser write access control flag 0b - Write access is not allowed in Secure User mode. 1b - Write access is allowed in Secure User mode.
8 SUX	SecureUser Execute SecureUser execute access control flag 0b - Execute access is not allowed in Secure User mode. 1b - Execute access is allowed in Secure User mode.
7 —	Reserved
6 NPR	NonsecurePriv Read NonsecurePriv read access control flag 0b - Read access is not allowed in Nonsecure Privilege mode. 1b - Read access is allowed in Nonsecure Privilege mode.
5 NPW	NonsecurePriv Write NonsecurePriv write access control flag 0b - Write access is not allowed in Nonsecure Privilege mode. 1b - Write access is allowed in Nonsecure Privilege mode.
4 NPX	NonsecurePriv Execute NonsecurePriv execute access control flag 0b - Execute access is not allowed in Nonsecure Privilege mode. 1b - Execute access is allowed in Nonsecure Privilege mode.
3 —	Reserved
2 NUR	NonsecureUser Read NonsecureUser read access control flag 0b - Read access is not allowed in Nonsecure User mode. 1b - Read access is allowed in Nonsecure User mode.

Table continues on the next page...

Table continued from the previous page...

Field	Function
1 NUW	NonsecureUser Write NonsecureUser write access control flag 0b - Write access is not allowed in Nonsecure User mode. 1b - Write access is allowed in Nonsecure User mode.
0 NUX	NonsecureUser Execute NonsecureUser execute access control flag 0b - Execute access is not allowed in Nonsecure User mode. 1b - Execute access is allowed in Nonsecure User mode.

21.5.1.6 MBC Global Access Control (MBC0_MEMN_GLBAC3)

Offset

Register	Offset
MBC0_MEMN_GLBAC3	2Ch

Function

These fully programmable R/W fields define the R/W/X (read/write/execute) access control flags with each of the 4 supported operating modes: SecurePriv, SecureUser, NonsecurePriv, Nonsecure User. The lock bit provides the ability to lock these control access flags and to lock the MBC_BLK_CFG[MBACSEL] field when a locked MBC_MEMN_GLBAC register is selected.

This register is not writable.

NOTE

The lock field (LK) in MBC0_MEMN_GLBAC1, MBC0_MEMN_GLBAC2, MBC0_MEMN_GLBAC3, MBC0_MEMN_GLBAC6 and MBC0_MEMN_GLBAC7 registers are set to lock state (LK= 1) at reset. Hence these registers cannot be changed and blocks that are assigned to one of these global access control indexes cannot change the access policy index until next reset.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	LK	0														
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	SPR	SPW	SPX	0	SUR	SUW	SUX	0	NPR	NPW	NPX	0	NUR	NUW	NUX
W																
Reset	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 LK	LOCK This 1-bit field provides a mechanism to limit writes to the this register to protect its contents. Once set, this bit remains asserted until the next reset. 0b - This register is not locked and can be altered. 1b - This register is locked and cannot be altered.
30-15 —	Reserved
14 SPR	SecurePriv Read SecurePriv read access control flag 0b - Read access is not allowed in Secure Privilege mode. 1b - Read access is allowed in Secure Privilege mode.
13 SPW	SecurePriv Write SecurePriv write access control flag 0b - Write access is not allowed in Secure Privilege mode. 1b - Write access is allowed in Secure Privilege mode.
12 SPX	SecurePriv Execute SecurePriv execute access control flag 0b - Execute access is not allowed in Secure Privilege mode. 1b - Execute access is allowed in Secure Privilege mode.
11 —	Reserved
10 SUR	SecureUser Read SecureUser read access control flag 0b - Read access is not allowed in Secure User mode. 1b - Read access is allowed in Secure User mode.
9 SUW	SecureUser Write SecureUser write access control flag 0b - Write access is not allowed in Secure User mode. 1b - Write access is allowed in Secure User mode.
8 SUX	SecureUser Execute SecureUser execute access control flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Execute access is not allowed in Secure User mode. 1b - Execute access is allowed in Secure User mode.
7 —	Reserved
6 NPR	NonsecurePriv Read NonsecurePriv read access control flag 0b - Read access is not allowed in Nonsecure Privilege mode. 1b - Read access is allowed in Nonsecure Privilege mode.
5 NPW	NonsecurePriv Write NonsecurePriv write access control flag 0b - Write access is not allowed in Nonsecure Privilege mode. 1b - Write access is allowed in Nonsecure Privilege mode.
4 NPX	NonsecurePriv Execute NonsecurePriv execute access control flag 0b - Execute access is not allowed in Nonsecure Privilege mode. 1b - Execute access is allowed in Nonsecure Privilege mode.
3 —	Reserved
2 NUR	NonsecureUser Read NonsecureUser read access control flag 0b - Read access is not allowed in Nonsecure User mode. 1b - Read access is allowed in Nonsecure User mode.
1 NUW	NonsecureUser Write NonsecureUser write access control flag 0b - Write access is not allowed in Nonsecure User mode. 1b - Write access is allowed in Nonsecure User mode.
0 NUX	NonsecureUser Execute NonsecureUser execute access control flag 0b - Execute access is not allowed in Nonsecure User mode. 1b - Execute access is allowed in Nonsecure User mode.

21.5.1.7 MBC Global Access Control (MBC0_MEMN_GLBAC4)

Offset

Register	Offset
MBC0_MEMN_GLBAC4	30h

Function

These fully programmable R/W fields define the R/W/X (read/write/execute) access control flags with each of the 4 supported operating modes: SecurePriv, SecureUser, NonsecurePriv, Nonsecure User. The lock bit provides the ability to lock these control access flags and to lock the MBC_BLK_CFG[MBACSEL] field when a locked MBC_MEMN_GLBAC register is selected.

NOTE

The lock field (LK) in MBC0_MEMN_GLBAC1, MBC0_MEMN_GLBAC2, MBC0_MEMN_GLBAC3, MBC0_MEMN_GLBAC6 and MBC0_MEMN_GLBAC7 registers are set to lock state (LK= 1) at reset. Hence these registers cannot be changed and blocks that are assigned to one of these global access control indexes cannot change the access policy index until next reset.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	LK	0														
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	SPR	SPW	SPX	0	SUR	SUW	SUX	0	NPR	NPW	NPX	0	NUR	NUW	NUX
W																
Reset	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0

Fields

Field	Function
31 LK	LOCK This 1-bit field provides a mechanism to limit writes to the this register to protect its contents. Once set, this bit remains asserted until the next reset. 0b - This register is not locked and can be altered. 1b - This register is locked and cannot be altered.
30-15 —	Reserved
14	SecurePriv Read

Table continues on the next page...

Table continued from the previous page...

Field	Function
SPR	SecurePriv read access control flag 0b - Read access is not allowed in Secure Privilege mode. 1b - Read access is allowed in Secure Privilege mode.
13 SPW	SecurePriv Write SecurePriv write access control flag 0b - Write access is not allowed in Secure Privilege mode. 1b - Write access is allowed in Secure Privilege mode.
12 SPX	SecurePriv Execute SecurePriv execute access control flag 0b - Execute access is not allowed in Secure Privilege mode. 1b - Execute access is allowed in Secure Privilege mode.
11 —	Reserved
10 SUR	SecureUser Read SecureUser read access control flag 0b - Read access is not allowed in Secure User mode. 1b - Read access is allowed in Secure User mode.
9 SUW	SecureUser Write SecureUser write access control flag 0b - Write access is not allowed in Secure User mode. 1b - Write access is allowed in Secure User mode.
8 SUX	SecureUser Execute SecureUser execute access control flag 0b - Execute access is not allowed in Secure User mode. 1b - Execute access is allowed in Secure User mode.
7 —	Reserved
6 NPR	NonsecurePriv Read NonsecurePriv read access control flag 0b - Read access is not allowed in Nonsecure Privilege mode. 1b - Read access is allowed in Nonsecure Privilege mode.

Table continues on the next page...

Table continued from the previous page...

Field	Function
5 NPW	NonsecurePriv Write NonsecurePriv write access control flag 0b - Write access is not allowed in Nonsecure Privilege mode. 1b - Write access is allowed in Nonsecure Privilege mode.
4 NPX	NonsecurePriv Execute NonsecurePriv execute access control flag 0b - Execute access is not allowed in Nonsecure Privilege mode. 1b - Execute access is allowed in Nonsecure Privilege mode.
3 —	Reserved
2 NUR	NonsecureUser Read NonsecureUser read access control flag 0b - Read access is not allowed in Nonsecure User mode. 1b - Read access is allowed in Nonsecure User mode.
1 NUW	NonsecureUser Write NonsecureUser write access control flag 0b - Write access is not allowed in Nonsecure User mode. 1b - Write access is allowed in Nonsecure User mode.
0 NUX	NonsecureUser Execute NonsecureUser execute access control flag 0b - Execute access is not allowed in Nonsecure User mode. 1b - Execute access is allowed in Nonsecure User mode.

21.5.1.8 MBC Global Access Control (MBC0_MEMN_GLBAC5)

Offset

Register	Offset
MBC0_MEMN_GLBAC5	34h

Function

These fully programmable R/W fields define the R/W/X (read/write/execute) access control flags with each of the 4 supported operating modes: SecurePriv, SecureUser, NonsecurePriv, Nonsecure User. The lock bit provides the ability to lock these control access flags and to lock the MBC_BLK_CFG[MBACSEL] field when a locked MBC_MEMN_GLBAC register is selected.

NOTE

The lock field (LK) in MBC0_MEMN_GLBAC1, MBC0_MEMN_GLBAC2, MBC0_MEMN_GLBAC3, MBC0_MEMN_GLBAC6 and MBC0_MEMN_GLBAC7 registers are set to lock state (LK= 1) at reset. Hence these registers cannot be changed and blocks that are assigned to one of these global access control indexes cannot change the access policy index until next reset.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	LK	0														
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	SPR	SPW	SPX	0	SUR	SUW	SUX	0	NPR	NPW	NPX	0	NUR	NUW	NUX
W																
Reset	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0

Fields

Field	Function
31 LK	LOCK This 1-bit field provides a mechanism to limit writes to the this register to protect its contents. Once set, this bit remains asserted until the next reset. 0b - This register is not locked and can be altered. 1b - This register is locked and cannot be altered.
30-15 —	Reserved
14 SPR	SecurePriv Read SecurePriv read access control flag 0b - Read access is not allowed in Secure Privilege mode. 1b - Read access is allowed in Secure Privilege mode.
13 SPW	SecurePriv Write SecurePriv write access control flag 0b - Write access is not allowed in Secure Privilege mode. 1b - Write access is allowed in Secure Privilege mode.
12 SPX	SecurePriv Execute SecurePriv execute access control flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Execute access is not allowed in Secure Privilege mode. 1b - Execute access is allowed in Secure Privilege mode.
11 —	Reserved
10 SUR	SecureUser Read SecureUser read access control flag 0b - Read access is not allowed in Secure User mode. 1b - Read access is allowed in Secure User mode.
9 SUW	SecureUser Write SecureUser write access control flag 0b - Write access is not allowed in Secure User mode. 1b - Write access is allowed in Secure User mode.
8 SUX	SecureUser Execute SecureUser execute access control flag 0b - Execute access is not allowed in Secure User mode. 1b - Execute access is allowed in Secure User mode.
7 —	Reserved
6 NPR	NonsecurePriv Read NonsecurePriv read access control flag 0b - Read access is not allowed in Nonsecure Privilege mode. 1b - Read access is allowed in Nonsecure Privilege mode.
5 NPW	NonsecurePriv Write NonsecurePriv write access control flag 0b - Write access is not allowed in Nonsecure Privilege mode. 1b - Write access is allowed in Nonsecure Privilege mode.
4 NPX	NonsecurePriv Execute NonsecurePriv execute access control flag 0b - Execute access is not allowed in Nonsecure Privilege mode. 1b - Execute access is allowed in Nonsecure Privilege mode.
3	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
2 NUR	NonsecureUser Read NonsecureUser read access control flag 0b - Read access is not allowed in Nonsecure User mode. 1b - Read access is allowed in Nonsecure User mode.
1 NUW	NonsecureUser Write NonsecureUser write access control flag 0b - Write access is not allowed in Nonsecure User mode. 1b - Write access is allowed in Nonsecure User mode.
0 NUX	NonsecureUser Execute NonsecureUser execute access control flag 0b - Execute access is not allowed in Nonsecure User mode. 1b - Execute access is allowed in Nonsecure User mode.

21.5.1.9 MBC Global Access Control (MBC0_MEMN_GLBAC6)

Offset

Register	Offset
MBC0_MEMN_GLBAC6	38h

Function

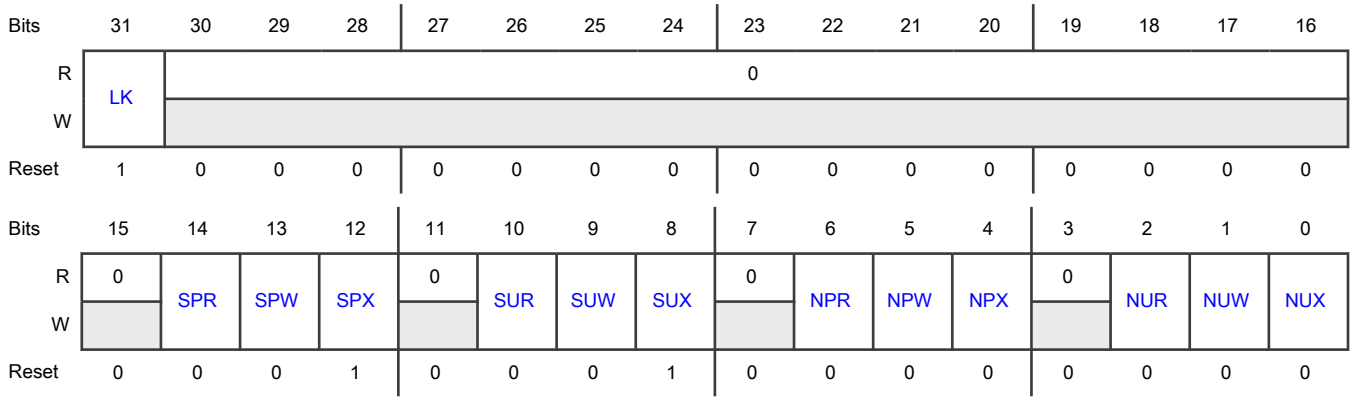
These fully programmable R/W fields define the R/W/X (read/write/execute) access control flags with each of the 4 supported operating modes: SecurePriv, SecureUser, NonsecurePriv, Nonsecure User. The lock bit provides the ability to lock these control access flags and to lock the MBC_BLK_CFG[MBACSEL] field when a locked MBC_MEMN_GLBAC register is selected.

This register is not writable.

NOTE

The lock field (LK) in MBC0_MEMN_GLBAC1, MBC0_MEMN_GLBAC2, MBC0_MEMN_GLBAC3, MBC0_MEMN_GLBAC6 and MBC0_MEMN_GLBAC7 registers are set to lock state (LK= 1) at reset. Hence these registers cannot be changed and blocks that are assigned to one of these global access control indexes cannot change the access policy index until next reset.

Diagram



Fields

Field	Function
31 LK	LOCK This 1-bit field provides a mechanism to limit writes to the this register to protect its contents. Once set, this bit remains asserted until the next reset. 0b - This register is not locked and can be altered. 1b - This register is locked and cannot be altered.
30-15 —	Reserved
14 SPR	SecurePriv Read SecurePriv read access control flag 0b - Read access is not allowed in Secure Privilege mode. 1b - Read access is allowed in Secure Privilege mode.
13 SPW	SecurePriv Write SecurePriv write access control flag 0b - Write access is not allowed in Secure Privilege mode. 1b - Write access is allowed in Secure Privilege mode.
12 SPX	SecurePriv Execute SecurePriv execute access control flag 0b - Execute access is not allowed in Secure Privilege mode. 1b - Execute access is allowed in Secure Privilege mode.
11 —	Reserved
10	SecureUser Read

Table continues on the next page...

Table continued from the previous page...

Field	Function
SUR	SecureUser read access control flag 0b - Read access is not allowed in Secure User mode. 1b - Read access is allowed in Secure User mode.
9 SUW	SecureUser Write SecureUser write access control flag 0b - Write access is not allowed in Secure User mode. 1b - Write access is allowed in Secure User mode.
8 SUX	SecureUser Execute SecureUser execute access control flag 0b - Execute access is not allowed in Secure User mode. 1b - Execute access is allowed in Secure User mode.
7 —	Reserved
6 NPR	NonsecurePriv Read NonsecurePriv read access control flag 0b - Read access is not allowed in Nonsecure Privilege mode. 1b - Read access is allowed in Nonsecure Privilege mode.
5 NPW	NonsecurePriv Write NonsecurePriv write access control flag 0b - Write access is not allowed in Nonsecure Privilege mode. 1b - Write access is allowed in Nonsecure Privilege mode.
4 NPX	NonsecurePriv Execute NonsecurePriv execute access control flag 0b - Execute access is not allowed in Nonsecure Privilege mode. 1b - Execute access is allowed in Nonsecure Privilege mode.
3 —	Reserved
2 NUR	NonsecureUser Read NonsecureUser read access control flag 0b - Read access is not allowed in Nonsecure User mode. 1b - Read access is allowed in Nonsecure User mode.

Table continues on the next page...

Table continued from the previous page...

Field	Function
1 NUW	NonsecureUser Write NonsecureUser write access control flag 0b - Write access is not allowed in Nonsecure User mode. 1b - Write access is allowed in Nonsecure User mode.
0 NUX	NonsecureUser Execute NonsecureUser execute access control flag 0b - Execute access is not allowed in Nonsecure User mode. 1b - Execute access is allowed in Nonsecure User mode.

21.5.1.10 MBC Global Access Control (MBC0_MEMN_GLBAC7)

Offset

Register	Offset
MBC0_MEMN_GLBAC7	3Ch

Function

These fully programmable R/W fields define the R/W/X (read/write/execute) access control flags with each of the 4 supported operating modes: SecurePriv, SecureUser, NonsecurePriv, Nonsecure User. The lock bit provides the ability to lock these control access flags and to lock the MBC_BLK_CFG[MBACSEL] field when a locked MBC_MEMN_GLBAC register is selected.

This register is not writable.

NOTE

The lock field (LK) in MBC0_MEMN_GLBAC1, MBC0_MEMN_GLBAC2, MBC0_MEMN_GLBAC3, MBC0_MEMN_GLBAC6 and MBC0_MEMN_GLBAC7 registers are set to lock state (LK= 1) at reset. Hence these registers cannot be changed and blocks that are assigned to one of these global access control indexes cannot change the access policy index until next reset.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	LK	0														
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	SPR	SPW	SPX	0	SUR	SUW	SUX	0	NPR	NPW	NPX	0	NUR	NUW	NUX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31 LK	LOCK This 1-bit field provides a mechanism to limit writes to the this register to protect its contents. Once set, this bit remains asserted until the next reset. 0b - This register is not locked and can be altered. 1b - This register is locked and cannot be altered.
30-15 —	Reserved
14 SPR	SecurePriv Read SecurePriv read access control flag 0b - Read access is not allowed in Secure Privilege mode. 1b - Read access is allowed in Secure Privilege mode.
13 SPW	SecurePriv Write SecurePriv write access control flag 0b - Write access is not allowed in Secure Privilege mode. 1b - Write access is allowed in Secure Privilege mode.
12 SPX	SecurePriv Execute SecurePriv execute access control flag 0b - Execute access is not allowed in Secure Privilege mode. 1b - Execute access is allowed in Secure Privilege mode.
11 —	Reserved
10 SUR	SecureUser Read SecureUser read access control flag 0b - Read access is not allowed in Secure User mode. 1b - Read access is allowed in Secure User mode.
9 SUW	SecureUser Write SecureUser write access control flag 0b - Write access is not allowed in Secure User mode. 1b - Write access is allowed in Secure User mode.
8 SUX	SecureUser Execute SecureUser execute access control flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Execute access is not allowed in Secure User mode. 1b - Execute access is allowed in Secure User mode.
7 —	Reserved
6 NPR	NonsecurePriv Read NonsecurePriv read access control flag 0b - Read access is not allowed in Nonsecure Privilege mode. 1b - Read access is allowed in Nonsecure Privilege mode.
5 NPW	NonsecurePriv Write NonsecurePriv write access control flag 0b - Write access is not allowed in Nonsecure Privilege mode. 1b - Write access is allowed in Nonsecure Privilege mode.
4 NPX	NonsecurePriv Execute NonsecurePriv execute access control flag 0b - Execute access is not allowed in Nonsecure Privilege mode. 1b - Execute access is allowed in Nonsecure Privilege mode.
3 —	Reserved
2 NUR	NonsecureUser Read NonsecureUser read access control flag 0b - Read access is not allowed in Nonsecure User mode. 1b - Read access is allowed in Nonsecure User mode.
1 NUW	NonsecureUser Write NonsecureUser write access control flag 0b - Write access is not allowed in Nonsecure User mode. 1b - Write access is allowed in Nonsecure User mode.
0 NUX	NonsecureUser Execute NonsecureUser execute access control flag 0b - Execute access is not allowed in Nonsecure User mode. 1b - Execute access is allowed in Nonsecure User mode.

21.5.1.11 MBC Memory Block Configuration Word (MBC0_DOM0_MEM0_BLK_CFG_W0 - MBC0_DOM0_MEM2_BLK_CFG_W0)

Offset

Register	Offset
MBC0_DOM0_MEM0_BLK_CFG_W0	40h
MBC0_DOM0_MEM0_BLK_CFG_W1	44h
MBC0_DOM0_MEM0_BLK_CFG_W2	48h
MBC0_DOM0_MEM0_BLK_CFG_W3	4Ch
MBC0_DOM0_MEM0_BLK_CFG_W4	50h
MBC0_DOM0_MEM0_BLK_CFG_W5	54h
MBC0_DOM0_MEM0_BLK_CFG_W6	58h
MBC0_DOM0_MEM0_BLK_CFG_W7	5Ch
MBC0_DOM0_MEM1_BLK_CFG_W0	180h
MBC0_DOM0_MEM2_BLK_CFG_W0	1A8h

Function

MBC[m]_DOM[d]_MEM[s]_BLK_CFG_W[w], where,

- m - mbc index
- d - domain index
- s - memory slave index
- w - word index

These registers are read/write for both the alternate view of the NSE bit and the 3-bit MBACSEL field. This is an array of 4 bit fields defining the block configuration for the given submemory. Each 4-bit field includes an alternative view of the associated NSE bit plus a 3-bit access control select that selects the global access control value that applies to the referenced memory block.

For a given memory block, B, if the value programmed in the MBACSELn field selects a locked MBC_MEMN_GLBACr register, the MBACSEL field is locked until the next reset. Depending on BLK_CFG_W, the NSEn/MBASELn fields correspond to different blocks. The following table describes the relationship between W (word index) and B (block number).

MEM0 supports up to 512 blocks ; W0 - W63

MEM1-3 supports up to 64 block ; W0 - W7

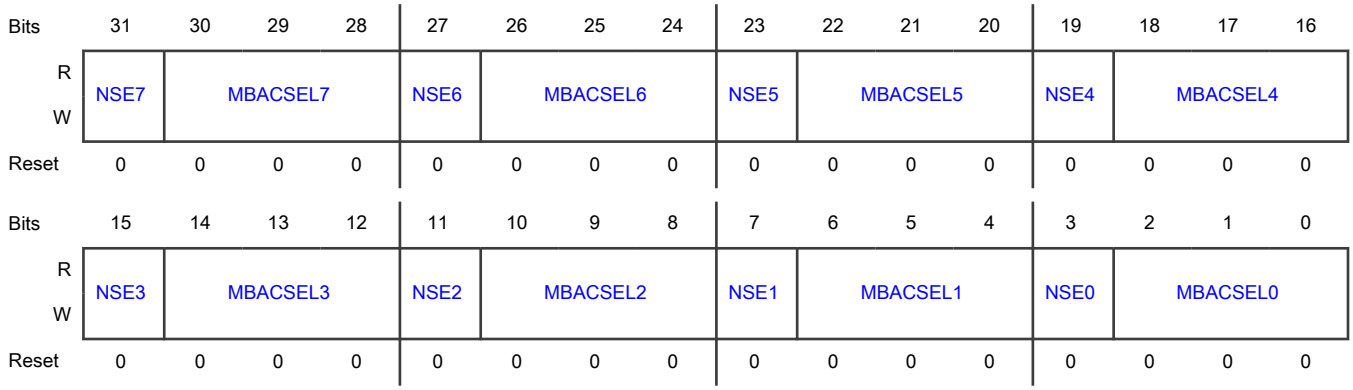
NOTE

This register is shown with fields such that it covers entire 32 bit register. However, actual fields may be less if number of blocks are such that fields don't align to 32 bits. These absent fields may be shown but user can refer to table below to deduce actual fields and treat absent fields as reserved.

Table 408. Block Config Word to Block Number Relationship

Block Config Word	NSE7/ MBACSEL7	NSE6/ MBACSEL6	NSE5/ MBACSEL5	NSE4/ MBACSEL4	NSE3/ MBACSEL3	NSE2/ MBACSEL2	NSE1/ MBACSEL1	NSE0/ MBACSEL0
W0	block 7	block 6	block 5	block 4	block 3	block 2	block 1	block 0
W1	block 15	block 14	block 13	block 12	block 11	block 10	block 9	block 8
W2	block 23	block 22	block 21	block 20	block 19	block 18	block 17	block 16
W3	block 31	block 30	block 29	block 28	block 27	block 26	block 25	block 24
W4	block 39	block 38	block 37	block 36	block 35	block 34	block 33	block 32
W5	block 47	block 46	block 45	block 44	block 43	block 42	block 41	block 40
W6	block 55	block 54	block 53	block 52	block 51	block 50	block 49	block 48
W7	block 63	block 62	block 61	block 60	block 59	block 58	block 57	block 56
For MEM0, block 64 - block 511								
W8	block 71	block 70	block 69	block 68	block 67	block 66	block 65	block 64
. . .								
W63	block 511	block 510	block 509	block 508	block 507	block 506	block 505	block 504

Diagram



Fields

Field	Function
31 NSE7	NonSecure Enable for block B 0b - Secure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]), nonsecure accesses to block B are not allowed. 1b - Secure accesses to block B are not allowed, nonsecure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]).
30-28 MBACSEL7	Memory Block Access Control Select for block B This field selects the global access control associated with block B. **NOTE** This field is locked (read-only) if MBACSEL7 = r and MBC_MEMN_GLBACr[LK] = 1. 000b - select MBC_MEMN_GLBAC0 access control policy for block B 001b - select MBC_MEMN_GLBAC1 access control policy for block B 010b - select MBC_MEMN_GLBAC2 access control policy for block B 011b - select MBC_MEMN_GLBAC3 access control policy for block B 100b - select MBC_MEMN_GLBAC4 access control policy for block B 101b - select MBC_MEMN_GLBAC5 access control policy for block B 110b - select MBC_MEMN_GLBAC6 access control policy for block B 111b - select MBC_MEMN_GLBAC7 access control policy for block B
27 NSE6	NonSecure Enable for block B 0b - Secure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]), nonsecure accesses to block B are not allowed. 1b - Secure accesses to block B are not allowed, nonsecure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]).
26-24 MBACSEL6	Memory Block Access Control Select for block B This field selects the global access control associated with block B. **NOTE** This field is locked (read-only) if MBACSEL6 = r and MBC_MEMN_GLBACr[LK] = 1. 000b - select MBC_MEMN_GLBAC0 access control policy for block B 001b - select MBC_MEMN_GLBAC1 access control policy for block B 010b - select MBC_MEMN_GLBAC2 access control policy for block B 011b - select MBC_MEMN_GLBAC3 access control policy for block B

Table continues on the next page...

Table continued from the previous page...

Field	Function
	100b - select MBC_MEMN_GLBAC4 access control policy for block B 101b - select MBC_MEMN_GLBAC5 access control policy for block B 110b - select MBC_MEMN_GLBAC6 access control policy for block B 111b - select MBC_MEMN_GLBAC7 access control policy for block B
23 NSE5	NonSecure Enable for block B 0b - Secure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]), nonsecure accesses to block B are not allowed. 1b - Secure accesses to block B are not allowed, nonsecure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]).
22-20 MBACSEL5	Memory Block Access Control Select for block B This field selects the global access control associated with block B. **NOTE** This field is locked (read-only) if MBACSEL5 = r and MBC_MEMN_GLBACr[LK] = 1. 000b - select MBC_MEMN_GLBAC0 access control policy for block B 001b - select MBC_MEMN_GLBAC1 access control policy for block B 010b - select MBC_MEMN_GLBAC2 access control policy for block B 011b - select MBC_MEMN_GLBAC3 access control policy for block B 100b - select MBC_MEMN_GLBAC4 access control policy for block B 101b - select MBC_MEMN_GLBAC5 access control policy for block B 110b - select MBC_MEMN_GLBAC6 access control policy for block B 111b - select MBC_MEMN_GLBAC7 access control policy for block B
19 NSE4	NonSecure Enable for block B 0b - Secure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]), nonsecure accesses to block B are not allowed. 1b - Secure accesses to block B are not allowed, nonsecure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]).
18-16 MBACSEL4	Memory Block Access Control Select for block B This field selects the global access control associated with block B. **NOTE** This field is locked (read-only) if MBACSEL4 = r and MBC_MEMN_GLBACr[LK] = 1.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	000b - select MBC_MEMN_GLBAC0 access control policy for block B 001b - select MBC_MEMN_GLBAC1 access control policy for block B 010b - select MBC_MEMN_GLBAC2 access control policy for block B 011b - select MBC_MEMN_GLBAC3 access control policy for block B 100b - select MBC_MEMN_GLBAC4 access control policy for block B 101b - select MBC_MEMN_GLBAC5 access control policy for block B 110b - select MBC_MEMN_GLBAC6 access control policy for block B 111b - select MBC_MEMN_GLBAC7 access control policy for block B
15 NSE3	NonSecure Enable for block B 0b - Secure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]), nonsecure accesses to block B are not allowed. 1b - Secure accesses to block B are not allowed, nonsecure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]).
14-12 MBACSEL3	Memory Block Access Control Select for block B This field selects the global access control associated with block B. **NOTE** This field is locked (read-only) if MBACSEL3 = r and MBC_MEMN_GLBACr[LK] = 1. 000b - select MBC_MEMN_GLBAC0 access control policy for block B 001b - select MBC_MEMN_GLBAC1 access control policy for block B 010b - select MBC_MEMN_GLBAC2 access control policy for block B 011b - select MBC_MEMN_GLBAC3 access control policy for block B 100b - select MBC_MEMN_GLBAC4 access control policy for block B 101b - select MBC_MEMN_GLBAC5 access control policy for block B 110b - select MBC_MEMN_GLBAC6 access control policy for block B 111b - select MBC_MEMN_GLBAC7 access control policy for block B
11 NSE2	NonSecure Enable for block B 0b - Secure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]), nonsecure accesses to block B are not allowed. 1b - Secure accesses to block B are not allowed, nonsecure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]).
10-8	Memory Block Access Control Select for block B

Table continues on the next page...

Table continued from the previous page...

Field	Function
MBACSEL2	This field selects the global access control associated with block B. **NOTE** This field is locked (read-only) if MBACSEL2 = r and MBC_MEMN_GLBACr[LK] = 1. 000b - select MBC_MEMN_GLBAC0 access control policy for block B 001b - select MBC_MEMN_GLBAC1 access control policy for block B 010b - select MBC_MEMN_GLBAC2 access control policy for block B 011b - select MBC_MEMN_GLBAC3 access control policy for block B 100b - select MBC_MEMN_GLBAC4 access control policy for block B 101b - select MBC_MEMN_GLBAC5 access control policy for block B 110b - select MBC_MEMN_GLBAC6 access control policy for block B 111b - select MBC_MEMN_GLBAC7 access control policy for block B
7 NSE1	NonSecure Enable for block B 0b - Secure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]), nonsecure accesses to block B are not allowed. 1b - Secure accesses to block B are not allowed, nonsecure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]).
6-4 MBACSEL1	Memory Block Access Control Select for block B This field selects the global access control associated with block B. **NOTE** This field is locked (read-only) if MBACSEL1 = r and MBC_MEMN_GLBACr[LK] = 1. 000b - select MBC_MEMN_GLBAC0 access control policy for block B 001b - select MBC_MEMN_GLBAC1 access control policy for block B 010b - select MBC_MEMN_GLBAC2 access control policy for block B 011b - select MBC_MEMN_GLBAC3 access control policy for block B 100b - select MBC_MEMN_GLBAC4 access control policy for block B 101b - select MBC_MEMN_GLBAC5 access control policy for block B 110b - select MBC_MEMN_GLBAC6 access control policy for block B 111b - select MBC_MEMN_GLBAC7 access control policy for block B
3 NSE0	NonSecure Enable for block B 0b - Secure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]), nonsecure accesses to block B are not allowed.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - Secure accesses to block B are not allowed, nonsecure accesses to block B are based on corresponding MBACSEL field in this register (MBCm_DOMd_MEMs_BLK_CFG_Ww[MBACSEL]).
2-0 MBACSEL0	Memory Block Access Control Select for block B This field selects the global access control associated with block B. **NOTE** This field is locked (read-only) if MBACSEL0 = r and MBC_MEMN_GLBACr[LK] = 1. 000b - select MBC_MEMN_GLBAC0 access control policy for block B 001b - select MBC_MEMN_GLBAC1 access control policy for block B 010b - select MBC_MEMN_GLBAC2 access control policy for block B 011b - select MBC_MEMN_GLBAC3 access control policy for block B 100b - select MBC_MEMN_GLBAC4 access control policy for block B 101b - select MBC_MEMN_GLBAC5 access control policy for block B 110b - select MBC_MEMN_GLBAC6 access control policy for block B 111b - select MBC_MEMN_GLBAC7 access control policy for block B

Chapter 22
Digital Tamper (TDET)

22.1 Chip-specific TDET information

Table 409. Reference links to related information[1]

Topic	Related module	Reference
Full description	TDET	TDET
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

22.1.1 Module instances

This device has one instance of the Digital Tamper module.

The device supports up to eight passive/active tamper pins. Digital tamper pin functionality can be used in all power modes including Deep power down and VBAT mode.

22.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

22.1.3 On-chip tamper inputs

In addition to detecting tampers for the digital tamper pins, a number of on-chip tamper sources from other modules are routed to the TDET. The table below shows the SR register fields associated with the on-chip tamper sources, the on-chip tamper source, and the flag associated with the tamper source in the module where the tamper originates:

Table 410. On-chip tamper sources

SR register fields	Description	Module	Tamper flag (tamper originating module)
TIF0	Clock monitor	VBAT	VBAT_STATUSA[CLOCK_DET]
IF1	Configuration error	VBAT	VBAT_STATUSA[CONFIG_DET]
TIF2	Voltage monitor	VBAT	VBAT_STATUSA[VOLT_DET]
TIF3	Temperature monitor	VBAT	VBAT_STATUSA[TEMP_DET]
TIF4 and TIF5	Reserved	-	-
TIF6	RAM zeroize tamper (ITRC_OUT3)	ITRC	ITRC_STATUS[OUT3_STATUS]
TIF7 to TIF9	Reserved	-	-

22.1.4 Device Wakeup from VBAT mode

A TDET interrupt asserts the VBAT.STATUSA[IRQ3_DET] field. This can be optionally configured as a source to assert the VBAT_WAKEUP_b pin. When the device is operating in the VBAT power mode, you can transition the device back to Active mode by configuring an external circuitry to recognize the VBAT_WAKEUP_b assertion and request for a system power on. See the VBAT chapter for more details.

22.1.5 TSR to RTC mapping

The following table shows how the TSR[TTS] bits map to the respective RTC registers.

Table 411. TSR to RTC mapping

TSR[TTS] bits	RTC registers
5:0	SECONDS[SEC_CNT]
11:6	HOURMIN[MIN_CNT]
16:12	HOURMIN[HOUR_CNT]
21:17	DAYS[DAY_CNT]
25:22	YEARMON[MON_CNT]
31:26	YEARMON[YROFST]

22.2 Overview

TDET detects potential on-chip and external device tampering, triggering an interrupt or chip reset. In addition to passive tamper detection where a static value is expected, TDET also provides active tamper detection on a board by streaming a known bit sequence and checking that their return values match what was sent.

22.2.1 Block diagram

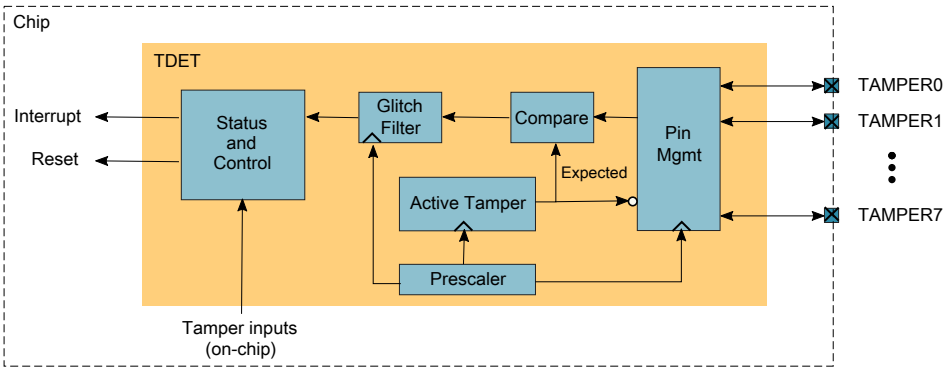


Figure 84. TDET block diagram

22.2.2 Features

TDET features include:

- 8 external tamper pins capable of generating interrupt or tamper event:
  - Configurable polarity and digital glitch filter with optional prescaler
  - Configurable for either passive or active tamper input

  - Supports software-initiated tamper pin assertion
  - Supports periodic sampling of the tamper pin
- 10 internal tamper sources plus software-initiated tamper capable of generating interrupt or tamper event
- Tamper seconds register records time of tamper event
- 2 active tamper shift registers each with configurable polynomial
- Register protection
  - Lock register requires VBAT POR or software reset to enable write access

22.3 Functional description

22.3.1 External tamper pins

Each tamper pin can be configured as a passive or active tamper input or as an active tamper output. Each pin has configurable polarity.

22.3.1.1 Pull-resistors for input pins

When configured as a tamper input (passive or active), a pull-resistor can be enabled (using PGFRn[TPE]) on the tamper pin to cause the tamper pin to assert or negate (based on PGFRn[TPS]) if the pin is floating. Disable the pull-resistor if the external tamper circuit is unable to overdrive the pull-resistor. In this case, the external circuit must ensure a floating tamper pin generates a tamper event.

22.3.1.2 Glitch filters

Each tamper pin input has an independently configured glitch filter that can filter out voltage transient widths as small as 30us or as large as 248ms. For an active tamper pin input or for a tamper pin input with dynamic polarity, enable its corresponding glitch filter to filter out the difference in propagation delay between the tamper pin expected value and the tamper pin input data traversing a board.

Clocking describes the clock source options for the glitch filters.

22.3.1.2.1 Glitch filter operation

The following pseudo-code describes glitch filter operation:

```
If (tamper_pin != expected)
  {
    If (counter == { filter_width, 1'b1 } )
    {
    Filtered_tamper = tamper_pin;
    Counter = 0;
    } else {
    Counter++;
    }
  }
Else if (counter > 0)
{
Counter --;
}
```

The following figures provide examples of glitch filter operation:

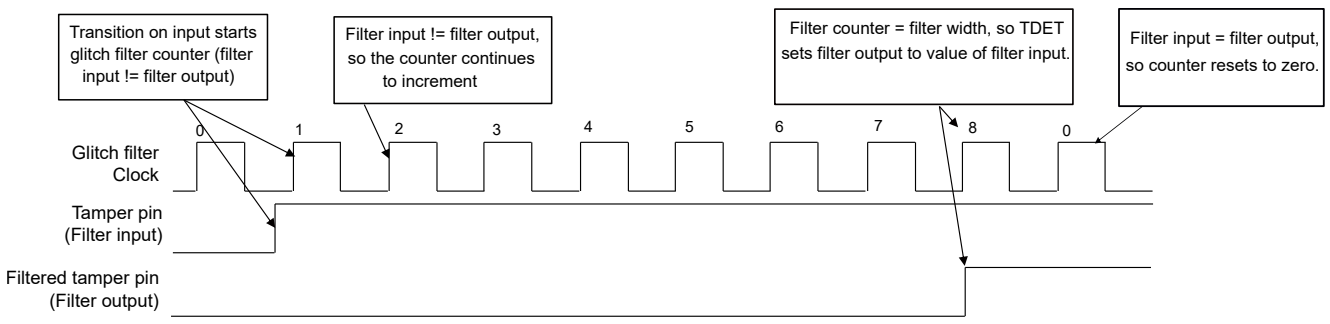


Figure 85. Tamper pin glitch filtering example: no glitch

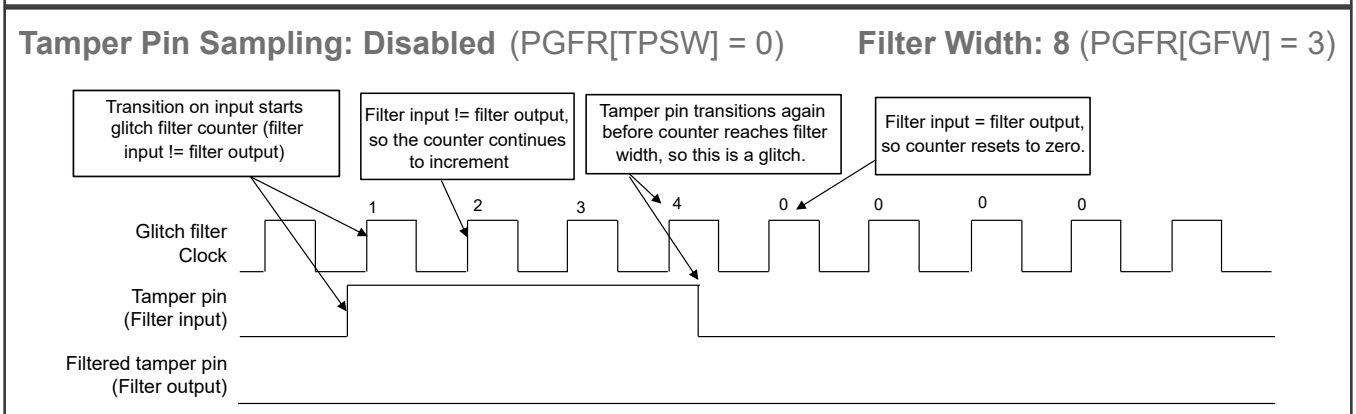


Figure 86. Tamper pin glitch filtering example: glitch

22.3.1.3 Tamper pin sampling

By default, a passive or active tamper input will continuously monitor the input to detect assertion of the tamper input. Tamper pin sampling can be enabled by configuring the tamper pin sample frequency and the tamper pin sample width for individual tamper pins. When pin sampling is enabled, the tamper pin internal pull resistor and input buffer will only be enabled for some of the time. This can be useful in reducing power consumption of the tamper pin, especially if the pull resistor is configured to assert the tamper pin while external logic is driving the tamper input to the opposite state.

When tamper pin sampling is enabled, the pull resistor is enabled for twice as long as the input buffer. This provides time for the pull resistor to affect the state of the pin before the input buffer is enabled. While the input buffer is disabled, the tamper pin state is internally driven as if the tamper pin is not asserted. If the glitch filter is enabled, the input buffer and pull resistor will remain enabled after the sample window if the glitch filter is still actively resolving the input state (for example, if the glitch filter is configured for a longer width than the sampling width, if the glitch filter detects an asserted tamper then it will keep both the input buffer and pull resistor enabled until the assertion propagates through the glitch filter or the input negates).

The tamper pin sampling can be configured to sample the tamper pin inputs every 8, 32, 128 or 512 cycles. The clock used in determining the frequency is the same clock used by the tamper pin glitch filter (software configurable to either 32.768 kHz or 512 Hz). The width of the tamper pin sampling can be configured for 1 cycle, 2 cycles or 4 cycles (with the pull resistor enabled for twice that length). Configuring the sample width to 4 cycles with a sample frequency of 8 cycles is not recommended as the pull resistor will remain enabled all the time.

22.3.1.4 Passive tamper

Use a passive tamper input pin to monitor an external tamper circuit with a static expected value. Configure the optional pull resistor and glitch filter as needed.

22.3.1.4.1 Configuring a passive tamper

To configure a tamper pin as a passive tamper:

1. Configure the tamper pin expected value to be 0 (PGFRn[TPEX] = 00). (Zero is the only static, non-active choice.)
2. If the tamper pin expected value needs to be 1, program its polarity to be inverted (PPR[TPPn] = 1).
3. Program the tamper pin direction to be an input (PDR[TPDn] = 0).
4. As needed, select the pull-resistor (PGFRn[TPS]) and enable it (PGFRn[TPE] = 1).
5. As needed, in the corresponding PGFRn:
   - Select the pin-sampling options (TPSF, TPSW).
   - Select the filter-clocking and filter-width options (GFP, GFW).
   - Enable the glitch filter (GFE).
6. Enable tamper detection (TER[TPEn] = 1) and/or interrupts (IER[TPIEn] = 1) for the pin as needed.

22.3.1.5 Active tamper

TDET can generate two active tamper values that can be used as active tamper outputs or active tamper inputs by any of the tamper pins. Each active tamper consists of a 16-bit linear feedback shift register (ATRn[ATSR]) with 16-bit configurable polynomial (ATRn[ATP]). Each shift register is clocked by a 1-Hz or 64-Hz clock (selected in CR[ATCSn]) generated by the TDET prescaler, which must be enabled (CR[DEN] = 1) for the active tamper to function.

If the active tamper polynomial is configured to a non-zero value and the shift register is configured to a non-zero value, the active tamper is enabled. Once enabled, the active tamper shift register is shifted right by one each clock period. In addition, it is XORed with the active tamper polynomial when the active tamper output is 1. The LSB of the active tamper shift register is the output of that active tamper.

Software writes to an active tamper register take effect immediately, and can alter the active tamper output if the LSB of the active tamper shift register is altered. Avoid writing the active tamper registers near the prescaler clock edge, which occurs when the TDET control register's prescaler field is near its maximum value.

22.3.1.5.1 Polarity considerations

TDET inverts tamper outputs before streaming their values to a pin. Inverting the output means a corresponding input stream is the inverse of the tamper expected value, which would be detected as tampering. Therefore, configure any active tamper output with its polarity opposite to that of any corresponding active tamper input (PPR[TPPn] = 1).

This hardware inversion feature also allows software to assert a tamper pin briefly, for less than the glitch filter period, to verify the tamper input has not become stuck. Software can check the current state of each tamper pin input (before the glitch filter) by reading the tamper pin input data field (PPR[TPIDn]).

22.3.1.5.2 Configuring an active tamper output

To configure a tamper pin as an active tamper output:

1. Select the clock source (CR[ATCSn]) for the chosen Active Tamper register.
2. Configure the chosen Active Tamper register (ATRn).
3. Configure the tamper pin expected value (PGFRn[TPEX]) to the desired active tamper output.
4. As needed, program the tamper pin polarity (PPR[TPPn]) to make it opposite to its corresponding active tamper input.
5. Program the tamper pin direction to be an output (PDR[TPDn] = 1).

Do not enable the chosen tamper pin as an input tamper source because the output data is always the inverse of the expected value which would trigger a tamper event.

22.3.1.5.3 Configuring an active tamper input

To configure a tamper pin as an active tamper input:

1. Configure the active tamper output that the active tamper input is to be paired with; see Configuring an active tamper output.
2. As needed, program the tamper pin polarity (PPR[TPPn]) to make it opposite to its corresponding active tamper output.
3. In the corresponding PGFRn:
   - Configure the tamper pin expected value (TPEX) to match the active tamper output to be paired with.
   - Select the pull-resistor, pin-sampling and filter-clocking options as needed (TPS, TPE, TPSF, TPSW, and GFP).
   - Program the filter width as needed to account for the propagation delay through the pins and the board (GFW).
   - Enable the glitch filter (GFE).
4. Enable the tamper pin as a tamper source (TER[TPEn] = 1), an interrupt source (IER[TPIEn] = 1), or both.

22.3.2 On-chip tamper inputs

Each on-chip tamper input source can be configured to generate an interrupt, set the Digital Tamper flag (SR[DTF]), or both. Each on-chip tamper source includes a status flag (SR[TIF*n*]), tamper enable (TER[TIE*n*]), and interrupt enable (IER[TIIE*n*]). The status flag can update whether the tamper or interrupt are enabled or not. Clearing a status flag requires the on-chip tamper source to negate and software to write 1 to the flag.

An enabled tamper source causes SR[DTF] to set if its individual tamper status flag is set. To clear SR[DTF], first clear any enabled tamper input status flags and then write 1 to SR[DTF].

NOTE

The Tamper Acknowledge flag (SR[TAF]) must be set to clear SR[DTF]. This requires a chip reset to occur between SR[DTF] being set due to a tamper event and when it is cleared by software.

22.3.3 Locking registers

If needed for additional security, use the Lock register (LR) to block write accesses to individual registers until the next VBAT POR or TDET software reset (CR[SWR]). Attempted write accesses to a locked register are ignored and do not generate a bus error.

Locking the Control register (CR) disables the software reset. Locking LR itself disables future updates to the Lock register.

22.3.4 Operation in chip power modes

TDET is always powered by the chip's backup power supply (VBAT) and remains operational regardless of the chip's power mode (power-up or power-down).

22.3.5 Clocking

The Digital Tamper clock and prescaler must be enabled (CR[DEN] = 1) for the pin glitch filters to operate. Each glitch filter can be clocked by a 32.768-kHz clock or 512-Hz prescaler clock (PGFRn[GFP]). The Active Tamper registers (ATRn) are clocked by the 1-Hz or 64-Hz prescaler clock (CR[ATCSn]). The Tamper Seconds register (TSR) is clocked by the 1-Hz prescaler clock.

22.3.6 Reset

The VBAT supply includes its own analog POR block that generates a power-on-reset signal whenever the VBAT domain is powered-up and initializes all registers to their default state for modules in the VBAT domain.

22.3.6.1 Software reset

Writing 1 to CR[SWR] forces the equivalent of a VBAT POR to the rest of TDET. This software-controlled reset does not affect the CR[SWR] bit. Write 0 to CR[SWR] to release TDET from software reset. The writes can occur back-to-back.

22.3.7 Interrupts

The Digital Tamper interrupt is asserted whenever a status flag and its corresponding interrupt enable bit are both set. It is always asserted on VBAT POR, TDET software reset and when the VBAT power supply is powered down.

22.4 External signals

Table 412. TDET signals

Signal	Description	I/O
TAMPER[7:0]	External tamper input or active tamper output	I/O

22.5 Initialization

As long as the system provides the required clocking in chip power-up and power-down modes, TDET is operational and ready to be configured for specific applications.

After a VBAT POR resets TDET, SR[DTF] is set. Clear SR[DTF] before configuring other TDET registers.

NOTE

Before attempting to clear DTF, confirm that a system reset has occurred (indicated by SR[TAF] = 1) to ensure a secure environment. If DTF = 1 and TAF = 0, first initiate a chip reset before clearing DTF. A chip reset must occur before DTF can be cleared.

22.6 Register definitions

All registers must be accessed using 32-bit writes. Writing to a register protected by the Lock register does not generate a bus error, but the write is ignored.

An attempt to access a TDET register results in a bus error under any of the following power-related conditions:

- VBAT domain is powered down.
- VBAT domain is electrically isolated.
- VBAT POR is asserted.

22.6.1 TDET register descriptions

22.6.1.1 TDET memory map

TDET0 base address: 4005_8000h

Offset	Register	Width (In bits)	Access	Reset value
10h	Control (CR)	32	RW	0000_0000h
14h	Status (SR)	32	RW	0000_0FFFh
18h	Lock (LR)	32	RW	00FF_3FFFh
1Ch	Interrupt Enable (IER)	32	RW	0000_0000h
20h	Tamper Seconds (TSR)	32	RW	0000_0000h
24h	Tamper Enable (TER)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
28h	Pin Direction (PDR)	32	RW	00FF_0000h
2Ch	Pin Polarity (PPR)	32	RW	0000_0000h
30h - 34h	Active Tamper (ATR0 - ATR1)	32	RW	0000_0000h
40h - 5Ch	Pin Glitch Filter (PGFR0 - PGFR7)	32	RW	0000_0000h

22.6.1.2 Control (CR)

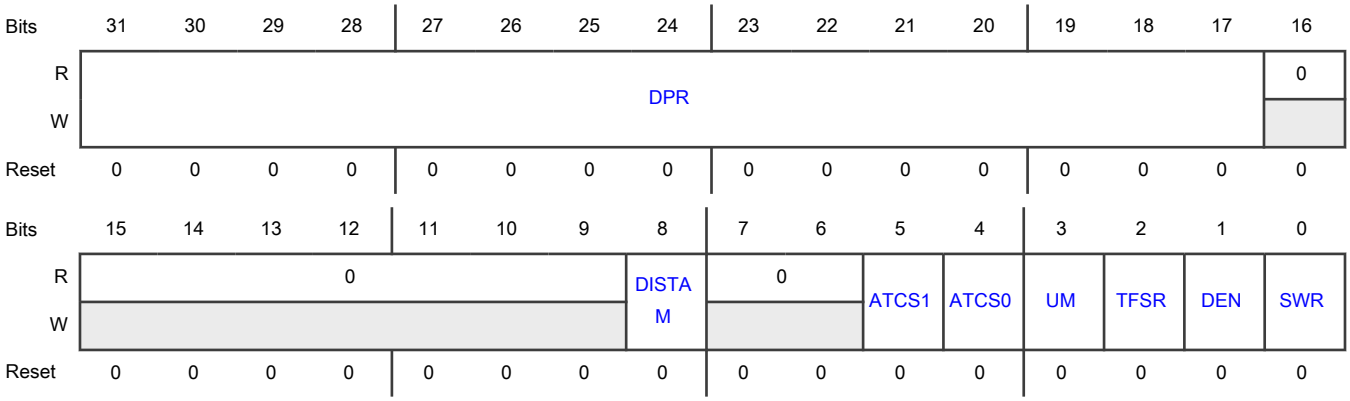
Offset

Register	Offset
CR	10h

Function

Contains the TDET prescaler and configuration options, as well as the software reset control bit.

Diagram



Fields

Field	Function
31-17 DPR	Digital Tamper Prescaler Contains the prescaler. When writing, configures the initial value of the prescaler. When reading, returns the current value of the prescaler. The glitch filters can be clocked by the 32.768 kHz clock or the 512 Hz prescaler clock output (when bit 22 transitions). The active tamper shift registers can be clocked by the 64 Hz prescaler clock output (when bit 25 transitions) or the 1 Hz prescaler clock output (when bit 31 transitions from a one to a zero).

Table continues on the next page...

Table continued from the previous page...

Field	Function
	NOTE DPR can be written only when the clocks are disabled (CR[DEN] = 0).
16-9 —	Reserved
8 DISTAM	Disable Prescaler On Tamper Allows the 32-KHz clock and prescaler to be automatically disabled after tamper detection (SR[DTF]=1) and until the system acknowledges the tamper (SR[TAF]=1). Disabling the prescaler after detecting a tamper event conserves power and freezes the state of the active tamper outputs and glitch filters. To ensure a clean transition, the prescaler is disabled at the end of a 1 Hz period. 0b - No effect 1b - Automatically disables the prescaler after tamper detection
7-6 —	Reserved
5-4 ATCSn	Active Tamper Clock Source Selects the clock source for the corresponding Active Tamper Shift Register. Do not change the active tamper clock source when the corresponding active tamper register is enabled (ATRn[ATP] ≠ 0). 0b - 1 Hz prescaler clock 1b - 64 Hz prescaler clock
3 UM	Update Mode When no tampering has been detected (SR[DTF]=0), enables software to clear tamper pin flags and on-chip tamper input flags in the Status register (SR[TPFn] and SR[TIFn]) even when SR is locked in the Lock register (LR[SRL]=0). UM applies to TPFn and TIFn that are enabled for interrupts only (IER[TPIEn] and IER[TIIEn] that equal 1 and their corresponding TER enable bits equal 0). 0b - No effect 1b - Allows the clearing of interrupts
2 TFSR	Tamper Force System Reset Enables generating a chip reset when tampering is detected (SR[DTF] = 1 and SR[TAF] = 0). 0b - Do not force chip reset 1b - Force chip reset
1 DEN	Digital Tamper Enable Enables the 32.768-kHz clock and the prescaler that generates the 512-Hz, 64-Hz and 1-Hz prescaler clocks. Must be enabled before enabling a glitch filter or active tamper. Must be disabled only after disabling all glitch filters and active tampers.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Disables TDET clock and prescaler 1b - Enables TDET clock and prescaler
0 SWR	Software Reset Resets all registers. Exceptions: • CR[SWR] itself is not affected; it is reset by VBAT POR only. 0b - No effect 1b - Perform a software reset

22.6.1.3 Status (SR)

Offset

Register	Offset
SR	14h

Function

Provides the status of all tamper flags, including the global Digital Tamper flag.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0								TPF7	TPF6	TPF5	TPF4	TPF3	TPF2	TPF1	TPF0
W									W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0				TIF9	TIF8	TIF7	TIF6	TIF5	TIF4	TIF3	TIF2	TIF1	TIF0	TAF	DTF
W					W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-24 —	Reserved
23-16	Tamper Pin n Flag

Table continues on the next page...

Table continued from the previous page...

Field	Function
TPFn	Indicates that tamper pin *n* does not equal its expected value and was not filtered by the glitch filter (if enabled). To clear, write 1 to the flag after its tamper pin equals its expected value and the glitch filter output has updated. 0b - Pin tamper not detected 1b - Pin tamper detected
15-12 —	Reserved
11-2 TIFn	Tamper Input n Flag Indicates that on-chip tamper input *n* has asserted. To clear, write 1 to a flag after the on-chip source negates. 0b - On-chip tamper not detected 1b - On-chip tamper detected
1 TAF	Tamper Acknowledge Flag Indicates a chip reset has occurred after the Digital Tamper Flag (SR[DTF]) was set. The chip acknowledges the tamper by resetting. TAF is cleared on VBAT POR, software reset or by software writing 1 to this flag whenever SR[DTF] is clear. 0b - Digital Tamper Flag (SR[DTF]) is clear or chip reset has not occurred after Digital Tamper Flag (SR[DTF]) was set. 1b - Chip reset has occurred after Digital Tamper Flag (SR[DTF]) was set.
0 DTF	Digital Tamper Flag Indicates tampering has been detected. Sets on VBAT POR, software reset, a write to the tamper seconds register or whenever an enabled tamper flag is set. Can be cleared by software by writing 1 to the flag provided the Tamper Acknowledge flag is set. 0b - TDET tampering not detected 1b - TDET tampering detected

22.6.1.4 Lock (LR)

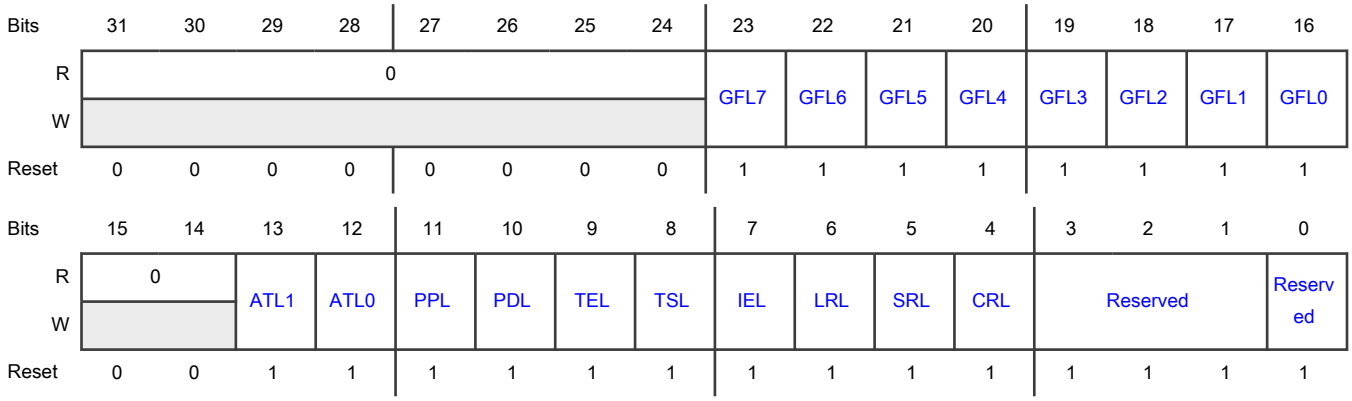
Offset

Register	Offset
LR	18h

Function

Clearing individual fields locks the corresponding register. Once cleared, fields can be set again (registers unlocked) only by VBAT POR or software reset.

Diagram



Fields

Field	Function
31-24 —	Reserved
23-16 GFLn	Glitch Filter Lock Locks the corresponding Pin Glitch Filter *n* register (PGFRn). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
15-14 —	Reserved
13-12 ATLn	Active Tamper Lock Locks the corresponding Active Tamper *n* register (ATRn). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
11 PPL	Pin Polarity Lock Locks the Pin Polarity register (PPR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
10 PDL	Pin Direction Lock Locks the Pin Direction register (PDR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
9	Tamper Enable Lock

Table continues on the next page...

Table continued from the previous page...

Field	Function
TEL	Locks the Tamper Enable register (TER). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
8 TSL	Tamper Seconds Lock Locks the Tamper Seconds register (TSR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
7 IEL	Interrupt Enable Lock Locks the Interrupt Enable register (IER). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
6 LRL	Lock Register Lock Locks the Lock register (LR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
5 SRL	Status Register Lock Locks the Status register (SR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
4 CRL	Control Register Lock Locks the Control register (CR). 0b - Locked and writes are ignored 1b - Not locked and writes complete as normal
3-1 —	Reserved
0 —	Reserved

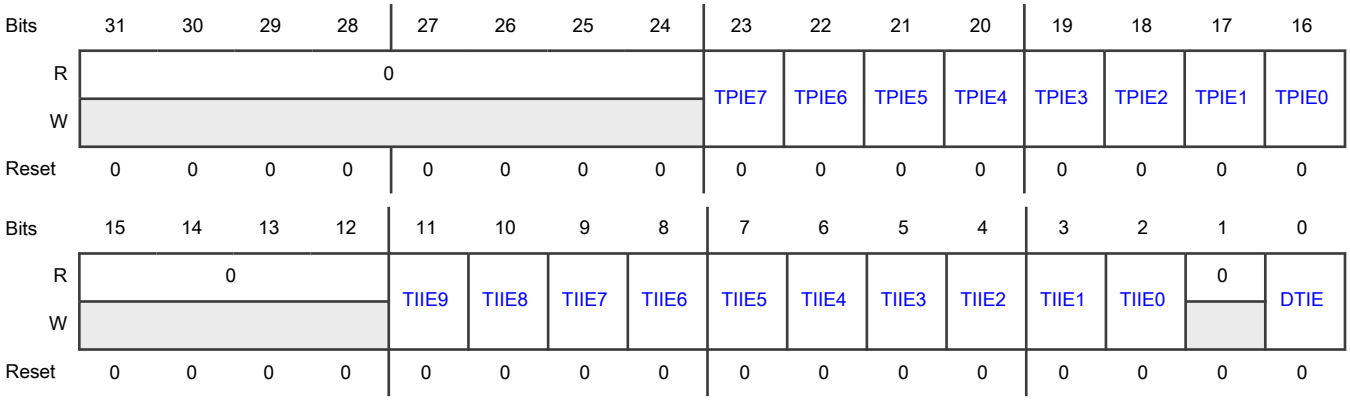
22.6.1.5 Interrupt Enable (IER)

Offset

Register	Offset
IER	1Ch

Function
Enables interrupts for the Digital Tamper flag and the individual tamper pins or on-chip tamper input flags.

Diagram



Fields

Field	Function
31-24 —	Reserved
23-16 TPIEn	Tamper Pin n Interrupt Enable Enables interrupts when the corresponding tamper pin *n* flag (SR[TPFn]) is set. 0b - Disables 1b - Enables
15-12 —	Reserved
11-2 TIIEn	Tamper Input n Interrupt Enable Enables interrupts when the corresponding on-chip tamper input *n* flag (SR[TIFn]) is set. 0b - Disables 1b - Enables
1	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
0 DTIE	Digital Tamper Interrupt Enable Enables interrupts when the Digital Tamper flag (SR[DTF]) is set. 0b - Disables 1b - Enables

22.6.1.6 Tamper Seconds (TSR)

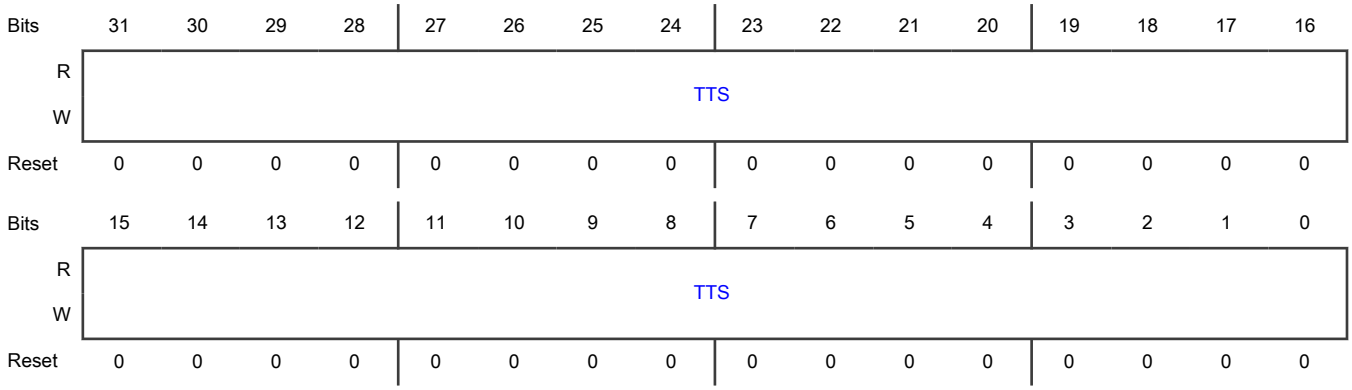
Offset

Register	Offset
TSR	20h

Function

Contains the time in seconds corresponding to when the Digital Tamper flag (SR[DTF]) was set.

Diagram



Fields

Field	Function
31-0 TTS	Tamper Time Seconds Reading this register returns the time in seconds at which SR[DTF] was set. Returns zero when SR[DTF] is clear. Writing to TSR[TTS] is considered tampering and sets SR[DTF].

22.6.1.7 Tamper Enable (TER)

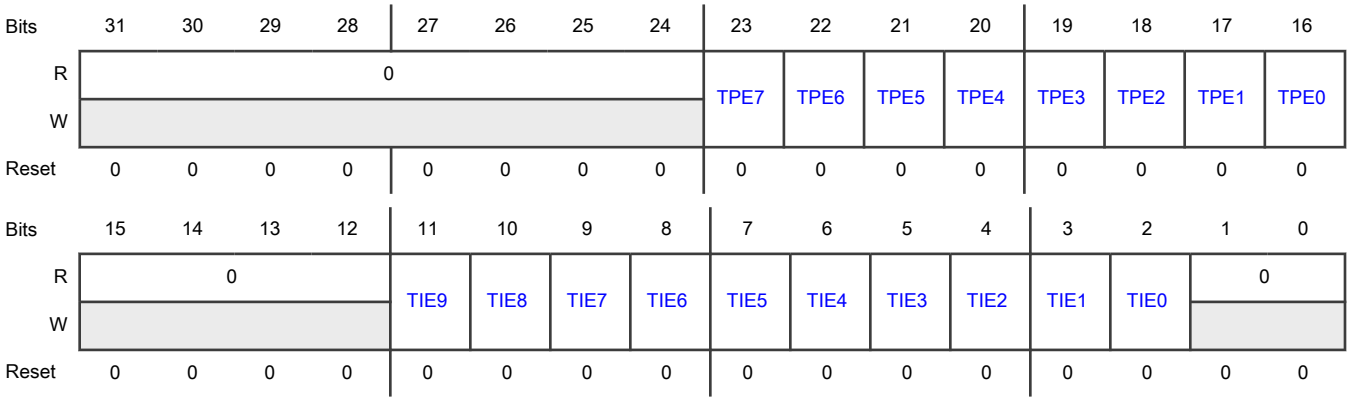
Offset

Register	Offset
TER	24h

Function

Enables tamper detection for tamper pins and on-chip tamper inputs.

Diagram



Fields

Field	Function
31-24 —	Reserved
23-16 TPEn	Tamper Pin Enable Enables tamper detection when the corresponding tamper pin *n* flag (SR[TPFn]) is set. 0b - Disables 1b - Enables
15-12 —	Reserved
11-2 TIEn	Tamper Input Enable Enables tamper detection when the corresponding on-chip tamper input *n* flag (SR[TIFn]) is set. 0b - Disables 1b - Enables
1-0 —	Reserved

22.6.1.8 Pin Direction (PDR)

Offset

Register	Offset
PDR	28h

Function

Configures the pin direction of tamper pins and provides read access to the current values of tamper pin output data.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0								TPOD7	TPOD6	TPOD5	TPOD4	TPOD3	TPOD2	TPOD1	TPOD0
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0								TPD7	TPD6	TPD5	TPD4	TPD3	TPD2	TPD1	TPD0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-24 —	Reserved
23-16 TPODn	Tamper Pin Output Data Contains the current value of tamper pin output data. 0b - Zero 1b - One
15-8 —	Reserved
7-0 TPDn	Tamper Pin Direction Configures the direction of the tamper pin. 0b - Input 1b - Output and drives the inverse of the expected value (tamper pin is asserted)

22.6.1.9 Pin Polarity (PPR)

Offset

Register	Offset
PPR	2Ch

Function

Configures the pin polarity of tamper pins and provides read access to the current values of tamper pin input data.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0								TPID7	TPID6	TPID5	TPID4	TPID3	TPID2	TPID1	TPID0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0								TPP7	TPP6	TPP5	TPP4	TPP3	TPP2	TPP1	TPP0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-24 —	Reserved
23-16 TPIDn	Tamper Pin n Input Data Contains the current value of the tamper pin *n* input data before the glitch filter. 0b - Zero 1b - One
15-8 —	Reserved
7-0 TPPn	Tamper Pin n Polarity Configures the polarity of the tamper pin *n* expected value. 0b - Not inverted 1b - Inverted

22.6.1.10 Active Tamper (ATR0 - ATR1)

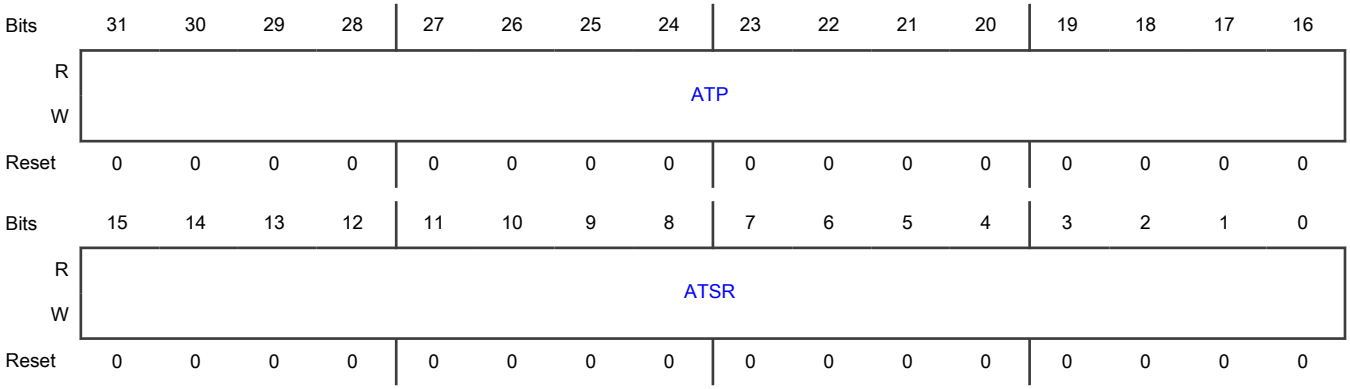
Offset

Register	Offset
ATR0	30h
ATR1	34h

Function

Configures the active tamper polynomial and contains the shift register itself.

Diagram



Fields

Field	Function
31-16 ATP	Active Tamper Polynomial Configures the polynomial of ATSR. When set to zero, ATSR is disabled. When programmed to a non-zero value, ATSR is enabled.
15-0 ATSR	Active Tamper Shift Register Shifts right by one every selected prescaler clock period once enabled (see ATP). Bit 0 of ATSR is the active tamper expected value. Initialize ATSR to a non-zero value but not at the same time as the selected prescaler clock pulse (see CR[DPR]). When the active tamper output is 1, ATSR is also XORed with ATP.

22.6.1.11 Pin Glitch Filter (PGFR0 - PGFR7)

Offset

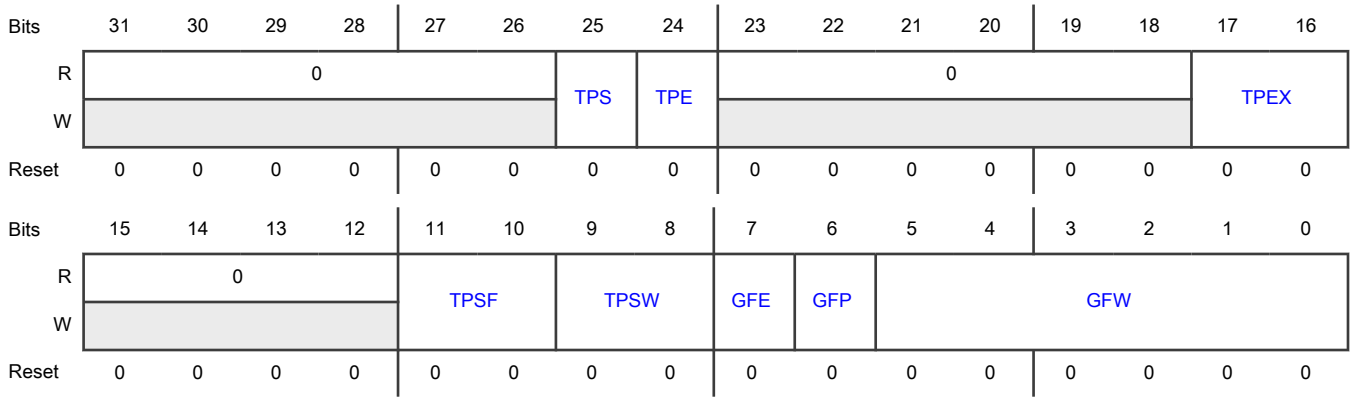
For n = 0 to 7:

Register	Offset
PGFRn	40h + (n × 4h)

Function

Configures the glitch filter, pin pull and sampling operation, as well as the expected value, for the corresponding tamper pin.

Diagram



Fields

Field	Function
31-26 —	Reserved
25 TPS	Tamper Pull Select Configures the direction of the tamper pin internal pull resistor such that the tamper pull resistor always asserts or negates the tamper pin. The tamper pull direction is therefore also dependent on the Tamper Pin Expected Value (PGFRn[TPEX]) and Tamper Pin Polarity (PPR[TPPn]) configuration. 0b - Asserts 1b - Negates
24 TPE	Tamper Pull Enable Enables the pull resistor on the tamper pin. 0b - Disables 1b - Enables
23-18 —	Reserved
17-16 TPEX	Tamper Pin Expected Selects the expected value for the corresponding tamper pin. 00b - Zero/passive tamper 01b - Active Tamper 0 output 10b - Active Tamper 1 output 11b - Active Tamper 0 output XORed with Active Tamper 1 output

Table continues on the next page...

Table continued from the previous page...

Field	Function
15-12 —	Reserved
11-10 TPSF	Tamper Pin Sample Frequency Configures the tamper pin sampling frequency (measured in number of glitch filter clock cycles). When tamper pin sampling is enabled, the tamper pin pull enable and input buffer enable periodically assert at a frequency determined by TPSF. The tamper pin pull enable and input buffer enable are always asserted whenever the glitch filter is actively filtering the input. The number of cycles refers to the number of cycles of the selected glitch filter clock (either 32.768 kHz or 512 Hz). 00b - Every 8 cycles 01b - Every 32 cycles 10b - Every 128 cycles 11b - Every 512 cycles
9-8 TPSW	Tamper Pin Sample Width Configures the tamper pin sampling width (measured in number of glitch filter clock cycles) during which the pin pull enable and input buffer enable are activated, or disables pin sampling. When tamper pin sampling is disabled, the tamper pin monitors the input continuously. When tamper pin sampling is enabled, the tamper pin pull enable and input buffer enable periodically assert for time widths determined by TPSW. The tamper pin pull enable and input buffer enable are always asserted whenever the glitch filter is actively filtering the input. The number of cycles refers to the number of cycles of the selected glitch filter clock (either 32.768 kHz or 512 Hz). 00b - Continuous monitoring, pin sampling disabled 01b - 2 cycles for pull enable and 1 cycle for input buffer enable 10b - 4 cycles for pull enable and 2 cycles for input buffer enable 11b - 8 cycles for pull enable and 4 cycles for input buffer enable
7 GFE	Glitch Filter Enable Enables the glitch filter. **NOTE** Do not change the glitch filter enable when the corresponding tamper pin is enabled. 0b - Bypasses 1b - Enables
6 GFP	Glitch Filter Prescaler Selects the clock source for the glitch filter.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	NOTE Do not change the prescaler when the glitch filter is enabled. 0b - 512 Hz prescaler clock 1b - 32.768 kHz clock
5-0 GFW	Glitch Filter Width Configures the number of clock edges during which the input must remain stable in order to pass through the glitch filter. The number of clock edges is (GFW + 1) * 2, supporting a configuration of between 2 and 128 clock edges. **NOTE** Do not change the glitch filter width when the glitch filter is enabled.

Chapter 23
Secure AHB bus and AHB Controller (AHBSC)

23.1 Chip-specific Secure AHB Controller information

Table 413. Reference links to related information[1]

Topic	Related module	Reference
Full description	Secure AHB Controller	Secure AHB Controller
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

23.1.1 Module instances

This device has one instance of the AHBSC module with four aliased base addresses.

23.1.2 Security considerations

The AHBSC module is instantiated to use four module slots—AHBSC, AHBSC_alias1, AHBSC_alias2, and AHBSC_alias3. The AHBSC registers can only be written when using a secure and privileged access. When a write is attempted using any other access level, this will generate a hypervisor interrupt instead of a hard fault. But read operation is allowed using any access levels. The hypervisor interrupt can be used as a mechanism to allow for switching directly to secure, privileged level from any other level by using a write access to the respective level's alias slot.

Given the above constraints, systems implementing hypervisor model should configure the four aliases as below using AHB_SECURE_CTRL_PERIPH_RULE0 register.

Table 414. AHBSC slots access level

Slot	Base address	Access level
AHBSC	0x4012_0000	Secure and privileged
AHBSC_alias1	0x4012_1000	Secure and non-privileged
AHBSC_alias2	0x4012_2000	Non-secure and privileged
AHBSC_alias3	0x4012_3000	Non-secure and non-privileged

The AHBSC registers can only be written when using the AHBSC (0x4012_0000) slot using a secure and privileged access. The AHBSC registers can be read from the alias addresses assuming the access has the appropriate access level, but the registers can never be written using the alias slots. When a write is attempted to one of the alias slots, this will generate a hypervisor interrupt instead of a hard fault. The hypervisor interrupt can be used as a mechanism to allow for switching directly to secure, privileged level from any other level by using a write access to the respective level's alias slot.

23.1.3 Secure AHB Controller mode

The Secure AHB Controller module supports strict mode. In strict mode, secure data access to non-secure address is not allowed. The following peripherals - GPIO, ELS, PUF, PKC, and AHBSC - can control access permission by themselves. To make

the peripherals accessed by both secure and non-secure masters in strict mode, alias slots are assigned for the peripherals. Software can configure each slot of a peripheral to different secure level. So masters with different secure level can access the peripheral independently.

23.1.4 Secure Interrupt Masking

TrustZone provides secure NVIC (NVIC_NS) and non-secure NVIC (NVIC_NS) capabilities. CPU0 has internal programmability to configure any interrupt as a secure interrupt, thus allowing it to be visible to NVIC_S while masking it from NVIC_NS. Refer to the ARM CM33 documents for more details. But on this SoC all interrupt sources are also connected to CPU1 which is a Coretex-M33 core without security extensions. Hence the security level of CPU1 is configured statically through MASTER_SEC_LEVEL registers. If CPU1 is used for non-secure application then interrupts from secure peripheral should be blocked by using SEC_CPU1_INT_MASKx registers.

23.2 Overview

The AHB Secure Controller enables the programming of the security attributes for all MPCs (Memory Protection Checkers), PPCs (Peripheral Protection Checkers), and MSWs (Master Security Wrappers). AHBSC supports the locking of SAU (Security Attribution Unit) settings, secure and non-secure MPU (Memory Protection Unit) settings (MPU_S/MPU_NS), secure and non-secure vector offset settings (VTOR_S/VTOR_NS) for CPU0. This enables the BootROM to safeguard certain security features, and prevents the possibility of enabling those settings dynamically, either unintentionally or maliciously.

This chapter discusses the basics of TrustZone for Armv8-M and then describes features that extend the TrustZone security foundation to enable a complete trusted execution environment.

23.2.1 Features

- Cortex-M33 with TrustZone support enabled
- Attribution Units: SAU (Security Attribution unit), IDAU (Device Attribution Unit)
- Secure MPU, Secure NVIC, Secure SYSTICK, Secure Stack Pointer
- Secure memory map aliasing
- Support for Arm AMBA 5.0 AHB secure bus
- Support for Arm AXI3 AXI bus
- Secure bus controller
- Memory and peripheral protection checkers
- Security attribution wrapper for AHB masters
- Interrupt masking
- Secure DMA and DMA masking
- Secure GPIO and GPIO masking
- Secure debug

23.3 Functional description

The AHB Secure Controller supports register programming for GPIO masking and interrupt masking. When a security violation is detected, an interrupt is raised by the AHB Secure Controller module. The AHB Secure Controller also logs violation information, like the address being accessed on a AHB slave port when the violation occurred, as well as the access type and security attributes of the master that generated the unauthorized access.

Only an application running in secure and privileged mode can write to the AHBSC to configure security attributes. From an application perspective, the highest tier (tier-4) thread from CPU0 can program security attributes for system slaves. Registers programmed in the AHB Secure Controller are retained during deep-sleep and power-down modes; however, after wake-up from deep power-down mode, these AHB Secure Controller registers must be re-programmed.

23.3.1 TrustZone for Armv8-M

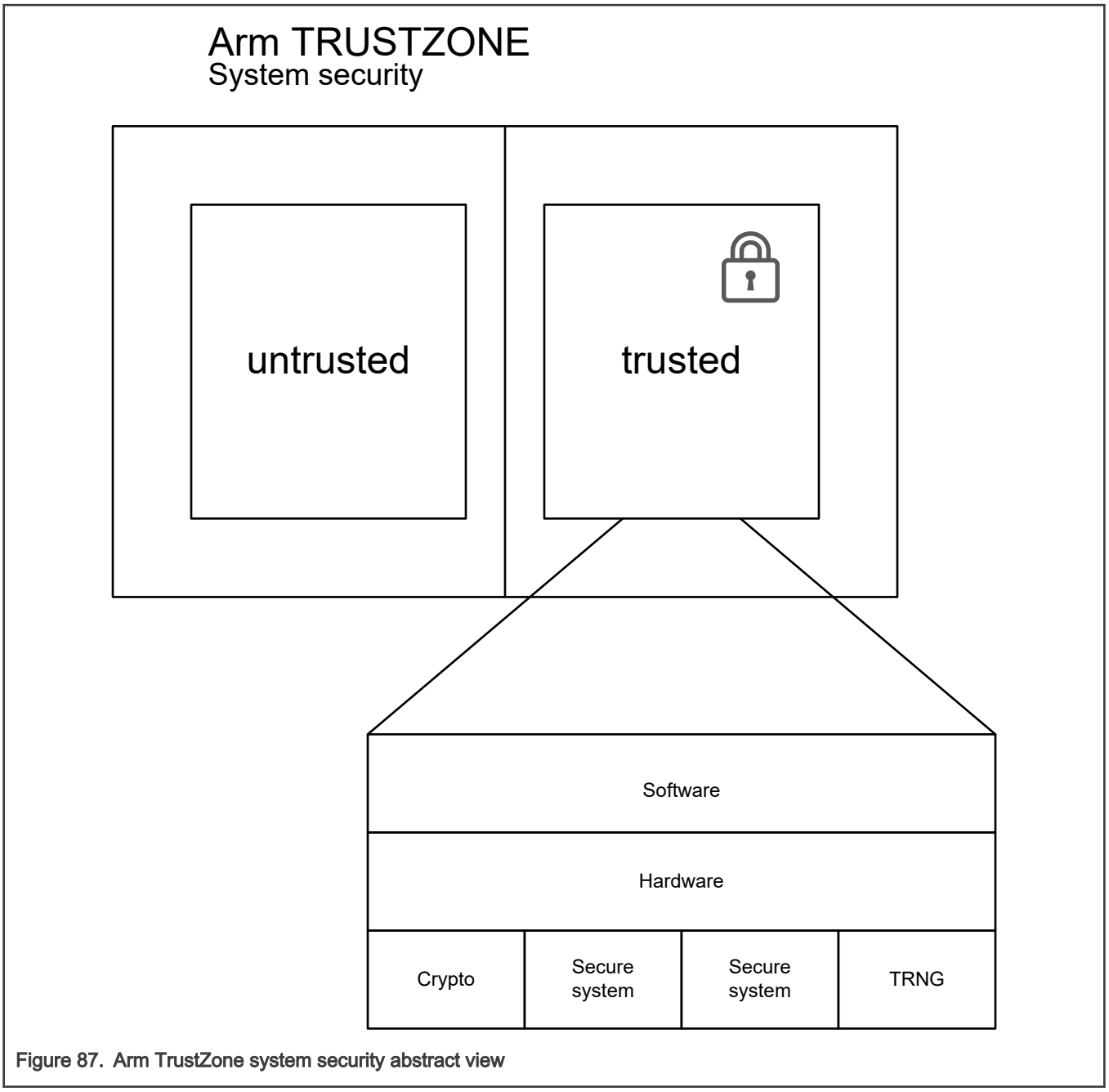


Figure 87. Arm TrustZone system security abstract view

TrustZone for Armv8-M is a new feature available on the Arm Cortex series that enables execution separation of trusted (Secure) software and access control isolation of trusted resources from non-trusted (Non-secure) software and resources, while running on the same CPU. This control of security is achieved by segmentation of memory arrays and peripherals into either Secure (S) or Non-secure (NS). TrustZone for Armv8 is optimized for energy efficient embedded applications that require real-time responsiveness.

The following rules apply to the M33 core if TZ-M functionality is enabled:

- CM33 CPU in Secure state (CPU-S) can execute instructions from secure memory (S-memory) only; it cannot execute from Non-secure memory (NS-memory).
- CPU-S can access data in both S-memory and NS-memory, that is, CPU-S can execute data reads from both S- and NS-memory, as well as execute data writes to S or NS-memory.

- CPU-NS can execute instructions only from NS-memory and cannot execute instructions from S-memory.
- CPU-NS can access data only in NS-memory; that is, CPU-NS can execute data

  reads from NS-memory only, and execute data writes to NS-memory only. CPU-NS cannot access data from S-memory.

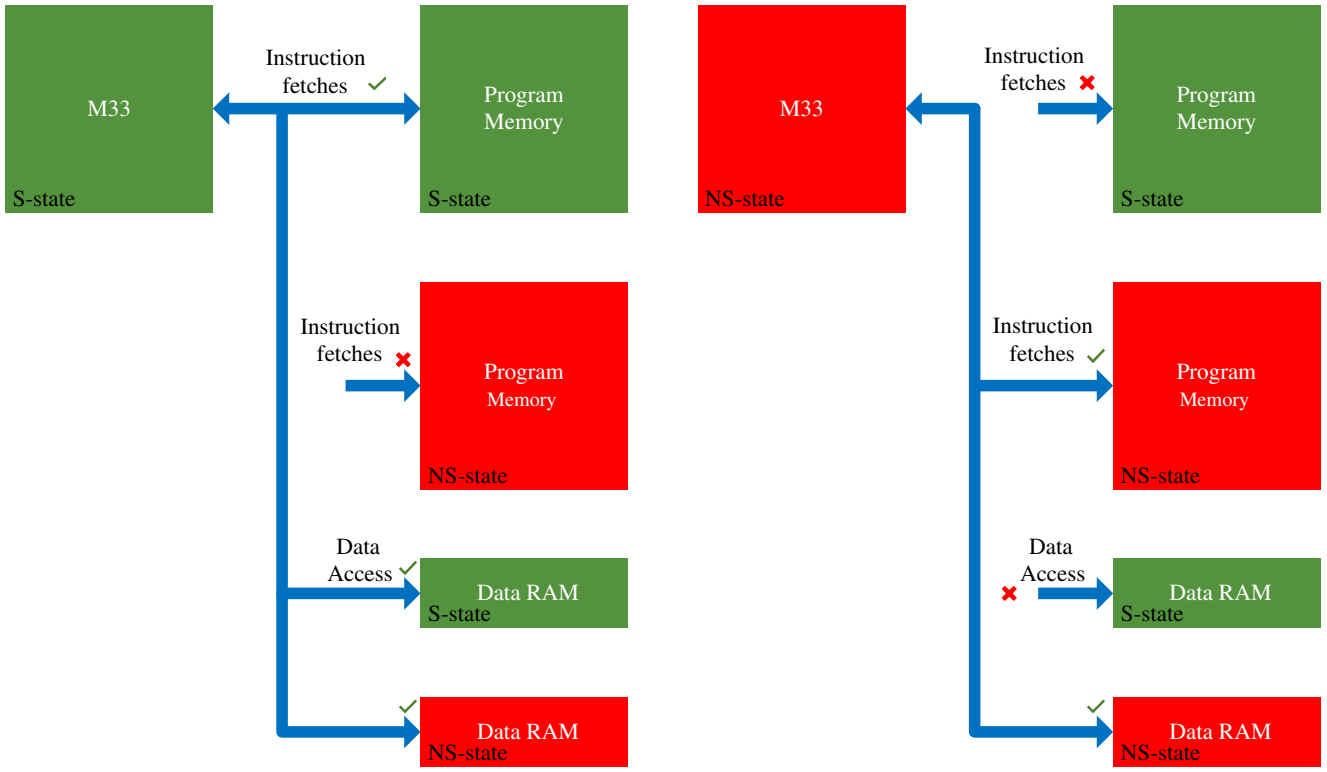


Figure 88. Secure state and Non-secure state view for TrustZone for Armv8

In summary:

- Secure code will not corrupt or modify NS code or data inadvertently or on purpose to create malfunction or hazard.
- S application code does not trust NS application code and disallows access to a CPU-NS.

To support secure state, Cortex-M33 architecture extends to include secure MPU, secure NVIC, secure SYSTICK, AHB Secure Controller and secure stack pointer with stack-threshold check.

23.3.1.1 State transitions

At reset release, CPU0 (CM33) is in Secure state.

CPU can call into NS application code from CPU-S state by executing newly introduced instructions:

- BXNS: Branch and Exchange Non Secure – branches to an address in NS-memory.
- BXLNS: Branch with Link Exchange Non Secure - calls a subroutine in NS memory.

On executing either the BXNS or BXLNS instructions the CPU-S will also change to the Non-secure state (CPU-NS) and thus be in the correct state for executing out of NS memory.

CPU cannot access S-memory directly when it is in NS-state. However, TZ-M provides a gateway into S-memory for NS-application code using a special region called Non-Secure Callable (NSC). The NSC region lies in S-memory and hence the CPU must be in CPU-S state to execute instructions in this region. The NSC region of S-memory provides a veneer for NS-application code to access functions in S-memory without divulging the specific address of the secure function.

When switching from CPU-NS to CPU-S, an additional gating factor is implemented in the form of a Secure Gate (SG) instruction. It is placed in the NSC region at the start of a secure function callable from a Non-secure code. When calling into NSC region,

the CPU-NS must target an address with the SG instruction. The SG instruction is the only instruction that can be executed when branching from a CPU-NS. On executing the SG instruction, the CPU will change from CPU-NS to the NSC region. If the CPU-NS calls into an address in the NSC region that is not an SG instruction, an secure fault is created.

The secure application code developer creates function calls inside the NSC region to S-application code, allowing the NS-application the ability to access functions inside S-memory.

23.3.2 Attribution units

TrustZone for the Armv8-M implementation consists of the Security Attribution unit (SAU) and Implementation Defined Attribution Unit (IDAU). Device Attribution Unit (DAU) connects to CPU0 via IDAU interface.

A combination of SAU and IDAU assign a specific security attribute (S, NS, or NSC) to a specific address from CPU0. Access from CPU0, dependent on its security status and the resultant security attribute set by the IDAU and SAU, is then compared by the secure AHB Controller to a specific checker which marks various access policies for memory and peripherals.

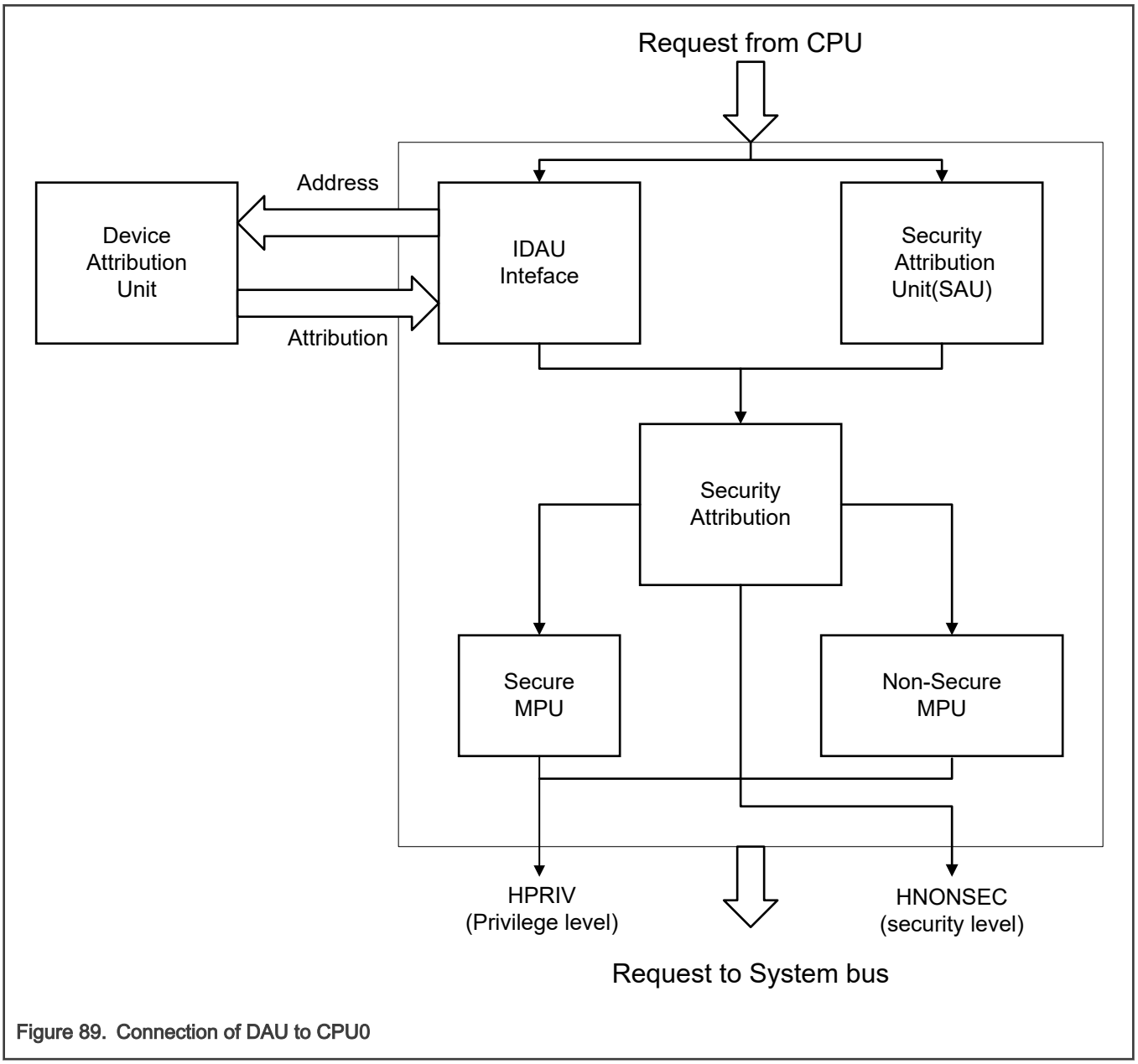


Figure 89. Connection of DAU to CPU0

23.3.2.1 Device Attribution Unit

The chip implements a simple Attribution unit that divides whole memory map into secure or Non-secure regions. All peripherals and memories are aliased at two locations.

- Address 0x0000_0000 to 0x1FFF_FFFF
  - Non-secure (always)
- Address 0x2000_0000 to 0xFFFF_FFFF
  - Non-secure if Address bit 28=0
  - Secure if Address bit 28=1

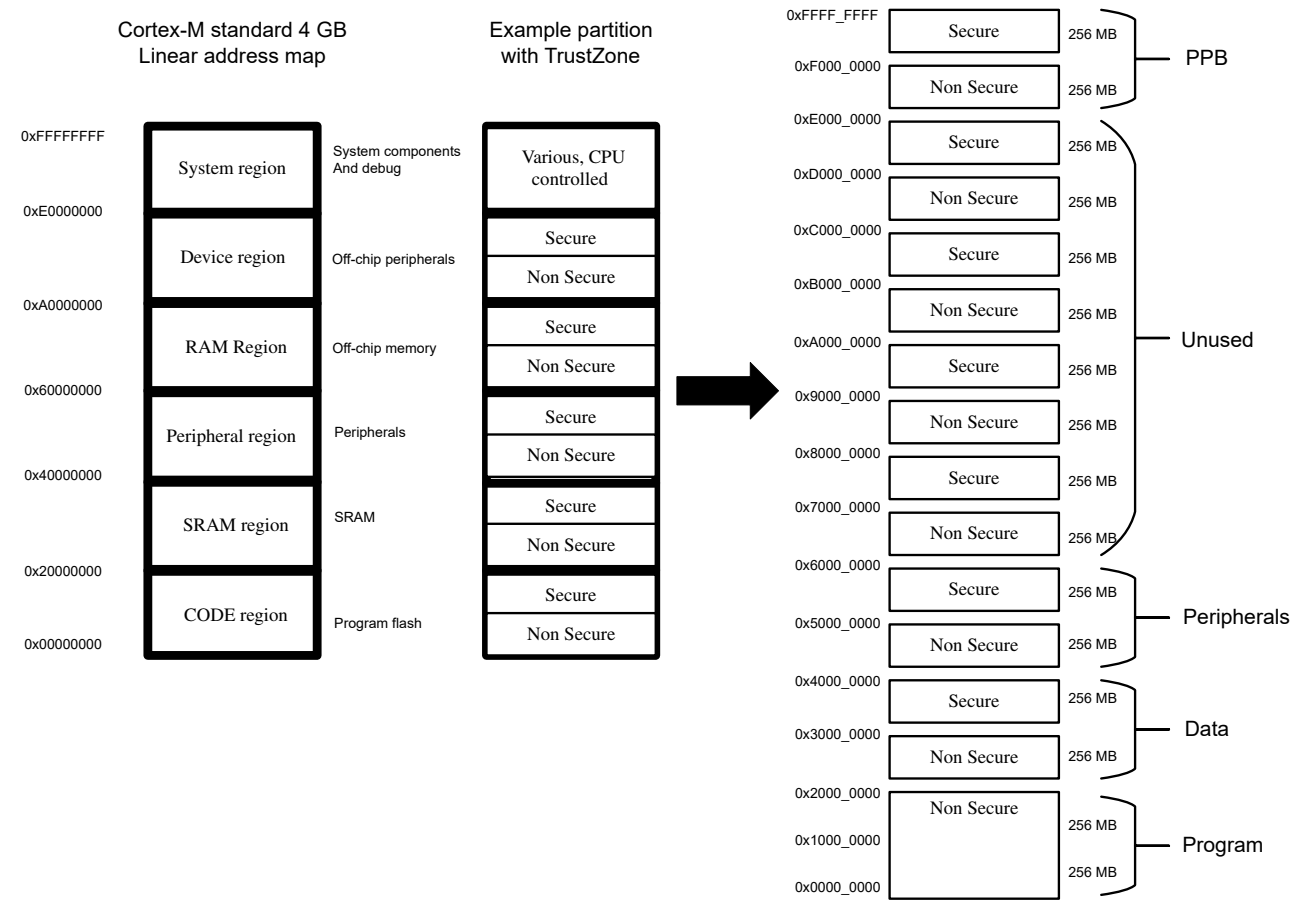


Figure 90. IDAU Implementation

23.3.2.2 Security Attribution Unit

The SAU is internal to CPU0 (CM33 with TZ). It monitors all addresses from the CPU0 and assigns an attribute if this address is S or NS. The SAU does not monitor addresses from bus masters other than the CPU0.

The SAU supports up to eight regional descriptors, each descriptor allows setting security state for a specific memory region from the following attributes.

- S – Secure.
- NS – Non-secure

- NSC – Non-secure Callable.

However, 0xF000_0000 to 0xFFFF_FFFF range is fixed as secure and SAU cannot program it to be NSC.

The SAU can only be configured by the CPU0 in the secure state. When enabled, the SAU will default all addresses as S. Only secure application code can program descriptors to create NSC or NS regions.

The IDAU works in conjunction with the SAU to assign a specific security attribute (S or NS) to a specific address. Both the IDAU and SAU will respond to a specific address and the CPU0 selects the higher of the two security attributes, where the highest state is Secure and the lowest state is NS. NSC attribute is defined by SAU. In IDAU NSC area can be defined as NS. Regions are aligned to 32-byte boundaries.

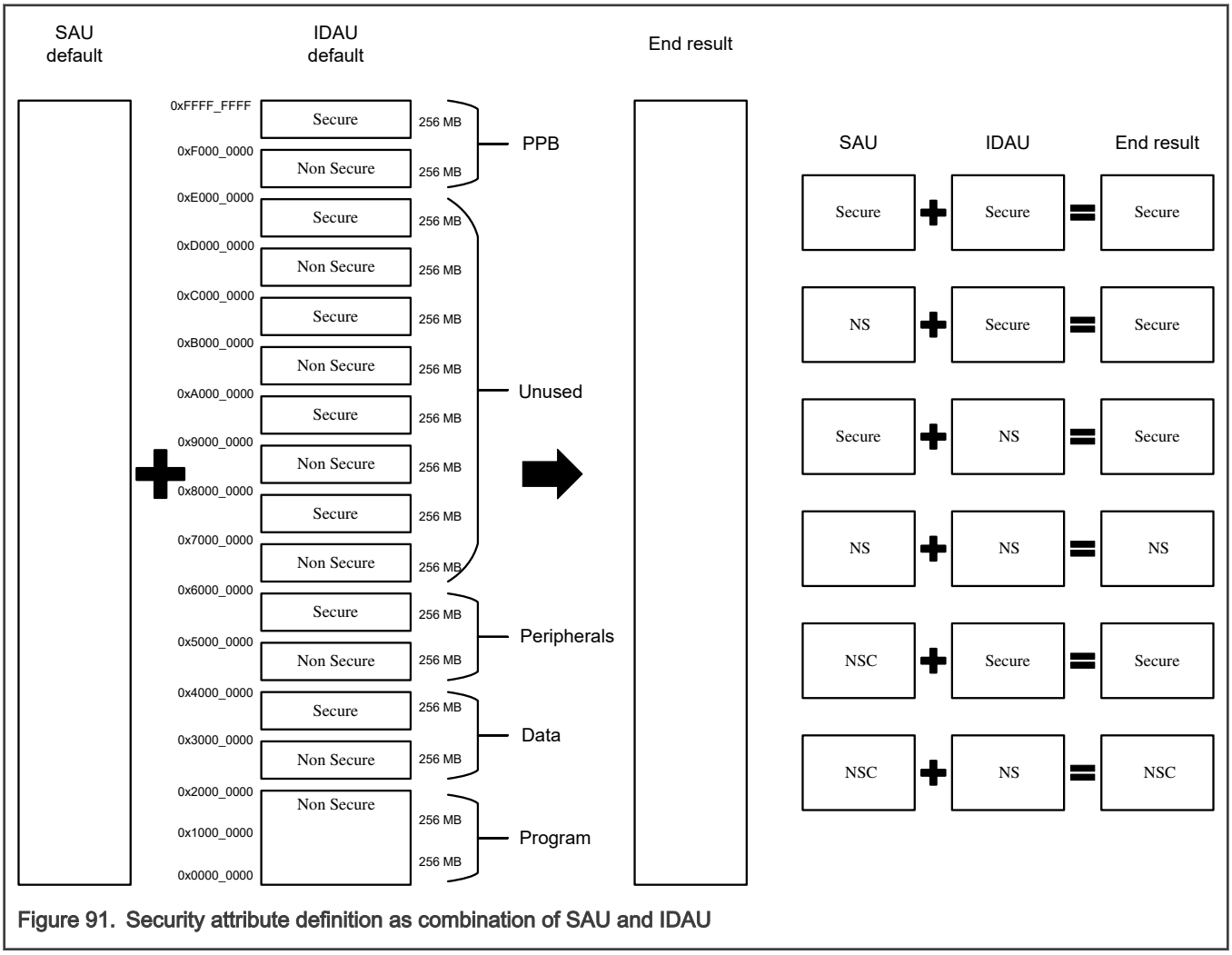


Figure 91. Security attribute definition as combination of SAU and IDAU

From a memory map perspective, the NS address space is an alias of the secure address space for the same physical program memory of 64kB address space at 0x0000_0000 to 0x0000_FFFF, Non-secure application code will fetch instructions in the 0x0000_0000 to 0x0000_FFFF. Non-secure (NS) space (address bit28 = 0) if the physical address space is configured as Non-secure, where secure application code will execute in 0x1000_0000 to 0x1000_FFFF Secure (S) space if the physical address space is configured as secure. Similarly, secure application code will access all peripherals in the 0x5000_0000 to 0x5FFF_FFFF space (address bit28=1), and NS application code will access NS peripherals at 0x4000_0000 to 0x4FFF_FFFF space. Details of SAU programmable registers can be found in Arm CM33 documents.

The figure below shows how SAU is used to create Secure and non secure alias.

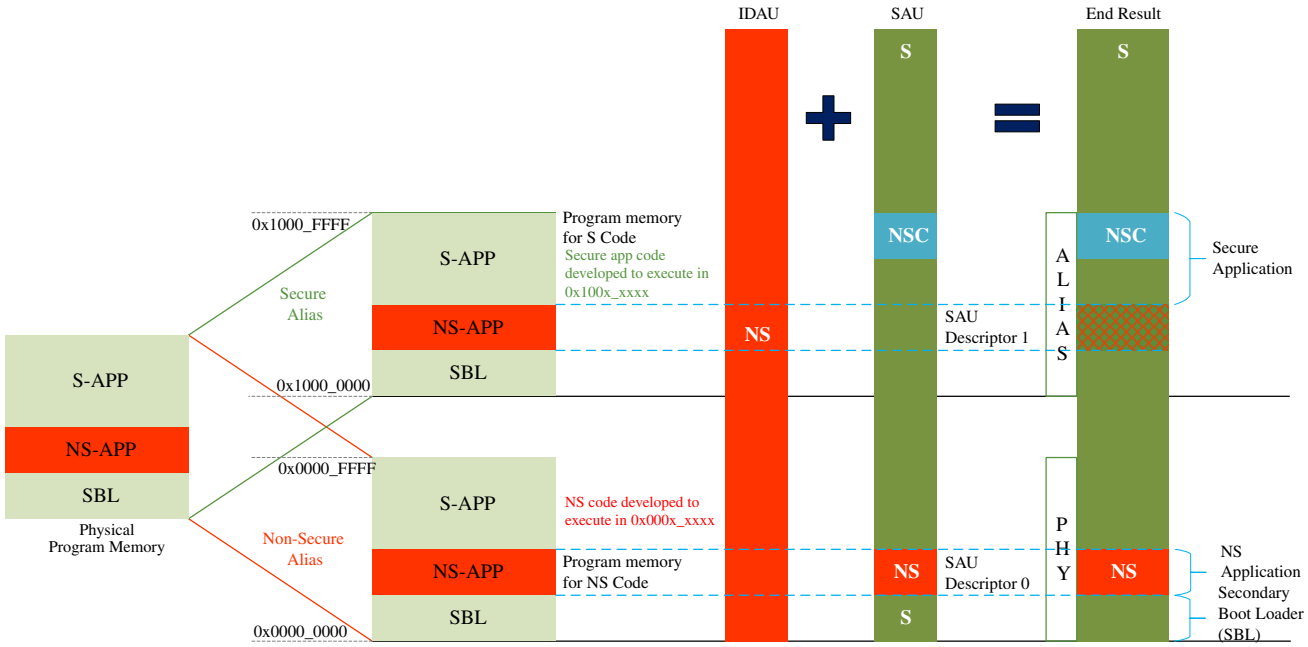


Figure 92. Program memory space aliasing and security attribution example

23.3.2.3 Region number and test target instruction

The IDAU generates a Region Number (RN) for each region, which can be used by application code to determine security level of that region. RN is a 8-bit number. In this device, IDAU returns region number as:

IDAU_RN[7:0] = ({4'h0, idau_addr_a[31:28]})

SAU will also return information on TT instruction indicating NS or S attribute.

Application can use Test Target instruction (TT) on start and end address of a region. Instruction returns RN and security attributes (NS or S).

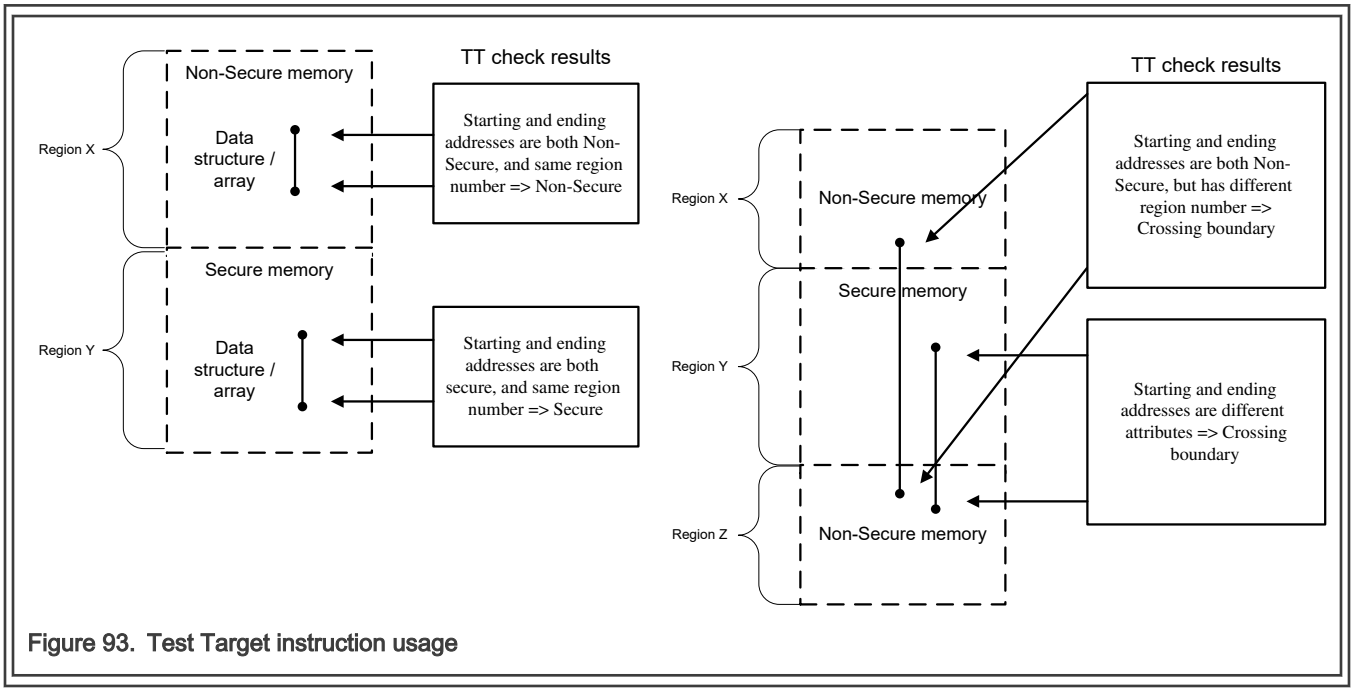


Figure 93. Test Target instruction usage

23.3.3 Secure AHB bus and Secure AHB Controller

CM33 TZ-M implementation consists of the IDAU and SAU modules, which filters address access from CPU0 based on specific security attribute (S, NS, or NSC) assigned to that address space. The the chip implements second layer of protection with secure AHB Bus to support secure trusted execution at system-level.

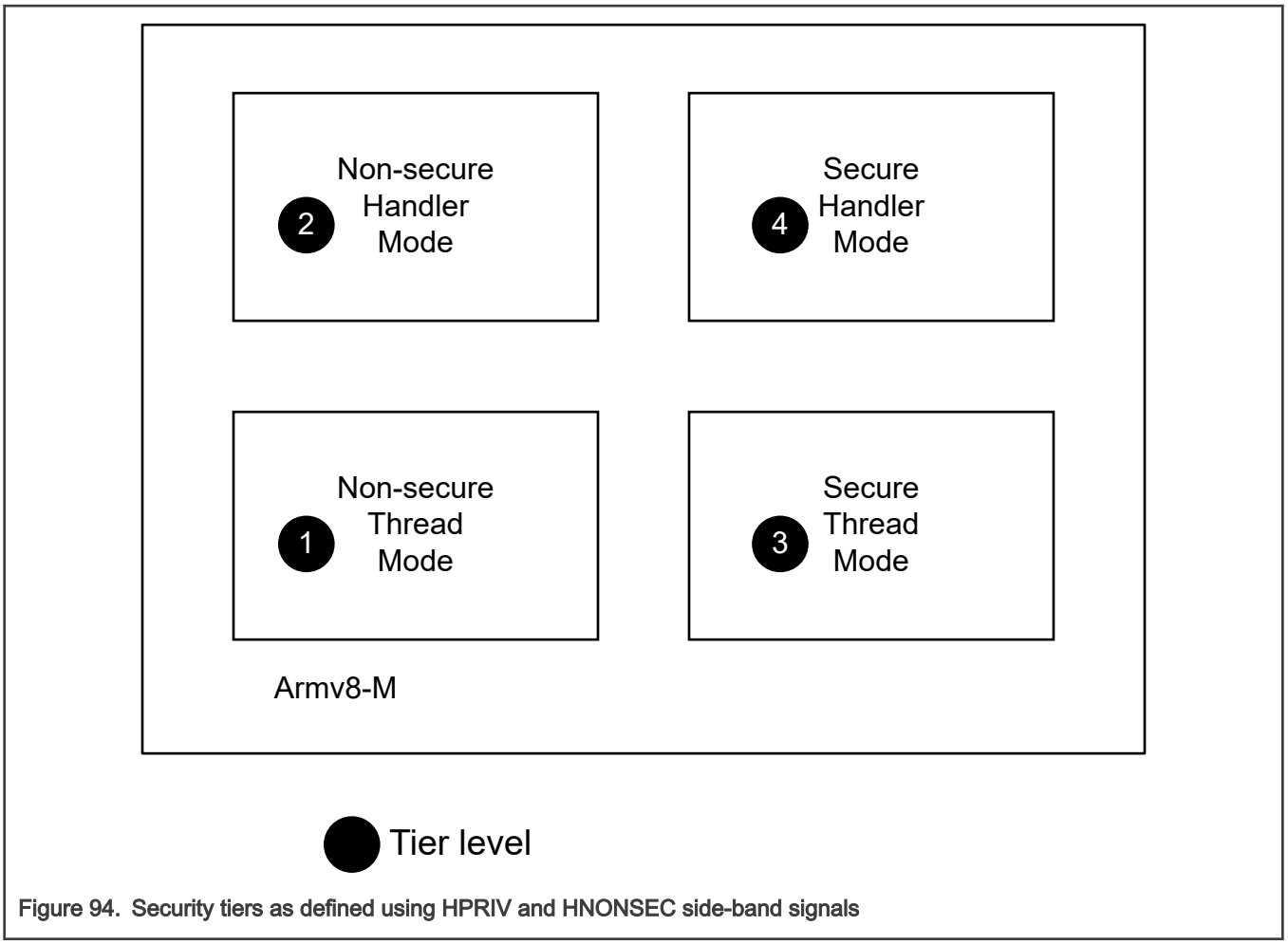


Figure 94. Security tiers as defined using HPRIV and HNONSEC side-band signals

The secure AHB Controller provides access policies for all the bus slaves via checker functions. All masters outputs security side-band signals HPRIV (privileged) and HNONSEC (Secure access) as indication of security attributes for a given access. Secure AHB Bus processes these signals and compares them against security attributes set for slaves in secure AHB Controller. Access is granted if security attribute of requested access is not violating the security attribute of the slave being accessed. Security violation exception interrupt is raised if violation occurs on data or instruction access. CPU0 switches to secure mode to handle the violation.

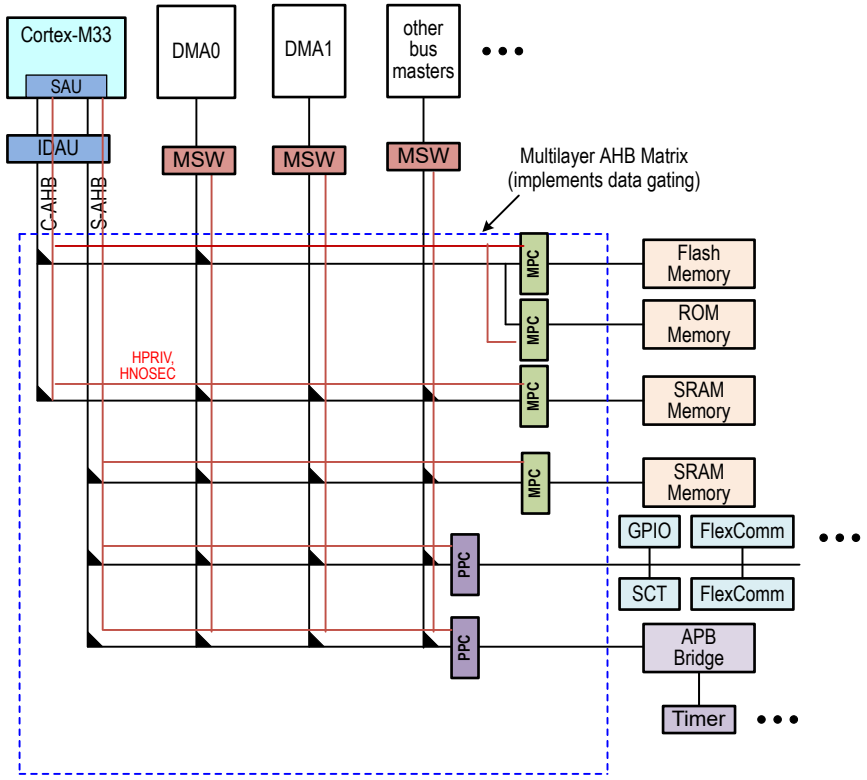


Figure 95. Example system view with secure AHB bus

The side-band signals create four security tiers. Data accesses are allowed from higher tier master to same or lower tier slave. However, instruction accesses are stricter, a master can access a slave only at same security tier. A special programmable option is available that allows treating all access in the system as instruction, meaning data access checker can also be as strict.

This protection is achieved using three primary components:

23.3.3.1 Memory Protection Checkers (MPC)

On-chip flash, ROM, RAM, and Fle can be protected against access from the application with lower tier security using Memory Protection Checkers (MPC). ROM, and each RAM bank has associated MPC. Flash has four MPCs. Flash, ROM and each RAM bank are divided into smaller sub-regions to offer more granularity for security tier assignment and filtering. Each sub-region can be assigned individual security tier by programing corresponding registers in secure AHB controller.

Table 415. Security tier granularity for on-chip memories

Memory	Total Size (KB)	Sub-region size (KB)	Sub-regions count
Flash00 – bank0	1024	32	32
Flash01 – bank1	1024	32	32
Flash02 – IFR0	64	16	4
Flash03 – IFR1	16	4	4
ROM	256	8	32
RAMX	96	4	24
RAMA	32	4	8

Table continues on the next page...

Table 415. Security tier granularity for on-chip memories (continued)

Memory	Total Size (KB)	Sub-region size (KB)	Sub-regions count
RAMB	32	4	8
RAMF	64	4	16
RAMG	64	4	16
RAMH	32	4	8
FlexSPI0 – region0	8192	256	32
FlexSPI0 – region1	8192	2048	4
FlexSPI0 – region2	16384	4096	4
FlexSPI0 – region3	32768	8192	4
FlexSPI0 – region4	65536	16384	4
FlexSPI0 – region5	65536	16384	4
FlexSPI0 – region6	65536	16384	4
FlexSPI0 – region7	8192	256	32

NOTE

All memories (SRAM/flash/Fle memory) which are executable should be configured as non-secure, non-privileged level (0b00) except the areas which host secure environment (SPE) code. This configuration helps to prevent a glitch attack being used to force the SPE to execute illegal code.

23.3.3.2 Peripheral Protection Checkers (PPC)

All peripherals on AHB slave ports as well as each peripheral on APB bridge can be protected against access from the application with lower tier security using Peripheral Protection Checkers (PPC). Each AHB port has associated PPC that offers granularity at individual slave level for security tier assignment and filtering. Each APB bridge has associated PPC that offers granularity for security tier assignment and filtering for each APB peripheral. Each peripheral can be assigned individual security tier by programing corresponding registers in secure AHB Controller.

NOTE

All peripheral registers spaces allocated or reserved should be assigned secure, privileged level (0b11).

23.3.3.3 Master Security Wrapper (MSW)

The TrustZone for ARMv8-M technology adds Secure and Non-secure states to Coretx-M33 processor

operation. In a simplified view, the program address determines the security state of the processor,

which can be either Secure or Non-secure.

- If the processor is running program code in Non-secure memory, the processor is in Non-secure state
- If the processor is running program code in Secure memory, the processor is in Secure state.
- If the processor is in Secure state, it must fetch instructions from Secure memory

The Coretx-M33 processor before doing any AHB bus transaction derives memory attribution by combining software configured SAU regions and IDAU defined region attributes. Based on processor's state, memory attribution and transaction type (code fetch or data access), the transaction level is determined. For example when processor in secure state and does a data access to non-secure attributed memory the transaction level is set as non-secure transaction. However other masters on chip don't have

notion of secure state, memory attribution and transaction level determination logic. Hence Master Security Wrapper (MSW) is implemented for each AHB Master except CPU0 to provide similar logic.

MSW allows application to set security state for each master through MASTER_SEC_LEVEL and MASTER_SEC_LEVEL_DP registers. The MSW implements simplified memory attribution logic based on address bit 28. If address bit 28 is set then the destination address is attributed as secure storage. The transaction level is determined based on the master type listed below.

1. Simple Master: AHB Bus Masters that perform data transaction only. Examples of this type of masters are USB-FS with built-in DMA and PKC master ports.
   - When the master is configured as secure and does data access to non-secure storage (address bit 28 is 0) then the transaction level is set as non-secure transaction.
   - When the master is configured as secure and does data access to secure storage (address bit 28 is 1) then the transaction level is set as secure transaction.
   - When the master is configured as non-secure, then irrespective of destination address the transaction level is set as non-secure transaction.
2. Smart Master: AHB Bus Masters that can execute code and perform data transaction. Examples of this type of masters are CoolFlux BSP32 DSP andSmart DMA(EZH).
   - When the master is configured as secure and does instruction fetch then irrespective of destination address the transaction level is set as secure transaction.
   - When the master is configured as secure and does data access to non-secure storage (address bit 28 is 0) then the transaction level is set as non-secure transaction.
   - When the master is configured as secure and does data access to secure storage (address bit 28 is 1) then the transaction level is set as secure transaction.
   - When the master is configured as non-secure, irrespective of transaction type or destination address the transaction level is set as non-secure transaction.
3. Masquerading Master: These masters inherit the processor state while configuring the channel. For example when Cortex-M33 in non-secure state issues AES encryption command (assuming the key properties are set to be used by non-secure context) then ELS master will inherit non-secure state when doing memory transactions to read/write data associated with AES command.
   - When the master is configured as secure and does data access to non-secure storage (address bit 28 is 0) then the transaction level is set as non-secure transaction.
   - When the master is configured as secure and does data access to secure storage (address bit 28 is 1) then the transaction level is set as secure transaction.
   - When the master is configured as non-secure, then irrespective of destination address the transaction level is set as non-secure transaction.

23.3.3.4 Secure AHB controller

Secure AHB controller allows programming security attributes for all MPCs, PPCs in addition to MSWs.

It also supports locking of SAU setting, secure and Non-secure MPU settings (MPU_S/MPU_NS), secure and Non-secure vector offset settings (VTOR_S/VTOR_NS) for CPU0. This enables BootROM to safeguard certain security features and disable the possibility of enabling those dynamically by unintentionally or with malicious intent.

It also supports register programing for GPIO masking. See SEC GPIO Mask register.

When a security violation is detected, an interrupt is raised by the secure AHB Controller module. It also logs violation information, (such as the address being accessed on a AHB slave port), when the violation occurred as well as access type and security attributes of the master that generated the unauthorized access.

Only an application running in secure and privileged mode can write to the AHBSC to configure security attributes. From an application perspective, the highest tier (tier-4) thread from CPU0 can program security attributes for system slaves. Registers

programmed in the AHB Secure Controller are retained during deep-sleep and power-down modes; however, after wake-up from deep power-down mode, these AHB Secure Controller registers must be re-programmed.

23.3.4 Example Configuration

There are two layers of TrustZone protection implemented. The first layer consists of two attribution units on the CM33 core:

- SAU (Security Attrrbution Unit)
- IDAU (Implementation Defined Attribution Unit)

The second layer consist of secure AHB bus and AHB secure controller implemented on the system level. The proper configuration of both layers is essential for secure trusted environment execution. The figure below shows an example of TrustZone configuration. The simplified device consists of CM33 core with security extension running in secure or normal mode, 64KB of code memory, 64KB data memory and four peripherals. The peripherals, code and data memory are connected to the core via secure AHB bus. Due to memory address aliasing (using address bit A28) the CM33 core sees all memories and peripherals twice in the memory map. For example, the 64KB code memory can be seen either as 0x0000 – 0xFFFF or 0x1000_0000 – 0x1000_FFFF. To keep figures simple, the data memory (0x2000_0000 – 0x2000_FFFF, 0x3000_0000 – 0x3000_FFFF) is not shown on all figures.

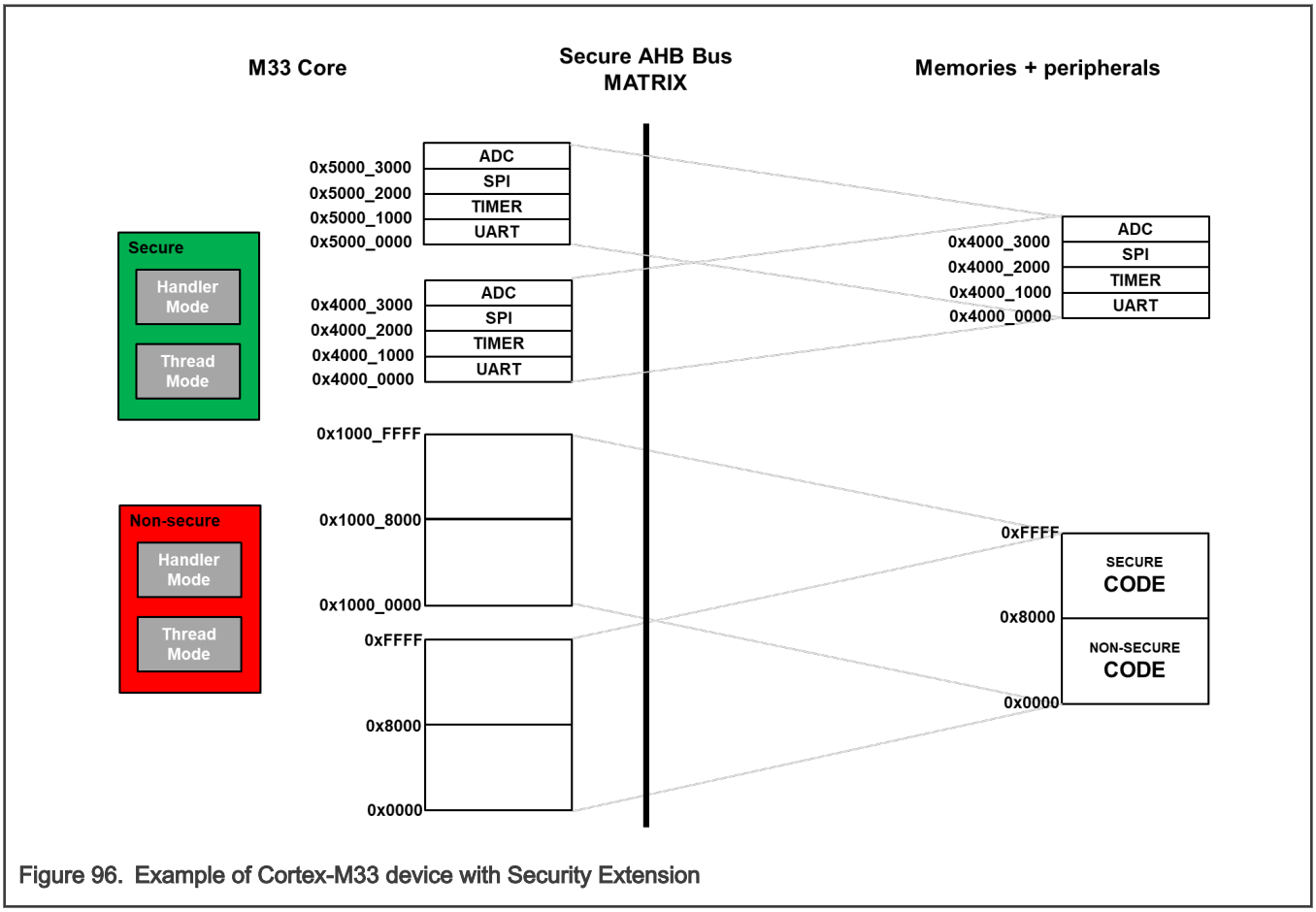


Figure 96. Example of Cortex-M33 device with Security Extension

23.3.4.1 Basic SAU Configuration

The TrustZone configuration starts with secure attribution map definition, which splits memories and peripherals between secure and non-secure domain. The security attribution map is defined by IDAU and SAU. The default security attribution map is defined by IDAU and it can be modified by SAU configuration. If the SAU is disabled or empty (none of SAU region is configured and enabled) the whole address space is secure. To assign some address space into non-secure domain, this address space must be configured in the SAU and concurrently the same address space must be marked as non-secure in IDAU. It means that address space marked as secure in IDAU can never be assigned to non-secure domain.

The simplest way to set up SAU is to configure every non-secure contiguous address space as one region in SAU. Using this approach, the SAU will be configured as follows:

Table 416. Basic SAU Configuration

	RBAR	RLAR		
	Base Address	Limit Address	NSC	ENABLE
Region 0 (code memory)	0x0000_0000	0x0000_7FFF	0	1
Region 1 (NSC memory)	0x1000_FC00	0x1000_FFFF	1	1
Region 2 (data memory)	0x2000_0000	0x2000_BFFF	0	1
Region 3 (UART)	0x4000_0000	0x4000_0FFF	0	0
Region 4 (TIMER)	0x4000_3000	0x4000_3FFF	0	0

The security attribution map after SAU configuration can be seen in the figure below. The SAU configuration also includes 1KB of secure, non-secure callable (NSC) memory placed on the top of secure memory. This region is used for veneer table. The veneer table contains all secure functions/services, which are callable from non-secure world. The NSC region is not shown on all figures as well as data memory.

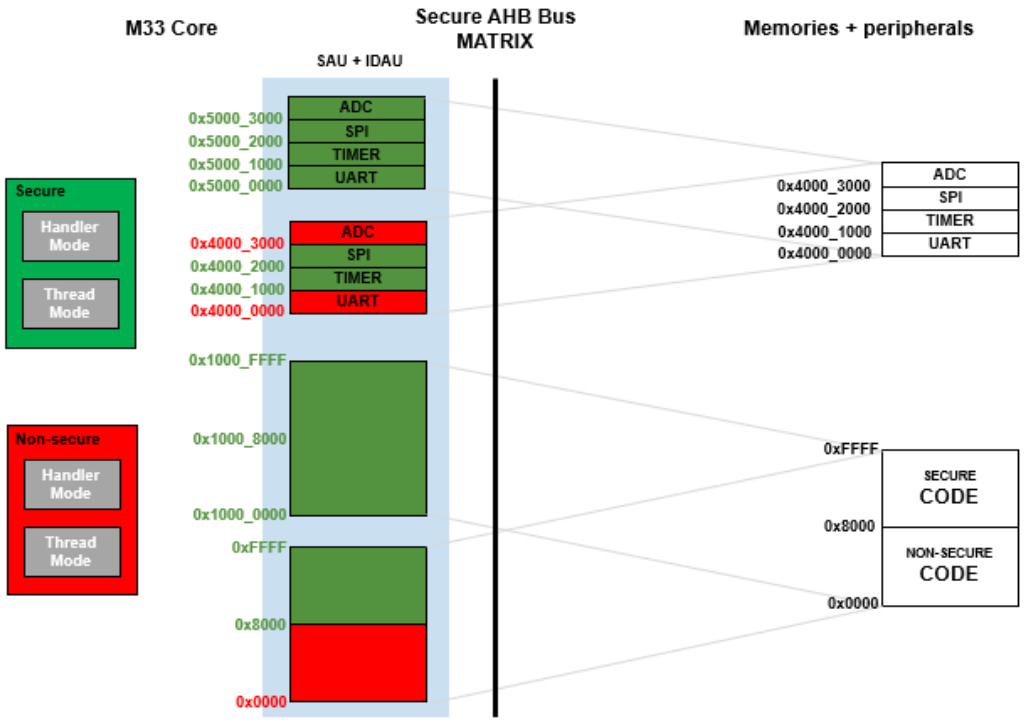


Figure 97. TrustZone Isolation after basic SAU configuration

After basic SAU configuration, the TrustZone isolation is fully functional, but there are following limitations:

- Every contiguous memory space requires one SAU region. Since there are 8 SAU regions only, this approach is suitable for simple applications.
- Since IDAU/SAU manages CM33 core transactions only, there is no any information about TrustZone isolation on system level. It means that TrustZone isolation is unknown to other bus masters in the system like DMA, USB and others.
- TrustZone isolation relies on SAU configuration only. In case of wrong SAU configuration (due to software error or glitch attack), TrustZone isolation is corrupted also.

Due to these limitations, this way of trusted execution environment configuration (based on basic SAU set up only) is not recommended. A second layer protection using the AHB Secure Controller must be employed. Second protection layer brings much higher flexibility in TrustZone configuration and higher isolation security.

23.3.4.2 Canonical SAU Configuration

Real applications are much more complex than presented example. Especially peripherals configuration can be challenging with 8 SAU regions only. Therefore, different approach for SAU configuration must be chosen. This approach is also called canonical SAU configuration. It relies on the principle, that all memories and peripherals have one secure and one non-secure alias. Said another way, every region in standard Cortex-M address space is divided into halves, where address space with address bit A28=1 means secure address and with A28=0 means non-secure address. The canonical SAU configuration can be seen in the table below.

Table 417. Canonical SAU Configuration

	RBAR	RLAR		
	Base Address	Limit Address	NSC	ENABLE
Region 0 (code memory)	0x0000_0000	0x1FFF_FFFF	0	1
Region 1 (data memory)	0x2000_0000	0x5FFF_FFFF	0	1

The canonical SAU configuration requires two regions only so there are still six SAU regions available for other purposes, for example for NSC regions definition. The presented example, configured using canonical SAU configuration, can be seen in the figure below.

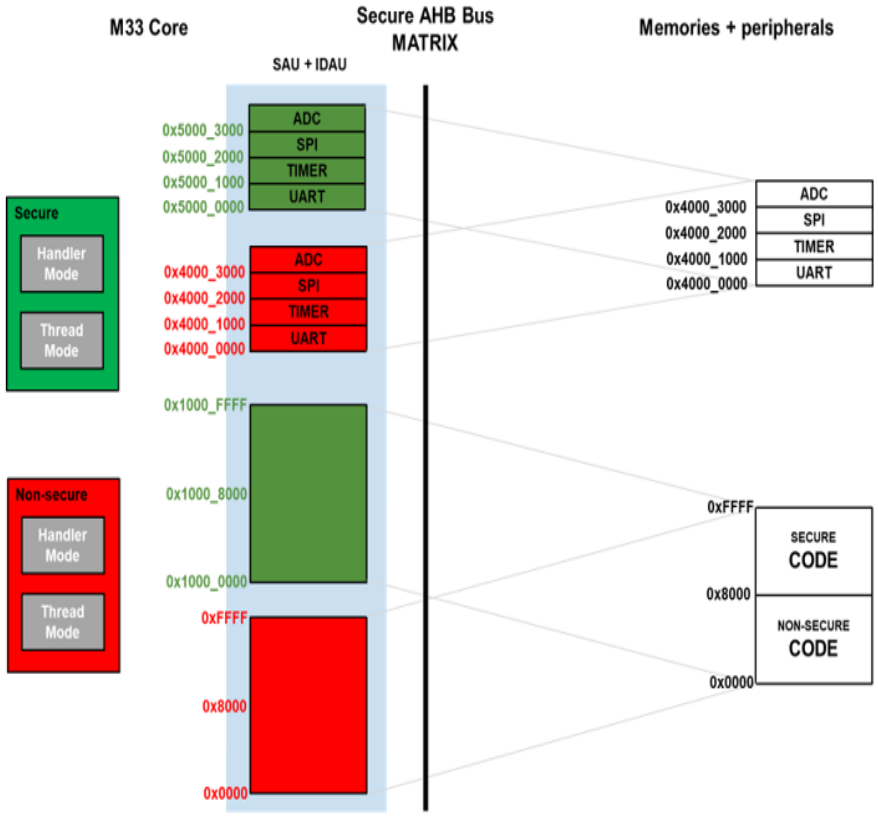


Figure 98. TrustZone Isolation after canonical SAU configuration

Comparing TrustZone Isolation after basic SAU configuration and TrustZone Isolation after canonical SAU configuration, it can be seen, that TrustZone isolation is not functional now. For example, secure code in address range 0x1000_8000 – 0x1000_FFFF is also available in non-secure mode in address range 0x8000 – 0xFFFF. This is because the SAU configuration is visible on core level only and thus system cannot know, whether address range 0x8000 – 0xFFFF is assigned to secure or non-secure domain. Moreover, the address range 0x8000 – 0xFFFF is assigned to both domains on the core level. To make TrustZone isolation functional, the AHB secure controller must be configured and enabled.

23.3.4.3 TrustZone Isolation after Canonical form of SAU and AHB Secure Controller Configuration

After proper AHB secure controller configuration the TrustZone isolation is fully functional. The non-secure software can still issue non-secure transaction with address 0x8000, but this transaction is blocked by AHB secure controller because the memory with physical address 0x8000 is already assigned to secure domain. It means that secure code is accessible via secure alias (0x1000_8000 – 0x1000FFFF) only.

Another effect of memory and peripheral checkers is also higher configuration flexibility. At canonical SAU configuration, user is not limited by number of SAU regions and flexibility is enhanced up to memory sub-region or peripheral level. Besides protection checkers, the AHB secure controller implements simplified IDAU, MSW for all non-core bus master (MSW) so they can also benefit from TrustZone isolation.

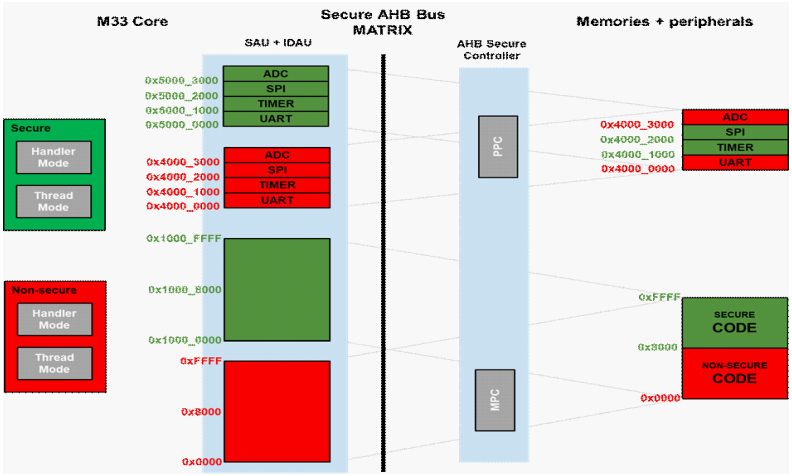


Figure 99. TrustZone Isolation after canonical form of SAU and AHB Secure Controller Configuration

23.3.4.4 Combined SAU and AHB Secure Controller Configuration

The last SAU configuration example combines both previous approaches together with AHB secure controller. The advantage of basic SAU configuration together with enabled AHB secure controller is the double checking each bus transaction. The first check is done on core level (SAU/IDAU), and the second check is performed on the system level (AHB secure controller). If some inconsistency is detected between SAU and AHB secure controller configurations (due to some software error or glitch attack), the access to the specific resource is blocked.

So, employing both SAU and AHB secure controller for TrustZone isolation makes isolation more robust compared to basic SAU configuration. The TrustZone example shown in figure below uses basic SAU configuration for code and data memories, and canonical SAU configuration for peripherals. This combination achieves the highest isolation robustness while allowing sufficient flexibility even for complex applications.

Table 418. Combined SAU Configuration

	RBAR register	RLAR register		
	Base Address	Limit Address	NSC bit	ENABLE bit
Region 0 (code memory)	0x0000_0000	0x0000_7FFF	0	1
Region 1 (NSC memory)	0x1000_FC00	0x1000_FFFF	1	1
Region 2 (data memory)	0x2000_0000	0x2000_BFFF	0	1
Region 3 (peripheral memory)	0x4000_0000	0x5FFF_FFFF	0	1

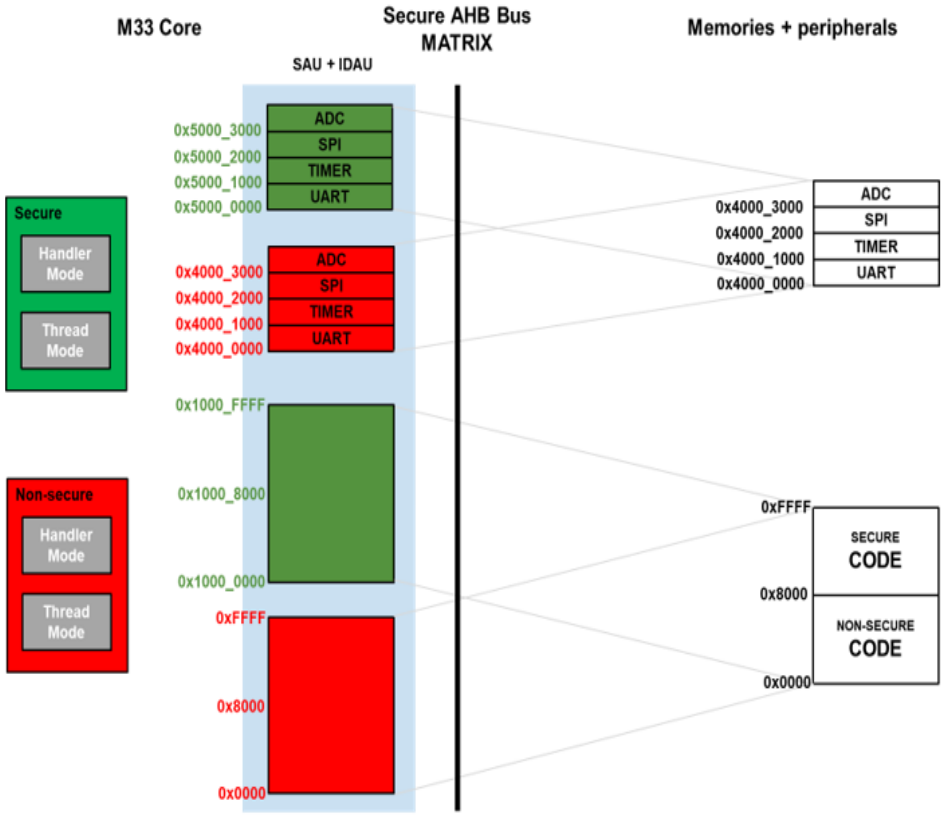


Figure 100. TrustZone Isolation after Combined SAU and AHB Secure Controller Configuration

23.4 Memory Map and Registers

This section includes the AHBSC module memory map and detailed descriptions of all registers.

23.4.1 AHBSC register descriptions

23.4.1.1 AHBSC memory map

AHBSC_alias3 base address: 4012_3000h

Offset	Register	Width (In bits)	Access	Reset value
10h - 1Ch	Flash Memory Rule (FLASH00_MEM_RULE0 - FLASH00_MEM_RULE3)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
20h - 2Ch	Flash Memory Rule (FLASH01_MEM_RULE0 - FLASH01_MEM_RULE3)	32	RW	0000_0000h
30h	Flash Memory Rule (FLASH02_MEM_RULE)	32	RW	0000_0000h
40h	Flash Memory Rule (FLASH03_MEM_RULE)	32	RW	0000_0000h
60h - 6Ch	ROM Memory Rule (ROM_MEM_RULE0 - ROM_MEM_RULE3)	32	RW	3333_3333h
80h - 88h	RAMX Memory Rule (RAMX_MEM_RULE0 - RAMX_MEM_RULE2)	32	RW	0000_0000h
A0h	RAMA Memory Rule 0 (RAMA_MEM_RULE)	32	RW	0000_0000h
C0h	RAMB Memory Rule (RAMB_MEM_RULE)	32	RW	0000_0000h
E0h - E4h	RAMC Memory Rule (RAMC_MEM_RULE0 - RAMC_MEM_RULE1)	32	RW	0000_0000h
100h - 104h	RAMD Memory Rule (RAMD_MEM_RULE0 - RAMD_MEM_RULE1)	32	RW	0000_0000h
120h - 124h	RAME Memory Rule (RAME_MEM_RULE0 - RAME_MEM_RULE1)	32	RW	0000_0000h
140h - 144h	RAMF Memory Rule (RAMF_MEM_RULE0 - RAMF_MEM_RULE1)	32	RW	0000_0000h
160h - 164h	RAMG Memory Rule (RAMG_MEM_RULE0 - RAMG_MEM_RULE1)	32	RW	0000_0000h
180h	RAMH Memory Rule (RAMH_MEM_RULE)	32	RW	0000_0000h
1A0h	APB Bridge Group 0 Memory Rule 0 (APB_PERIPHERAL_GROUP0_MEM_RULE0)	32	RW	3333_3333h
1A4h	APB Bridge Group 0 Memory Rule 1 (APB_PERIPHERAL_GROUP0_MEM_RULE1)	32	RW	3333_3333h
1A8h	APB Bridge Group 0 Rule 2 (APB_PERIPHERAL_GROUP0_MEM_RULE2)	32	RW	3333_3333h
1ACh	APB Bridge Group 0 Memory Rule 3 (APB_PERIPHERAL_GROUP0_MEM_RULE3)	32	RW	3333_3333h
1B0h	APB Bridge Group 1 Memory Rule 0 (APB_PERIPHERAL_GROUP1_MEM_RULE0)	32	RW	3333_3333h
1B4h	APB Bridge Group 1 Memory Rule 1 (APB_PERIPHERAL_GROUP1_MEM_RULE1)	32	RW	3333_3333h
1BCh	APB Bridge Group 1 Memory Rule 2 (APB_PERIPHERAL_GROUP1_MEM_RULE2)	32	RW	3333_3333h
1C0h	AIPS Bridge Group 0 Memory Rule 0 (AIPS_BRIDGE_GROUP0_MEM_RULE0)	32	RW	3333_3333h
1C4h	AIPS Bridge Group 0 Memory Rule 1 (AIPS_BRIDGE_GROUP0_MEM_RULE1)	32	RW	3333_3333h
1C8h	AIPS Bridge Group 0 Memory Rule 2 (AIPS_BRIDGE_GROUP0_MEM_RULE2)	32	RW	3333_3333h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
1CCh	AIPS Bridge Group 0 Memory Rule 3 (AIPS_BRIDGE_GROUP0_MEM_RULE3)	32	RW	3333_3333h
1D0h	AHB Peripheral 0 Slave Port 12 Slave Rule 0 (AHB_PERIPHERAL0_SLAVE_PORT_P12_SLAVE_RULE0)	32	RW	3333_3333h
1D4h	AHB Peripheral 0 Slave Port 12 Slave Rule 1 (AHB_PERIPHERAL0_SLAVE_PORT_P12_SLAVE_RULE1)	32	RW	3333_3333h
1D8h	AHB Peripheral 0 Slave Port 12 Slave Rule 2 (AHB_PERIPHERAL0_SLAVE_PORT_P12_SLAVE_RULE2)	32	RW	3333_3333h
1E0h	AIPS Bridge Group 1 Rule 0 (AIPS_BRIDGE_GROUP1_MEM_RULE0)	32	RW	3333_3333h
1E4h	AIPS Bridge Group 1 Rule 1 (AIPS_BRIDGE_GROUP1_MEM_RULE1)	32	RW	3333_3330h
1F0h	AHB Peripheral 1 Slave Port 13 Slave Rule 0 (AHB_PERIPHERAL1_SLAVE_PORT_P13_SLAVE_RULE0)	32	RW	3333_3333h
1F4h	AHB Peripheral 1 Slave Port 13 Slave Rule 1 (AHB_PERIPHERAL1_SLAVE_PORT_P13_SLAVE_RULE1)	32	RW	3333_3333h
1F8h	AHB Peripheral 1 Slave Port 13 Slave Rule 2 (AHB_PERIPHERAL1_SLAVE_PORT_P13_SLAVE_RULE2)	32	RW	3333_3333h
200h	AIPS Bridge Group 2 Rule 0 (AIPS_BRIDGE_GROUP2_MEM_RULE0)	32	RW	3333_3333h
204h	AIPS Bridge Group 2 Memory Rule 1 (AIPS_BRIDGE_GROUP2_MEM_RULE1)	32	RW	3333_3333h
220h	AIPS Bridge Group 3 Rule 0 (AIPS_BRIDGE_GROUP3_MEM_RULE0)	32	RW	3333_3333h
224h	AIPS Bridge Group 3 Memory Rule 1 (AIPS_BRIDGE_GROUP3_MEM_RULE1)	32	RW	3333_3333h
228h	AIPS Bridge Group 3 Rule 2 (AIPS_BRIDGE_GROUP3_MEM_RULE2)	32	RW	3333_3333h
22Ch	AIPS Bridge Group 3 Rule 3 (AIPS_BRIDGE_GROUP3_MEM_RULE3)	32	RW	3333_3333h
240h	AIPS Bridge Group 4 Rule 0 (AIPS_BRIDGE_GROUP4_MEM_RULE0)	32	RW	3333_3333h
244h	AIPS Bridge Group 4 Rule 1 (AIPS_BRIDGE_GROUP4_MEM_RULE1)	32	RW	3333_3333h
248h	AIPS Bridge Group 4 Rule 2 (AIPS_BRIDGE_GROUP4_MEM_RULE2)	32	RW	3333_3333h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
24Ch	AIPS Bridge Group 4 Rule 3 (AIPS_BRIDGE_GROUP4_MEM_RULE3)	32	RW	3333_3333h
250h	AHB Secure Control Peripheral Rule 0 (AHB_SECURE_CTRL_PERIPHERAL_RULE0)	32	RW	0000_3333h
270h - 27Ch	FLEXSPI0 Region 0 Memory Rule (FLEXSPI0_REGION0_MEM_RULE0 - FLEXSPI0_REGION0_MEM_RULE3)	32	RW	0000_0000h
280h	FLEXSPI0 Region 1 Memory Rule 0 (FLEXSPI0_REGION1_MEM_RULE0)	32	RW	0000_0000h
290h	FLEXSPI0 Region 2 Memory Rule 0 (FLEXSPI0_REGION2_MEM_RULE0)	32	RW	0000_0000h
2A0h	FLEXSPI0 Region 3 Memory Rule 0 (FLEXSPI0_REGION3_MEM_RULE0)	32	RW	0000_0000h
2B0h	FLEXSPI0 Region 4 Memory Rule 0 (FLEXSPI0_REGION4_MEM_RULE0)	32	RW	0000_0000h
2C0h	FLEXSPI0 Region 5 Memory Rule 0 (FLEXSPI0_REGION5_MEM_RULE0)	32	RW	0000_0000h
2D0h	FLEXSPI0 Region 6 Memory Rule 0 (FLEXSPI0_REGION6_MEM_RULE0)	32	RW	0000_0000h
2E0h - 2ECh	FLEXSPI0 Region 7 Memory Rule (FLEXSPI0_REGION7_MEM_RULE0 - FLEXSPI0_REGION7_MEM_RULE3)	32	RW	0000_0000h
2F0h	FLEXSPI0 Region 8 Memory Rule 0 (FLEXSPI0_REGION8_MEM_RULE0)	32	RW	0000_0000h
300h	FLEXSPI0 Region 9 Memory Rule 0 (FLEXSPI0_REGION9_MEM_RULE0)	32	RW	0000_0000h
310h	FLEXSPI0 Region 10 Memory Rule 0 (FLEXSPI0_REGION10_MEM_RULE0)	32	RW	0000_0000h
320h	FLEXSPI0 Region 11 Memory Rule 0 (FLEXSPI0_REGION11_MEM_RULE0)	32	RW	0000_0000h
330h	FLEXSPI0 Region 12 Memory Rule 0 (FLEXSPI0_REGION12_MEM_RULE0)	32	RW	0000_0000h
340h	FLEXSPI0 Region 13 Memory Rule 0 (FLEXSPI0_REGION13_MEM_RULE0)	32	RW	0000_0000h
E00h - E7Ch	Security Violation Address (SEC_VIO_ADDR0 - SEC_VIO_ADDR31)	32	R	0000_0000h
E80h - EFCh	Security Violation Miscellaneous Information at Address (SEC_VIO_MISC_INFO0 - SEC_VIO_MISC_INFO31)	32	R	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
F00h	Security Violation Info Validity for Address (SEC_VIO_INFO_VALID)	32	RW	0000_0000h
F80h	GPIO Mask for Port 0 (SEC_GPIO_MASK0)	32	RW	FFFF_FFFFh
F84h	GPIO Mask for Port 1 (SEC_GPIO_MASK1)	32	RW	FFFF_FFFFh
F98h	Secure Interrupt Mask 0 for CPU1 (SEC_CPU1_INT_MASK0)	32	RW	FFFF_FFFFh
F9Ch	Secure Interrupt Mask 1 for CPU1 (SEC_CPU1_INT_MASK1)	32	RW	FFFF_FFFFh
FA0h	Secure Interrupt Mask 2 for CPU1 (SEC_CPU1_INT_MASK2)	32	RW	FFFF_FFFFh
FA4h	Secure Interrupt Mask 3 for CPU1 (SEC_CPU1_INT_MASK3)	32	RW	FFFF_FFFFh
FA8h	Secure Interrupt Mask 4 for CPU1 (SEC_CPU1_INT_MASK4)	32	RW	0FFF_FFFFh
FBCh	Secure Mask Lock (SEC_GP_REG_LOCK)	32	RW	AAAA_AAAAh
FD0h	Master Secure Level (MASTER_SEC_LEVEL)	32	RW	8000_0000h
FD4h	Master Secure Level (MASTER_SEC_ANTI_POL_REG)	32	RW	BFFF_FFFFh
FECh	Miscellaneous CPU0 Control Signals (CPU0_LOCK_REG)	32	RW	See section
FF0h	Miscellaneous CPU1 Control Signals (CPU1_LOCK_REG)	32	RW	See section
FF8h	Secure Control Duplicate (MISC_CTRL_DP_REG)	32	RW	0000_86A6h
FFCh	Secure Control (MISC_CTRL_REG)	32	RW	0000_86A6h

23.4.1.2 Flash Memory Rule (FLASH00_MEM_RULE0 - FLASH00_MEM_RULE3)

Offset

Register	Offset
FLASH00_MEM_RULE0	10h
FLASH00_MEM_RULE1	14h
FLASH00_MEM_RULE2	18h
FLASH00_MEM_RULE3	1Ch

Function

These 4 registers control access rules for Flash00 slaves.

Table 419. FLASH00_MEM_RULE0

Rule	Address Access
0	0x00000000 – 0x00007FFF
1	0x00008000 – 0x0000FFFF

Table continues on the next page...

Table 419. FLASH00_MEM_RULE0 (continued)

Rule	Address Access
2	0x00010000 – 0x00017FFF
3	0x00018000 – 0x0001FFFF
4	0x00020000 – 0x00027FFF
5	0x00028000 – 0x0002FFFF
6	0x00030000 – 0x00037FFF
7	0x00038000 – 0x0003FFFF

Table 420. FLASH00_MEM_RULE1

Rule	Address Access
0	0x00040000 – 0x0047FFF
1	0x00048000 – 0x0004FFFF
2	0x00050000 – 0x00057FFF
3	0x00058000 – 0x0005FFFF
4	0x00060000 – 0x00067FFF
5	0x00068000 – 0x0006FFFF
6	0x00070000 – 0x00077FFF
7	0x00078000 – 0x0007FFFF

Table 421. FLASH00_MEM_RULE2

Rule	Address Access
0	0x00080000 – 0x00087FFF
1	0x00088000 – 0x0008FFFF
2	0x00090000 – 0x00097FFF
3	0x00098000 – 0x0009FFFF
4	0x000A0000 – 0x000A7FFF
5	0x000A8000 – 0x000AFFFF
6	0x000B0000 – 0x000B7FFF
7	0x000B8000 – 0x000BFFFF

Table 422. FLASH00_MEM_RULE3

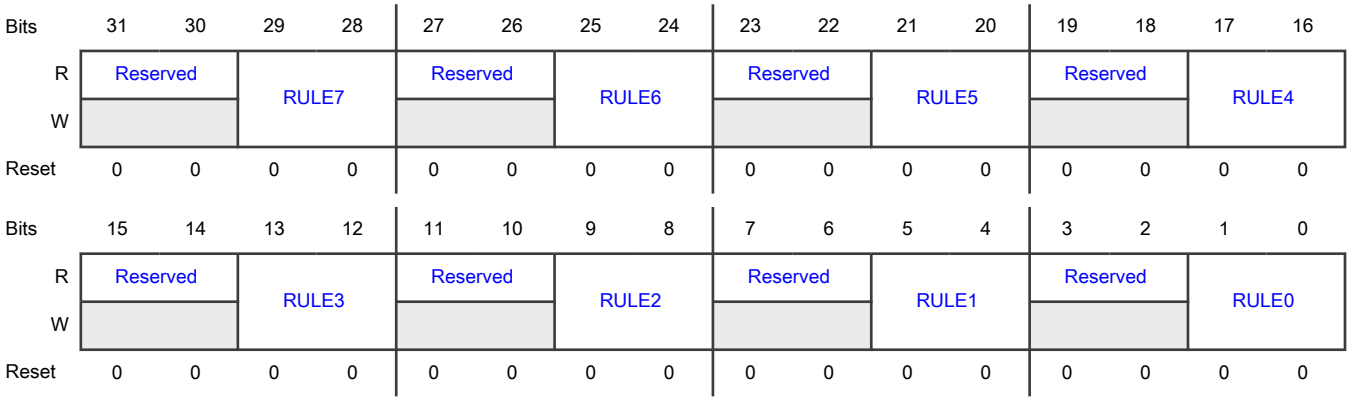
Rule	Address Access
0	0x000C0000 – 0x000C7FFF
1	0x000C8000 – 0x000CFFFF
2	0x000D0000 – 0x000D7FFF

Table continues on the next page...

Table 422. FLASH00_MEM_RULE3 (continued)

Rule	Address Access
3	0x000D8000 – 0x000DFFFF
4	0x000E0000 – 0x000E7FFF
5	0x000E8000 – 0x000EFFFF
6	0x000F0000 – 0x000F7FFF
7	0x000F8000 – 0x000FFFFF

Diagram



Fields

Field	Function
31-30 —	Reserved, write as 1 (or 0b11).
29-28 RULE7	Rule 7 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved, write as 1 (or 0b11).
25-24 RULE6	Rule 6 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved, write as 1 (or 0b11).
21-20 RULE5	Rule 5 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved, write as 1 (or 0b11).
17-16 RULE4	Rule 4 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved, write as 1 (or 0b11).
13-12 RULE3	Rule 3 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved, write as 1 (or 0b11).
9-8 RULE2	Rule 2 Defines the minimal security level to access the ROM memory at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved, write as 1 (or 0b11).
5-4 RULE1	Rule 1 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved, write as 1 (or 0b11).
1-0 RULE0	Rule 0 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.3 Flash Memory Rule (FLASH01_MEM_RULE0 - FLASH01_MEM_RULE3)

Offset

Register	Offset
FLASH01_MEM_RULE0	20h
FLASH01_MEM_RULE1	24h
FLASH01_MEM_RULE2	28h
FLASH01_MEM_RULE3	2Ch

Function

These 4 registers control access rules for Flash01 slaves.

Table 423. FLASH01_MEM_RULE0

Rule	Address Access
0	0x00100000 – 0x00107FFF
1	0x00108000 – 0x0010FFFF
2	0x00110000 – 0x00117FFF
3	0x00118000 – 0x0011FFFF
4	0x00120000 – 0x00127FFF
5	0x00128000 – 0x0012FFFF
6	0x00130000 – 0x00137FFF
7	0x00138000 – 0x0013FFFF

Table 424. FLASH01_MEM_RULE1

Rule	Address Access
0	0x00140000 – 0x00147FFF
1	0x00148000 – 0x0014FFFF
2	0x00150000 – 0x00157FFF
3	0x00158000 – 0x0015FFFF
4	0x00160000 – 0x00167FFF
5	0x00168000 – 0x0016FFFF
6	0x00170000 – 0x00177FFF
7	0x00178000 – 0x0017FFFF

Table 425. FLASH01_MEM_RULE2

Rule	Address Access
0	0x00180000 – 0x00187FFF
1	0x00188000 – 0x0018FFFF
2	0x00190000 – 0x00197FFF
3	0x00198000 – 0x0019FFFF
4	0x001A0000 – 0x001A7FFF
5	0x001A8000 – 0x001AFFFF
6	0x001B0000 – 0x001B7FFF
7	0x001B8000 – 0x001BFFFF

Table 426. FLASH01_MEM_RULE3

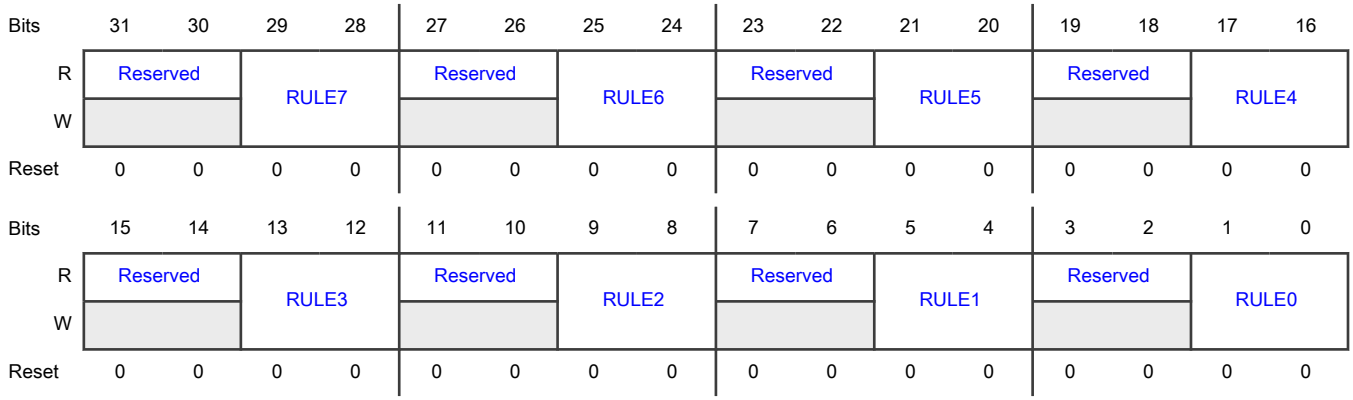
Rule	Address Access
0	0x001C0000 – 0x001C7FFF

Table continues on the next page...

Table 426. FLASH01_MEM_RULE3 (continued)

Rule	Address Access
1	0x001C8000 – 0x001CFFFF
2	0x001D0000 – 0x001D7FFF
3	0x001D8000 – 0x001DFFFF
4	0x001E0000 – 0x001E7FFF
5	0x001E8000 – 0x001EFFFF
6	0x001F0000 – 0x001F7FFF
7	0x001F8000 – 0x001FFFFF

Diagram



Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the ROM memory at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved, write as 1 (or 0b11).
17-16 RULE4	Rule 4 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved, write as 1 (or 0b11).
13-12 RULE3	Rule 3 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved, write as 1 (or 0b11).
9-8	Rule 2

Table continues on the next page...

Table continued from the previous page...

Field	Function
RULE2	Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved, write as 1 (or 0b11).
5-4 RULE1	Rule 1 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved, write as 1 (or 0b11).
1-0 RULE0	Rule 0 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.4 Flash Memory Rule (FLASH02_MEM_RULE)

Offset

Register	Offset
FLASH02_MEM_RULE	30h

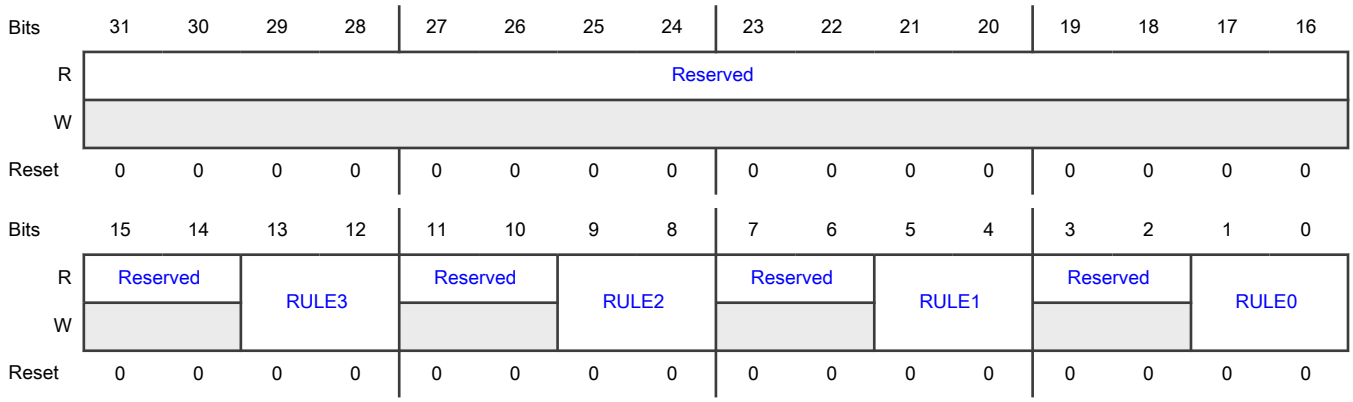
Function

This register controls access rules for Flash02 slaves.

Table 427. FLASH02_MEM_RULE

Rule	Address Access
0	0x01000000 – 0x01003FFF
1	0x01004000 – 0x00107FFF
2	0x01008000 – 0x0100BFFF
3	0x0100C000 – 0x0100FFFF
4	-
5	-
6	-
7	-

Diagram



Fields

Field	Function
31-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8	Rule 2

Table continues on the next page...

Table continued from the previous page...

Field	Function
RULE2	Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.5 Flash Memory Rule (FLASH03_MEM_RULE)

Offset

Register	Offset
FLASH03_MEM_RULE	40h

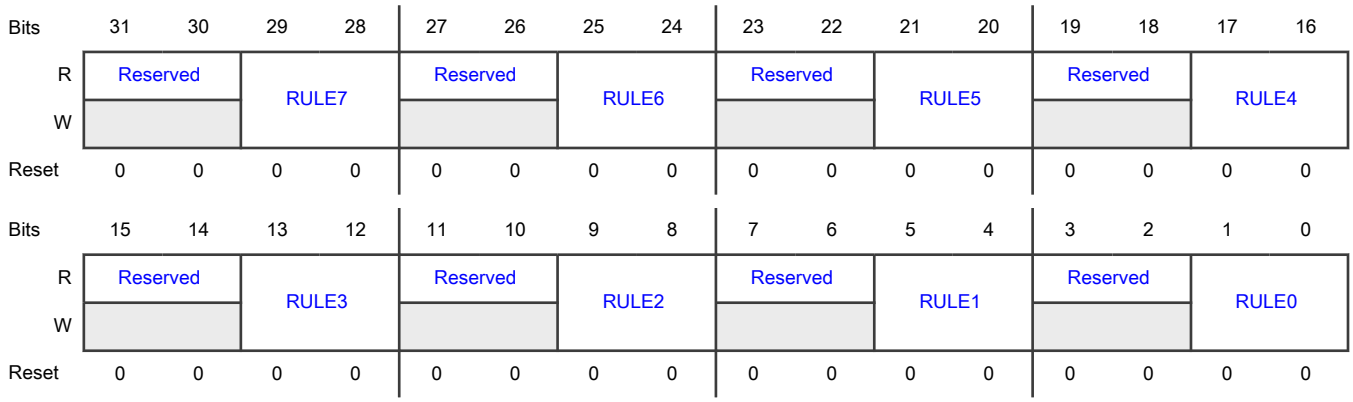
Function

This register controls access rules for Flash02 slaves.

Table 428. FLASH03_MEM_RULE

Rule	Address Access
0	0x01100000 – 0x01100FFF
1	0x01101000 – 0x00111FFF
2	0x01102000 – 0x01102FFF
3	0x01103000 – 0x01103FFF
4	-
5	-
6	-
7	-

Diagram



Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24	Rule 6

Table continues on the next page...

Table continued from the previous page...

Field	Function
RULE6	Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
9-8 RULE2	Rule 2 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.6 ROM Memory Rule (ROM_MEM_RULE0 - ROM_MEM_RULE3)

Offset

Register	Offset
ROM_MEM_RULE0	60h
ROM_MEM_RULE1	64h
ROM_MEM_RULE2	68h
ROM_MEM_RULE3	6Ch

Function

These 4 registers control access rules for ROM slaves.

NOTE

The BOOT ROM code may change the register reset value to 0x00000000 before jumping to user code.

Table 429. ROM_MEM_RULE0

Rule	Address Access
0	0x0300_0000 – 0x0300_1FFF
1	0x0300_2000 – 0x0300_3FFF
2	0x0300_4000 – 0x0300_5FFF
3	0x0300_6000 – 0x0300_7FFF
4	0x0300_8000 – 0x0300_9FFF
5	0x0300_A000 – 0x0300_BFFF
6	0x0300_C000 – 0x0300_DFFF
7	0x0300_E000 – 0x0300_FFFF

Table 430. ROM_MEM_RULE1

Rule	Address Access
0	0x0301_0000 – 0x0301_1FFF
1	0x0301_2000 – 0x0301_3FFF
2	0x0301_4000 – 0x0301_5FFF
3	0x0301_6000 – 0x0301_7FFF
4	0x0301_8000 – 0x0301_9FFF
5	0x0301_A000 – 0x0301_BFFF
6	0x0301_D000 – 0x0301_DFFF
7	0x0301_E000 – 0x0301_FFFF

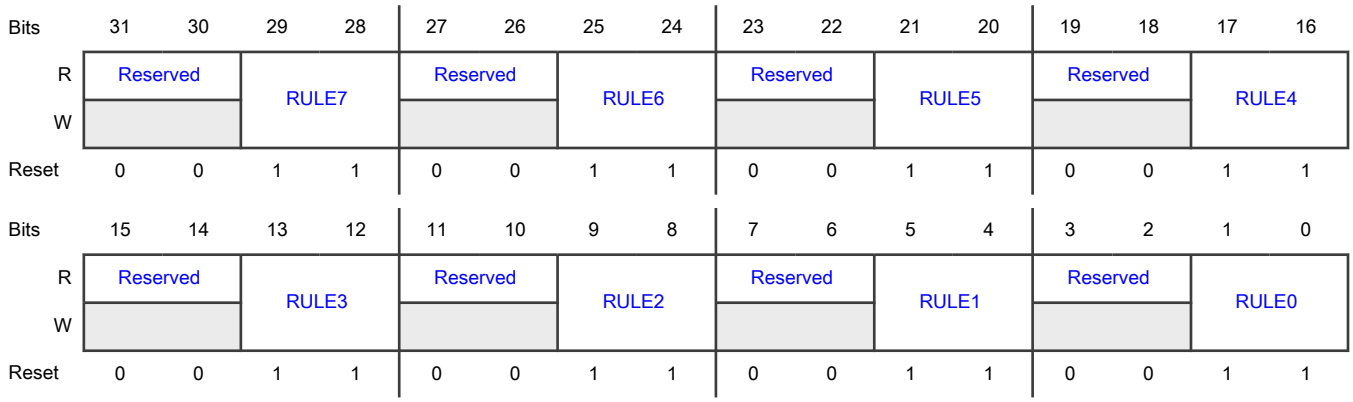
Table 431. ROM_MEM_RULE2

Rule	Address Access
0	0x0302_0000 – 0x0302_1FFF
1	0x0302_2000 – 0x0302_3FFF
2	0x0302_4000 – 0x0302_5FFF
3	0x0302_6000 – 0x0302_7FFF
4	0x0302_8000 – 0x0302_9FFF
5	0x0302_A000 – 0x0302_BFFF
6	0x0302_C000 – 0x0302_DFFF
7	0x0302_E000 – 0x0302_FFFF

Table 432. ROM_MEM_RULE3

Rule	Address Access
0	0x0303_0000 – 0x0303_1FFF
1	0x0303_2000 – 0x0303_3FFF
2	0x0303_4000 – 0x0303_5FFF
3	0x0303_6000 – 0x0303_7FFF
4	0x0303_8000 – 0x0303_9FFF
5	0x0303_A000 – 0x0303_BFFF
6	0x0303_C000 – 0x0303_DFFF
7	0x0303_E000 – 0x0303_FFFF

Diagram



Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24	Rule 6

Table continues on the next page...

Table continued from the previous page...

Field	Function
RULE6	Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
9-8 RULE2	Rule 2 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the ROM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.7 RAMX Memory Rule (RAMX_MEM_RULE0 - RAMX_MEM_RULE2)

Offset

Register	Offset
RAMX_MEM_RULE0	80h
RAMX_MEM_RULE1	84h
RAMX_MEM_RULE2	88h

Function

Table 433. RAMX_MEM_RULE0

Rule	Address Access
0	0x0400_0000 – 0x0400_0FFF
1	0x0400_1000 – 0x0400_1FFF
2	0x0400_2000 – 0x0400_2FFF
3	0x0400_3000 – 0x0400_3FFF
4	0x0400_4000 – 0x0400_4FFF
5	0x0400_5000 – 0x0400_5FFF
6	0x0400_6000 – 0x0400_6FFF
7	0x0400_7000 - 0x0400_7FFF

Table 434. RAMX_MEM_RULE1

Rule	Address Access
0	0x0400_8000 – 0x0400_8FFF
1	0x0400_9000 – 0x0400_9FFF
2	0x0400_A000 – 0x0400_AFFF
3	0x0400_B000 – 0x0400_BFFF
4	0x0400_C000 – 0x0400_CFFF
5	0x0400_D000 – 0x0400_DFFF
6	0x0400_E000 – 0x0400_EFFF
7	0x0400_F000 - 0x0400_FFFF

Table 435. RAMX_MEM_RULE2

Rule	Address Access
0	0x0401_0000 – 0x0401_0FFF
1	0x0401_1000 – 0x0401_1FFF
2	0x0401_2000 – 0x0401_2FFF
3	0x0401_3000 – 0x0401_3FFF
4	0x0401_4000 – 0x0401_4FFF
5	0x0401_5000 – 0x0401_5FFF
6	0x0401_6000 – 0x0401_6FFF
7	0x0401_7000 - 0x0401_7FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the RAM memory at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.8 RAMA Memory Rule 0 (RAMA_MEM_RULE)

Offset

Register	Offset
RAMA_MEM_RULE	A0h

Function

Table 436. RAMA_MEM_RULE0

Rule	Address Access
0	0x20000000 – 0x20000FFF
1	0x20001000 - 0x20001FFF
2	0x20002000 - 0x20002FFF
3	0x20003000 - 0x20003FFF
4	0x20004000 - 0x20004FFF
5	0x20005000 - 0x20005FFF
6	0x20006000 - 0x20006FFF
7	0x20007000 - 0x20007FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the RAM memory at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.9 RAMB Memory Rule (RAMB_MEM_RULE)

Offset

Register	Offset
RAMB_MEM_RULE	C0h

Function

This register controls access to the RAM. Each rule field controls access to a specific range in RAM.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 437. RAMB_MEM_RULE0

Rule	Address Access
0	0x20008000 – 0x20008FFF
1	0x20009000 - 0x20009FFF
2	0x2000A000 - 0x2000AFFF
3	0x2000B000 - 0x2000BFFF
4	0x2000C000 - 0x2000CFFF
5	0x2000D000 - 0x2000DFFF
6	0x2000E000 - 0x2000EFFF
7	0x2000F000 - 0x2000FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the RAM memory at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the RAM memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.10 RAMC Memory Rule (RAMC_MEM_RULE0 - RAMC_MEM_RULE1)

Offset

Register	Offset
RAMC_MEM_RULE0	E0h
RAMC_MEM_RULE1	E4h

Function

This register controls access to the RAM. Each rule field controls access to a specific range in RAM.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 438. RAMC_MEM_RULE0

Rule	Address Access
0	0x20010000 – 0x20010FFF
1	0x20011000 - 0x20001FFF
2	0x20012000 - 0x20012FFF
3	0x20013000 - 0x20013FFF
4	0x20014000 - 0x20014FFF
5	0x20015000 - 0x20015FFF

Table continues on the next page...

Table 438. RAMC_MEM_RULE0 (continued)

Rule	Address Access
6	0x20016000 - 0x20016FFF
7	0x20017000 - 0x20018FFF

Table 439. RAMC_MEM_RULE1

Rule	Address Access
0	0x20018000 – 0x20018FFF
1	0x20019000 - 0x20009FFF
2	0x2001A000 - 0x2001AFFF
3	0x2001B000 - 0x2001BFFF
4	0x2001C000 - 0x2001CFFF
5	0x2001D000 - 0x2001DFFF
6	0x2001E000 - 0x2001EFFF
7	0x2001F000 - 0x2001FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the RAM at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.11 RAMD Memory Rule (RAMD_MEM_RULE0 - RAMD_MEM_RULE1)

Offset

Register	Offset
RAMD_MEM_RULE0	100h
RAMD_MEM_RULE1	104h

Function

This register controls access to the RAM. Each rule field controls access to a specific range in RAM.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 440. RAMD_MEM_RULE0

Rule	Address Access
0	0x20020000 – 0x20020FFF
1	0x20021000 - 0x20021FFF
2	0x20022000 - 0x20022FFF
3	0x20023000 - 0x20023FFF
4	0x20024000 - 0x20024FFF
5	0x20025000 - 0x20025FFF
6	0x20026000 - 0x20026FFF
7	0x20027000 - 0x20028FFF

Table 441. RAMD_MEM_RULE1

Rule	Address Access
0	0x20028000 – 0x20028FFF
1	0x20029000 - 0x20009FFF
2	0x2002A000 - 0x2002AFFF
3	0x2002B000 - 0x2002BFFF
4	0x2002C000 - 0x2002CFFF
5	0x2002D000 - 0x2002DFFF
6	0x2002E000 - 0x2002EFFF
7	0x2002F000 - 0x2002FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the RAM at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.12 RAME Memory Rule (RAME_MEM_RULE0 - RAME_MEM_RULE1)

Offset

Register	Offset
RAME_MEM_RULE0	120h
RAME_MEM_RULE1	124h

Function

This register controls access to the RAM. Each rule field controls access to a specific range in RAM.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 442. RAME_MEM_RULE0

Rule	Address Access
0	0x20030000 – 0x20030FFF
1	0x20031000 - 0x20031FFF
2	0x20032000 - 0x20032FFF
3	0x20033000 - 0x20033FFF
4	0x20034000 - 0x20034FFF
5	0x20035000 - 0x20035FFF

Table continues on the next page...

Table 442. RAME_MEM_RULE0 (continued)

Rule	Address Access
6	0x20036000 - 0x20036FFF
7	0x20037000 - 0x20038FFF

Table 443. RAME_MEM_RULE1

Rule	Address Access
0	0x20038000 – 0x20038FFF
1	0x20039000 - 0x20009FFF
2	0x2003A000 - 0x2003AFFF
3	0x2003B000 - 0x2003BFFF
4	0x2003C000 - 0x2003CFFF
5	0x2003D000 - 0x2003DFFF
6	0x2003E000 - 0x2003EFFF
7	0x2003F000 - 0x2003FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the RAM at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.13 RAMF Memory Rule (RAMF_MEM_RULE0 - RAMF_MEM_RULE1)

Offset

Register	Offset
RAMF_MEM_RULE0	140h
RAMF_MEM_RULE1	144h

Function

This register controls access to the RAM. Each rule field controls access to a specific range in RAM.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 444. RAMF_MEM_RULE0

Rule	Address Access
0	0x20040000 – 0x20040FFF
1	0x20041000 - 0x20041FFF
2	0x20042000 - 0x20042FFF
3	0x20043000 - 0x20043FFF
4	0x20044000 - 0x20044FFF
5	0x20045000 - 0x20045FFF
6	0x20046000 - 0x20046FFF
7	0x20047000 - 0x20048FFF

Table 445. RAMF_MEM_RULE1

Rule	Address Access
0	0x20048000 – 0x20048FFF
1	0x20049000 - 0x20009FFF
2	0x2004A000 - 0x2004AFFF
3	0x2004B000 - 0x2004BFFF
4	0x2004C000 - 0x2004CFFF
5	0x2004D000 - 0x2004DFFF
6	0x2004E000 - 0x2004EFFF
7	0x2004F000 - 0x2004FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the RAM at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.14 RAMG Memory Rule (RAMG_MEM_RULE0 - RAMG_MEM_RULE1)

Offset

Register	Offset
RAMG_MEM_RULE0	160h
RAMG_MEM_RULE1	164h

Function

This register controls access to the RAM. Each rule field controls access to a specific range in RAM.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 446. RAMG_MEM_RULE0

Rule	Address Access
0	0x20050000 – 0x20050FFF
1	0x20051000 - 0x20051FFF
2	0x20052000 - 0x20052FFF
3	0x20053000 - 0x20053FFF
4	0x20054000 - 0x20054FFF
5	0x20055000 - 0x20055FFF

Table continues on the next page...

Table 446. RAMG_MEM_RULE0 (continued)

Rule	Address Access
6	0x20056000 - 0x20056FFF
7	0x20057000 - 0x20058FFF

Table 447. RAMG_MEM_RULE1

Rule	Address Access
0	0x20058000 – 0x20058FFF
1	0x20059000 - 0x20009FFF
2	0x2005A000 - 0x2005AFFF
3	0x2005B000 - 0x2005BFFF
4	0x2005C000 - 0x2005CFFF
5	0x2005D000 - 0x2005DFFF
6	0x2005E000 - 0x2005EFFF
7	0x2005F000 - 0x2005FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the RAM at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.15 RAMH Memory Rule (RAMH_MEM_RULE)

Offset

Register	Offset
RAMH_MEM_RULE	180h

Function

This register controls access to the RAM. Each rule field controls access to a specific range in RAM.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 448. RAMH_MEM_RULE

Rule	Address Access
0	0x20060000 – 0x20060FFF
1	0x20061000 - 0x20061FFF
2	0x20062000 - 0x20062FFF
3	0x20063000 - 0x20063FFF
4	0x20064000 - 0x20064FFF
5	0x20065000 - 0x20065FFF
6	0x20066000 - 0x20066FFF
7	0x20067000 - 0x20067FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
29-28 RULE7	Rule 7 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the RAM at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	11b - Secure and privilege user access allowed

23.4.1.16 APB Bridge Group 0 Memory Rule 0 (APB_PERIPHERAL_GROUP0_MEM_RULE0)

Offset

Register	Offset
APB_PERIPHERAL_GROUP0_MEM_RULE0	1A0h

Function

This register controls access to the APB Bridge Group 0. Each rule field controls access to a specific range in APB Bridge Group 0.

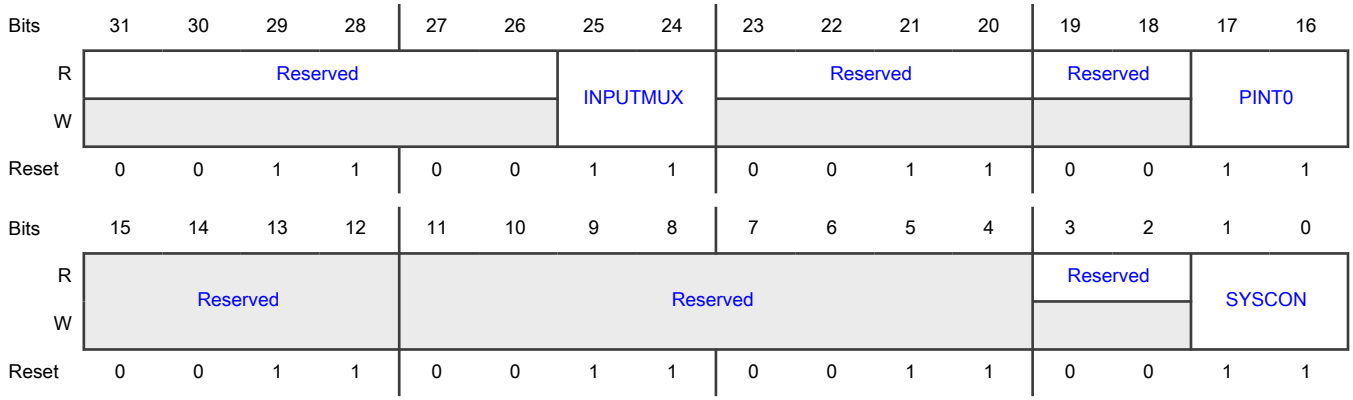
NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 449. APB Bridge Group 0 Rule 0

Rule	Address Access
SYSCON	0x4000 0000 – 0x4000 0FFF
Reserved	0x4000 1000 – 0x4000 1FFF
Reserved	0x4000 2000 – 0x4000 2FFF
PINT	0x4000 4000 – 0x4000 4FFF
Reserved	0x4000 5000 – 0x4000 5FFF
INPUTMUX	0x4000 6000 – 0x4000 6FFF
Reserved	0x4000 7000 – 0x4000 7FFF

Diagram



Fields

Field	Function
31-26 —	Reserved, write as 1 (or 0b11).
25-24 INPUTMUX	INPUTMUX Defines the minimal security level to access INPUTMUX. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-20 —	Reserved, write as 1 (or 0b11).
19-18 —	Reserved, write as 1 (or 0b11).
17-16 PINT0	PINT0 Defines the minimal security level to access PINT0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-12 —	Reserved, write as 1 (or 0b11).
11-4 —	Reserved, write as 1 (or 0b11).
3-2 —	Reserved, write as 1 (or 0b11).
1-0 SYSCON	SYSCON Defines the minimal security level to access SYSCON. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.17 APB Bridge Group 0 Memory Rule 1 (APB_PERIPHERAL_GROUP0_MEM_RULE1)

Offset

Register	Offset
APB_PERIPHERAL_GROUP0_MEM_RULE1	1A4h

Function

This register controls access to the APB Bridge Peripheral 0. Each rule field controls access to a specific range in APB Bridge Peripheral 0.

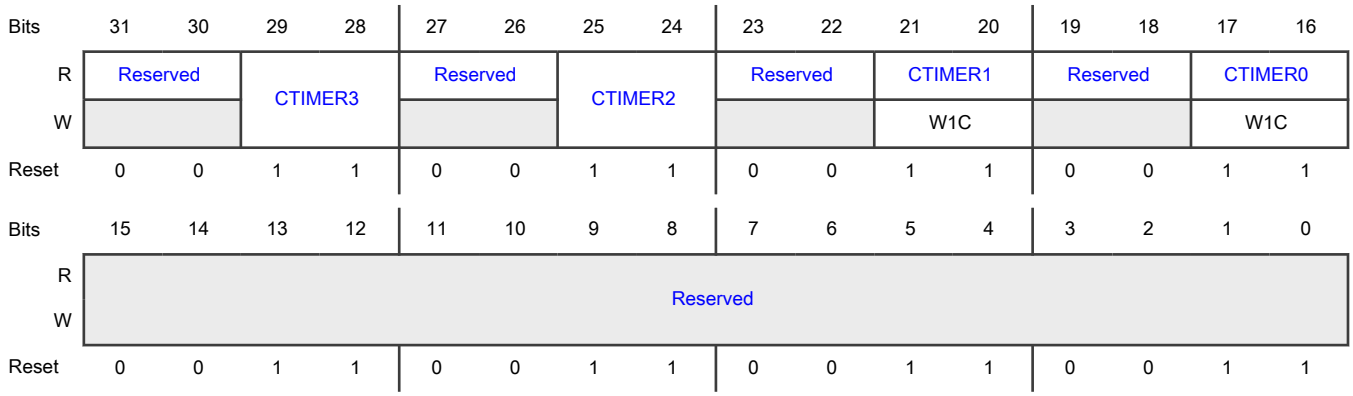
NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 450. APB_PERIPHERAL_GROUP0_MEM_RULE1

Rule	Address Access
Reserved	0x4000 8000 – 0x4000 8FFF
Reserved	0x4000 9000 – 0x4000 9FFF
Reserved	0x4000 A000 – 0x4000 AFFF
Reserved	0x4000 B000 – 0x4000 BFFF
CTIMER0	0x4000 C000 – 0x4000 CFFF
CTIMER1	0x4000 D000 – 0x4000 DFFF
CTIMER2	0x4000 E000 – 0x4000 EFFF
CTIMER3	0x4000 F000 – 0x4000 FFFF

Diagram



Fields

Field	Function
31-30 —	Reserved, write as 1 (or 0b11).
29-28 CTIMER3	CTIMER3 Defines the minimal security level to access CTIMER3. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved, write as 1 (or 0b11).
25-24 CTIMER2	CTIMER2 Defines the minimal security level to access CTIMER2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved, write as 1 (or 0b11).
21-20 CTIMER1	CTIMER1 Defines the minimal security level to access the CTIMER1 memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved, write as 1 (or 0b11).
17-16 CTIMER0	CTIMER0 Defines the minimal security level to access the CTIMER0 memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	11b - Secure and privilege user access allowed
15-0 —	Reserved

23.4.1.18 APB Bridge Group 0 Rule 2 (APB_PERIPHERAL_GROUP0_MEM_RULE2)

Offset

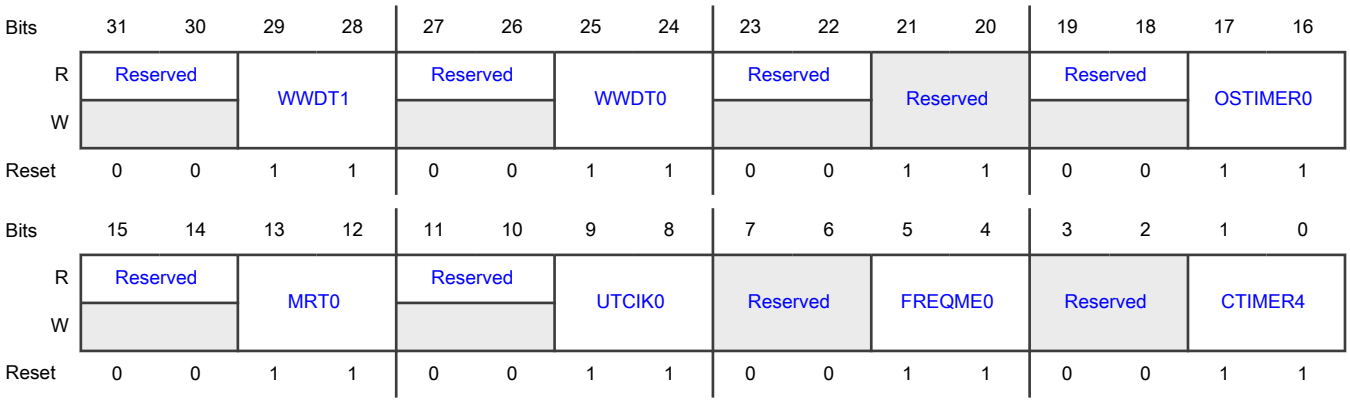
Register	Offset
APB_PERIPHERAL_GROUP0_MEM_RULE2	1A8h

Function

Table 451. APB_PERIPHERAL_GROUP0_MEM_RULE2

Rule	Address Access
CTIMER4	0x40010000 – 0x40010FFF
FREQME	0x40011000 – 0x40011FFF
UTICK0	0x40012000 – 0x40012FFF
MRT0	0x40013000 – 0x40013FFF
OSTIMER0	0x40014000 – 0x40014FFF
Reserved	0x40015000 – 0x40015FFF
WWDT0	0x40016000 – 0x40016FFF
WWDT1	0x40017000 – 0x40017FFF

Diagram



Fields

Field	Function
31-30 —	Reserved, write as 1 (or 0b11).
29-28 WWDT1	WWDT1 Defines the minimal security level to access WWDT1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved, write as 1 (or 0b11).
25-24 WWDT0	WWDT0 Defines the minimal security level to access WWDT0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved, write as 1 (or 0b11).
21-20 —	Reserved, write as 1 (or 0b11).
19-18 —	Reserved, write as 1 (or 0b11).
17-16 OSTIMER0	OSTIMER0 Defines the minimal security level to access OSTIMER0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved, write as 1 (or 0b11).
13-12	MRT0

Table continues on the next page...

Table continued from the previous page...

Field	Function
MRT0	Defines the minimal security level to access MRT0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved, write as 1 (or 0b11).
9-8 UTCIK0	UTCIK0 Defines the minimal security level to access PROBE_IS (SYNC). 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved, write as 1 (or 0b11).
5-4 FREQME0	FREQME0 Defines the minimal security level to access the FREQME0 memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved
1-0 CTIMER4	CTIMER4 Defines the minimal security level to access the Coolflux memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.19 APB Bridge Group 0 Memory Rule 3 (APB_PERIPHERAL_GROUP0_MEM_RULE3)

Offset

Register	Offset
APB_PERIPHERAL_GROUP0_MEM_RULE3	1ACh

Function

This register controls access to the APB Bridge Group 0. Each rule field controls access to a specific range in APB Bridge Peripheral 0.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 452. APB_PERIPHERAL_GROUP0_MEM_RULE3

Rule	Address Access
Reserved	0x4001 8000 – 0x4001 8FFF
Reserved	0x4001 9000 – 0x4001 9FFF
Reserved	0x4001 A000 – 0x4001 AFFF
CACHE64 POLSEL0	0x4001 B000 – 0x4001 BFFF
Reserved	0x4001 C000 – 0x4001 CFFF
Reserved	0x4001 D000 – 0x4001 DFFF
Reserved	0x4001 E000 – 0x4001 EFFF
Reserved	0x4001 F000 – 0x4001 FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		Reserved		Reserved		Reserved		Reserved				Reserved		Reserved	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		CACHE64_POLSEL0		Reserved											
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved, write as 1 (or 0b11).
29-28 —	Reserved, write as 1 (or 0b11).
27-26 —	Reserved, write as 1 (or 0b11).
25-24 —	Reserved, write as 1 (or 0b11).
23-20 —	Reserved, write as 1 (or 0b11).
19-18 —	Reserved, write as 1 (or 0b11).
17-16 —	Reserved, write as 1 (or 0b11).
15-14 —	Reserved, write as 1 (or 0b11).
13-12 CACHE64_POLSEL0	CACHE64_POLSEL0 Defines the minimal security level to access the CACHE64 POLSEL memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-0 —	Reserved

23.4.1.20 APB Bridge Group 1 Memory Rule 0 (APB_PERIPHERAL_GROUP1_MEM_RULE0)

Offset

Register	Offset
APB_PERIPHERAL_GROUP1_MEM_RULE0	1B0h

Function

This register controls access to the APB Bridge Group 1. Each rule field controls access to a specific range in APB Bridge Group 1.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 453. APB_PERIPHERAL_GROUP1_RULE0

Rule	Address Access
I3C0	0x4002 1000 – 0x4002 1FFF
I3C1	0x4002 2000 – 0x4002 2FFF
Reserved	0x4002 3000 – 0x4002 3FFF
Reserved	0x4002 4000 – 0x4002 4FFF
Digital glitch detector	0x4002 5000 – 0x4002 5FFF
ITRC	0x4002 6000 – 0x4002 6FFF
Reserved	0x4002 7000 – 0x4002 7FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved				Reserved		ITRC		Reserved		GDET		Reserved			
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved				Reserved		I3C1		Reserved		I3C0		Reserved		Reserved	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-28 —	Reserved, write as 1 (or 0b11).
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 ITRC	ITRC Defines the minimal security level to access ITRC. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved, write as 1 (or 0b11).
21-20 GDET	GDET Defines the minimal security level to access GDET. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-12 —	Reserved, write as 1 (or 0b11).
11-10 —	Reserved, write as 1 (or 0b11).
9-8 I3C1	I3C1 Defines the minimal security level to access I3C1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved, write as 1 (or 0b11).
5-4 I3C0	I3C0 Defines the minimal security level to access I3C0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
1-0 —	Reserved, write as 1 (or 0b11).

23.4.1.21 APB Bridge Group 1 Memory Rule 1 (APB_PERIPHERAL_GROUP1_MEM_RULE1)

Offset

Register	Offset
APB_PERIPHERAL_GROUP1_MEM_RULE1	1B4h

Function

This register controls access to the APB Bridge Group 1. Each rule field controls access to a specific range in APB Bridge Group 1.

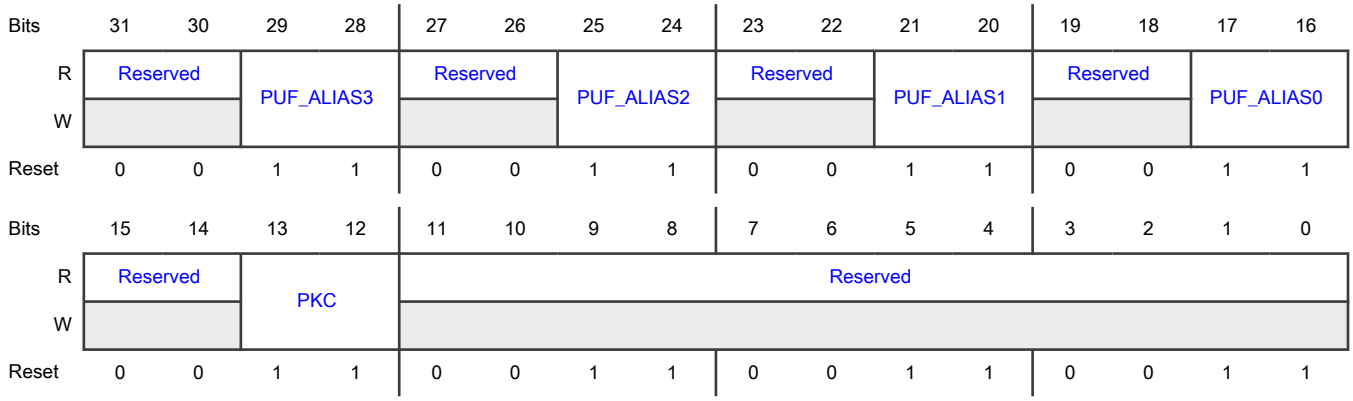
NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 454. APB_PERIPHERAL_GROUP1_RULE1

Rule	Address Access
Reserved	0x4002 8000 – 0x4002 8FFF
Reserved	0x4002 9000 – 0x4002 9FFF
Reserved	0x4002 A000 – 0x4002 AFFF
PKC	0x4002 B000 – 0x4002 BFFF
PUF_ALIAS0	0x4002 C000 – 0x4002 CFFF
PUF_ALIAS1	0x4002 D000 – 0x4002 DFFF
PUF_ALIAS2	0x4002 E000 – 0x4002 EFFF
PUF_ALIAS3	0x4002 F000 – 0x4002 FFFF

Diagram



Fields

Field	Function
31-30 —	Reserved, write as 1 (or 0b11).
29-28 PUF_ALIAS3	PUF_ALIAS3 Defines the minimal security level to access PUF_ALIAS3. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved, write as 1 (or 0b11).
25-24 PUF_ALIAS2	PUF_ALIAS2 Defines the minimal security level to access PUF_ALIAS2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved, write as 1 (or 0b11).
21-20 PUF_ALIAS1	PUF_ALIAS1 Defines the minimal security level to access PUF_ALIAS1. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved, write as 1 (or 0b11).
17-16 PUF_ALIAS0	PUF_ALIAS0 Defines the minimal security level to access PUF. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved, write as 1 (or 0b11).
13-12 PKC	PKC Defines the minimal security level to access PKC. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-0 —	Reserved, write as 1 (or 0b11).

23.4.1.22 APB Bridge Group 1 Memory Rule 2 (APB_PERIPHERAL_GROUP1_MEM_RULE2)

Offset

Register	Offset
APB_PERIPHERAL_GROUP1_MEM_RULE2	1BCh

Function

This register controls access to the APB Bridge Group 1. Each rule field controls access to a specific range in APB Bridge Group 1.

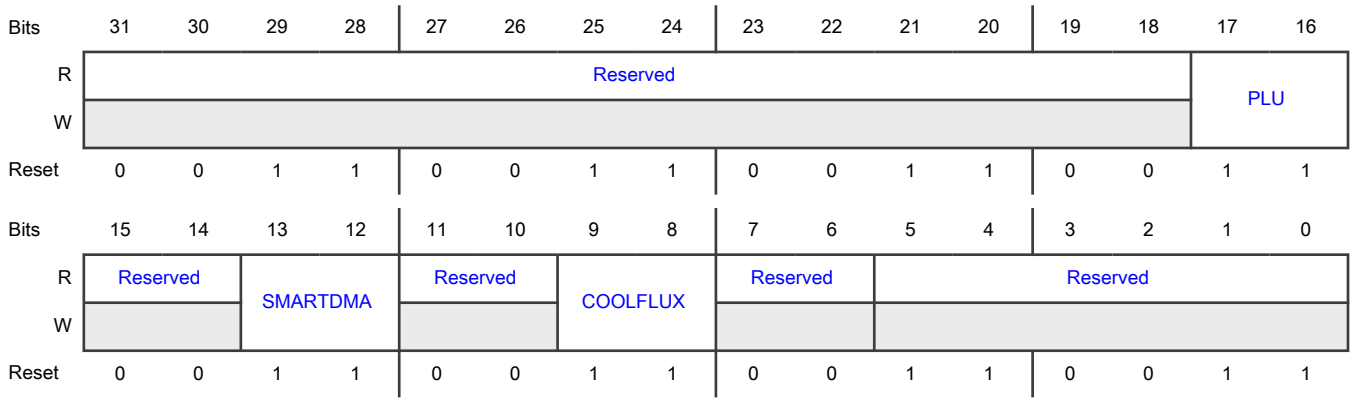
NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 455. APB_PERIPHERAL_GROUP1_RULE2

Rule	Address Access
Reserved	0x4003 0000 – 0x4003 0FFF
Reserved	0x4003 1000 – 0x4003 1FFF
COOLFLUX	0x4003 2000 – 0x4003 2FFF
SmartDMA	0x4003 3000 – 0x4003 3FFF
PLU	0x4003 4000 – 0x4003 4FFF
Reserved	0x4003 5000 – 0x4003 5FFF
Reserved	0x4003 6000 – 0x4003 6FFF
Reserved	0x4003 7000 – 0x4003 7FFF

Diagram



Fields

Field	Function
31-18 —	Reserved, write as 1 (or 0b11).
17-16 PLU	PLU Defines the minimal security level to access PLU. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved, write as 1 (or 0b11).
13-12	SmartDMA

Table continues on the next page...

Table continued from the previous page...

Field	Function
SMARTDMA	Defines the minimal security level to access SmartDMA. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved, write as 1 (or 0b11).
9-8 COOLFLUX	COOLFLUX Defines the minimal security level to access COOLFLUX. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved, write as 1 (or 0b11).
5-0 —	Reserved, write as 1 (or 0b11).

23.4.1.23 AIPS Bridge Group 0 Memory Rule 0 (AIPS_BRIDGE_GROUP0_MEM_RULE0)

Offset

Register	Offset
AIPS_BRIDGE_GROUP0_MEM_RULE0	1C0h

Function

This register controls access to the AIPS Bridge Peripheral 0. Each rule field controls access to a specific range in AIPS Bridge Peripheral 0.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 456. AIPS_BRIDGE_GROUP0_RULE0

Rule	Address Access
GPIO_ALIAS0	0x4004 0000 – 0x4004 0FFF
GPIO_ALIAS1	0x4004 1000 – 0x4004 1FFF
PORT5	0x4004 2000 – 0x4004 2FFF
FMU0	0x4004 3000 – 0x4004 3FFF
SCG0	0x4004 4000 – 0x4004 4FFF
SPC0	0x4004 5000 – 0x4004 5FFF
WUU0	0x4004 6000 – 0x4004 6FFF
TRO0	0x4004 7000 – 0x4004 7FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		TRO0		Reserved		WUU0		Reserved		SPC0		Reserved		SCG0	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		FMU0		Reserved		PORT5		Reserved		GPIO5_ALIAS1		Reserved		GPIO5_ALIAS0	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 TRO0	TRO0 Defines the minimal security level to access TRO0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24	WUU0

Table continues on the next page...

Table continued from the previous page...

Field	Function
WUU0	Defines the minimal security level to access WUU0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 SPC0	SPC0 Defines the minimal security level to access the SPC0 Controller. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 SCG0	SCG0 Defines the minimal security level to access the SCG0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 FMU0	FMU0 Defines the minimal security level to access the FMU0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
9-8 PORT5	PORT5 Defines the minimal security level to access PORT5. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 GPIO5_ALIAS1	GPIO5_ALIAS2 Defines the minimal security level to access GPIO5_ALIAS1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 GPIO5_ALIAS0	GPIO5_ALIAS0 Defines the minimal security level to access GPIO5. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.24 AIPS Bridge Group 0 Memory Rule 1 (AIPS_BRIDGE_GROUP0_MEM_RULE1)

Offset

Register	Offset
AIPS_BRIDGE_GROUP0_MEM_RULE1	1C4h

Function

This register controls access to the AIPS Bridge Peripheral 0. Each rule field controls access to a specific range in AIPS Bridge Peripheral 0.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 457. AIPS_BRIDGE_GROUP0_PER_RULE1

Rule	Address Access
Reserved	0x4004 8000 – 0x4004 8FFF
Reserved	0x4004 9000 – 0x4004 9FFF
LPTMR0	0x4004 A000 – 0x4004 AFFF
LPTMR1	0x4004 B000 – 0x4004 BFFF
RTC0	0x4004 C000 – 0x4004 CFFF
Reserved	0x4004 D000 – 0x4004 DFFF
FMU_TEST	0x4004 E000 – 0x4004 EFFF
Reserved	0x4004 F000 – 0x4004 FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		Reserved		Reserved		FMU_TEST		Reserved				Reserved		RTC	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		LPTMR1		Reserved		LPTMR0		Reserved							
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 —	Reserved
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 FMU_TEST	FMU_TEST Defines the minimal security level to access FMU_TEST.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-20 —	Reserved; the value read from a reserved bit is not defined.
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RTC	RTC Defines the minimal security level to access the RTC. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 LPTMR1	LPTMR1 Defines the minimal security level to access LPTMR1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 LPTMR0	LPTMR0 Defines the minimal security level to access LPTMR0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-0	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	

23.4.1.25 AIPS Bridge Group 0 Memory Rule 2 (AIPS_BRIDGE_GROUP0_MEM_RULE2)

Offset

Register	Offset
AIPS_BRIDGE_GROUP0_MEM_RULE2	1C8h

Function

This register controls access to the AIPS Bridge Peripheral 0. Each rule field controls access to a specific range in AIPS Bridge Peripheral 0.

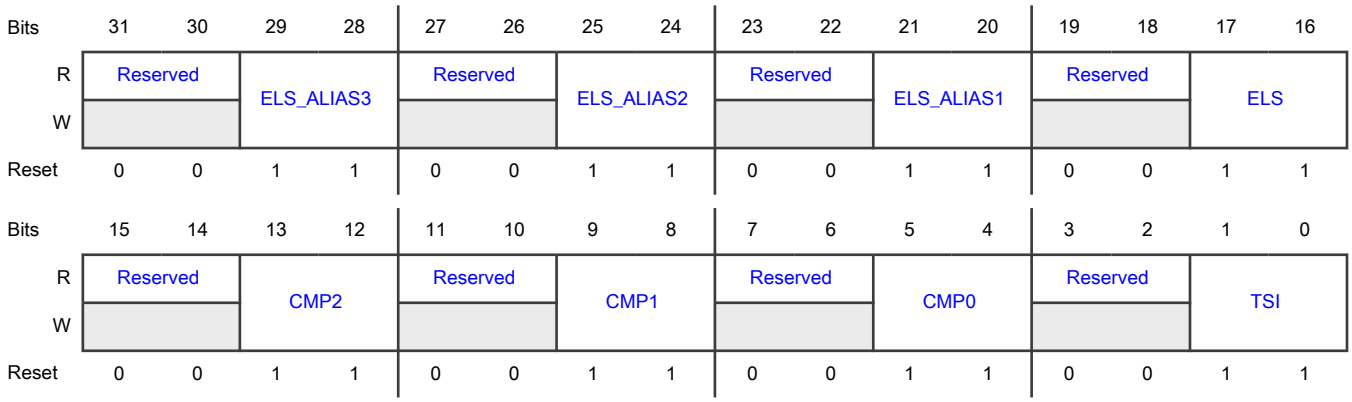
NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 458. AIPS_BRIDGE_GROUP0_RULE2

Rule	Address Access
TSI	0x4005 0000 – 0x4005 0FFF
CMP0	0x4005 1000 – 0x4005 1FFF
CMP1	0x4005 2000 – 0x4005 2FFF
CMP2	0x4005 3000 – 0x4005 3FFF
ELS	0x4005 4000 – 0x4005 4FFF
ELS_ALIAS1	0x4005 5000 – 0x4005 5FFF
ELS_ALIAS2	0x4005 6000 – 0x4005 6FFF
ELS_ALIAS3	0x4005 7000 – 0x4005 7FFF

Diagram



Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 ELS_ALIAS3	ELS_ALIAS3 Defines the minimal security level to access ELS_ALIAS3. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 ELS_ALIAS2	ELS_ALIAS2 Defines the minimal security level to access ELS_ALIAS2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 ELS_ALIAS1	ELS_ALIAS1 Defines the minimal security level to access the ELS_ALIAS1 Controller. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 ELS	ELS Defines the minimal security level to access the ELS. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 CMP2	CMP2 Defines the minimal security level to access the CMP2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 CMP1	CMP1 Defines the minimal security level to access CMP1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 CMP0	CMP0 Defines the minimal security level to access CMP0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 TSI	TSI Defines the minimal security level to access TSI. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.26 AIPS Bridge Group 0 Memory Rule 3 (AIPS_BRIDGE_GROUP0_MEM_RULE3)

Offset

Register	Offset
AIPS_BRIDGE_GROUP0_MEM_RULE3	1CCh

Function

This register controls access to the AIPS Bridge Peripheral 0. Each rule field controls access to a specific range in AIPS Bridge Peripheral 0.

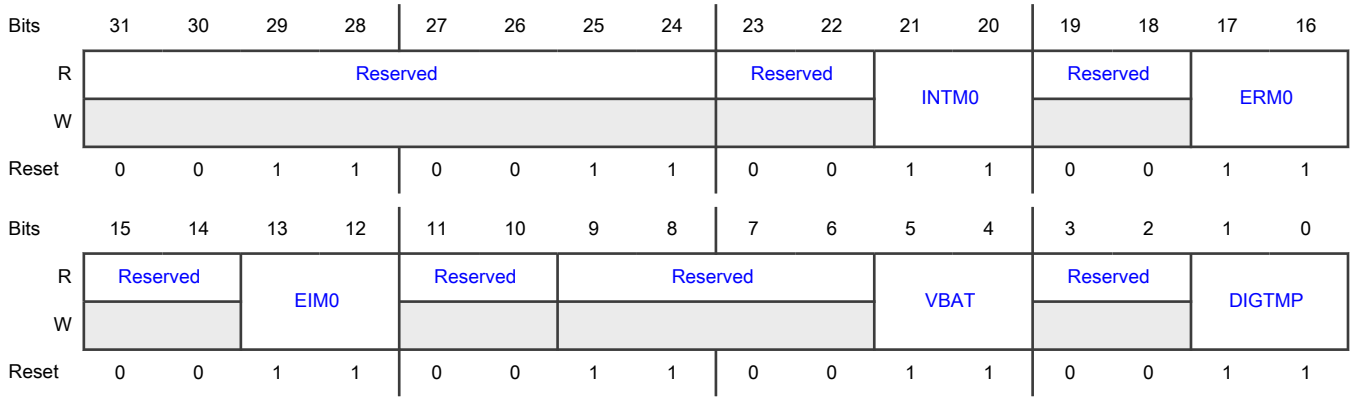
NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 459. AIPS_BRIDGE_GROUP0_RULE3

Rule	Address Access
DIGTMP	0x4005 8000 – 0x4005 8FFF
VBAT	0x4005 9000 – 0x4005 9FFF
Reserved	0x4005 A000 – 0x4005 AFFF
EIM0	0x4005 B000 – 0x4005 BFFF
ERM0	0x4005 C000 – 0x4005 CFFF
INTM0	0x4005 D000 – 0x4005 DFFF
Reserved	0x4005 E000 – 0x4005 EFFF
Reserved	0x4005 F000 – 0x4005 FFFF

Diagram



Fields

Field	Function
31-24 —	Reserved; the value read from a reserved bit is not defined.
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 INTM0	INTM0 Defines the minimal security level to access the INTM0 Controller. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 ERM0	ERM0 Defines the minimal security level to access the ERM0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12	EIM0

Table continues on the next page...

Table continued from the previous page...

Field	Function
EIM0	Defines the minimal security level to access the EIM0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 VBAT	VBAT Defines the minimal security level to access VBAT. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 DIGTMP	DIGTMP Defines the minimal security level to access DIGTMP. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.27 AHB Peripheral 0 Slave Port 12 Slave Rule 0 (AHB_PERIPHERAL0_SLAVE_PORT_P12_SLAVE_RULE0)

Offset

Register	Offset
AHB_PERIPHERAL0_SLAVE_PORT_P12_SLAVE_RULE0	1D0h

Function

This register controls access to the AHB Peripheral 0 Slave. Each rule field controls access to a specific peripheral in AHB Peripheral 0.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 460. AHB_PERIPHERAL0_MEM_RULE_RULE0

Rule	Address Access
eDMA0_15	40090000 - 40090FFF
SCT0	40091000 - 40091FFF
LP_FLEXCOMM0	40092000 - 40092FFF
LP_FLEXCOMM1	40093000 - 40093FFF
LP_FLEXCOMM2	40094000 - 40094FFF
LP_FLEXCOMM3	40095000 - 40095FFF
GPIO0_ALIAS0	40096000 - 40096FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		GPIO0_ALIAS0		Reserved		LP_FLEXCOMM3		Reserved		LP_FLEXCOMM2		Reserved		LP_FLEXCOMM1	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		LP_FLEXCOMM0		Reserved		SCT0		Reserved		eDMA0_CH15		Reserved			
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved, write as 1 (or 0b11).
29-28 GPIO0_ALIAS0	GPIO0_ALIAS0 Defines the minimal security level to access GPIO0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
27-26 —	Reserved, write as 1 (or 0b11).
25-24 LP_FLEXCOMM3	LP_FLEXCOMM3 Defines the minimal security level to access LP_FLEXCOMM3. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved, write as 1 (or 0b11).
21-20 LP_FLEXCOMM2	LP_FLEXCOMM2 Defines the minimal security level to access LP_FLEXCOMM2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved, write as 1 (or 0b11).
17-16 LP_FLEXCOMM1	LP_FLEXCOMM1 Defines the minimal security level to access LP_FLEXCOMM1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved, write as 1 (or 0b11).
13-12 LP_FLEXCOMM0	LP_FLEXCOMM0 Defines the minimal security level to access LP_FLEXCOMM0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	11b - Secure and privilege user access allowed
11-10 —	Reserved, write as 1 (or 0b11).
9-8 SCT0	SCT0 Defines the minimal security level to access SCT0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved, write as 1 (or 0b11).
5-4 eDMA0_CH15	eDMA0_CH15 Defines the minimal security level to access eDMA0_CH15. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-0 —	Reserved, write as 1 (or 0b11).

23.4.1.28 AHB Peripheral 0 Slave Port 12 Slave Rule 1 (AHB_PERIPHERAL0_SLAVE_PORT_P12_SLAVE_RULE1)

Offset

Register	Offset
AHB_PERIPHERAL0_SLAVE_PORT_P12_SLAVE_RULE1	1D4h

Function

This register controls access to the AHB Peripheral 0 Slave. Each rule field controls access to a specific peripheral in AHB Peripheral 0.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 461. AHB_PERIPHERAL0_MEM_RULE_RULE0

Rule	Address Access
GPIO0_ALIAS1	40097000 - 40097FFF
GPIO1_ALIAS0	40098000 - 40098FFF
GPIO1_ALIAS1	40099000 - 40099FFF
GPIO2_ALIAS0	4009A000 - 4009AFFF
GPIO2_ALIAS1	4009B000 - 4009BFFF
GPIO3_ALIAS0	4009C000 - 4009CFFF
GPIO3_ALIAS1	4009D000 - 4009DFFF
GPIO4_ALIAS0	4009E000 - 4009EFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		GPIO4_ALIAS0		Reserved		GPIO3_ALIAS1		Reserved		GPIO3_ALIAS0		Reserved		GPIO2_ALIAS1	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		GPIO2_ALIAS0		Reserved		GPIO1_ALIAS1		Reserved		GPIO1_ALIAS0		Reserved		GPIO0_ALIAS1	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved, write as 1 (or 0b11).
29-28 GPIO4_ALIAS0	GPIO4_ALIAS0 Defines the minimal security level to access GPIO4. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved, write as 1 (or 0b11).
25-24	GPIO3_ALIAS1

Table continues on the next page...

Table continued from the previous page...

Field	Function
GPIO3_ALIAS1	Defines the minimal security level to access GPIO3_ALIAS1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved, write as 1 (or 0b11).
21-20 GPIO3_ALIAS0	GPIO3_ALIAS0 Defines the minimal security level to access GPIO3. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved, write as 1 (or 0b11).
17-16 GPIO2_ALIAS1	GPIO2_ALIAS1 Defines the minimal security level to access GPIO2_ALIAS1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved, write as 1 (or 0b11).
13-12 GPIO2_ALIAS0	GPIO2_ALIAS0 Defines the minimal security level to access GPIO2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved, write as 1 (or 0b11).

Table continues on the next page...

Table continued from the previous page...

Field	Function
9-8 GPIO1_ALIAS1	GPIO1_ALIAS1 Defines the minimal security level to access GPIO1_ALIAS1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved, write as 1 (or 0b11).
5-4 GPIO1_ALIAS0	GPIO1_ALIAS0 Defines the minimal security level to access GPIO1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved, write as 1 (or 0b11).
1-0 GPIO0_ALIAS1	GPIO0_ALIAS1 Defines the minimal security level to access GPIO0_ALIAS1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.29 AHB Peripheral 0 Slave Port 12 Slave Rule 2 (AHB_PERIPHERAL0_SLAVE_PORT_P12_SLAVE_RULE2)

Offset

Register	Offset
AHB_PERIPHERAL0_SLAVE_PORT_P12_SLAVE_RULE2	1D8h

Function

This register controls access to the AHB Peripheral 0 Slave. Each rule field controls access to a specific peripheral in AHB Peripheral 0.

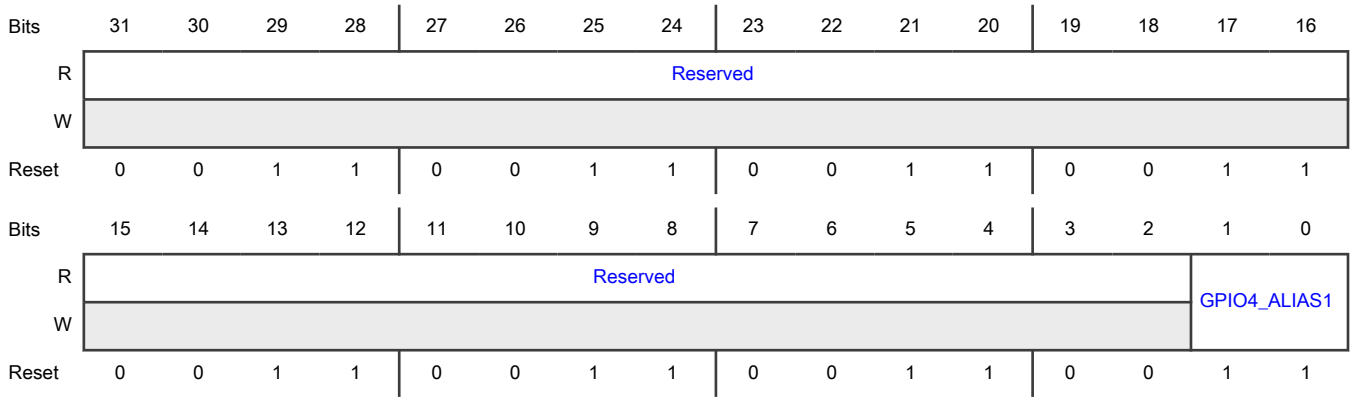
NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 462. AHB_PERIPHERAL0_MEM_RULE_RULE0

Rule	Address Access
GPIO4_ALIAS1	4009F000 - 4009FFFF

Diagram



Fields

Field	Function
31-2 —	Reserved, write as 1 (or 0b11).
1-0 GPIO4_ALIAS1	GPIO4_ALIAS1 Defines the minimal security level to access GPIO4_ALIAS1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.30 AIPS Bridge Group 1 Rule 0 (AIPS_BRIDGE_GROUP1_MEM_RULE0)

Offset

Register	Offset
AIPS_BRIDGE_GROUP1_MEM_RULE0	1E0h

Function

This register controls access to the AIPS Bridge Group 1. Each rule field controls access to a specific range in the AIPS Bridge Group 1.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 463. AIPS_BRIDGE_GROUP1_PER_RULE0

Rule	Address Access
eDMA0_MP	0x4008 0000 – 0x4008 1FFF
eDMA0_CH0	0x4008 1000 – 0x4008 1FFF
eDMA0_CH1	0x4008 2000 – 0x4008 2FFF
eDMA0_CH2	0x4008 3000 – 0x4008 3FFF
eDMA0_CH3	0x4008 4000 – 0x4008 4FFF
eDMA0_CH4	0x4008 5000 – 0x4008 5FFF
eDMA0_CH5	0x4008 6000 – 0x4008 6FFF
eDMA0_CH6	0x4008 7000 – 0x4008 7FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		eDMA0_CH6		Reserved		eDMA0_CH5		Reserved		eDMA0_CH4		Reserved		eDMA0_CH3	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		eDMA0_CH2		Reserved		eDMA0_CH1		Reserved		eDMA0_CH0		Reserved		eDMA0_MP	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 eDMA0_CH6	eDMA0_CH6 Defines the minimal security level to access eDMA0_CH6. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 eDMA0_CH5	eDMA0_CH5 Defines the minimal security level to access eDMA0_CH5. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 eDMA0_CH4	eDMA0_CH4 Defines the minimal security level to access eDMA0_CH4. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 eDMA0_CH3	FLEXSPI0 Registers Defines the minimal security level to access eDMA0_CH3. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 eDMA0_CH2	eDMA0_CH2 Defines the minimal security level to access eDMA0_CH2.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 eDMA0_CH1	eDMA0_CH1 Defines the minimal security level to access eDMA0_CH1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 eDMA0_CH0	eDMA0_CH0 Defines the minimal security level to access eDMA0_CH0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 eDMA0_MP	eDMA0_MP Defines the minimal security level to access eDMA0_MP. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.31 AIPS Bridge Group 1 Rule 1 (AIPS_BRIDGE_GROUP1_MEM_RULE1)

Offset

Register	Offset
AIPS_BRIDGE_GROUP 1_MEM_RULE1	1E4h

Function

This register controls access to the AIPS Bridge Group 1. Each rule field controls access to a specific range in the AIPS Bridge Group 1.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 464. AIPS_BRIDGE_GROUP1_PER_RULE1

Rule	Address Access
eDMA0_CH7	0x4008 8000 – 0x4008 8FFF
eDMA0_CH8	0x4008 9000 – 0x4008 9FFF
eDMA0_CH9	0x4008 A000 – 0x4008 AFFF
eDMA0_CH10	0x4008 B000 – 0x4008 BFFF
eDMA0_CH11	0x4008 C000 – 0x4008 CFFF
eDMA0_CH12	0x4008 D000 – 0x4008 DFFF
eDMA0_CH13	0x4008 E000 – 0x4008 EFFF
eDMA0_CH14	0x4008 F000 – 0x4008 FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		eDMA0_CH14		Reserved		eDMA0_CH13		Reserved		eDMA0_CH12		Reserved		eDMA0_CH11	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		eDMA0_CH10		Reserved		eDMA0_CH9		Reserved		eDMA0_CH8		Reserved		eDMA0_CH7	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 eDMA0_CH14	eDMA0_CH14 Defines the minimal security level to access eDMA0_CH14. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 eDMA0_CH13	eDMA0_CH13 Defines the minimal security level to access eDMA0_CH13. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 eDMA0_CH12	eDMA0_CH12 Defines the minimal security level to access eDMA0_CH12. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 eDMA0_CH11	FLEXSPI0 Defines the minimal security level to access eDMA0_CH11. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 eDMA0_CH10	eDMA0_CH10 Defines the minimal security level to access eDMA0_CH10. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 eDMA0_CH9	eDMA0_CH9 Defines the minimal security level to access eDMA0_CH9. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 eDMA0_CH8	eDMA0_CH8 Defines the minimal security level to access eDMA0_CH8. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 eDMA0_CH7	eDMA0_CH7 Defines the minimal security level to access eDMA0_CH7. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.32 AHB Peripheral 1 Slave Port 13 Slave Rule 0 (AHB_PERIPHERAL1_SLAVE_PORT_P13_SLAVE_RULE0)

Offset

Register	Offset
AHB_PERIPHERAL1_SLAVE_PORT_P13_SLAVE_RULE0	1F0h

Function

This register controls access to the AHB Peripheral 1 Slave. Each rule field controls access to a specific peripheral in AHB Peripheral 1.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

0x400B0000 - 0x400B0FFF	eDMA1_CH15
0x400B1000 - 0x400B1FFF	SEMA42
0x400B2000 - 0x400B2FFF	MAILBOX
0x400B3000 - 0x400B3FFF	PKC_RAM
0x400B4000 - 0x400B4FFF	FLEXCOMM4
0x400B5000 - 0x400B5FFF	FLEXCOMM5
0x400B6000 - 0x400B6FFF	FLEXCOMM6

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		FLEXCOMM6		Reserved		FLEXCOMM5		Reserved		FLEXCOMM4		Reserved		PKC_RAM	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		MAILBOX		Reserved		SEMA42		Reserved		eDMA1_CH15		Reserved			
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved, write as 1 (or 0b11).
29-28 FLEXCOMM6	FLEXCOMM6 Defines the minimal security level to access FLEXCOMM6. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved, write as 1 (or 0b11).
25-24 FLEXCOMM5	FLEXCOMM5 Defines the minimal security level to access FLEXCOMM5. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved, write as 1 (or 0b11).
21-20 FLEXCOMM4	FLEXCOMM4 Defines the minimal security level to access FLEXCOMM4. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved, write as 1 (or 0b11).
17-16 PKC_RAM	PKC_RAM Defines the minimal security level to access PKC_RAM. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved, write as 1 (or 0b11).
13-12 MAILBOX	MAILBOX Defines the minimal security level to access MAILBOX. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved, write as 1 (or 0b11).
9-8 SEMA42	SEMA42 Defines the minimal security level to access SEMA42. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved, write as 1 (or 0b11).
5-4 eDMA1_CH15	eDMA1_CH15 Defines the minimal security level to access eDMA1_CH15.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-0 —	Reserved, write as 1 (or 0b11).

23.4.1.33 AHB Peripheral 1 Slave Port 13 Slave Rule 1 (AHB_PERIPHERAL1_SLAVE_PORT_P13_SLAVE_RULE1)

Offset

Register	Offset
AHB_PERIPHERAL1_SLAVE_PORT_P13_SLAVE_RULE1	1F4h

Function

This register controls access to the AHB Peripheral 1 Slave. Each rule field controls access to a specific peripheral in AHB Peripheral 1.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

0x400B7000 - 0x400B7FFF	FLEXCOMM7
0x400B8000 - 0x400B8FFF	FLEXCOMM8
0x400B9000 - 0x400B9FFF	FLEXCOMM9
0x400BA000 - 0x400BAFFF	USB_FS_OTG_RAM
0x400BB000 - 0x400BBFFF	CDOG0
0x400BC000 - 0x400BCFFF	CDOG1
0x400BD000 - 0x401BDFFF	DEBUG_MAILBOX
0x400BE000 - 0x401BEFFF	NPU
0x400BF000 - 0x400BFFFF	POWERQUAD

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		NPU		Reserved		DEBUG_MAILBOX		Reserved		CDOG1		Reserved		CDOG0	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		USB_FS_OTG_RAM		Reserved		FLEXCOMM9		Reserved		FLEXCOMM8		Reserved		FLEXCOMM7	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved, write as 1 (or 0b11).
29-28 NPU	NPU Defines the minimal security level to access NPU. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved, write as 1 (or 0b11).
25-24 DEBUG_MAILBOX	DEBUG_MAILBOX Defines the minimal security level to access DEBUG_MAILBOX. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved, write as 1 (or 0b11).
21-20 CDOG1	CDOG1 Defines the minimal security level to access CDOG1. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved, write as 1 (or 0b11).
17-16 CDOG0	CDOG0 Defines the minimal security level to access CDOG0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved, write as 1 (or 0b11).
13-12 USB_FS_OTG_RAM	USB FS OTG RAM Defines the minimal security level to access USB FS OTG RAM. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved, write as 1 (or 0b11).
9-8 FLEXCOMM9	FLEXCOMM9 Defines the minimal security level to access FLEXCOMM9. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved, write as 1 (or 0b11).
5-4 FLEXCOMM8	FLEXCOMM8 Defines the minimal security level to access FLEXCOMM8.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved, write as 1 (or 0b11).
1-0 FLEXCOMM7	FLEXCOMM7 Defines the minimal security level to access FLEXCOMM7. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.34 AHB Peripheral 1 Slave Port 13 Slave Rule 2 (AHB_PERIPHERAL1_SLAVE_PORT_P13_SLAVE_RULE2)

Offset

Register	Offset
AHB_PERIPHERAL1_SLAVE_PORT_P13_SLAVE_RULE2	1F8h

Function

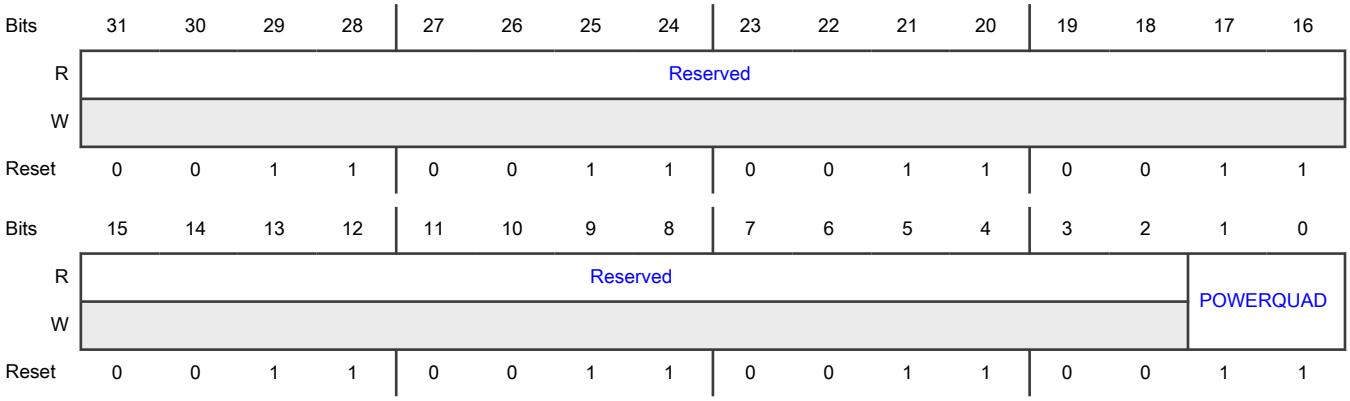
This register controls access to the AHB Peripheral 1 Slave. Each rule field controls access to a specific peripheral in AHB Peripheral 1.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

0x400BF000 - 0x400BFFFF	POWERQUAD

Diagram



Fields

Field	Function
31-2 —	Reserved, write as 1 (or 0b11).
1-0 POWERQUAD	POWERQUAD Defines the minimal security level to access POWERQUAD. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.35 AIPS Bridge Group 2 Rule 0 (AIPS_BRIDGE_GROUP2_MEM_RULE0)

Offset

Register	Offset
AIPS_BRIDGE_GROUP2_MEM_RULE0	200h

Function

This register controls access to the AIPS Bridge Group 2 Slave. Each rule field controls access to a specific peripheral in AIPS Bridge Group 2 Slave.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 465. AIPS Bridge Group 2

Rule	Address Access
eDMA1_MP	400A0000 - 400A0FFF
eDMA1_CH0	400A1000 - 400A1FFF
eDMA1_CH1	400A2000 - 400A2FFF
eDMA1_CH2	400A3000 - 400A3FFF
eDMA1_CH3	400A4000 - 400A4FFF
eDMA1_CH4	400A5000 - 400A5FFF
eDMA1_CH5	400A6000 - 400A6FFF
eDMA1_CH6	400A7000 - 400A7FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		eDMA1_CH6		Reserved		eDMA1_CH5		Reserved		eDMA1_CH4		Reserved		eDMA1_CH3	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		eDMA1_CH2		Reserved		eDMA1_CH1		Reserved		eDMA1_CH0		Reserved		eDMA1_MP	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 eDMA1_CH6	eDMA1_CH6 Defines the minimal security level to access eDMA1_CH6. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24	eDMA1_CH5

Table continues on the next page...

Table continued from the previous page...

Field	Function
eDMA1_CH5	Defines the minimal security level to access eDMA1_CH5. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 eDMA1_CH4	eDMA1_CH4 Defines the minimal security level to access eDMA1_CH4. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 eDMA1_CH3	eDMA1_CH3 Defines the minimal security level to access eDMA1_CH3. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 eDMA1_CH2	eDMA1_CH2 Defines the minimal security level to access eDMA1_CH2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
9-8 eDMA1_CH1	eDMA1_CH1 Defines the minimal security level to access eDMA1_CH1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 eDMA1_CH0	eDMA1_CH0 Defines the minimal security level to access eDMA1_CH0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 eDMA1_MP	eDMA1_MP Defines the minimal security level to access eDMA1_MP. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.36 AIPS Bridge Group 2 Memory Rule 1 (AIPS_BRIDGE_GROUP2_MEM_RULE1)

Offset

Register	Offset
AIPS_BRIDGE_GROUP2_MEM_RULE1	204h

Function

This register controls access to the AIPS Bridge Group 2 Slave. Each rule field controls access to a specific peripheral inAIPS Bridge Group 2 Slave.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 466. AIPS BRIDGE GROUP2 MEM RULE1

Rule	Address Access
eDMA1_CH7	400A8000 - 400A8FFF
eDMA1_CH8	400A9000 - 400A9FFF
eDMA1_CH9	400AA000 - 400AAFFF
eDMA1_CH10	400AB000 - 400ABFFF
eDMA1_CH11	400AC000 - 400ACFFF
eDMA1_CH12	400AD000 - 400ADFFF
eDMA1_CH13	400AE000 - 400AEFFF
eDMA1_CH14	400AF000 - 400AFFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		eDMA1_CH14		Reserved		eDMA1_CH13		Reserved		eDMA1_CH12		Reserved		eDMA1_CH11	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		eDMA1_CH10		Reserved		eDMA1_CH9		Reserved		eDMA1_CH8		Reserved		eDMA1_CH7	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 eDMA1_CH14	eDMA1_CH14 Defines the minimal security level to access eDMA1_CH14. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
25-24 eDMA1_CH13	eDMA1_CH13 Defines the minimal security level to access eDMA1_CH13. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 eDMA1_CH12	eDMA1_CH12 Defines the minimal security level to access eDMA1_CH12. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 eDMA1_CH11	eDMA1_CH11 Defines the minimal security level to access eDMA1_CH11. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 eDMA1_CH10	eDMA1_CH10 Defines the minimal security level to access eDMA1_CH10. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 eDMA1_CH9	eDMA1_CH9 Defines the minimal security level to access eDMA1_CH9. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 eDMA1_CH8	eDMA1_CH8 Defines the minimal security level to access eDMA1_CH8. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 eDMA1_CH7	eDMA1_CH7 Defines the minimal security level to access eDMA1_CH7. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.37 AIPS Bridge Group 3 Rule 0 (AIPS_BRIDGE_GROUP3_MEM_RULE0)

Offset

Register	Offset
AIPS_BRIDGE_GROUP3_MEM_RULE0	220h

Function

This register controls access to the AIPS Bridge Group 2 Slave. Each rule field controls access to a specific peripheral in AIPS Bridge Group 2 Slave.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 467. AIPS Bridge Group 3

Rule	Address Access
EWM0	400C0000 - 400C0FFF
LPCAC	400C1000 - 400C1FFF
FLEXSPI-CMX	400C2000 - 400C2FFF
Reserved	400C3000 - 400C3FFF
Reserved	400C4000 - 400C4FFF
SFA	400C5000 - 400C5FFF
Reserved	400C6000 - 400C6FFF
MBC	400C7000 - 400C7FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		MBC		Reserved				Reserved		SFA		Reserved		Reserved	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved				Reserved		FLEXSPI_CMX		Reserved		LPCAC		Reserved		EWM0	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 MBC	MBC Defines the minimal security level to access MBC. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-24 —	Reserved; the value read from a reserved bit is not defined.
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 SFA	SFA Defines the minimal security level to access SFA. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 —	Reserved; the value read from a reserved bit is not defined.
15-12 —	Reserved; the value read from a reserved bit is not defined.
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 FLEXSPI_CMX	FLEXSPI_CMX Defines the minimal security level to access FLEXSPI_CMX. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 LPCAC	LPCAC Defines the minimal security level to access LPCAC.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 EWM0	EWM0 Defines the minimal security level to access EWM0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.38 AIPS Bridge Group 3 Memory Rule 1 (AIPS_BRIDGE_GROUP3_MEM_RULE1)

Offset

Register	Offset
AIPS_BRIDGE_GROUP3_MEM_RULE1	224h

Function

This register controls access to the AIPS Bridge Group 3 Slave. Each rule field controls access to a specific peripheral inAIPS Bridge Group 3 Slave.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 468. AIPS BRIDGE GROUP3 MEM RULE1

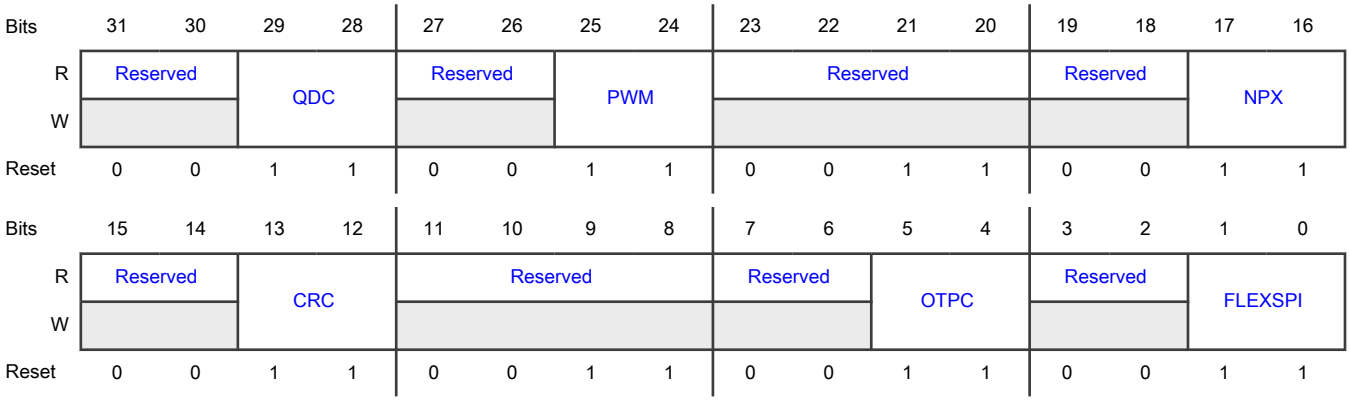
Rule	Address Access
FLEXSPI	400C8000 - 400C8FFF
OTPC	400C9000 - 400C9FFF
Reserved	400CA000 - 400CAFFF
CRC	400CB000 - 400CBFFF
NPX	400CC000 - 400CCFFF
Reserved	400CD000 - 400CDFFF

Table continues on the next page...

Table 468. AIPS BRIDGE GROUP3 MEM RULE1 (continued)

Rule	Address Access
PWM	400CE000 - 400CEFFF
QDC	400CF000 - 400CFFFF

Diagram



Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 QDC	QDC Defines the minimal security level to access QDC. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 PWM	PWM Defines the minimal security level to access PWM. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-20	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 NPX	NPX Defines the minimal security level to access NPX. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 CRC	CRC Defines the minimal security level to access CRC. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-8 —	Reserved; the value read from a reserved bit is not defined.
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 OTPC	OTPC Defines the minimal security level to access OTPC. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0	FLEXSPI Defines the minimal security level to access FLEXSPI.

Table continues on the next page...

Table continued from the previous page...

Field	Function
FLEXSPI	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.39 AIPS Bridge Group 3 Rule 2 (AIPS_BRIDGE_GROUP3_MEM_RULE2)

Offset

Register	Offset
AIPS_BRIDGE_GROUP3_MEM_RULE2	228h

Function

This register controls access to the AIPS Bridge Group 3. Each rule field controls access to a specific range in the AIPS Bridge Group 3.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 469. AIPS_BRIDGE_GROUP3_PER_RULE2

Rule	Address Access
PWM1	0x400D 0000 – 0x400D 0FFF
QDC1	0x400D 1000 – 0x400D 1FFF
EVTG0	0x400D 2000 – 0x400D 2FFF
Reserved	0x400D 3000 – 0x400D 3FFF
CAN0 Region 0	0x400D 4000 – 0x400D 4FFF
CAN0 Region 1	0x400D 5000 – 0x400D 5FFF
CAN0 Region 2	0x400D 6000 – 0x400D 6FFF
CAN0 Region 3	0x400D 7000 – 0x400D 7FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		CAN0_RULE3		Reserved		CAN0_RULE2		Reserved		CAN0_RULE1		Reserved		CAN0_RULE0	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved				Reserved		EVTG		Reserved		QDC1		Reserved		PWM1	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved
29-28 CAN0_RULE3	CAN0 RULE3 Defines the minimal security level to access CAN0 Region 3. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved
25-24 CAN0_RULE2	CAN0 RULE2 Defines the minimal security level to access CAN0 Region 2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved
21-20 CAN0_RULE1	CAN0 RULE1 Defines the minimal security level to access CAN0 Region 1. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved
17-16 CAN0_RULE0	CAN0 RULE0 Defines the minimal security level to access CAN0 Region 0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-12 —	Reserved; the value read from a reserved bit is not defined.
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 EVTG	EVTG Defines the minimal security level to access EVTG. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 QDC1	QDC1 Defines the minimal security level to access QDC1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
1-0 PWM1	PWM1 Defines the minimal security level to access PWM1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.40 AIPS Bridge Group 3 Rule 3 (AIPS_BRIDGE_GROUP3_MEM_RULE3)

Offset

Register	Offset
AIPS_BRIDGE_GROUP3_MEM_RULE3	22Ch

Function

This register controls access to the AIPS Bridge Group 3. Each rule field controls access to a specific range in the AIPS Bridge Group 3.

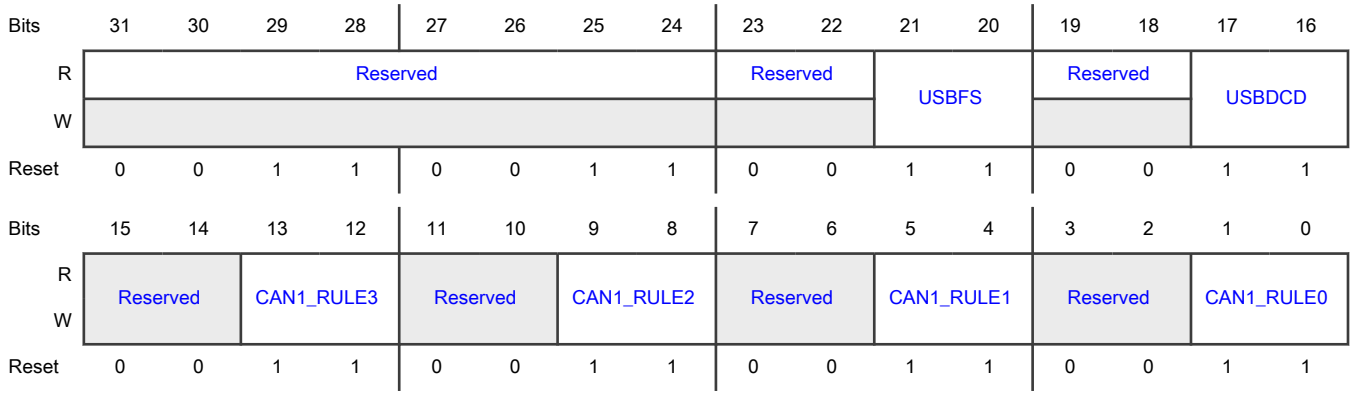
NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 470. AIPS_BRIDGE_GROUP3_PER_RULE3

Rule	Address Access
CAN1 Region 1	0x400D 8000 – 0x400D 8FFF
CAN1 Region 2	0x400D 9000 – 0x400D 9FFF
CAN1 Region 3	0x400D A000 – 0x400D AFFF
CAN1 Region 4	0x400D B000 – 0x400D BFFF
USBDCD	0x400D C000 – 0x400D CFFF
USBFS	0x400D D000 – 0x400D DFFF
Reserved	0x400D E000 – 0x400D DFFF
Reserved	0x400D F000 – 0x400D FFFF

Diagram



Fields

Field	Function
31-24 —	Reserved; the value read from a reserved bit is not defined.
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 USBFS	USBFS Defines the minimal security level to access USBFS. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 USBDCD	USBDCD Defines the minimal security level to access USBDCD. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved
13-12	CAN1 RULE3

Table continues on the next page...

Table continued from the previous page...

Field	Function
CAN1_RULE3	Defines the minimal security level to access CAN1 Region 3. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved
9-8 CAN1_RULE2	CAN1 RULE2 Defines the minimal security level to access CAN1 Region 2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved
5-4 CAN1_RULE1	CAN1 RULE1 Defines the minimal security level to access CAN1 Region 1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved
1-0 CAN1_RULE0	CAN1 RULE0 Defines the minimal security level to access CAN1 Region 0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.41 AIPS Bridge Group 4 Rule 0 (AIPS_BRIDGE_GROUP4_MEM_RULE0)

Offset

Register	Offset
AIPS_BRIDGE_GROUP4_MEM_RULE0	240h

Function

This register controls access to the AIPS Bridge Group 4 Slave. Each rule field controls access to a specific peripheral in AIPS Bridge Group 4 Slave.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 471. AIPS Bridge Group 4

Rule	Address Access
ENET	0x400E 0000 – 0x400E 1FFF
Reserved	0x400E 2000 – 0x400E 2FFF
EMVSIM0	0x400E 3000 – 0x400E 3FFF
EMVSIM1	0x400E 4000 – 0x400E 4FFF
FLEXIO	0x400E 5000 – 0x400E 5FFF
SAI0	0x400E 6000 – 0x400E 6FFF
SAI1	0x400E 7000 – 0x400E 7FFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		SAI1		Reserved		SAI0		Reserved		FLEXIO		Reserved		EMVSIM1	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		EMVSIM0		Reserved								ENET			
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
29-28 SAI1	SAI1 Defines the minimal security level to access SAI1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 SAI0	SAI0 Defines the minimal security level to access SAI0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 FLEXIO	FLEXIO Defines the minimal security level to access FLEXIO. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 EMVSIM1	EMVSIM1 Defines the minimal security level to access EMVSIM1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 EMVSIM0	EMVSIM0 Defines the minimal security level to access EMVSIM0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-4 —	Reserved; the value read from a reserved bit is not defined.
3-0 ENET	ENET Defines the minimal security level to access ENET. 0000b - Non-secure and non-privilege user access allowed 0001b - Non-secure and privilege access allowed 0010b - Secure and non-privilege user access allowed 0011b - Secure and privilege user access allowed

23.4.1.42 AIPS Bridge Group 4 Rule 1 (AIPS_BRIDGE_GROUP4_MEM_RULE1)

Offset

Register	Offset
AIPS_BRIDGE_GROUP4_MEM_RULE1	244h

Function

This register controls access to the AIPS Bridge Group 4 Slave. Each rule field controls access to a specific peripheral in AIPS Bridge Group 4 Slave.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 472. AIPS Bridge Group 4

Rule	Address Access
SINC0	0x400E 8000 – 0x400E 8FFF

Table continues on the next page...

Table 472. AIPS Bridge Group 4 (continued)

Rule	Address Access
uSDHC0	0x400E 9000 – 0x400E 9FFF
USBHSPHY	0x400E A000 – 0x400E AFFF
USBHS	0x400E DE00 – 0x400E BFFF
MICD	0x400E C000 – 0x400E CFFF
ADC0	0x400E D000 – 0x400E DFFF
ADC1	0x400E E000 – 0x400E EFFF
DAC0	0x400E F000 – 0x400E FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		DAC0		Reserved		ADC1		Reserved		ADC0		Reserved		MICD	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		USBHS		Reserved		USBHSPHY		Reserved		uSDHC0		Reserved		SINC0	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 DAC0	DAC0 Defines the minimal security level to access DAC0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 ADC1	ADC1 Defines the minimal security level to access ADC1.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 ADC0	ADC0 Defines the minimal security level to access ADC0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 MICD	MICD Defines the minimal security level to access MICD. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 USBHS	USBHS Defines the minimal security level to access USBHS. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8	USBHSPHY

Table continues on the next page...

Table continued from the previous page...

Field	Function
USBHSPHY	Defines the minimal security level to access USBHSPHY. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 uSDHC0	uSDHC0 Defines the minimal security level to access uSDHC0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 SINC0	SINC0 Defines the minimal security level to access SINC0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.43 AIPS Bridge Group 4 Rule 2 (AIPS_BRIDGE_GROUP4_MEM_RULE2)

Offset

Register	Offset
AIPS_BRIDGE_GROUP4_MEM_RULE2	248h

Function

This register controls access to the AIPS Bridge Group 4 Slave. Each rule field controls access to a specific peripheral in AIPS Bridge Group 4 Slave.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 473. AIPS Bridge Group 4

Rule	Address Access
OPAMP0	0x400E 8000 – 0x400E 8FFF
VREF	0x400E 9000 – 0x400E 9FFF
DAC	0x400E A000 – 0x400E AFFF
OPAMP1	0x400E DE00 – 0x400E BFFF
HPDAC0	0x400E C000 – 0x400E CFFF
OPAMP2	0x400E D000 – 0x400E DFFF
PORT0	0x400E E000 – 0x400E EFFF
DAC0	0x400E F000 – 0x400E FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		PORT1		Reserved		PORT0		Reserved		OPAMP2		Reserved		HPDAC0	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		OPAMP1		Reserved		DAC		Reserved		VREF		Reserved		OPAMP0	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 PORT1	PORT1 Defines the minimal security level to access PORT1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24	PORT0

Table continues on the next page...

Table continued from the previous page...

Field	Function
PORT0	Defines the minimal security level to access PORT0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 OPAMP2	OPAMP2 Defines the minimal security level to access OPAMP2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 HPDAC0	HPDAC0 Defines the minimal security level to access HPDAC0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 OPAMP1	OPAMP1 Defines the minimal security level to access OPAMP1. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
9-8 DAC	DAC Defines the minimal security level to access DAC. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 VREF	VREF Defines the minimal security level to access VREF. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 OPAMP0	OPAMP0 Defines the minimal security level to access OPAMP0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.44 AIPS Bridge Group 4 Rule 3 (AIPS_BRIDGE_GROUP4_MEM_RULE3)

Offset

Register	Offset
AIPS_BRIDGE_GROUP4_MEM_RULE3	24Ch

Function

This register controls access to the AIPS Bridge Group 4 Slave. Each rule field controls access to a specific peripheral in AIPS Bridge Group 4 Slave.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 474. AIPS Bridge Group 4

Rule	Address Access
PORT2	0x400F 8000 – 0x400F 8FFF
PORT3	0x400F 9000 – 0x400F 9FFF
PORT4	0x400F A000 – 0x400F AFFF
Reserved	0x400F DE00 – 0x400F BFFF
Reserved	0x400F C000 – 0x400F CFFF
Reserved	0x400F D000 – 0x400F DFFF
MTR0	0x400F E000 – 0x400F EFFF
ATX0	0x400F F000 – 0x400F FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		ATX0		Reserved		MTR0		Reserved		Reserved		Reserved		Reserved	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		Reserved		Reserved		PORT4		Reserved		PORT3		Reserved		PORT2	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 ATX0	ATX0 Defines the minimal security level to access ATX0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
25-24 MTR0	MTR0 Defines the minimal security level to access MTR0. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 —	Reserved
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 —	Reserved
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 —	Reserved
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 PORT4	PORT4 Defines the minimal security level to access PORT4. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4	PORT3

Table continues on the next page...

Table continued from the previous page...

Field	Function
PORT3	Defines the minimal security level to access PORT3. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 PORT2	PORT2 Defines the minimal security level to access PORT2. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.45 AHB Secure Control Peripheral Rule 0 (AHB_SECURE_CTRL_PERIPHERAL_RULE0)

Offset

Register	Offset
AHB_SECURE_CTRL_PERIPHERAL_RULE0	250h

Function

This register controls access to the AHB Secure Control Peripheral. Each rule field controls access to a specific range in the AHB Secure Control Peripheral.

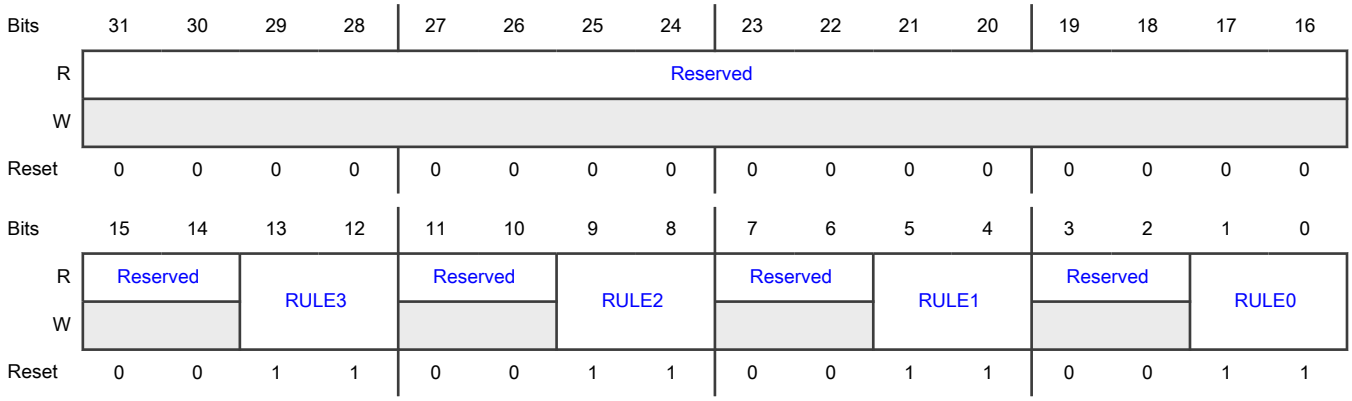
NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 475. AHB_SECURE_CTRL_PERIPH_RULE0

Rule	Address Access
RULE0	0x4012 0000 – 0x4012 0FFF
RULE1	0x4012 1000 – 0x4012 1FFF
RULE2	0x4012 2000 – 0x4012 2FFF
RULE3	0x4012 3000 – 0x4012 3FFF

Diagram



Fields

Field	Function
31-14 —	Reserved, write as 1 (or 0b11).
13-12 RULE3	Rule 3 Defines the minimal security level to access the memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved, write as 1 (or 0b11).
9-8 RULE2	Rule 2 Defines the minimal security level to access the memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved, write as 1 (or 0b11).
5-4 RULE1	Rule 1 Defines the minimal security level to access the memory at a specific address range. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved, write as 1 (or 0b11).
1-0 RULE0	Rule 0 Defines the minimal security level to access the memory at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.46 FLEXSPI0 Region 0 Memory Rule (FLEXSPI0_REGION0_MEM_RULE0 - FLEXSPI0_REGION0_MEM_RULE3)

Offset

Register	Offset
FLEXSPI0_REGION0_MEM_RULE0	270h
FLEXSPI0_REGION0_MEM_RULE1	274h
FLEXSPI0_REGION0_MEM_RULE2	278h
FLEXSPI0_REGION0_MEM_RULE3	27Ch

Function

These 4 registers control access to the FLEXSPI0 External Memory interface. For each register, each rule field controls access to a specific range in FLEXSPI0 External Memory interface.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 476. FLEXSPI0_REGION0_MEM_RULE0

Rule	Address Access
0	0x8000 0000 – 0x8003 FFFF

Table continues on the next page...

Table 476. FLEXSPI0_REGION0_MEM_RULE0 (continued)

Rule	Address Access
1	0x8004 0000 – 0x8007 FFFF
2	0x8008 0000 – 0x800B FFFF
3	0x800C 0000 – 0x800F FFFF
4	0x8010 0000 – 0x8013 FFFF
5	0x8014 0000 – 0x8017 FFFF
6	0x8018 0000 – 0x801B FFFF
7	0x801C 0000 – 0x801F FFFF

Table 477. FLEXSPI0_REGION0_MEM_RULE1

Rule	Address Access
0	0x8020 0000 – 0x8023 FFFF
1	0x8024 0000 – 0X8027 FFFF
2	0x8028 0000 – 0x802B FFFF
3	0x802C 0000 – 0x802 FFFF
4	0x8030 0000 – 0x8033 FFFF
5	0x8034 0000 – 0x8037 FFFF
6	0x8038 0000 – 0x803B FFFF
7	0x803C 0000 – 0x803F FFFF

Table 478. FLEXSPI0_REGION0_MEM_RULE2

Rule	Address Access
0	0x8040 0000 – 0x8043 FFFF
1	0x8044 0000 – 0x8047 FFFF
2	0x8048 0000 – 0x804B FFFF
3	0x804C 0000 – 0x804F FFFF
4	0x8050 0000 – 0x8053 FFFF
5	0x8054 0000 – 0x8057 FFFF
6	0x8058 0000 – 0x805B FFFF
7	0x805C 0000 – 0x805F FFFF

Table 479. FLEXSPI0_REGION0_MEM_RULE3

Rule	Address Access
0	0x8060 0000 – 0x8063 FFFF
1	0x8064 0000 – 0x8067 FFFF

Table continues on the next page...

Table 479. FLEXSPI0_REGION0_MEM_RULE3 (continued)

Rule	Address Access
2	0x8068 0000 – 0x806B FFFF
3	0x806C 0000 – 0x806 FFFF
4	0x8070 0000 – 0x8073 FFFF
5	0x8074 0000 – 0x8077 FFFF
6	0x8078 0000 – 0x807B FFFF
7	0x807C 0000 – 0x807F FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
9-8 RULE2	Rule 2 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.47 FLEXSPI0 Region a Memory Rule 0 (FLEXSPI0_REGION1_MEM_RULE0 - FLEXSPI0_REGION6_MEM_RULE0)

Offset

Register	Offset
FLEXSPI0_REGION1_MEM_RULE0	280h
FLEXSPI0_REGION2_MEM_RULE0	290h
FLEXSPI0_REGION3_MEM_RULE0	2A0h
FLEXSPI0_REGION4_MEM_RULE0	2B0h
FLEXSPI0_REGION5_MEM_RULE0	2C0h
FLEXSPI0_REGION6_MEM_RULE0	2D0h

Function

These 4 registers control access to the FLEXSPI0 external memory interface. For each register, each rule field controls access to a specific range in FLEXSPI0 external memory interface.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

0x80800000 - 80FFFFFF	FLEXSPI0 REGION1
0x81000000 - 81FFFFFF	FLEXSPI0 REGION2
0x82000000 - 83FFFFFF	FLEXSPI0 REGION3
0x84000000 - 87FFFFFF	FLEXSPI0 REGION4
0x88000000 - 8BFFFFFF	FLEXSPI0 REGION5
0x8C000000 - 8FFFFFFF	FLEXSPI0 REGION6

Table 480. FLEXSPI0_REGION1_MEM_RULE0

Rule	Address Access
0	0x8080 0000 – 0x809F FFFF
1	0x80A0 0000 – 0x80BF FFFF
2	0x80C0 0000 – 0x80DF FFFF
3	0x80E0 0000 – 0x80FF FFFF

Table 481. FLEXSPI0_REGION2_MEM_RULE0

Rule	Address Access
0	0x8100 0000 – 0x813F FFFF
1	0x8140 0000 – 0x817F FFFF
2	0x8180 0000 – 0x81BF FFFF
3	0x81C0 0000 – 0x81FF FFFF

Table 482. FLEXSPI0_REGION3_MEM_RULE0

Rule	Address Access
0	0x8200 0000 – 0x827F FFFF
1	0x8280 0000 – 0x82FF FFFF
2	0x8300 0000 – 0x837F FFFF
3	0x8380 0000 – 0x83FF FFFF

Table 483. FLEXSPI0_REGION4_MEM_RULE0

Rule	Address Access
0	0x8400 0000 – 0x84FF FFFF
1	0x8500 0000 – 0x85FF FFFF
2	0x8600 0000 – 0x86FF FFFF
3	0x8700 0000 – 0x87FF FFFF

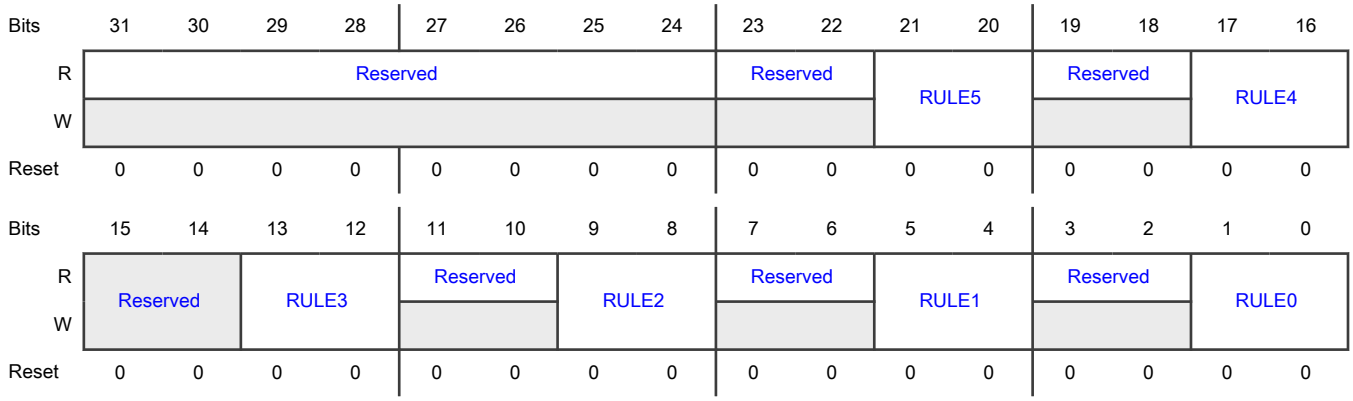
Table 484. FLEXSPI0_REGION5_MEM_RULE0

Rule	Address Access
0	0x8800 0000 – 0x88FF FFFF
1	0x8900 0000 – 0x89FF FFFF
2	0x8A00 0000 – 0x8AFF FFFF
3	0x8B00 0000 – 0x8BFF FFFF

Table 485. FLEXSPI0_REGION6_MEM_RULE0

Rule	Address Access
0	0x8C00 0000 – 0x8CFF FFFF
1	0x8D00 0000 – 0x8DFF FFFF
2	0x8E00 0000 – 0x8EFF FFFF
3	0x8F00 0000 – 0x8FFF FFFF

Diagram



Fields

Field	Function
31-24 —	Reserved; the value read from a reserved bit is not defined.
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
13-12 RULE3	Rule 3 Defines the minimal security level to access the FLEXSPI0 external memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the FLEXSPI0 external memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the FLEXSPI0 external memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the FLEXSPI0 external memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.48 FLEXSPI0 Region 7 Memory Rule (FLEXSPI0_REGION7_MEM_RULE0 - FLEXSPI0_REGION7_MEM_RULE3)

Offset

Register	Offset
FLEXSPI0_REGION7_MEM_RULE0	2E0h
FLEXSPI0_REGION7_MEM_RULE1	2E4h
FLEXSPI0_REGION7_MEM_RULE2	2E8h
FLEXSPI0_REGION7_MEM_RULE3	2ECh

Function

These 4 registers control access to the FLEXSPI0 External Memory interface. For each register, each rule field controls access to a specific range in FLEXSPI0 External Memory interface.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

Table 486. FLEXSPI0_REGION7_MEM_RULE0

Rule	Address Access
0	0xA000 0000 – 0xA003 FFFF
1	0xA004 0000 – 0xA007 FFFF
2	0xA008 0000 – 0xA00B FFFF
3	0xA00C 0000 – 0xA00F FFFF
4	0xA010 0000 – 0xA013 FFFF
5	0xA014 0000 – 0xA017 FFFF
6	0xA018 0000 – 0xA01B FFFF
7	0xA01C 0000 – 0xA01F FFFF

Table 487. FLEXSPI0_REGION0_MEM_RULE1

Rule	Address Access
0	0xA020 0000 – 0xA023 FFFF
1	0xA024 0000 – 0xA027 FFFF
2	0xA028 0000 – 0xA02B FFFF
3	0xA02C 0000 – 0xA02 FFFF
4	0xA030 0000 – 0xA033 FFFF
5	0xA034 0000 – 0xA037 FFFF
6	0xA038 0000 – 0xA03B FFFF
7	0xA03C 0000 – 0xA03F FFFF

Table 488. FLEXSPI0_REGION0_MEM_RULE2

Rule	Address Access
0	0xA040 0000 – 0xA043 FFFF
1	0xA044 0000 – 0xA047 FFFF
2	0xA048 0000 – 0xA04B FFFF
3	0xA04C 0000 – 0xA04F FFFF
4	0xA050 0000 – 0xA053 FFFF
5	0xA054 0000 – 0xA057 FFFF
6	0xA058 0000 – 0xA05B FFFF
7	0xA05C 0000 – 0xA05F FFFF

Table 489. FLEXSPI0_REGION0_MEM_RULE3

Rule	Address Access
0	0xA060 0000 – 0xA063 FFFF
1	0xA064 0000 – 0xA067 FFFF
2	0xA068 0000 – 0xA06B FFFF
3	0xA06C 0000 – 0xA06 FFFF
4	0xA070 0000 – 0xA073 FFFF
5	0xA074 0000 – 0xA077 FFFF
6	0xA078 0000 – 0xA07B FFFF
7	0xA07C 0000 – 0xA07F FFFF

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		RULE7		Reserved		RULE6		Reserved		RULE5		Reserved		RULE4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		RULE3		Reserved		RULE2		Reserved		RULE1		Reserved		RULE0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved; the value read from a reserved bit is not defined.
29-28 RULE7	Rule 7 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
27-26 —	Reserved; the value read from a reserved bit is not defined.
25-24 RULE6	Rule 6 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
23-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 RULE5	Rule 5 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved; the value read from a reserved bit is not defined.
13-12 RULE3	Rule 3 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
5-4 RULE1	Rule 1 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.49 FLEXSPI0 Region a Memory Rule 0 (FLEXSPI0_REGION8_MEM_RULE0 - FLEXSPI0_REGION13_MEM_RULE0)

Offset

Register	Offset
FLEXSPI0_REGION8_MEM_RULE0	2F0h
FLEXSPI0_REGION9_MEM_RULE0	300h
FLEXSPI0_REGION10_MEM_RULE0	310h
FLEXSPI0_REGION11_MEM_RULE0	320h
FLEXSPI0_REGION12_MEM_RULE0	330h
FLEXSPI0_REGION13_MEM_RULE0	340h

Function

These 4 registers control access to the FLEXSPI0 external memory interface. For each register, each rule field controls access to a specific range in FLEXSPI0 external memory interface.

NOTE

Any rule can be set when MIS_CTRL_REG[WRITE_LOCK] = 10.

0xA0800000 - 0xA0FFFFFF	FLEXSPI0 REGION8
0xA1000000 - 0xA1FFFFFF	FLEXSPI0 REGION9
0xA2000000 - 0xA3FFFFFF	FLEXSPI0 REGION10
0xA4000000 - 0xA7FFFFFF	FLEXSPI0 REGION11
0xA8000000 - 0xABFFFFFF	FLEXSPI0 REGION12
0xAC000000 - 0xAFFFFFFF	FLEXSPI0 REGION13

Table 490. FLEXSPI0_REGION8_MEM_RULE0

Rule	Address Access
0	0xA080 0000 – 0xA09F FFFF
1	0xA0A0 0000 – 0xA0BF FFFF
2	0xA0C0 0000 – 0xA0DF FFFF
3	0xA0E0 0000 – 0xA0FF FFFF

Table 491. FLEXSPI0_REGION9_MEM_RULE0

Rule	Address Access
0	0xA100 0000 – 0xA13F FFFF
1	0xA140 0000 – 0xA17F FFFF
2	0xA180 0000 – 0xA1BF FFFF
3	0xA1C0 0000 – 0xA1FF FFFF

Table 492. FLEXSPI0_REGION10_MEM_RULE0

Rule	Address Access
0	0xA200 0000 – 0xA27F FFFF
1	0xA280 0000 – 0xA2FF FFFF
2	0xA300 0000 – 0xA37F FFFF
3	0xA380 0000 – 0xA3FF FFFF

Table 493. FLEXSPI0_REGION11_MEM_RULE0

Rule	Address Access
0	0x8400 0000 – 0xA4FF FFFF

Table continues on the next page...

Table 493. FLEXSPI0_REGION11_MEM_RULE0 (continued)

Rule	Address Access
1	0xA500 0000 – 0xA5FF FFFF
2	0xA600 0000 – 0xA6FF FFFF
3	0xA700 0000 – 0xA7FF FFFF

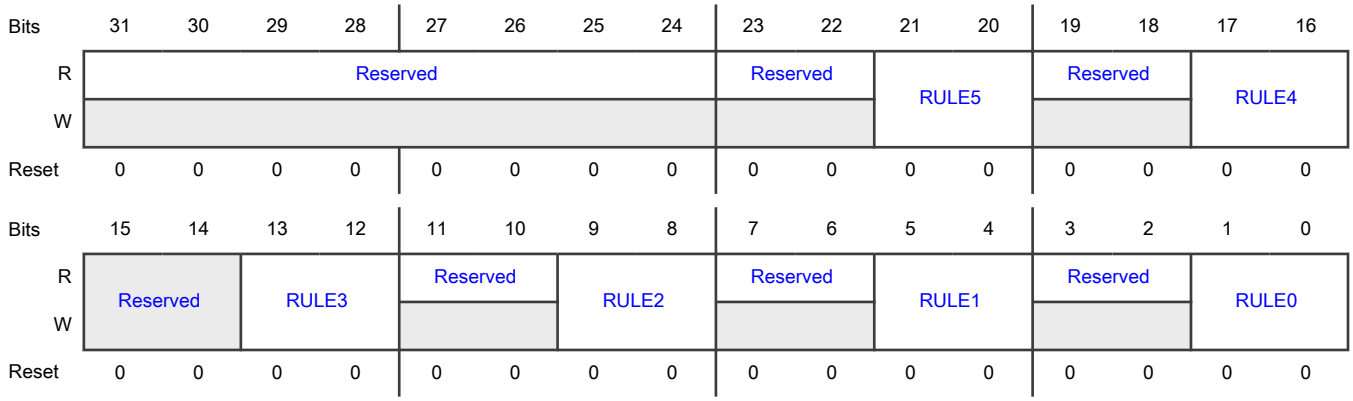
Table 494. FLEXSPI0_REGION12_MEM_RULE0

Rule	Address Access
0	0xA800 0000 – 0xA8FF FFFF
1	0xA900 0000 – 0xA9FF FFFF
2	0xAA00 0000 – 0xAAFF FFFF
3	0xAB00 0000 – 0xABFF FFFF

Table 495. FLEXSPI0_REGION13_MEM_RULE0

Rule	Address Access
0	0xAC00 0000 – 0xACFF FFFF
1	0xAD00 0000 – 0xADFF FFFF
2	0xAE00 0000 – 0xAEFF FFFF
3	0xAF00 0000 – 0xAFFF FFFF

Diagram



Fields

Field	Function
31-24 —	Reserved; the value read from a reserved bit is not defined.
23-22	Reserved; the value read from a reserved bit is not defined.

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
21-20 RULE5	Rule 5 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
19-18 —	Reserved; the value read from a reserved bit is not defined.
17-16 RULE4	Rule 4 Defines the minimal security level to access the FLEXSPI0 External Memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
15-14 —	Reserved
13-12 RULE3	Rule 3 Defines the minimal security level to access the FLEXSPI0 external memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
11-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 RULE2	Rule 2 Defines the minimal security level to access the FLEXSPI0 external memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed

Table continues on the next page...

Table continued from the previous page...

Field	Function
	01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
7-6 —	Reserved; the value read from a reserved bit is not defined.
5-4 RULE1	Rule 1 Defines the minimal security level to access the FLEXSPI0 external memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed
3-2 —	Reserved; the value read from a reserved bit is not defined.
1-0 RULE0	Rule 0 Defines the minimal security level to access the FLEXSPI0 external memory interface at a specific address range. 00b - Non-secure and non-privilege user access allowed 01b - Non-secure and privilege access allowed 10b - Secure and non-privilege user access allowed 11b - Secure and privilege user access allowed

23.4.1.50 Security Violation Address (SEC_VIO_ADDR0 - SEC_VIO_ADDR31)

Offset

For a = 0 to 31:

Register	Offset
SEC_VIO_ADDRa	E00h + (a × 4h)

Function

This is the most recent security violation address for AHB layer a. However, please refer to SEC_VIO_INFO_VALID register to verify if that slave has valid violation.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SEC_VIO_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SEC_VIO_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 SEC_VIO_ADDR	Security violation address for AHB layer a reset value 0

23.4.1.51 Security Violation Miscellaneous Information at Address (SEC_VIO_MISC_INFO0 - SEC_VIO_MISC_INFO31)

Offset

For a = 0 to 31:

Register	Offset
SEC_VIO_MISC_INFOa	E80h + (a × 4h)

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved			SEC_VIO_INFO_MASTER					SEC_VIO_INFO_MASTER_SEC_LEVEL				Reserved		SEC_VIO...	SEC_VIO...
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-13 —	Reserved; the value read from a reserved bit is not defined.
12-8 SEC_VIO_INFO_MASTER	Security violation master number 0_0000b - M33 Code 0_0001b - M33 System 0_0010b - CPU1 (Mirco-CM33) Code 0_0011b - SMARTDMA Instruction 0_0100b - CPU1 (Mirco-CM33) system 0_0101b - SMARTDMA Data 0_0110b - eDMA0 0_0111b - eDMA1 0_1000b - PKC 0_1001b - ELS S50 0_1010b - PKC M0 0_1011b - NPU Operands 0_1100b - DSP Instruction 0_1101b - DSPX 0_1110b - DSPY 1_0000b - NPU Data 1_0001b - USB FS 1_0010b - Ethernet 1_0011b - USB HS 1_0100b - uSDHC
7-4 SEC_VIO_INFO_MASTER_SEC_LEVEL	Security Violation Info Master Security Level • Bits [5:4]: master security level and privilege level • Bits [7:6]: anti-pole value for master sec level and privilege level
3-2 —	Reserved; the value read from a reserved bit is not defined.
1 SEC_VIO_INFO_DATA_ACCESS	Security Violation Info Data Access Indicates whether the security violations occurred on a data access or a code access 0b - Code 1b - Data

Table continues on the next page...

Table continued from the previous page...

Field	Function
0 SEC_VIO_INFO_WRITE	Security violation access read/write indicator Indicates whether the security violation occurred on a read or write access 0b - Read access 1b - Write access

23.4.1.52 Security Violation Info Validity for Address (SEC_VIO_INFO_VALID)

Offset

Register	Offset
SEC_VIO_INFO_VALID	F00h

Function

This register describes if security violation happened on a given AHB layer. If VIO_INFO_VALIDn=1, look at SEC_VIO_ADDRn and SEC_VIO_MISC_INFOn registers for the detailed violation information.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved													VIO_INF...	VIO_INF...	VIO_INF...
W														W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...	VIO_INF...
W	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-19 —	Reserved; the value read from a reserved bit is not defined.
18-0 VIO_INFO_VALIDn	Violation information valid flag for AHB port n Write 1 to a bit to clear it. 0b - Not valid 1b - Valid

23.4.1.53 GPIO Mask for Port 0 (SEC_GPIO_MASK0)

Offset

Register	Offset
SEC_GPIO_MASK0	F80h

Function

Pin Interrupt and Pattern Match (PINT) block GPIO mask for port 0 pins. This register is used to block leakage of secure interface pin states to the non-secure world PINT block if it is used in. Peripherals can be configured as secure, using a SEC_RULE for each peripheral.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...	PIO0_PI...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-0 PIOn_PINn_SEC_MASK	Mask bit 0b - Masked 1b - Not masked

23.4.1.54 GPIO Mask for Port 1 (SEC_GPIO_MASK1)

Offset

Register	Offset
SEC_GPIO_MASK1	F84h

Function

Pin Interrupt and Pattern Match (PINT) block GPIO mask for port 1 pins. This register is used to block leakage of secure interface pin states to the non-secure world PINT block if it is used in. Peripherals can be configured as secure, using a SEC_RULE for each peripheral.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...	PIO1_PI...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-0 PIOn_PINn_SEC_MASK	Mask bit 0b - Masked 1b - Not masked

23.4.1.55 Secure Interrupt Mask 0 for CPU1 (SEC_CPU1_INT_MASK0)

Offset

Register	Offset
SEC_CPU1_INT_MASK0	F98h

Function

This register can block different peripheral interrupt from IRQ bus to go to CPU1. If CPU1 is not a secure master, application might want to disable secure peripheral interrupts reaching that CPU.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	INT31_M...	INT30_M...	INT29_M...	INT28_M...	INT27_M...	INT26_M...	INT25_M...	INT24_M...	INT23_M...	INT22_M...	INT21_M...	INT20_M...	INT19_M...	INT18_M...	INT17_M...	INT16_M...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	INT15_M...	INT14_M...	INT13_M...	INT12_M...	INT11_M...	INT10_M...	INT9_MA...	INT8_MA...	INT7_MA...	INT6_MA...	INT5_MA...	INT4_MA...	INT3_MA...	INT2_MA...	INT1_MA...	INT0_MA...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-0 INTn_MASK	Mask bit 0b - Masked 1b - Not masked

23.4.1.56 Secure Interrupt Mask 1 for CPU1 (SEC_CPU1_INT_MASK1)

Offset

Register	Offset
SEC_CPU1_INT_MASK1	F9Ch

Function

This register can block different peripheral interrupt from IRQ bus to go to CPU1. If CPU1 is not a secure master, application might want to disable secure peripheral interrupts reaching that CPU.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	INT63_M...	INT62_M...	INT61_M...	INT60_M...	INT59_M...	INT58_M...	INT57_M...	INT56_M...	INT55_M...	INT54_M...	INT53_M...	INT52_M...	INT51_M...	INT50_M...	INT49_M...	INT48_M...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	INT47_M...	INT46_M...	INT45_M...	INT44_M...	INT43_M...	INT42_M...	INT41_M...	INT40_M...	INT39_M...	INT38_M...	INT37_M...	INT36_M...	INT35_M...	INT34_M...	INT33_M...	INT32_M...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-0 INTn_MASK	Mask bit 0b - Masked 1b - Not masked

23.4.1.57 Secure Interrupt Mask 2 for CPU1 (SEC_CPU1_INT_MASK2)

Offset

Register	Offset
SEC_CPU1_INT_MASK2	FA0h

Function

This register can block different peripheral interrupt from IRQ bus to go to CPU1. If CPU1 is not a secure master, application might want to disable secure peripheral interrupts reaching that CPU.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	INT95_M...	INT94_M...	INT93_M...	INT92_M...	INT91_M...	INT90_M...	INT89_M...	INT88_M...	INT87_M...	INT86_M...	INT85_M...	INT84_M...	INT83_M...	INT82_M...	INT81_M...	INT80_M...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	INT79_M...	INT78_M...	INT77_M...	INT76_M...	INT75_M...	INT74_M...	INT73_M...	INT72_M...	INT71_M...	INT70_M...	INT69_M...	INT68_M...	INT67_M...	INT66_M...	INT65_M...	INT64_M...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-0 INTn_MASK	Mask bit 0b - Masked 1b - Not masked

23.4.1.58 Secure Interrupt Mask 3 for CPU1 (SEC_CPU1_INT_MASK3)

Offset

Register	Offset
SEC_CPU1_INT_MASK3	FA4h

Function

This register can block different peripheral interrupt from IRQ bus to go to CPU1. If CPU1 is not a secure master, application might want to disable secure peripheral interrupts reaching that CPU.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	INT127_...	INT126_...	INT125_...	INT124_...	INT123_...	INT122_...	INT121_...	INT120_...	INT119_...	INT118_...	INT117_...	INT116_...	INT115_...	INT114_...	INT113_...	INT112_...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	INT111_...	INT110_...	INT109_...	INT108_...	INT107_...	INT106_...	INT105_...	INT104_...	INT103_...	INT102_...	INT101_...	INT100_...	INT99_M...	INT98_M...	INT97_M...	INT96_M...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-0 INTn_MASK	Mask bit 0b - Masked 1b - Not masked

23.4.1.59 Secure Interrupt Mask 4 for CPU1 (SEC_CPU1_INT_MASK4)

Offset

Register	Offset
SEC_CPU1_INT_MASK4	FA8h

Function

This register can block different peripheral interrupt from IRQ bus to go to CPU1. If CPU1 is not a secure master, application might want to disable secure peripheral interrupts reaching that CPU.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	INT159_...	INT158_...	INT157_...	INT156_...	INT155_...	INT154_...	INT153_...	INT152_...	INT151_...	INT150_...	INT149_...	INT148_...	INT147_...	INT146_...	INT145_...	INT144_...
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	INT143_...	INT142_...	INT141_...	INT140_...	INT139_...	INT138_...	INT137_...	INT136_...	INT135_...	INT134_...	INT133_...	INT132_...	INT131_...	INT130_...	INT129_...	INT128_...
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-0 INTn_MASK	Mask bit 0b - Masked 1b - Not masked

23.4.1.60 Secure Mask Lock (SEC_GP_REG_LOCK)

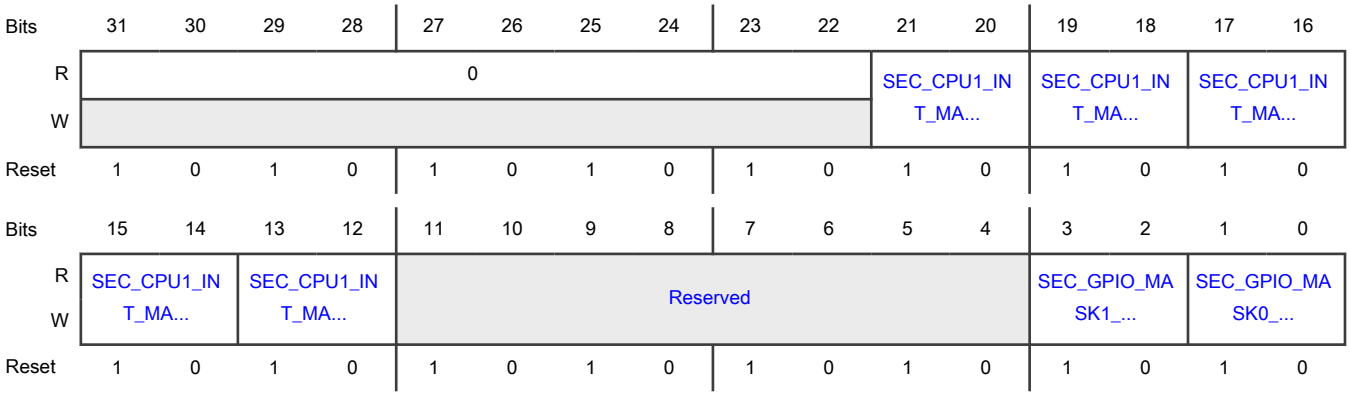
Offset

Register	Offset
SEC_GP_REG_LOCK	FBCh

Function

For the security of general purpose register access control. Each pair of lock bits can be written once. After changing the default value, the lock bits are locked too.

Diagram



Fields

Field	Function
31-22 —	Reserved; the value read from a reserved bit is not defined.
21-20 SEC_CPU1_INT_MASK4_LOCK	SEC_CPU1_INT_MASK4 Lock 00b - Reserved 01b - SEC_CPU1_INT_MASK4 cannot be written 10b - SEC_CPU1_INT_MASK4 can be written 11b - Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
19-18 SEC_CPU1_INT_MASK3_LOCK	SEC_CPU1_INT_MASK3 Lock 00b - Reserved 01b - SEC_CPU1_INT_MASK3 cannot be written 10b - SEC_CPU1_INT_MASK3 can be written 11b - Reserved
17-16 SEC_CPU1_INT_MASK2_LOCK	SEC_CPU1_INT_MASK2 Lock 00b - Reserved 01b - SEC_CPU1_INT_MASK2 cannot be written 10b - SEC_CPU1_INT_MASK2 can be written 11b - Reserved
15-14 SEC_CPU1_INT_MASK1_LOCK	SEC_CPU1_INT_MASK1 Lock 00b - Reserved 01b - SEC_GPIO_MASK1 cannot be written 10b - SEC_GPIO_MASK1 can be written 11b - Reserved
13-12 SEC_CPU1_INT_MASK0_LOCK	SEC_CPU1_INT_MASK0 Lock 00b - Reserved 01b - SEC_GPIO_MASK0 cannot be written 10b - SEC_GPIO_MASK0 can be written 11b - Reserved
11-4 —	Reserved
3-2 SEC_GPIO_MASK1_LOCK	Secure GPIO _MASK1 Lock 00b - Reserved 01b - SEC_GPIO_MASK1 cannot be written 10b - SEC_GPIO_MASK1 can be written 11b - Reserved
1-0 SEC_GPIO_MASK0_LOCK	Secure GPIO _MASK0 Lock 00b - Reserved 01b - SEC_GPIO_MASK0 cannot be written 10b - SEC_GPIO_MASK0 can be written 11b - Reserved

23.4.1.61 Master Secure Level (MASTER_SEC_LEVEL)

Offset

Register	Offset
MASTER_SEC_LEVEL	FD0h

Function

This register allows configuring security level for each master on AHB. Expectation is that application makes a static choice up front; programs and locks this register with the help of ROM. Once LOCK (bit 31-30) is applied, it can be unlocked only by system reset.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MASTER_SEC_LEVE...		USDHC		USB_HS		ETHERNET		USB_FS		Reserved		COOLFLUXI		NPUO	
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PQ		Reserved		PKC		eDMA1		eDMA0		SMARTDMA		CPU1		Reserved	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 MASTER_SEC_LEVEL_LOCK	Master SEC Level Lock 00b - Reserved 01b - MASTER_SEC_LEVEL_LOCK cannot be written 10b - MASTER_SEC_LEVEL_LOCK can be written 11b - Reserved
29-28 USDHC	uSDHC 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master
27-26 USB_HS	USB HS 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master

Table continues on the next page...

Table continued from the previous page...

Field	Function
	10b - Secure and non-privileged Master 11b - Secure and privileged Master
25-24 ETHERNET	Ethernet 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master
23-22 USB_FS	USB_FS 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master
21-20 —	Reserved
19-18 COOLFLUXI	Coolflux Instruction 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master
17-16 NPUO	NPU Operands 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master
15-14 PQ	PowerQuad 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master
13-12 —	Reserved
11-10	PKC

Table continues on the next page...

Table continued from the previous page...

Field	Function
PKC	00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master
9-8 eDMA1	eDMA1 **NOTE** The final eDMA channel transaction security level is controlled by both eDMA channel configuration and this bit field. It is the least level of both. For example, if eDMA channel transaction security level is non-secure and non-privileged with its channel configuration, but this field configures eDMA as secure and privileged master, the final channel transaction security level is non-secure and non-privileged. 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master
7-6 eDMA0	eDMA0 **NOTE** The final eDMA channel transaction security level is controlled by both eDMA channel configuration and this bit field. It is the least level of both. For example, if eDMA channel transaction security level is non-secure and non-privileged with its channel configuration, but this field configures eDMA as secure and privileged master, the final channel transaction security level is non-secure and non-privileged. 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master
5-4 SMARTDMA	SMARTDMA Data **NOTE** TrustZone aware peripherals (DMA, RGPIO, FMU, FlexSPI, CMX_PERFMON. and PUF_CTRL) will always see SmartDMA accesses as non-privileged even if a privileged option is selected here. When accessing these peripherals always use either non-secure and non-privileged or secure and non-privileged setting. 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master

Table continues on the next page...

Table continued from the previous page...

Field	Function
	11b - Secure and privileged Master
3-2 CPU1	CPU1 **NOTE** The final privilege level is controlled by both CPU1 itself and this bit field. It is the least level of both. For example, if CPU1 initiates a non-privileged transaction, but this field configures CPU1 as privileged master, the final privilege level is non-privileged. The secure and non-secure attribute is solely controlled by this bit field because CPU1 can only initiate non-secure transaction by itself. 00b - Non-secure and non-privileged Master 01b - Non-secure and privileged Master 10b - Secure and non-privileged Master 11b - Secure and privileged Master
1-0 —	Reserved; the value read from a reserved bit is not defined.

23.4.1.62 Master Secure Level (MASTER_SEC_ANTI_POL_REG)

Offset

Register	Offset
MASTER_SEC_ANTI_POL_REG	FD4h

Function

The fields in this register must be equal to the same fields in the MASTER_SEC_LEVEL register, but with inverse polarity.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MASTER_SEC_LEVE...		USDHC		USB_HS		ETHERNET		USB_FS		Reserved		COOLFLUXI		NPUO	
W																
Reset	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PQ		Reserved		PKC		eDMA1		eDMA0		SMARTDMA		CPU1		Reserved	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fields

Field	Function
31-30 MASTER_SEC_LEVEL_ANTIPOL_LOCK	Master SEC Level Antipol Lock 00b - Reserved 01b - MASTER_SEC_LEVEL_LOCK cannot be written 10b - MASTER_SEC_LEVEL_LOCK can be written 11b - Reserved
29-28 USDHC	uSDHC 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
27-26 USB_HS	USB HS 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
25-24 ETHERNET	Ethernet 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
23-22 USB_FS	USB_FS 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
21-20 —	Reserved
19-18 COOLFLUXI	Coolflux Instruction 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master

Table continues on the next page...

Table continued from the previous page...

Field	Function
17-16 NPUO	NPU Operands 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
15-14 PQ	PowerQuad 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
13-12 —	Reserved
11-10 PKC	PKC 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
9-8 eDMA1	eDMA1 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
7-6 eDMA0	eDMA0 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
5-4 SMARTDMA	SMARTDMA Data **NOTE** TrustZone aware peripherals (DMA, RGPIO, FMU, FlexSPI, CMX_PERFMON. and PUF_CTRL) will always see SmartDMA accesses as non-privileged even if a privileged option is selected here. When accessing these peripherals always use either non-secure and non-privileged or secure and non-privileged setting.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
3-2 CPU1	CPU1 00b - Secure and privileged Master 01b - Secure and non-privileged Master 10b - Non-secure and privileged Master 11b - Non-secure and non-privileged Master
1-0 —	Reserved; the value read from a reserved bit is not defined.

23.4.1.63 Miscellaneous CPU0 Control Signals (CPU0_LOCK_REG)

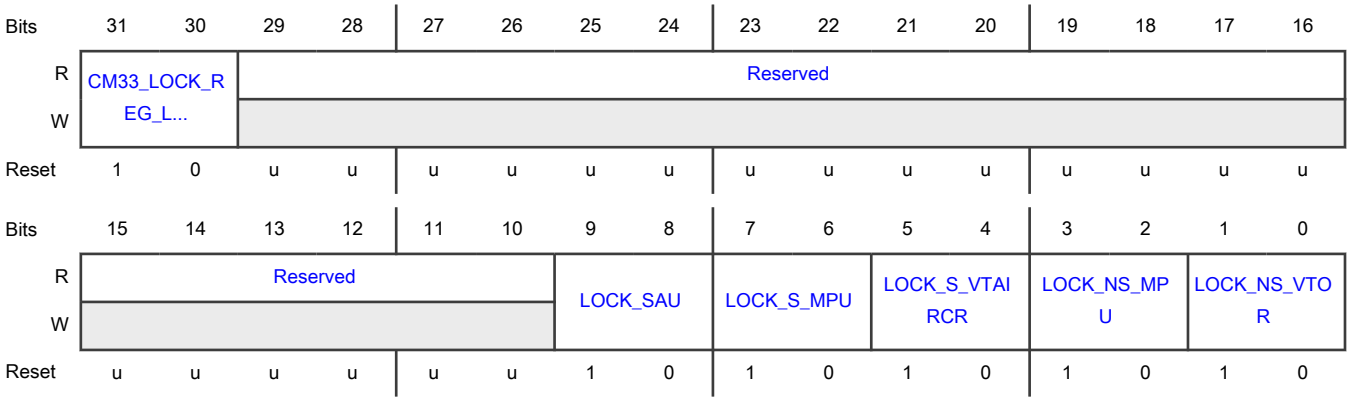
Offset

Register	Offset
CPU0_LOCK_REG	FECh

Function

This register drives certain input ports of CPU0, providing capability to lock the settings or enhanced security.

Diagram



Fields

Field	Function
31-30 CM33_LOCK_REG_LOCK	CM33_LOCK_REG_LOCK Disables write access to this register itself. Once locked, only system reset can unlock (to enable write access). 00b - Reserved 01b - CM33_LOCK_REG_LOCK is 1 10b - CM33_LOCK_REG_LOCK is 0 11b - Reserved
29-10 —	Reserved; the value read from a reserved bit is not defined.
9-8 LOCK_SAU	LOCK_SAU SAU is the Security Attribution Unit. 00b - Reserved 01b - CM33 (CPU0) LOCK_SAU is 1 10b - CM33 (CPU0) LOCK_SAU is 0 11b - Reserved
7-6 LOCK_S_MPU	LOCK_S_MPU 00b - Reserved 01b - CM33 (CPU0) LOCK_S_MPU is 1 10b - CM33 (CPU0) LOCK_S_MPU is 0 11b - Reserved
5-4 LOCK_S_VTAIRCR	LOCK_S_VTAIRCR 00b - Reserved 01b - CM33 (CPU0) LOCK_S_VTAIRCR is 1 10b - CM33 (CPU0) LOCK_S_VTAIRCR is 0 11b - Reserved
3-2 LOCK_NS_MPU	LOCK_NS_MPU 00b - Reserved 01b - CM33 (CPU0) LOCK_NS_MPU is 1 10b - CM33 (CPU0) LOCK_NS_MPU is 0 11b - Reserved
1-0	LOCK_NS_VTOR 00b - Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
LOCK_NS_VTOR	01b - CM33 (CPU0) LOCKNSVTOR is 1 10b - CM33 (CPU0) LOCKNSVTOR is 0 11b - Reserved

23.4.1.64 Miscellaneous CPU1 Control Signals (CPU1_LOCK_REG)

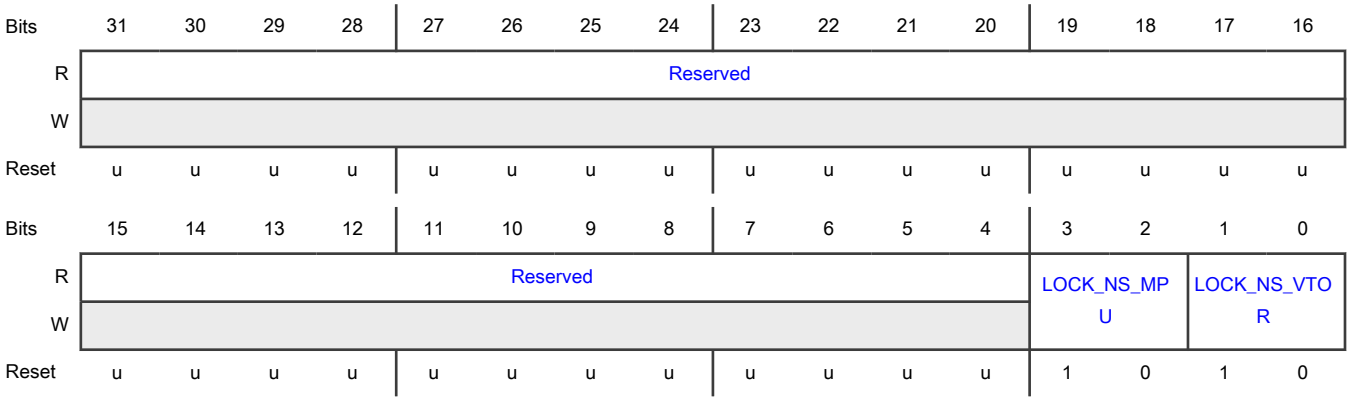
Offset

Register	Offset
CPU1_LOCK_REG	FF0h

Function

This register drives certain input ports of CPU1, providing capability to lock the settings or enhanced security.

Diagram



Fields

Field	Function
31-4 —	Reserved; the value read from a reserved bit is not defined.
3-2 LOCK_NS_MPU	LOCK_NS_MPU 00b - Reserved 01b - CM33 (CPU0) LOCK_NS_MPU is 1 10b - CM33 (CPU0) LOCK_NS_MPU is 0 11b - Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
1-0 LOCK_NS_VTOR	LOCK_NS_VTOR 00b - Reserved 01b - CM33 (CPU0) LOCKNSVTOR is 1 10b - CM33 (CPU0) LOCKNSVTOR is 0 11b - Reserved

23.4.1.65 Secure Control Duplicate (MISC_CTRL_DP_REG)

Offset

Register	Offset
MISC_CTRL_DP_REG	FF8h

Function

This register is duplicate of MISC_CTRL_REG. A secondary register with duplicate programing is implemented to provide better protection against malicious hacking attacks such as glitch attack.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IDAU_ALL_NS		Reserved		DISABLE_STRICT_...		DISABLE_VIOLATI...		ENABLE_NS_PRIV_...		ENABLE_S_PRIV_C...		ENABLE_SECURE_C...		WRITE_LOCK	
Reset	1	0	0	0	0	1	1	0	1	0	1	0	0	1	1	0

Fields

Field	Function
31-16 —	Reserved
15-14 IDAU_ALL_NS	IDAU All Non-Secure IDAU is the Implementation Defined Attribution Unit.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	IDAU will be enabled if the value doesn't match MISC_CTRL_REG[IDAU_ALL_NS], independently of its value. 00b - Reserved 01b - IDAU is disabled, which means that all memories are attributed as non-secure memory. 10b - IDAU is enabled (restrictive mode) 11b - Reserved
13-12 ——	Reserved
11-10 DISABLE_STRICT_MODE	Disable Strict Mode A master can perform both data and code access and execute code. Master strict mode only works when ENABLE_SECURE_CHECKING is on. If ENABLE_SECURE_CHECKING is off, then master strict mode is disabled, independently of its value. When enabling ENABLE_SECURE_CHECKING, strict modes start as enabled. Master strict mode will be on if the value doesn't match MISC_CTRL_REG[DISABLE_STRICT_MODE], independently of its value. 00b - Reserved 01b - Master can access memories and peripherals at the same level or below that level. 10b - Master can access memories and peripherals at same level only 11b - Reserved
9-8 DISABLE_VIOLATION_ABORT	Disable Violation Abort The violation access will not cause an abort if the value doesn't match MISC_CTRL_REG[DISABLE_VIOLATION_ABORT], independently of its value. 00b - Reserved 01b - The violation detected by the secure checker will not cause an abort, but a secure_violation_irq (interrupt request) will still be asserted and serviced by ISR. 10b - The violation detected by the secure checker will cause an abort. 11b - Reserved
7-6 ENABLE_NS_PRIV_CHECK	Enable Non-Secure Privilege Checking The bit field enables privilege checking when non-secure mode access occurs. Non-secure privilege checking will be enabled if the value doesn't match MISC_CTRL_REG[ENABLE_S_PRIV_CHECK], independently of its value. 00b - Reserved 01b - Enables the privilege checking of non-secure mode access. 10b - Disables the privilege checking of non-secure mode access. 11b - Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
5-4 ENABLE_S_PRIV_CHECK	Enable Secure Privilege Checking The bit field enables privilege checking when secure mode access occurs. Secure privilege checking will be enabled if the value doesn't match MISC_CTRL_REG[ENABLE_S_PRIV_CHECK], independently of its value. 00b - Reserved 01b - Enables the privilege checking of secure mode access. 10b - Disables the privilege checking of secure mode access. 11b - Reserved
3-2 ENABLE_SECURE_CHECKING	Enable Secure Checking The bit fields provide privilege checking when secure mode access occurs. Reset value of these bits is 2'b01 before boot ROM code but it can be changed to 2'b10 by boot ROM code before jumping to user code. Secure checking will be enabled if the value does not match MISC_CTRL_DP_REG[ENABLE_SECURE_CHECKING], independently of its value. 00b - Reserved 01b - Enables secure checking. Violation can be detected when the security level of a transaction does not meet the security rule of the slave or memory to be accessed. 10b - Disables secure checking. Even if the security level of a transaction does not conform to the security rule of the slave or memory, it will not be detected as a violation. 11b - Reserved
1-0 WRITE_LOCK	Write Lock The bit fields can lock writing to MISC_CTRL_REG, MISC_CTRL_DP_REG and all the secure control rule registers. After lock, these registers will be read-only, and cannot be unlocked. If this does not match MISC_CTRL_REG[1:0], the registers under control will also be write-locked. 00b - Reserved 01b - Writes to this register and to the Memory and Peripheral RULE registers are not allowed 10b - Writes to this register and to the Memory and Peripheral RULE registers are allowed 11b - Reserved

23.4.1.66 Secure Control (MISC_CTRL_REG)

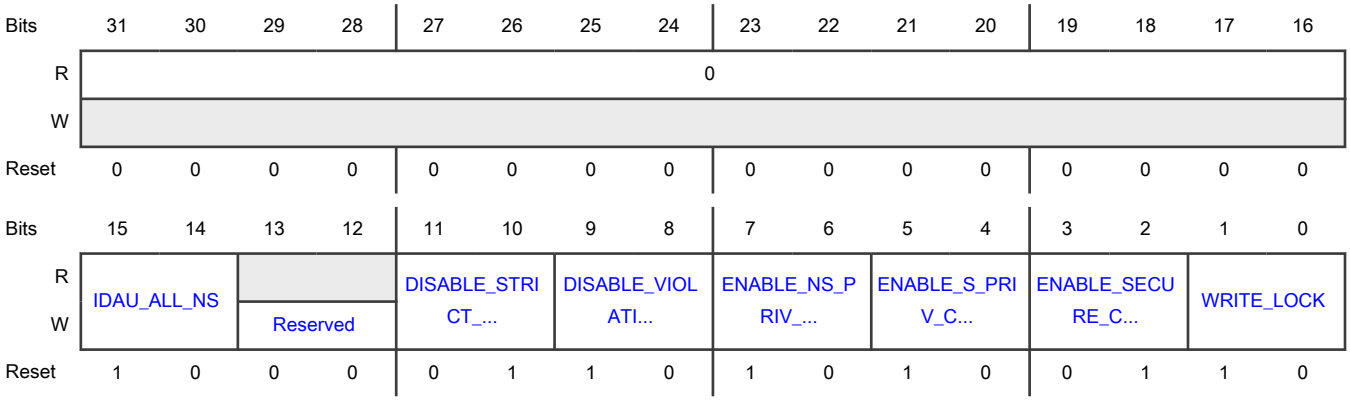
Offset

Register	Offset
MISC_CTRL_REG	FFCh

Function

This register provides more control over certain system behavior as described in individual fields.

Diagram



Fields

Field	Function
31-16 —	Reserved
15-14 IDAU_ALL_NS	IDAU All Non-Secure IDAU is the Implementation Defined Attribution Unit. IDAU will be enabled if the value doesn't match MISC_CTRL_DP_REG[IDAU_ALL_NS], independently of its value. 00b - Reserved 01b - IDAU is disabled, which means that all memories are attributed as non-secure memory. 10b - IDAU is enabled (restrictive mode) 11b - Reserved
13-12 —	Reserved
11-10 DISABLE_STRICT_MODE	Disable Strict Mode A master can perform both data and code access and execute code. Master strict mode only works when ENABLE_SECURE_CHECKING is on. If ENABLE_SECURE_CHECKING is off, then master strict mode is disabled, independently of its value. When enabling ENABLE_SECURE_CHECKING, strict modes start as enabled. Master strict mode will be on if the value doesn't match MISC_CTRL_DP_REG[DISABLE_STRICT_MODE], independently of its value. 00b - Reserved 01b - Master strict mode is on and can access memories and peripherals at the same level or below that level 10b - Master strict mode is disabled and can access memories and peripherals at same level only 11b - Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
9-8 DISABLE_VIOLATION_ABORT	Disable Violation Abort The violation access will not cause an abort if the value doesn't match MISC_CTRL_DP_REG[DISABLE_VIOLATION_ABORT], independently of its value. 00b - Reserved 01b - The violation detected by the secure checker will not cause an abort, but a secure_violation_irq (interrupt request) will still be asserted and serviced by ISR. 10b - The violation detected by the secure checker will cause an abort. 11b - Reserved
7-6 ENABLE_NS_PRIV_CHECK	Enable Non-Secure Privilege Checking The bit fields provide privilege checking enable when non-secure mode access occurs. Non-secure privilege checking will be enabled if the value doesn't match MISC_CTRL_DP_REG[ENABLE_NS_PRIV_CHECK], independently of its value. 00b - Reserved 01b - Enables privilege checking of non-secure mode access. 10b - Disables privilege checking of non-secure mode access is disabled. 11b - Reserved
5-4 ENABLE_S_PRIV_CHECK	Enable Secure Privilege Checking The bit fields provide privilege checking enable when secure mode access occurs. Secure privilege checking will be enabled if the value doesn't match MISC_CTRL_DP_REG[ENABLE_S_PRIV_CHECK], independently of its value. 00b - Reserved 01b - Enables privilege checking of secure mode access. 10b - Disables privilege checking of secure mode access. 11b - Reserved
3-2 ENABLE_SECURE_CHECKING	Enable Secure Checking The bit fields provide privilege checking when secure mode access occurs. Reset value of these bits is 2'b01 before boot ROM code but it can be changed to 2'b10 by boot ROM code before jumping to user code. Secure checking will be enabled if the value does not match MISC_CTRL_DP_REG[ENABLE_SECURE_CHECKING], independently of its value. 00b - Reserved 01b - Enables secure checking. Violation can be detected when the security level of a transaction does not meet the security rule of the slave or memory to be accessed. 10b - Disables secure checking. Even if the security level of a transaction does not conform to the security rule of the slave or memory, it will not be detected as a violation. 11b - Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
1-0 WRITE_LOCK	Write Lock The bit fields can lock writing to MISC_CTRL_REG, MISC_CTRL_DP_REG and all the secure control rule registers. After lock, these registers will be read-only, and cannot be unlocked. If this does not match MISC_CTRL_DP_REG[1:0], the registers under control will also be write-locked. 00b - Reserved 01b - Writes to this register and to the Memory and Peripheral RULE registers are not allowed 10b - Writes to this register and to the Memory and Peripheral RULE registers are allowed 11b - Reserved

Chapter 24
Flash Controller (FMC)

24.1 Chip-specific FMC information

Table 496. Reference links to related information[1]

Topic	Related module	Reference
Full description	NPX	NPX
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer to the MCX Nx4x Reference Manual.

NOTE

See NVM_CTRL register in SYSCON chapter of MCX Nx4x Security Reference Manual for details on flash speculation and flash cache control.

24.1.1 Module instances

This device has one instance of the NVM PRINCE Encryption and Decryption (NPX) module, NPX0.

24.1.2 Security considerations

For secure applications, NXP recommends that this module is configured for secure and privileged access only to prevent unintentional or malicious modification of the system operation by nonsecure software. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

24.1.3 PRINCE On-the-Fly encrypt/decrypt

The device supports upto four decrypt/encrypt regions, each region can be up to 2 MB wide. All four regions have start address 0x0 and all regions overlap. Each sub-region has 32 KB granularity and crypto enable per sub-region is register programmable. Each region has a dedicated pair of Secret Key and IV. This allows multiple images to reside in the Flash with independent encryption base. It needs to be ensured through programming that a sub-region is not setup for de/encryption by more than one KEY/IV pair.

24.1.4 ECC error check

The flash includes ECC checks on the memory contents. Any correctable, single-bit errors are automatically corrected. Non-correctable, double-bit errors are reported through the Error Recording Module (ERM). See the ERM chapter in MCX Nx4x Reference Manual.

24.2 Overview

The Flash Memory Controller (FMC) is a memory interface and acceleration unit providing:

- An interface between the device and the nonvolatile memory
- Buffers that can accelerate flash memory transfers
  - A flash-phrase-sized buffer (128 bits) holds the most recently accessed flash phrase.
  - An optional flash-phrase-sized speculation buffer can prefetch the next flash phrase.

- An optional 4-way, set-associative cache can store previously accessed flash phrases.

- An NVM Prince Counter Mode (CM) module called NPX to optionally perform On-The-Fly (OTF) read decryption and write encryption of flash memory contents

The FMC manages the interface between the device and the flash memory. The FMC receives status information describing the configuration of the memory and uses this information to ensure a proper interface. The following table shows the supported read and write operations.

Flash memory type	Read	Write
Program, IFR, IFR1 flash memory	8-bit, 16-bit, and 32-bit reads	16-byte and 128-byte writes

The FMC is controlled by a programmer's model external to the FMC module; see the chip-specific FMC information for details. The FMC's programming model provides a very configurable, high performance flexible memory controller, which can be optimized for the runtime characteristics of specific applications.

NOTE

Program the FMC's controls only while the flash controller is idle. Changing the configuration settings while a flash access is in progress can cause non-deterministic, unpredictable behavior.

24.2.1 Features

- Interface between the device and the flash memory:
  - The FMC's input bus supports 8-bit, 16-bit, and 32-bit read operations to flash memory.
  - The FMC's flash memory interface fetches a 128-bit flash phrase.
  - For input read requests, the FMC fetches a flash phrase with the desired read data from flash memory.
  - The flash memory interface can write aligned 16-byte flash write phrases or aligned 128-byte flash write pages to flash memory.
  - The FMC has a 16-byte aligned write buffer. This buffer is used once for aligned 16-byte flash write phrases or eight times for aligned 128-byte flash write pages.
  - The FMC's input bus supports 32-bit write operations for flash memory writes to fill the FMC's write buffer.
  - For input write requests, the FMC must receive the 4-word write of an aligned phrase in order.
- Acceleration of data transfer from flash memory to the device:
  - A flash-phrase-sized buffer that holds the current decrypted flash phrase fetched due to a FMC read request. Subsequent FMC read requests *that hit in the current buffer* return data with no wait states.
  - A flash-phrase-sized prefetch speculation buffer with controls for prefetching on instructions and/or data reads. When prefetching is enabled, idle FMC-to-flash interface cycles are used to fetch the next sequential flash phrase and hold it in the prefetch buffer. Subsequent FMC read requests *that hit in the speculation buffer* return data with no wait states.
  - Input controls:
    - to disable data type speculation
    - to disable all speculation
    - to invalidate the current and speculation buffers

    See the chip-specific section for details about controls.
  - The flash cache has input controls:
    - to disable instruction caching
    - to disable operand caching
    - to disable all caching

- to clear the cache

See the chip-specific section for details about controls.

— The size of the flash cache in bytes is calculated as follows:

flash cache size = [number of ways] × [number of sets] × [flash phrase size (in bytes)]

For example, a flash cache with 4 ways, 1 set, and a 128-bit flash phrase (= 16 bytes) has a total flash cache size = 64 bytes

(4 ways) × (1 set) × (16 bytes per flash phrase) = 64 bytes, the size of the flash cache

NOTE

Clear the speculation buffer and flash cache before accessing recently modified flash addresses. The flash cache has a specific clear control bit. To clear the speculation buffer, first disable then re-enable the speculation via other modules.

24.3 Functional description

The FMC is a flash interface and acceleration unit, with flexible buffers for user configuration.

- The FMC's input bus can operate faster than the flash memory.
- The FMC-to-flash interface has flow control to add wait states as needed (for input bus reads that need flash accesses).
- The FMC also contains various configurable buffers that hold recent flash accesses. If an input bus read hits a valid buffer, then that access will complete with no wait states.

24.3.1 Modes of operation

The FMC only operates when a bus master accesses the flash memory.

For any device power mode where the flash memory cannot be accessed, FMC is disabled automatically.

24.3.2 Default configuration

After system reset, the FMC is configured to provide a significant level of buffering for transfers from the flash memory. For all banks:

- The current and speculation buffers are cleared by reset.
- Prefetch support for data and instructions is enabled.
- The cache is cleared by reset.
- The cache is configured for data or instruction replacement.

24.3.3 Configuration options

The default configuration provides a high degree of flash acceleration, but advanced users may want to customize the FMC buffer configurations to maximize throughput for their use cases. When reconfiguring the FMC for custom use cases, do not program the FMC's control registers while the flash memory is being accessed. Instead, change the control registers with a routine executing from RAM in Supervisor mode.

The FMC's cache and buffering controls allow other modules to tune resources to suit specific application requirements. The cache and buffer are each controlled individually. The controls enable buffering and prefetching per memory bank and access type (instruction fetch or data reference).

As an application example: if both instruction fetches and data references are accessing flash memory, then control is available to send instruction fetches, data references, or both to the cache or the single-entry buffer. Likewise, speculation can be enabled or disabled for either type of access.

In another application example, the cache can be configured for replacement from bank 0, while the single-entry buffer can be enabled for bank 1 only. This configuration is ideal for applications that use bank 0 for program space and bank 1 for data space.

For best performance, the FMC cache and speculation buffering should be enabled for instruction, data fetching, or both. The following is recommended for best performance:

- If the speculation buffer is enabled for both instruction and data speculation, enable the flash cache for both instruction and data caching.
- If the speculation buffer is enabled for instruction speculation only, enable the flash cache for at least instruction caching.

24.3.4 Wait states

Because the core, crossbar switch, and bus masters can be clocked at a higher frequency than the flash clock, flash memory accesses that do not hit in the speculation buffer or cache usually require wait states.

FMC does not allow the configuration of wait states directly. Wait states can be controlled via FMU FCTRL[RWSC].

24.3.5 Speculative reads

The FMC has a single buffer that reads ahead to the next phrase in the flash memory if there is an idle cycle. Speculative prefetching is programmable for instruction and data accesses. Because many code accesses are sequential, using the speculative prefetch buffer improves performance in most cases.

See the chip-specific section for information about controlling speculative reads.

When speculative reads are enabled, the FMC immediately requests the next sequential phrase address after a read completes. By requesting the next phrase immediately, speculative reads can help to reduce or even eliminate wait states when accessing sequential code and/or data.

24.3.6 NPX submodule

NPX is the FMC NVM On-The-Fly PRINCE Encryption and Decryption submodule. A trend in embedded processor design is an increasing need for hardware to support cryptographic calculations that are required for system security. There are emerging customer requirements to protect application code and data stored in flash memories in an encrypted form, and to have the embedded processor provide hardware support for "on-the-fly" decryption (meaning that the data is automatically decrypted as the data is loaded or saved). The flash memory image of the code and data is always stored in an encrypted format, and in response to processor references to the address space, the memory image is decrypted on the fly, returning the original value to the requesting bus master.

The FMC Controller's NPX (NVM PRINCE Encryption and Decryption) submodule provides both "on-the-fly" decryption and encryption. The "on-the-fly" encryption is used to program the flash with encrypted data, without needing external support to pre-encrypt the program data. Using this method, there is no need for external knowledge of keys for the system to benefit from the additional security of encrypted flash memory.

A cryptographic symmetric key block cipher algorithm is used for the encryption and decryption. The specific algorithm used is PRINCE. PRINCE is a symmetric key cipher, originally developed by NXP-Leuven and 3 European universities, that provides a lightweight hardware implementation cost, but still provides strong cryptographic protection. PRINCE operates on 64-bit (8-byte) data blocks with 128-bit secret keys.

The On-the-Fly (OTF) PRINCE engine operates in conjunction with the internal flash memory controller.

The NPX OTF encrypt/decrypt engine provides superior cryptographic capabilities without compromising system performance for the embedded processor applications that require enhanced security.

24.3.6.1 NPX features

- PRINCE On-the-Fly Encryption and Decryption
  - 128-bit keys and 64-bit data block sizes
- Hardware supports bitmaps for marking each 32-KByte region of flash as not being encrypted or as using 1 of 4 independent encryption/decryption keys. The keys are 128 bits each and unreadable.

- Functionally acts as a slave submodule to the Flash Memory Controller (FMC)
  - Logically connected between the FMC flash cache and the FMC flash interface
  - Programming model is mapped to the FMC peripheral address space
- Hardware microarchitecture
  - 128-bit (16-byte) input/output buses match flash read/write width
  - Single cycle encryption or decryption engine using 2 parallel PRINCE instantiations
  - Optimized for 32-bit WRAP4 bursts (CPU cache miss fetch size, typical DMA fetch size)

24.3.6.2 NPX block diagram

The NPX is directly connected to the Flash Memory Controller (FMC).

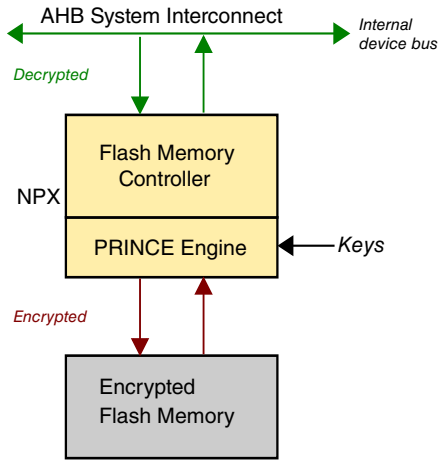


Figure 101. NPX block diagram

24.3.6.3 NPX modes of operation

The NPX operating modes are controlled and reported in the NPX Control Register (NPXCR):

- Global Decryption Enable (GDE) field
  - If GDE=0, then all decryption is disabled.
  - If GDE=1, then for reads that hit a valid context (memory region), NPX performs on-the-fly data decryptions.
- Global Encryption Enable (GEE) field
  - If GEE=0, then all encryption is disabled.
  - If GEE=1, then for writes that hit a valid context (memory region), NPX performs on-the-fly data encryptions.

24.3.6.4 Obfuscating cache data

When the flash cache is enabled, each 32-bit word of data is automatically exclusive OR'ed with the cache obfuscation mask (CACMSK) before being stored in the cache. Cache data is unmasked when read. To modify CACMSK:

1. Disable the flash cache.

2. Modify CACMSK.
3. Clear the cache.
4. Re-enable the cache.

24.3.6.5 Remapping flash addresses

An address remap mechanism allows for the swapping (both ways) of a specified flash address range between flash bank 0 and flash bank 1. This remapping can facilitate software updates in the field without the need for address modifications in software. For example, to use the remap mechanism to update software in the lower portion of bank 0, follow these steps:

1. Load the updated software into the lower portion of flash bank1.
2. Enable remapping to automatically use the updated software.

In REMAP, remapping is enabled when LIM[20:16] = LIMDP[28:24], and LIM[20:16] is nonzero. LIM[20:16] and LIMDP[28:24] define the *remap_address*[19:15] for the remapping, providing an address range granularity of 32 KB. For a remapped FMC access to address *access_address*, see Table 497.

Table 497. Remapping *access_address*

When *access_address* originates in...	And is less than or equal to...	The access remaps to...	In...
flash bank 0	*remap_address*[19:15]	bank1_base + access_address	bank 1
flash bank 1	bank1_base + *remap_address*[19:15]	*access_address* – bank1_base	bank 0

The address ranges that may be swapped depends on the total flash size available. For non-power-of-2 total flash sizes, the upper half of flash bank 0 cannot be swapped, as shown in Table 498.

Table 498. Remapping address ranges

For a flash size of...	This bank 0 range...	Remaps to this bank 1 range...	This bank 0 range cannot be remapped...
2 MB	0x00_0000 – 0x0F_FFFF	0x10_0000 – 0x1F_FFFF	
1536 KB	0x00_0000 – 0x07_FFFF	0x10_0000 – 0x17_FFFF	0x08_0000 – 0x0F_FFFF
1 MB	0x00_0000 – 0x07_FFFF	0x08_0000 – 0x0F_FFFF	
768 KB	0x00_0000 – 0x03_FFFF	0x08_0000 – 0x0B_FFFF	0x04_0000 – 0x07_FFFF
512 KB	0x00_0000 – 0x03_FFFF	0x04_0000 – 0x07_FFFF	
384 KB	0x0_0000 – 0x1_FFFF	0x4_0000 – 0x5_FFFF	0x2_0000 – 0x3_FFFF
256 KB	0x0_0000 – 0x1_FFFF	0x2_0000 – 0x3_FFFF	—
192 KB	0x0_0000 – 0x0_FFFF	0x2_0000 – 0x2_FFFF	0x1_0000 – 0x1_FFFF
128 KB	0x0_0000 – 0x0_FFFF	0x1_0000 – 0x1_FFFF	—
96 KB	0x0_0000 – 0x0_7FFF	0x1_0000 – 0x1_7FFF	0x0_8000 – 0x0_FFFF
64 KB	0x0_0000 – 0x0_7FFF	0x0_8000 – 0x0_FFFF	—
48 KB	0x0_0000 – 0x0_3FFF	0x0_8000 – 0x0_BFFF	0x0_4000 – 0x0_7FFF
32 KB	0x0_0000 – 0x0_3FFF	0x0_4000 – 0x0_7FFF	—

When LIM = LIMDP = 0, FMC disables the remap function. When LIM = LIMDP = a nonzero value, the remapping address range is (LIM + 1) × 32 KB. For example:

- When LIM = LIMDP = 1, the range is ≤ 64 KB.
- When LIM = LIMDP = 2, the range is ≤ 128 KB.

24.3.6.6 NPX initialization

The operating configuration of the NPX is controlled by programmable bits in NPXCR.

24.3.6.7 Interrupts

This module has no interrupts.

24.4 External signals

The FMC has no external signals.

24.5 Initialization and application information

The FMC does not require user initialization. Flash acceleration features are enabled by default.

The FMC has no visibility into flash memory erase and program cycles because the Flash Memory module manages them directly. To prevent the possibility of returning stale data when an application is executing flash memory commands, disable and/or flush FMC's current buffer, speculation buffer, and cache. While you can disable these features individually, we recommend disabling all of them when executing flash memory commands.

See the chip-specific section for details about the registers used to disable features.

24.6 Register descriptions

The NPX (NVM Prince Module) register set has a control register and a status register, plus information for 4 memory contexts. Each memory context has a 64-bit Block Valid bits register and a 64-bit Block IV register. The Block Valid registers hold bitmaps to indicate if a given 32-KByte block within the 2MByte maximum size flash is encrypted/decrypted and by what key. The Block IV registers hold additional encrypt/decrypt information.

The NPX register set also has a Flash Cache Mask register and a Flash Remap Control register.

NOTE

See the chip-specific information for FMC controls.

NOTE

When defining the memory contexts, software must ensure the contexts do not overlap.

NOTE

Any access to an undefined memory area results in a bus error.

24.6.1 NPX register descriptions

24.6.1.1 NPX memory map

NPX0 base address: 400C_C000h

Offset	Register	Width (In bits)	Access	Reset value
0h	NPX Control Register (NPXCR)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
8h	NPX Status Register (NPXSR)	32	R	0000_0004h
10h	Flash Cache Obfuscation Mask (CACMSK)	32	W	0000_0000h
20h	Data Remap (REMAP)	32	RW	0000_0000h
40h	Bitmap of Valid Control for Memory Context 0 (VMAPCTX0_WD0)	32	RW	0000_0000h
44h	Bitmap of Valid Control for Memory Context 0 (VMAPCTX0_WD1)	32	RW	0000_0000h
48h	Block Initial Vector for Memory Context 0 (BIVCTX0_WD0)	32	W	0000_0000h
4Ch	Block Initial Vector for Memory Context 0 (BIVCTX0_WD1)	32	W	0000_0000h
50h	Bitmap of Valid Control for Memory Context 1 (VMAPCTX1_WD0)	32	RW	0000_0000h
54h	Bitmap of Valid Control for Memory Context 1 (VMAPCTX1_WD1)	32	RW	0000_0000h
58h	Block Initial Vector for Memory Context 1 (BIVCTX1_WD0)	32	W	0000_0000h
5Ch	Block Initial Vector for Memory Context 1 (BIVCTX1_WD1)	32	W	0000_0000h
60h	Bitmap of Valid Control for Memory Context 2 (VMAPCTX2_WD0)	32	RW	0000_0000h
64h	Bitmap of Valid Control for Memory Context 2 (VMAPCTX2_WD1)	32	RW	0000_0000h
68h	Block Initial Vector for Memory Context 2 (BIVCTX2_WD0)	32	W	0000_0000h
6Ch	Block Initial Vector for Memory Context 2 (BIVCTX2_WD1)	32	W	0000_0000h
70h	Bitmap of Valid Control for Memory Context 3 (VMAPCTX3_WD0)	32	RW	0000_0000h
74h	Bitmap of Valid Control for Memory Context 3 (VMAPCTX3_WD1)	32	RW	0000_0000h
78h	Block Initial Vector for Memory Context 3 (BIVCTX3_WD0)	32	W	0000_0000h
7Ch	Block Initial Vector for Memory Context 3 (BIVCTX3_WD1)	32	W	0000_0000h

24.6.1.2 NPX Control Register (NPXCR)

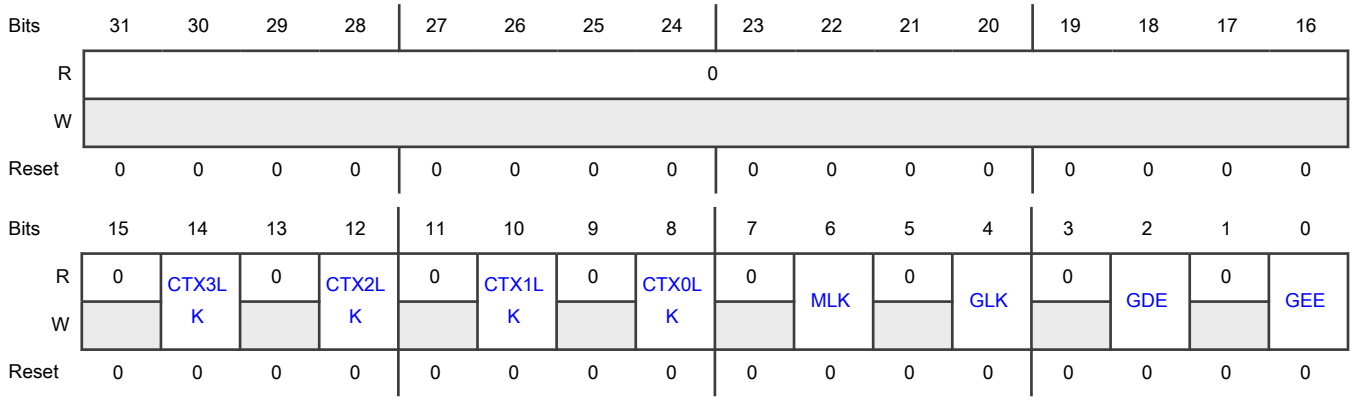
Offset

Register	Offset
NPXCR	0h

Function

Contains enables for Global Lock, Context Locks, Mask Lock, Global Decryption and Global Encryption.

Diagram



Fields

Field	Function
31-15 —	This read-only field is reserved and always has the value 0
14 CTX3LK	Lock Enable for Context 3 Affects the register access properties of VMAPCTX3. BIVCTX3 is not affected and remains write-only. Set by software (sticky); cleared only by a reset. 0b - Lock disabled: VMAPCTX3 remains read-write 1b - Lock enabled: cannot write to VMAPCTX3 (becomes read-only)
13 —	This read-only field is reserved and always has the value 0
12 CTX2LK	Lock Enable for Context 2 Affects the register access properties of VMAPCTX2. BIVCTX2 is not affected and remains write-only. Set by software (sticky); cleared only by a reset. 0b - Lock disabled: VMAPCTX2 remains read-write 1b - Lock enabled: cannot write to VMAPCTX2 (becomes read-only)
11 —	This read-only field is reserved and always has the value 0
10 CTX1LK	Lock Enable for Context 1 Affects the register access properties of VMAPCTX1. BIVCTX1 is not affected and remains write-only. Set by software (sticky); cleared only by a reset. 0b - Lock disabled: VMAPCTX1 remains read-write 1b - Lock enabled: cannot write to VMAPCTX1 (becomes read-only)

Table continues on the next page...

Table continued from the previous page...

Field	Function
9 —	This read-only field is reserved and always has the value 0
8 CTX0LK	Lock Enable for Context 0 Affects the register access properties of VMAPCTX0. BIVCTX0 is not affected and remains write-only. Set by software (sticky); cleared only by a reset. 0b - Lock disabled: VMAPCTX0 remains read-write 1b - Lock enabled: cannot write to VMAPCTX0 (becomes read-only)
7 —	This read-only field is reserved and always has the value 0
6 MLK	Mask Lock Enable Set by software (sticky); cleared only by a reset. 0b - Lock disabled. Subsequent reads return 0. 1b - Lock enabled: cannot write to mask. Subsequent reads return 1.
5 —	This read-only field is reserved and always has the value 0
4 GLK	Global Lock Enable Affects the register access properties of VMAPCTX*n*, NPXCR (NPX Control Register), and CACMSK (Flash Cache Obfuscation Mask). BIVCTX*n* is not affected and remains write-only. Set by software (sticky); cleared only by a reset. 0b - Lock disabled. Subsequent reads return 0. 1b - Lock enabled: cannot write to VMAPCTXn, NPXCR, or CACMSK. Subsequent reads return 1.
3 —	This read-only field is reserved and always has the value 0
2 GDE	Global Decryption Enable Set by software; cleared by software; cleared by POR (Power-On Reset). 0b - Global decryption disabled. NPX on-the-fly decryption is globally disabled. Subsequent reads return 0. 1b - Global decryption enabled. NPX on-the-fly decryption is globally enabled. Subsequent reads return 1.
1 —	This read-only field is reserved and always has the value 0

Table continues on the next page...

Table continued from the previous page...

Field	Function
0 GEE	Global Encryption Enable Set by software; cleared by software; cleared by POR (Power-On Reset). 0b - Global encryption disabled. NPX on-the-fly encryption is disabled. Subsequent reads return 0. 1b - Global encryption enabled. NPX on-the-fly encryption is enabled if the flash access hits in a valid memory context. Subsequent reads return 1.

24.6.1.3 NPX Status Register (NPXSR)

Offset

Register	Offset
NPXSR	8h

Function

Contains the Key 0-3 validity flags and the number of supported memory contexts (NUMCTX).

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0				V3	V2	V1	V0	0				NUMCTX			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Fields

Field	Function
31-12 —	This read-only field is reserved and always has the value 0
11-8 Vn	Key n Valid V*n* indicates that the corresponding memory context *n* and its key are valid. The context *n* and its key are considered valid when the corresponding VMAPCTX*n* is non-zero.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Not valid 1b - Valid
7-4 —	This read-only field is reserved and always has the value 0
3-0 NUMCTX	Number of implemented memory contexts Contains the number of supported memory contexts. In this device, there are 4 memory contexts. 0000b - No (zero) implemented memory contexts 0001b - 1 implemented memory contexts 0010b - 2 implemented memory contexts 0011b - 3 implemented memory contexts 0100b - 4 implemented memory contexts

24.6.1.4 Flash Cache Obfuscation Mask (CACMSK)

Offset

Register	Offset
CACMSK	10h

Function

Contains the obfuscation mask for the flash cache.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	OBMASK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	OBMASK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 OBMASK	Obfuscation Mask

24.6.1.5 Data Remap (REMAP)

Offset

Register	Offset
REMAP	20h

Function

Provides a data remapping mechanism.

To write to REMAP, the remap lock must be disabled (REMAPLK = 0). To control the remapping, follow these options:

- To write to LIMDP, write 0x*NNNN*A5A5 (where *NNNN* is data) to REMAP. In this case, bits 28 to 24 are written to LIMDP. All other fields of REMAP remain unchanged.
- To write to LIM, write 0x*NNNN*5A5A (where *NNNN* is data) to REMAP. In this case, bits 20 to 16 are written to LIM.
- Any other writes to REMAP are ignored.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0				LIMDP				0				LIM			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved															REMAPLK
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-29 —	Always reads as 0. When writing, see the register description above.
28-24	LIMDP Remapping Address

Table continues on the next page...

Table continued from the previous page...

Field	Function
LIMDP	Defines remap_address[19:15] for remapping. Its value should be lower than the lower-half flash range.
23-21 —	Always reads as 0. When writing, see the register description above.
20-16 LIM	LIM Remapping Address Defines remap_address[19:15] for remapping. Its value should be lower than the lower-half flash range.
15-1 —	Always reads as 0. When writing, see the register description above.
0 REMAPLK	Remap Lock Enable To set the remap lock, write 0xXXXXC3C3 (X = don't care) to REMAP. All other fields of REMAP remain unchanged. Set by software (sticky); cleared only by a reset. 0b - Lock disabled: can write to REMAP 1b - Lock enabled: cannot write to REMAP

24.6.1.6 Bitmap of Valid Control for Memory Context 0 (VMAPCTX0_WD0)

Offset

Register	Offset
VMAPCTX0_WD0	40h

Function

Each VMAPCTX*n*_WD*m* register contains a 32-bit bitmap that controls whether the corresponding 32-KByte block of flash (out of a maximum size of 2 MBytes) is encrypted/decrypted within Memory Context *n* using KEY*n*.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	VAL31	VAL30	VAL29	VAL28	VAL27	VAL26	VAL25	VAL24	VAL23	VAL22	VAL21	VAL20	VAL19	VAL18	VAL17	VAL16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	VAL15	VAL14	VAL13	VAL12	VAL11	VAL10	VAL9	VAL8	VAL7	VAL6	VAL5	VAL4	VAL3	VAL2	VAL1	VAL0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 VALi	Block valid enable for encryption/decryption Set bit *i* to enable encryption/decryption for the corresponding 32-KByte block *i*. Set by software; cleared by software; cleared by POR (Power-On Reset). 0b - Disable 1b - Enable

24.6.1.7 Bitmap of Valid Control for Memory Context 0 (VMAPCTX0_WD1)

Offset

Register	Offset
VMAPCTX0_WD1	44h

Function

Each VMAPCTX*n*_WD*m* register contains a 32-bit bitmap that controls whether the corresponding 32-KByte block of flash (out of a maximum size of 2 MBytes) is encrypted/decrypted within Memory Context *n* using KEY*n*.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	VAL63	VAL62	VAL61	VAL60	VAL59	VAL58	VAL57	VAL56	VAL55	VAL54	VAL53	VAL52	VAL51	VAL50	VAL49	VAL48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	VAL47	VAL46	VAL45	VAL44	VAL43	VAL42	VAL41	VAL40	VAL39	VAL38	VAL37	VAL36	VAL35	VAL34	VAL33	VAL32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 VALi	Block valid enable for encryption/decryption Set bit *i* to enable encryption/decryption for the corresponding 32-KByte block *i*. Set by software; cleared by software; cleared by POR (Power-On Reset). 0b - Disable 1b - Enable

24.6.1.8 Block Initial Vector for Memory Context 0 (BIVCTX0_WD0)

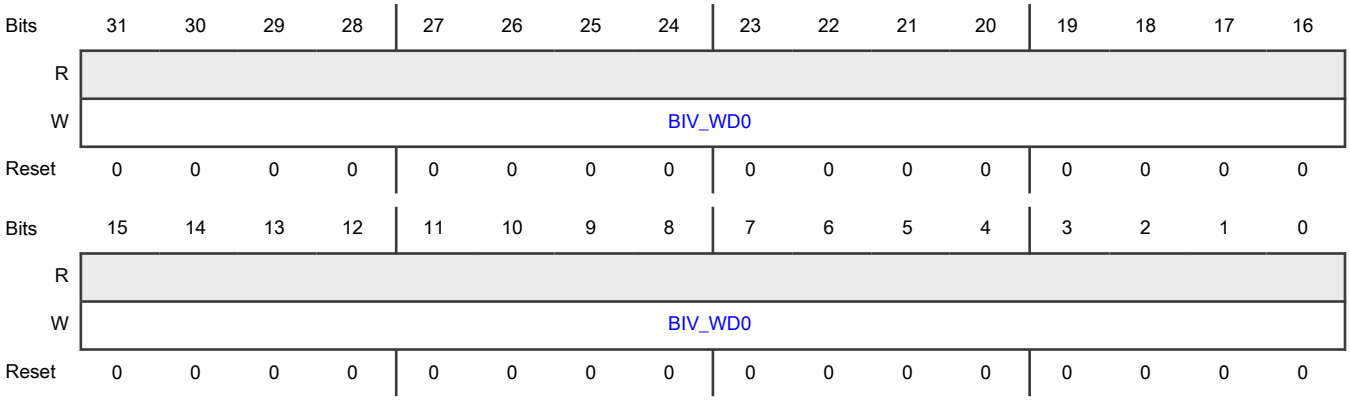
Offset

Register	Offset
BIVCTX0_WD0	48h

Function

For a given CTX*n*, the two combined BIVCTX*n*_WD*m* registers contain the 64-bit initial vector used as the nonce to encrypt/decrypt this block.

Diagram



Fields

Field	Function
31-0 BIV_WD0	Block Initial Vector Word0 Contains bits 31 to 0 of the block initial vector. Set by software; cleared by software; cleared by POR (Power-On Reset).

24.6.1.9 Block Initial Vector for Memory Context 0 (BIVCTX0_WD1)

Offset

Register	Offset
BIVCTX0_WD1	4Ch

Function

For a given CTX*n*, the two combined BIVCTX*n*_WD*m* registers contain the 64-bit initial vector used as the nonce to encrypt/decrypt this block.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	BIV_WD1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	BIV_WD1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 BIV_WD1	Block Initial Vector Word1 Contains bits 63 to 32 of the block initial vector. Set by software; cleared by software; cleared by POR (Power-On Reset).

24.6.1.10 Bitmap of Valid Control for Memory Context 1 (VMAPCTX1_WD0)

Offset

Register	Offset
VMAPCTX1_WD0	50h

Function

Each VMAPCTX*n*_WD*m* register contains a 32-bit bitmap that controls whether the corresponding 32-KByte block of flash (out of a maximum size of 2 MBytes) is encrypted/decrypted within Memory Context *n* using KEY*n*.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	VAL31	VAL30	VAL29	VAL28	VAL27	VAL26	VAL25	VAL24	VAL23	VAL22	VAL21	VAL20	VAL19	VAL18	VAL17	VAL16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	VAL15	VAL14	VAL13	VAL12	VAL11	VAL10	VAL9	VAL8	VAL7	VAL6	VAL5	VAL4	VAL3	VAL2	VAL1	VAL0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 VALi	Block valid enable for encryption/decryption Set bit *i* to enable encryption/decryption for the corresponding 32-KByte block *i*. Set by software; cleared by software; cleared by POR (Power-On Reset). 0b - Disable 1b - Enable

24.6.1.11 Bitmap of Valid Control for Memory Context 1 (VMAPCTX1_WD1)

Offset

Register	Offset
VMAPCTX1_WD1	54h

Function

Each VMAPCTX*n*_WD*m* register contains a 32-bit bitmap that controls whether the corresponding 32-KByte block of flash (out of a maximum size of 2 MBytes) is encrypted/decrypted within Memory Context *n* using KEY*n*.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	VAL63	VAL62	VAL61	VAL60	VAL59	VAL58	VAL57	VAL56	VAL55	VAL54	VAL53	VAL52	VAL51	VAL50	VAL49	VAL48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	VAL47	VAL46	VAL45	VAL44	VAL43	VAL42	VAL41	VAL40	VAL39	VAL38	VAL37	VAL36	VAL35	VAL34	VAL33	VAL32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 VALi	Block valid enable for encryption/decryption Set bit *i* to enable encryption/decryption for the corresponding 32-KByte block *i*. Set by software; cleared by software; cleared by POR (Power-On Reset). 0b - Disable 1b - Enable

24.6.1.12 Block Initial Vector for Memory Context 1 (BIVCTX1_WD0)

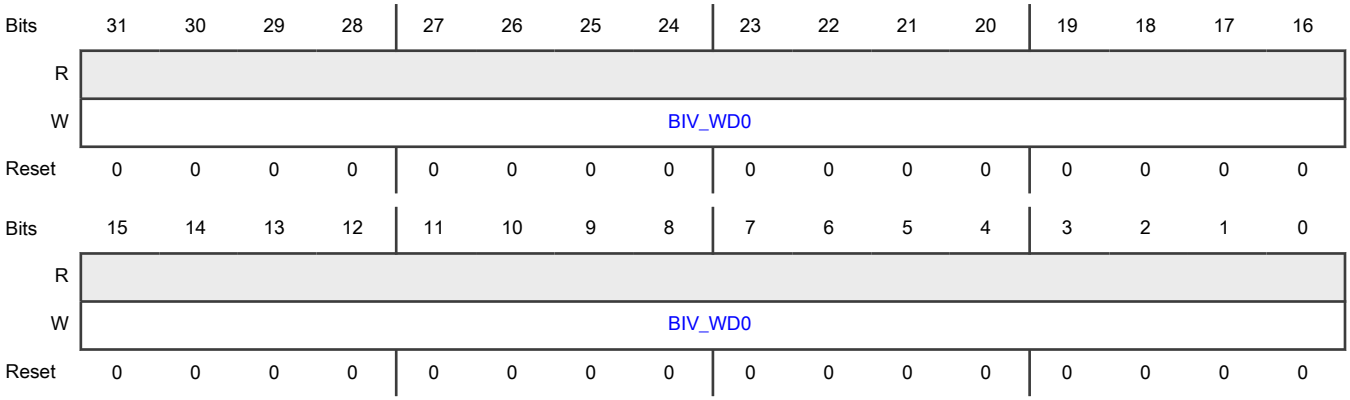
Offset

Register	Offset
BIVCTX1_WD0	58h

Function

For a given CTX*n*, the two combined BIVCTX*n*_WD*m* registers contain the 64-bit initial vector used as the nonce to encrypt/decrypt this block.

Diagram



Fields

Field	Function
31-0 BIV_WD0	Block Initial Vector Word0 Contains bits 31 to 0 of the block initial vector. Set by software; cleared by software; cleared by POR (Power-On Reset).

24.6.1.13 Block Initial Vector for Memory Context 1 (BIVCTX1_WD1)

Offset

Register	Offset
BIVCTX1_WD1	5Ch

Function

For a given CTX*n*, the two combined BIVCTX*n*_WD*m* registers contain the 64-bit initial vector used as the nonce to encrypt/decrypt this block.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	BIV_WD1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	BIV_WD1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 BIV_WD1	Block Initial Vector Word1 Contains bits 63 to 32 of the block initial vector. Set by software; cleared by software; cleared by POR (Power-On Reset).

24.6.1.14 Bitmap of Valid Control for Memory Context 2 (VMAPCTX2_WD0)

Offset

Register	Offset
VMAPCTX2_WD0	60h

Function

Each VMAPCTX*n*_WD*m* register contains a 32-bit bitmap that controls whether the corresponding 32-KByte block of flash (out of a maximum size of 2 MBytes) is encrypted/decrypted within Memory Context *n* using KEY*n*.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	VAL31	VAL30	VAL29	VAL28	VAL27	VAL26	VAL25	VAL24	VAL23	VAL22	VAL21	VAL20	VAL19	VAL18	VAL17	VAL16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	VAL15	VAL14	VAL13	VAL12	VAL11	VAL10	VAL9	VAL8	VAL7	VAL6	VAL5	VAL4	VAL3	VAL2	VAL1	VAL0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 VALi	Block valid enable for encryption/decryption Set bit *i* to enable encryption/decryption for the corresponding 32-KByte block *i*. Set by software; cleared by software; cleared by POR (Power-On Reset). 0b - Disable 1b - Enable

24.6.1.15 Bitmap of Valid Control for Memory Context 2 (VMAPCTX2_WD1)

Offset

Register	Offset
VMAPCTX2_WD1	64h

Function

Each VMAPCTX*n*_WD*m* register contains a 32-bit bitmap that controls whether the corresponding 32-KByte block of flash (out of a maximum size of 2 MBytes) is encrypted/decrypted within Memory Context *n* using KEY*n*.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	VAL63	VAL62	VAL61	VAL60	VAL59	VAL58	VAL57	VAL56	VAL55	VAL54	VAL53	VAL52	VAL51	VAL50	VAL49	VAL48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	VAL47	VAL46	VAL45	VAL44	VAL43	VAL42	VAL41	VAL40	VAL39	VAL38	VAL37	VAL36	VAL35	VAL34	VAL33	VAL32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 VALi	Block valid enable for encryption/decryption Set bit *i* to enable encryption/decryption for the corresponding 32-KByte block *i*. Set by software; cleared by software; cleared by POR (Power-On Reset). 0b - Disable 1b - Enable

24.6.1.16 Block Initial Vector for Memory Context 2 (BIVCTX2_WD0)

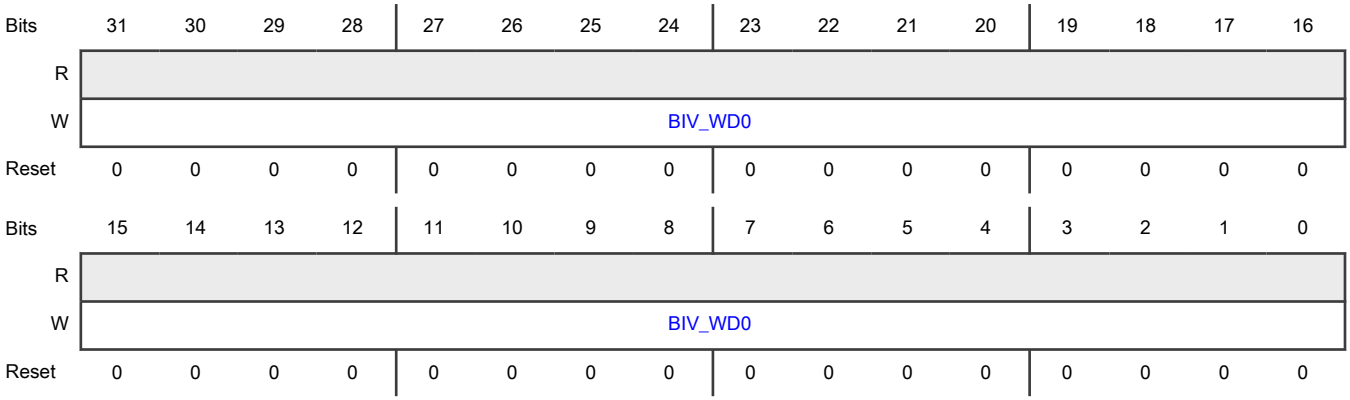
Offset

Register	Offset
BIVCTX2_WD0	68h

Function

For a given CTX*n*, the two combined BIVCTX*n*_WD*m* registers contain the 64-bit initial vector used as the nonce to encrypt/decrypt this block.

Diagram



Fields

Field	Function
31-0 BIV_WD0	Block Initial Vector Word0 Contains bits 31 to 0 of the block initial vector. Set by software; cleared by software; cleared by POR (Power-On Reset).

24.6.1.17 Block Initial Vector for Memory Context 2 (BIVCTX2_WD1)

Offset

Register	Offset
BIVCTX2_WD1	6Ch

Function

For a given CTX*n*, the two combined BIVCTX*n*_WD*m* registers contain the 64-bit initial vector used as the nonce to encrypt/decrypt this block.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	BIV_WD1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	BIV_WD1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 BIV_WD1	Block Initial Vector Word1 Contains bits 63 to 32 of the block initial vector. Set by software; cleared by software; cleared by POR (Power-On Reset).

24.6.1.18 Bitmap of Valid Control for Memory Context 3 (VMAPCTX3_WD0)

Offset

Register	Offset
VMAPCTX3_WD0	70h

Function

Each VMAPCTX*n*_WD*m* register contains a 32-bit bitmap that controls whether the corresponding 32-KByte block of flash (out of a maximum size of 2 MBytes) is encrypted/decrypted within Memory Context *n* using KEY*n*.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	VAL31	VAL30	VAL29	VAL28	VAL27	VAL26	VAL25	VAL24	VAL23	VAL22	VAL21	VAL20	VAL19	VAL18	VAL17	VAL16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	VAL15	VAL14	VAL13	VAL12	VAL11	VAL10	VAL9	VAL8	VAL7	VAL6	VAL5	VAL4	VAL3	VAL2	VAL1	VAL0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 VALi	Block valid enable for encryption/decryption Set bit *i* to enable encryption/decryption for the corresponding 32-KByte block *i*. Set by software; cleared by software; cleared by POR (Power-On Reset). 0b - Disable 1b - Enable

24.6.1.19 Bitmap of Valid Control for Memory Context 3 (VMAPCTX3_WD1)

Offset

Register	Offset
VMAPCTX3_WD1	74h

Function

Each VMAPCTX*n*_WD*m* register contains a 32-bit bitmap that controls whether the corresponding 32-KByte block of flash (out of a maximum size of 2 MBytes) is encrypted/decrypted within Memory Context *n* using KEY*n*.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	VAL63	VAL62	VAL61	VAL60	VAL59	VAL58	VAL57	VAL56	VAL55	VAL54	VAL53	VAL52	VAL51	VAL50	VAL49	VAL48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	VAL47	VAL46	VAL45	VAL44	VAL43	VAL42	VAL41	VAL40	VAL39	VAL38	VAL37	VAL36	VAL35	VAL34	VAL33	VAL32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 VALi	Block valid enable for encryption/decryption Set bit *i* to enable encryption/decryption for the corresponding 32-KByte block *i*. Set by software; cleared by software; cleared by POR (Power-On Reset). 0b - Disable 1b - Enable

24.6.1.20 Block Initial Vector for Memory Context 3 (BIVCTX3_WD0)

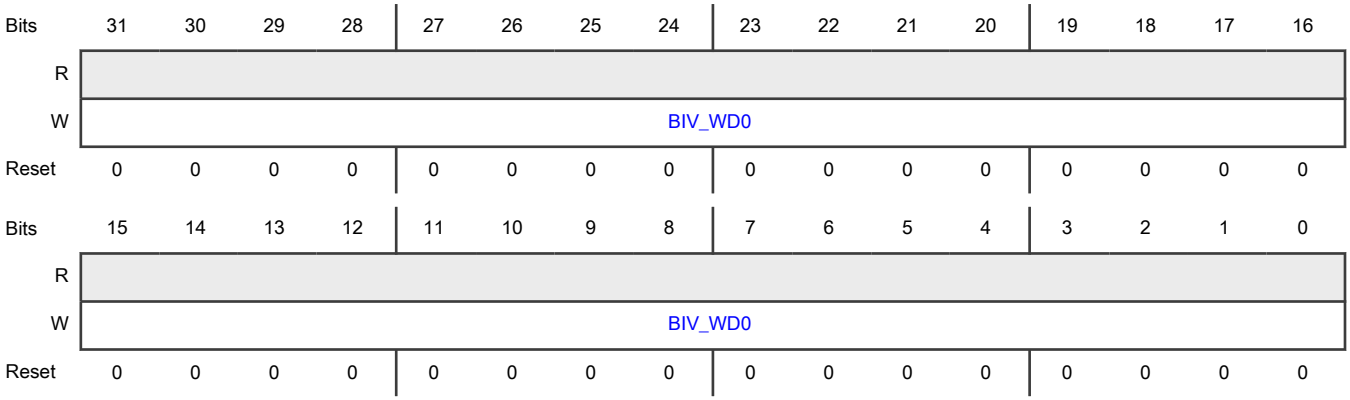
Offset

Register	Offset
BIVCTX3_WD0	78h

Function

For a given CTX*n*, the two combined BIVCTX*n*_WD*m* registers contain the 64-bit initial vector used as the nonce to encrypt/decrypt this block.

Diagram



Fields

Field	Function
31-0 BIV_WD0	Block Initial Vector Word0 Contains bits 31 to 0 of the block initial vector. Set by software; cleared by software; cleared by POR (Power-On Reset).

24.6.1.21 Block Initial Vector for Memory Context 3 (BIVCTX3_WD1)

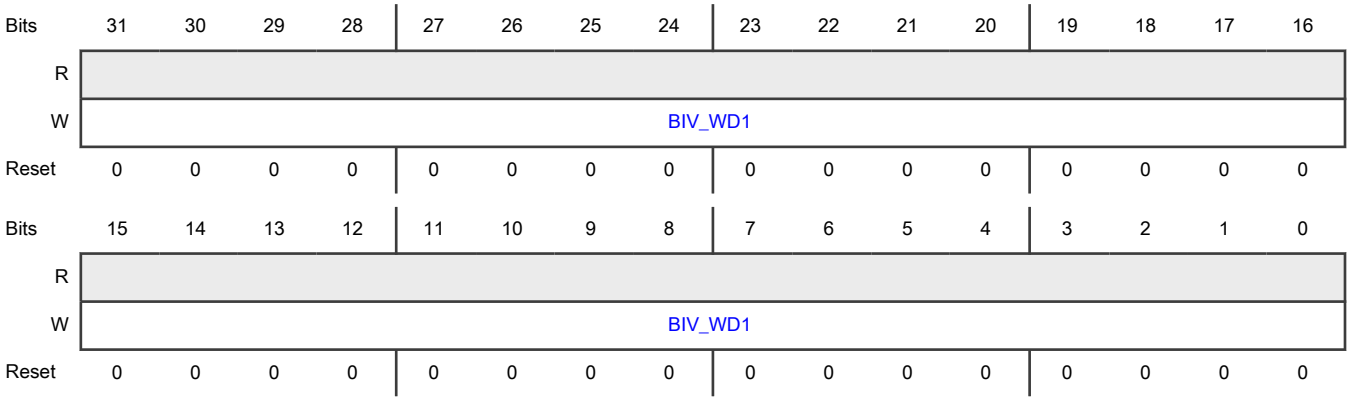
Offset

Register	Offset
BIVCTX3_WD1	7Ch

Function

For a given CTX*n*, the two combined BIVCTX*n*_WD*m* registers contain the 64-bit initial vector used as the nonce to encrypt/decrypt this block.

Diagram



Fields

Field	Function
31-0 BIV_WD1	Block Initial Vector Word1 Contains bits 63 to 32 of the block initial vector. Set by software; cleared by software; cleared by POR (Power-On Reset).

Chapter 25
FlexSPI

25.1 Chip-specific FlexSPI information

Table 499. Reference links to related information[1]

Topic	Related module	Reference
Full description	FlexSPI	FlexSPI
System memory map		See the section "System memory map"
Clocking		See the chapter "Clocking"
Power management		See the chapter "Power management"
Signal multiplexing	Port control	See the chapter "Signal Multiplexing"

1. For all the reference sections and chapters mentioned in this table, refer the MCX Nx4x Reference Manual.

25.1.1 Module instances

This device has one instance of the FlexSPI module, FLEXSPI0.

25.1.2 Security considerations

The FlexSPI module implements a Domain control feature that can be used to restrict IP command access and LUT write access. The IP command domain control feature can be used to restrict non-secure and/or non-privileged masters to a specific address range (a secure master must configure the allowed address range for the non-secure IP commands).

While the domain control feature allows for some sharing of the FlexSPI module between secure and non-secure masters, NXP recommends configuring the FlexSPI for secure and privileged access only to prevent unintentional or malicious modification of the FlexSPI operational registers. The secure AHB controller controls the security level for access to peripherals and does default to secure and privileged access for all peripherals.

25.1.3 FlexSPI AHB regions

There are multiple FlexSPI regions in the system memory map, but the data accessed is aliased across the multiple regions. Refer to the System memory map section (in Core Overview chapter) to see FlexSPI AHB spaces on this device. Also, refer to FlexSPI regions mapping to the CACHE64_CTRL and CACHE64_POLSEL modules in their respective chapters in the MCX Nx4x Reference Manual.

25.1.4 Controller ID allocation

The Controller ID (MSTRID) values used for the FlexSPI module are same as the AHB bus matrix port numbers. Refer to Memory chapter (AHB bus matrix ports table) in MCX Nx4x Reference Manual to find the controller port/MSTRID value used for each controller.

25.1.5 FlexSPI connection modes

The table below shows details about the Parallel and Individual modes for flash memory.

Table 500. Relation between modes and bus size for FlexSPI instances

Mode	Effective Bus Size	Data Signals	Function	Devices Attached to Bus	Chip Selects
INDIVIDUAL	4 bit	A_DATA0- A_DATA3	Single QSPI chip support(1b,2b,4b)	Up to 2	A_SS0_B, A_SS1_B
INDIVIDUAL	4 bit	B_DATA0- B_DATA3	Single QSPI chip support(1b,2b,4b)	Up to 2	B_SS0_B, B_SS1_B
INDIVIDUAL	8 bit	A_DATA0- A_DATA7	Single Octal/ Hyperbus/Xccela chip support(1x8b)	Up to 2	A_SS0_B, A_SS1_B
INDIVIDUAL	8 bit	B_DATA0- B_DATA7	Single Octal/ Hyperbus/Xccela chip support(1x8b)	Up to 2	B_SS0_B, B_SS1_B
PARALLEL	8 bit	A_DATA0- A_DATA3 | B_DATA0- B_DATA3	Two QSPI chips in parallel(2x4b)	2 parallel chips x2 = up to 4	(A_SS0_B | B_SS0_B), (A_SS1_B | B_SS1_B)
PARALLEL	16 bit	A_DATA0- A_DATA7 | B_DATA0- B_DATA7	Two Octal/ Hyperbus/Xccela chips in parallel(2x8b)	2 parallel chips x2 = up to 4	(A_SS0_B | B_SS0_B), (A_SS1_B | B_SS1_B)

25.1.6 Loading IPED encryption keys

See details of the IPED encryption keys and their respective slot numbers in ELS key store state after boot.

Also refer Figure 4.

25.2 Overview

FlexSPI supports two SPI channels and up to four external devices. Each channel supports Single/Dual/Quad/Octal mode data transfer (1/2/4/8 bidirectional data lines).

The FlexSPI configuration depends on the chip configuration. See the system-level section for boot information and pinmux for chip-specific information regarding the modes and number of devices supported.

FlexSPI supports communication with both serial flash memory and serial RAM devices. While many of the descriptions, registers, and fields specifically reference flash memory, almost all information can also be applied to serial RAM. Flash memory is used as an example in the tables and figures in this chapter.

NOTE

Terminology in this chapter has been updated to align with JEDEC standard *Expanded Serial Peripheral Interface (xSPI) for Non Volatile Memory Devices, Version 1.0*.

Table 501. Updated terms

Updated term	Deprecated term
Controller	Master
Target	Slave

25.2.1 Block diagram

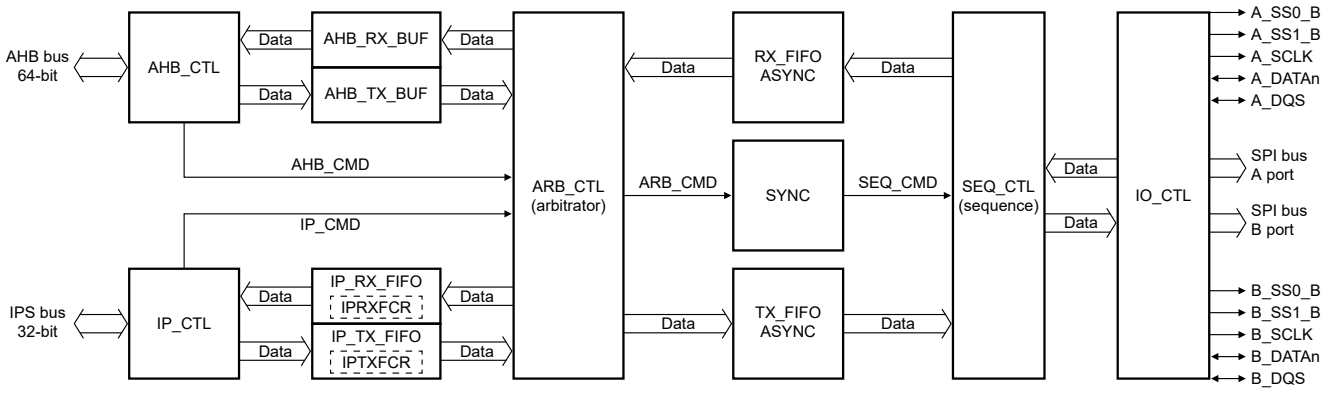


Figure 102. Block diagram

25.2.2 Features

FlexSPI supports:

- Flexible sequence engine (lookup table) to support various vendor devices
  - Serial NOR flash memory and other devices with SPI protocol similar to serial NOR flash memory
  - Serial NAND flash memory
  - HyperBus devices (HyperFlash/HyperRAM)
  - FPGA devices
- Flash memory access modes
  - Single, Dual, Quad, Octal mode
  - Single Data Transfer Rate (SDR) and Double Data Transfer Rate (DDR) mode
  - Individual/Parallel mode
- Sampling clock mode
  - Internal dummy read strobe looped back internally
  - Internal dummy read strobe looped back from pad
  - SCLK clock output looped back from pad
  - Flash-memory-provided read strobe
- Automatic data learning to select the correct sample clock phase
- Memory mapped read and write access by AHB bus

- AHB receive buffer implemented to reduce read latency. Total AHB receive buffer size: 1024 bytes.
- 16 AHB controllers with programmable priority for read access from each
- 8 flexible and configurable buffers in AHB receive buffer
- AHB transmit buffer implemented to buffer all write data from one AHB burst. AHB transmit buffer size is 64 bytes.

NOTE

All AHB controllers share this AHB transmit buffer. There is no AHB controller number limitation for write access.

- Software-triggered flash memory read and write access by IP bus
  - IP receive FIFO implemented to buffer all read data from external devices. FIFO size: 1024 bytes
  - IP transmit FIFO implemented to buffer all write data to external devices. FIFO size: 1024 bytes
  - DMA support to read IP receive FIFO
  - DMA support to fill IP transmit FIFO
  - SCLK stops when IP receive FIFO is full during reading flash memory data
  - SCLK stops when IP transmit FIFO is empty during writing flash memory data

25.3 Functional description

25.3.1 Flash connection

There are two FlexSPI interface ports (port A and port B). Each port supports up to two flash devices by providing two chip select outputs.

NOTE

FlexSPI configuration depends on the chip configuration. See the chip-specific FlexSPI information regarding the number of devices supported.

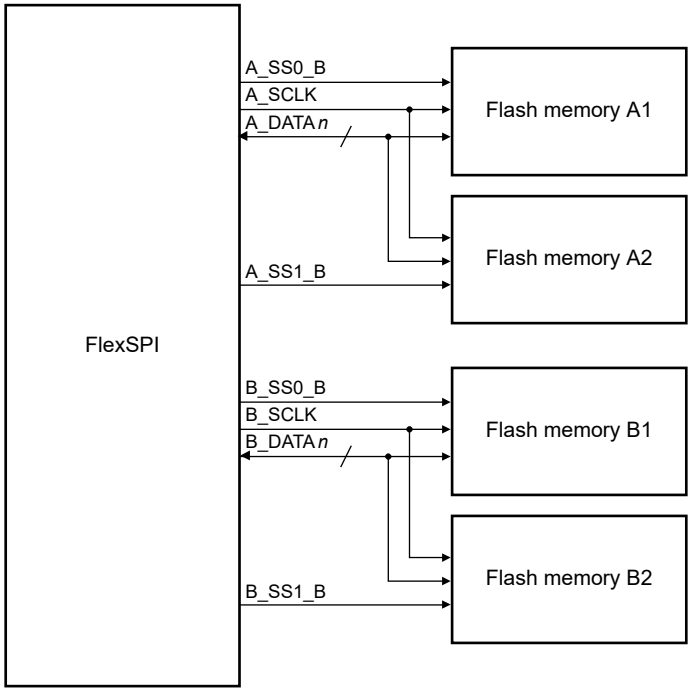


Figure 103. FlexSPI connection diagram with four devices.

NOTE

- Flash A1 and Flash A2 can be two flash chip packages or two flash dies in the same package. There is no difference to FlexSPI. This statement is also true for Flash B1 and Flash B2.
- Flash A1 and Flash B1 can be accessed in parallel, using parallel mode. FlexSPI merges or splits the flash read and program data automatically. This statement is also true for Flash A2 and Flash B2.
- In parallel mode, Flash A1 and Flash A2 cannot be accessed at the same time. This statement is also true for Flash B1 and Flash B2.
- In individual mode, flash devices cannot be accessed at the same time. But these four devices can be accessed separately.

There is a combination mode (MCR0[COMBINATIONEN] = 1) used to provide octal flash memory support by combining Port A (A_DATA[3:0]) and Port B (B_DATA[3:0]) together. Normal mode can also support Octal mode by using all eight data lines from Port A or Port B. Figure 104 shows the connection diagram for combination mode.

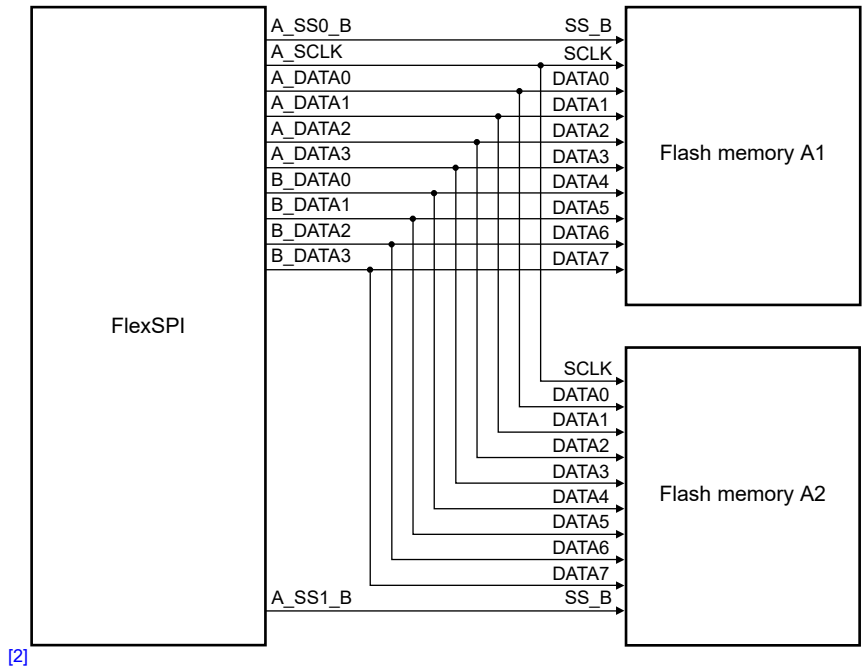


Figure 104. FlexSPI connection diagram for combination mode

25.3.2 Flash memory access mode

25.3.2.1 SPI clock mode

FlexSPI only supports SPI clock mode 0: clock polarity (CPOL) = 0 and clock phase (CPHA) = 0. When the SPI bus is idle, SCLK stays at a logic low state.

25.3.2.2 Individual mode and Parallel mode

In individual mode, read and write data are received and transmitted on port A or port B.

In parallel mode, read and write data are received and transmitted on port A and port B in parallel. FlexSPI merges or splits the read and program data automatically (READ/WRITE instruction). Only read and program data is merged or split (READ/WRITE instruction). For other instructions (such as Command, Address, Mode, and Data size), the same associated information is transmitted to the devices connected to port A and port B. For more details, see Executing instructions on SPI interface.

IPCR1[IPAREN] (for IP command) or AHBCR[APAREN] (for AHB command) statistically determines Individual and Parallel mode.

NOTE

FlexSPI does not support 16-bit read and write for the SPI interface.

25.3.2.3 SDR mode and DDR mode

In SDR mode, flash memory receives data on the rising edge of SCLK and transmits data on the falling edge of SCLK.

In DDR mode, flash memory receives and transmits data on both the rising and falling edges of SCLK.

[2] Parallel mode access is not available in combination mode.

The instruction (opcode) in the LUT sequence dynamically determines SDR and DDR modes. There is no static configuration register field setting for SDR and DDR modes. See FlexSPI input timing and FlexSPI output timing for details about input and output timing.

25.3.2.4 Single, Dual, Quad, and Octal mode

In Single mode, FlexSPI transmits and receives data on one data pin (DATA0 for transmitting, DATA1 for receiving).

In Dual mode, FlexSPI transmits and receives data on two data pins (DATA0–DATA1 for both transmitting and receiving).

In Quad mode, FlexSPI transmits and receives data on four data pins (DATA0–DATA3 for both transmitting and receiving).

In Octal mode, FlexSPI transmits and receives data on eight data pins (DATA0–DATA7 for both transmitting and receiving).

Instruction (num_pads) in the LUT sequence dynamically determine Single, Dual, Quad, and Octal modes. There is no static configuration register field setting for Single, Dual, Quad, and Octal modes.

25.3.3 Modes of operation

FlexSPI operates with these modes.

Table 502. Operating modes

Mode	Description
Module Disable	This mode is a low-power mode for FlexSPI. In Module Disable mode, the AHB clock and serial clock domains are gated off internally, but the IPS bus clock domain is not gated off. Read and write access to the control and status register are available, but the LUT, IP receive FIFO, and IP transmit FIFO cannot be accessed. Serial memory access is also not available. Write 1 to MCR0[MDIS] to enter this mode. Write 0 to MCR0[MDIS] to exit this mode.
Doze	This mode is a low-power mode for the chip. When the chip requires FlexSPI to enter Doze mode and MCR0[DOZEEN] = 1, FlexSPI enters Doze mode after all transactions are completed (STS0[ARBIDLE] = 1). In Doze mode, the AHB clock and serial clock domains are gated off internally, but the IPS bus clock is not gated off. Read and write access to the control and status register are available, but the LUT, IP receive FIFO, the IP transmit FIFO cannot be accessed. Serial memory access is also not available. This mode is entered via system request, and exited by deasserting this system request.
Stop	This mode is a low-power mode for the chip. When the chip requires the FlexSPI to enter Stop mode, FlexSPI waits for all transactions to complete (STS0[ARBIDLE] = 1) and return ACK handshake to the system. After the ACK handshake is returned, FlexSPI gates off the AHB clock and serial clock domains internally. The system can gate off the AHB bus clock, IPS bus clock, and serial clock at the system level. This mode is entered via system request. This mode is exited by deasserting this system request, and the ACK handshake message is also deasserted immediately.
Normal	No clock is gated off internally. Normal register access and serial memory access are available.

25.3.4 AHB access memory map

FlexSPI allocates AHB memory space for each memory device starting from the FlexSPI region base address of the system memory map. The FLSHxCR0[FLSHSZ] field determines the amount of memory allocated for each memory device in KB, Fx_SIZE. The memory is then allocated sequentially and contiguously.

NOTE

The maximum flash size supported for each device is 512 MB. The maximum total flash size supported (for all four devices) is also 512 MB.

When MCR2[SAMEDEVICEEN] = 1:

- FA1_SIZE = FLSHA1CR0[FLSHSZ] × 1 KB
- FA2_SIZE = FLSHA1CR0[FLSHSZ] × 1 KB
- FB1_SIZE = FLSHA1CR0[FLSHSZ] × 1 KB
- FB2_SIZE = FLSHA1CR0[FLSHSZ] × 1 KB

When MCR2[SAMEDEVICEEN] = 0:

- FA1_SIZE = FLSHSZ × 1 KB
- FA2_SIZE = FLSHA2CR0[FLSHSZ] × 1 KB
- FB1_SIZE = FLSHB1CR0[FLSHSZ] × 1 KB
- FB2_SIZE = FLSHB2CR0[FLSHSZ] × 1 KB

Flash B1 and Flash B2 size settings are ignored in parallel mode (FLSHB1CR0[FLSHSZ], FLSHB2CR0[FLSHSZ]). To execute in parallel mode, Flash B1 should be the same device as Flash A1, and Flash B2 should be the same device as Flash A2.

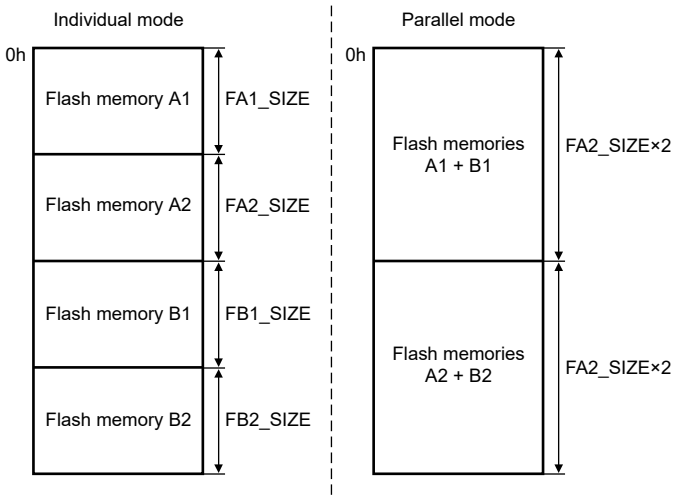


Figure 105. AHB access memory map in individual and parallel mode

AHB access memory map in individual mode:

- Flash A1 address range: 0000_0000h to FA1_SIZE
- Flash A2 address range: FA1_SIZE to (FA1_SIZE + FA2_SIZE)
- Flash B1 address range: (FA1_SIZE + FA2_SIZE) to (FA1_SIZE + FA2_SIZE + FB1_SIZE)
- Flash B2 address range: (FA1_SIZE + FA2_SIZE + FB1_SIZE) to (FA1_SIZE + FA2_SIZE + FB1_SIZE + FB2_SIZE)

AHB access memory map in parallel mode:

- Flash A1+B1 address range: 0000_0000h to FA1_SIZE × 2
- Flash A2+B2 address range: FA1_SIZE × 2 to (FA1_SIZE × 2 + FA2_SIZE × 2)

NOTE

The address used in Figure 105 and equations in this topic is the address presented to the memory. The base address for FlexSPI in the system memory map has already been removed, so it not used or shown here.

25.3.4.1 Dual image use case using HADDRSTART, HADDREND, and HADDROFFSET registers

The FlexSPI controller has a remap feature that supports the storage of dual images in memory. The HADDRSTART and HADDREND registers can be used to configure an address range to which an offset (that the HADDROFFSET register configures) is applied. This feature allows a single system bus address range to be mapped to two different memory ranges: the range that is remapped or the range that is not remapped. You can enable and disable the remapping to switch between accessing the two images at the same system memory address. This feature is used for booting.

NOTE

- Remapping in this table includes the REMAP register (HADDRSTART[0]) and SWAP register (HADDRSTART[2]).
- The swap function impacts both the AHB and IPS address remapping with the same mechanism. The remap function only impacts the AHB address.

Table 503. FlexSPI address translation

	AHB or IPS address	Memory address	Description
Enable remap			
AHB WR	ADDR	ADDR + OFFSET	The remap function is available if HADDRSTART <= ADDR < HADDREND, otherwise no change in address.
AHB RD	ADDR	ADDR + OFFSET	The remap function is available if HADDRSTART <= ADDR <HADDREND, otherwise no change in address.
Disable remap			
AHB WR	ADDR	ADDR	No change in address.
AHB RD	ADDR	ADDR	No change in address.

NOTE

- HADDRSTART[0] is used to enable the remap feature.
- The ADDR above is the address from the IPS or AHB interface. The OFFSET is same as the HADDROFFSET register.

25.3.4.2 Flash address sent to flash memory devices

The AHB address (AHB command) or IPCR0[SFAR] (IP command) determines the flash memory access start address. See Flash memory access via AHB command and Flash memory access via IP command for more details.

For AHB commands, the FlexSPI controller removes the flash memory base address automatically when sending flash addresses to flash devices. The flash address is sent to devices in two parts: row address and column address. For flash devices that do not support the column address, set FLSHxCR1[CAS] to 0. This setting causes all flash address bits to be sent to flash devices as row addresses.

For word-addressable flash devices, the last bit of the address is not needed, because flash memory is read and programmed in terms of two bytes. For parallel mode, Flash A1 and Flash B1 (or Flash A2 and Flash B2) are accessed in parallel. As a result, the flash memory address sent to flash devices should be divided by 2. Table 504 indicates the relationship of row address, column address, and flash address (FA).

Table 504. Flash address, row address, and column address

Parallel mode	Word-addressable	Row address	Column address	Comment
0	0	FA[31:CAS]	FA[CAS-1:0]	There is no limitation on FA and data size alignment.
0	1	FA[31:CAS+1]	FA[CAS:1]	FA and data size should be two-byte aligned.
1	0	FA[31:CAS+1]	FA[CAS:1]	FA and data size should be two-byte aligned.
1	1	FA[31:CAS+2]	FA[CAS+1:2]	FA and data size should be four-byte aligned.

NOTE

- FA is the flash address without base address.
- If the bit number of the row or column flash address is more than the valid value, the high position bits are supplemented with zero. See Programmable sequence engine for details about row or column address instructions.

When parallel mode is enabled or word-addressable flash memory is used, there are limitations on the flash memory start address and data size. You can address this requirement by aligning the AHB bus access address (for AHB command) or IP command address IPCR0[SFAR] (for IP command) in software. There are two ways to avoid these limitations in the specified case.

1. For AHB Read Command only:

   When AHBCR[READADDROPT] and AHBCR[PREFETCHEN] are both 1, FlexSPI guarantees the start address and data size for the flash memory access are eight-byte aligned by hardware.

   When AHBCR[READADDROPT] is 1, FlexSPI fetches redundant data to guarantee the flash memory start address is eight-byte aligned. When prefetch is enabled (AHBCR[PREFETCHEN] is 1), the size of the eight-byte-aligned AHB receive buffer determines the flash memory read data size.

2. For AHB Write Command and Individual Mode only:

   By default, FlexSPI guarantees the flash memory write access start address and data size are two-byte aligned when using DQS as write mask (FLSHCR4[WMENB or WMENA] = 1).

   This feature is not applied in parallel mode and must be used if the external device supports write mask feature.

   For example, for odd start address:

   - To write 6 bytes, FlexSPI sends 8 bytes, the first and last byte are both masked.
   - To write 7 bytes, FlexSPI sends 8 bytes, only the first byte is masked.
   - To write 8 bytes, FlexSPI sends 10 bytes, the first and last byte are both masked.

   For even start address:

   - To write 6 bytes, FlexSPI sends 6 bytes, no bytes are masked.
   - To write 7 bytes, FlexSPI sends 8 bytes, last byte is masked.
   - To write 8 bytes, FlexSPI sends 8 bytes, no bytes are masked.

25.3.5 Lookup table (LUT)

The LUT is an internal memory that preserves a number of preprogrammed sequences. Each sequence consists of up to eight instructions which are executed sequentially. When an IP command or an AHB command triggers a flash memory access, the FlexSPI controller:

1. Fetches the sequence from LUT (sequence index or number).
2. Executes the flash memory access to generate a valid flash transaction on the SPI interface.

Figure 106 shows the LUT structures, sequences, and instructions.

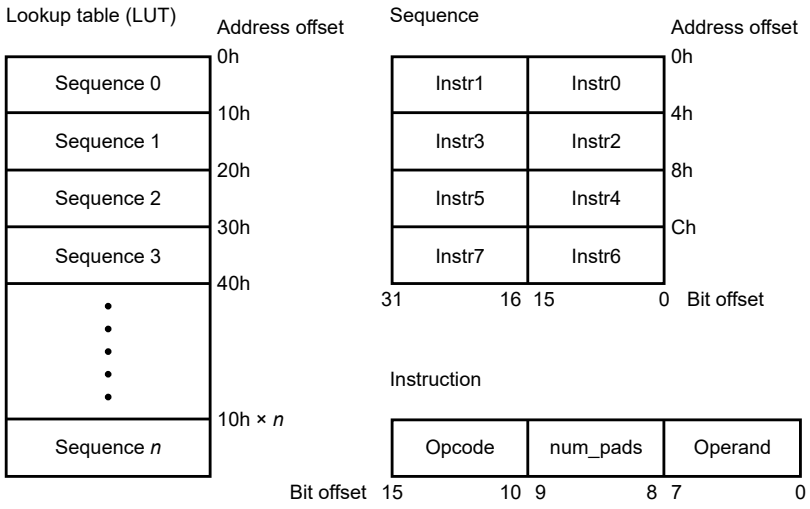


Figure 106. LUT structure, sequences, and instructions

NOTE

If the number of instructions needed for a flash transaction is less than eight, the STOP instruction (code 0000_0000h) must be programmed for unneeded instructions.

For IP and AHB write commands, the FlexSPI controller always executes from instruction pointer 0. For AHB read commands, the FlexSPI controller executes from a saved instruction start pointer. The FlexSPI controller saves the instruction start pointers separately for each flash device. All these saved instruction pointers are zero before the JMP_ON_CS instruction is executed. When the JMP_ON_CS instruction is executed, the operand in the JMP_ON_CS instruction is saved as instruction start pointer. See Execute-In-Place (XIP) enhanced mode.

The reset value of the LUT is unknown because it is implemented as internal memory. The LUT must be programmed according to the connected device. To protect its contents during a code runover, the LUT can be locked or unlocked to prevent unwanted changes after the LUT has been configured. The key to lock or unlock the LUT is **5AF05AF0h**.

To lock the LUT:

1. Write the key (5AF05AF0h) to LUT Key (LUTKEY).
2. Write 1 to LUTCR[LOCK] and write 0 to LUTCR[UNLOCK]. When there is another register write access to FlexSPI between these two write accesses, this LUT is not successfully locked.

To unlock the LUT:

1. Write the key (5AF05AF0h) to the LUT Key Register LUT Key (LUTKEY).
2. Write 0 to LUTCR[LOCK] and write 1 to LUTCR[UNLOCK]. When there is another register write access to FlexSPI between these two write accesses, this LUT is not successfully locked.

The lock status of the LUT can be read from LUTCR[LOCK] and LUTCR[UNLOCK].

25.3.6 Programmable sequence engine

The FlexSPI controller implements a programmable sequence engine that executes the sequence from the LUT. The FlexSPI controller executes the instructions sequentially, and generates flash transactions on the SPI interface accordingly. Table 505 is a complete list of the supported instructions.

Table 505. Instruction set

Name	Opcode	num_pads	Action on SPI interface	Transmit data	Bits/bytes/cycle number
CMD_SDR	00_0001h	00h - one pad (Single mode) 01h - two pads (Dual mode) 02h - four pads (Quad mode) 03h - eight pads (Octal mode)	Transmit command code to flash memory	Command code: Operand[7:0]	Bit number: 8
CMD_DDR	00_0021h				
RADDR_SDR	00_0002h		Transmit row address to flash memory See Flash address sent to flash memory devices	Row_Address[31:0] Row_Address comes from AHB bus (AHB command) or IPCR0[SFAR] (IP command).	Bit number: operand[7:0] Value of operand determines number of bits sent in Row_Address.
RADDR_DDR	00_0022h				
CADDR_SDR	00_0003h		Transmit column address to flash memory. See Flash address sent to flash memory devices.	Column_Address[31:0] Column_Address comes from AHB bus (AHB command) or IPCR0[SFAR] (IP command).	Bit number: operand[7:0] Value of operand determines number of bits sent in Column_Address.
CADDR_DDR	00_0023h				
MODE1_SDR	00_0004h		Transmit mode bits to flash memory	Mode bits: Operand[0]	Bit number: 1
MODE1_DDR	00_0024h				
MODE2_SDR	00_0005h			Mode bits: Operand[1:0]	Bit number: 2
MODE2_DDR	00_0025h				
MODE4_SDR	00_0006h			Mode bits: Operand[3:0]	Bit number: 4
MODE4_DDR	00_0026h				
MODE8_SDR	00_0007h			Mode bits: Operand[7:0]	Bit number: 8
MODE8_DDR	00_0027h				
WRITE_SDR	00_0008h		Transmit program data to flash device	Program data in IP_TX_FIFO or AHB_TX_BUF	AHB burst size and burst type (AHB command) or IPCR1[IDATSZ] (IP command) determines byte number (data size). For details about flash read or program data size, see Flash memory access via AHB
WRITE_DDR	00_0028h				
READ_SDR	00_0009h		Receive read data from flash device	-	
READ_DDR	00_0029h				

Table continues on the next page...

Table 505. Instruction set (continued)

Name	Opcode	num_pads	Action on SPI interface	Transmit data	Bits/bytes/cycle number
			Read data is put into AHB_RX_BUF or IP_RX_FIFO.		command and Flash memory access via IP command.
DATSZ_SDR	00_000Bh		Transmit read or program data size (byte number) to flash device	Internal logic Read or program data size for current command sequence	Bit number: operand[7:0] Never set operand to zero or greater than 64 for DATSZ instruction.
DATSZ_DDR	00_002Bh				
DUMMY_SDR	00_000Ch		Leave data lines undriven by FlexSPI controller. Turnaround cycles are provided from host driving to device driving. num_pads determines number of pads in input mode.	-	Dummy cycle number (in serial root clock): Operand[7:0] Dummy cycle (N), described in data sheet of flash device, is in number of SCLK cycles. This number may be configurable. In SDR mode, SCLK cycle is same as serial root clock. Operand value must be set to N. In DDR mode, SCLK cycle is double serial root clock cycle. Operand value must be set to 2N, 2N-1 or 2N+1 depending on definition of dummy in data sheet of flash device. See Flash memory access sequence examples and dummy cycle definition in data sheet of external memory.
DUMMY_DDR	00_002Ch				
DUMMY_RWDS_SDR	00_000Dh		Similar to DUMMY_SDR/ DUMMY_DDR instruction. Difference is in dummy cycle number. DQS pin is called RWDS in HyperBus specification. See Dummy instruction for details.	-	For read command, dummy cycle number (in serial root clock): (operand[7:0] × 4 - 1) if RWDS (DQS pin) is high; (operand[7:0] × 2 - 1) if RWDS (DQS pin) is low; For write command, dummy cycle number (in serial root clock): (operand[7:0] × 4 - 2) if RWDS (DQS pin) is high;
DUMMY_RWDS_DDR	00_002Dh				

Table continues on the next page...

Table 505. Instruction set (continued)

Name	Opcode	num_pads	Action on SPI interface	Transmit data	Bits/bytes/cycle number
			Set operand to "Latency count" for HyperBus devices.		(operand[7:0] × 2 - 2) if RWDS (DQS pin) is low;
LEARN_SDR	00_000Ah	00h - one pad (Single mode) 01h - two pads (Dual mode) 02h - four pads (Quad mode) 03h - eight pads (Octal mode)	Receive read data or preamble bit from flash device FlexSPI controller compares data line bits with the DLPR register to determine a correct sampling clock phase.	-	Bit number: operand[7:0] Never set operand to zero for LEARN instruction. Value of operand indicates number of bits to receive and compare to DLPR value. For example, 8-bit pattern 5Ah on each data line must set operand to 8.
LEARN_DDR	00_002Ah				
JMP_ON_CS	00_001Fh	Num_pads setting ignored.	Stop execution, deassert CS and save operand[7:0] as instruction start pointer for next sequence. Normally this instruction is used to support Execute-In-Place enhanced mode. See Execute-In-Place (XIP) enhanced mode. This instruction is only allowed for AHB read commands. When using this instruction in IP command or AHB write command, interrupt status flag set (INTR[IPCMDERR] or INTR[AHBCMDERR]).	-	No transaction on SPI interface.
STOP	00_0000h		Stop execution and deassert CS. Next command	-	

Table continues on the next page...

Table 505. Instruction set (continued)

Name	Opcode	num_pads	Action on SPI interface	Transmit data	Bits/bytes/cycle number
			sequence (to same flash device) starts from instruction pointer 0.		

The programmable sequence engine allows configuration of the LUT according to the connected external serial device. The flexible LUT structure easily adapts to new command or protocol changes from different vendors.

The DDR sequence is a flash memory access sequence that contains DDR instructions other than DUMMY_DDR, and may contain SDR instructions. The output and input timing on FlexSPI differs for SDR and DDR sequences. When included as part of a DDR sequence, SDR instructions execute differently from SDR instructions in an SDR sequence. See FlexSPI input timing and FlexSPI output timing.

25.3.6.1 Executing instructions on SPI interface

This section describes the execution of instructions on the SPI interface. For all instructions that receive bits from or transmit bits to flash devices:

- The bit order in one byte is higher on DATA7 than DATA0.
- The bit order is higher on port B than port A.

This order is shown in Figure 107.

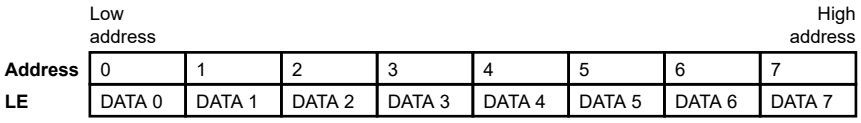


Figure 107. Bus bit order

Table 506. SPI instructions

Instructions	Description
Command (CMD_SDR, CMD_DDR)	Normally used to transmit a command code to the external device. The command code is the 8-bit operand in the instructions. Command code is sent to both port A and port B in parallel mode. See Flash memory access sequence examples.
Address (RADDR_SDR, RADDR_DDR, CADDR_SDR, CADDR_DDR)	Normally used to send the flash memory access start address (row address or column address) to an external device. FlexSPI determines the bits of the row address or column address according to the AHB access address or IP command address. See Flash address sent to flash memory devices. The operand value in the instruction code represents the number of address bits to send. The bits in the row address or column address are sent to both port A and port B in parallel mode. For example, when NUM_PADSn = 8, a memory reading the flash memory address on four SCK edges requires the sum of bits in CADDR and RADDR instructions to be 32. Otherwise, FlexSPI does not generate four SCK edges of address phase. See Flash memory access sequence examples.

Table continues on the next page...

Table 506. SPI instructions (continued)

Instructions	Description
Mode (MODEx_SDR, MODEx_DDR)	Normally used to send mode bits to external devices. Mode bits are the lower bits of an operand. For example, the bit number is 1 for MODE1_SDR and MODE1_DDR, 2 for MODE2_SDR and MODE2_DDR, 4 for MODE4_SDR and MODE4_DDR, and 8 for MODE8_SDR and MODE8_DDR. The pad number should not be greater than the mode bit number. For example, you cannot set NUM_PADSn to 11b (Octal mode) for MODE4_* instructions. Mode bits are sent to both port A and port B in parallel mode. See Flash memory access sequence examples for more details.
Data Size (DATSZ_SDR, DATSZ_DDR)	Used to send the size of program data or read data (byte number) to external devices. This instruction is normally used in FPGA applications where the memory space in external device acts like a FIFO. The external device requires data size information to determine how much data are popped from or pushed into the internal FIFO. The operation value in the data size instructions is the bit number. Data size is sent to both port A and port B in parallel mode. See Flash memory access sequence examples.
Write (WRITE_SDR, WRITE_DDR)	Normally used to send program data to external device. Programming data is fetched from IP_TX_FIFO (IP Command) or AHB_TX_Buffer (AHB command). For details about flash program data size, see Flash memory access via AHB command and Flash memory access via IP command. The byte order for program data is always from low to high. Odd bytes are sent on port A and even bytes are sent on port B in parallel mode. See Flash memory access sequence examples for more details.
Read (READ_SDR, READ_DDR)	Normally used to receive flash memory data from external devices. Received data is put into IP_RX_FIFO (IP Command) or AHB_RX_Buffer (AHB command). For information about flash memory read data size, see Flash memory access via AHB command and Flash memory access via IP command. The byte order for reading data is always from low to high. Odd bytes are received from port A and even bytes are received from port B in parallel mode. See Flash memory access sequence examples.
Dummy (DUMMY_SDR, DUMMY_DDR, DUMMY_RWDS_SDR, DUMMY_RWDS_DDR)	Used to provide turnaround cycles on the SPI interface. During dummy instruction, the FlexSPI controller and external devices do not drive the SPI interface. See Programmable sequence engine for details about dummy cycle number. DUMMY_RWDS_DDR can be used for a HyperBus device that uses RWDS pin to indicate whether extra latency is needed. DUMMY_RWDS_SDR is reserved. The FlexSPI controller checks the DQS pin input level at the fourth cycle after SCLK output toggling is enabled. The DQS pin is called RWDS in HyperBus specification. See Flash memory access sequence examples. **NOTE** FlexSPI releases the bus after at least one cycle. To avoid data contention, the NUM_PADSn value for the dummy commands should be configured to match the number of data lines used for the external memory.
Learn (LEARN_SDR, LEARN_DDR)	Used to determine the correct sample clock phase for flash memory read data sampling. External device drives read data (or data learning pattern) bits on the FlexSPI interface. The FlexSPI controller compares the data line bits to the DLPR register to determine a correct sampling clock phase.

Table continues on the next page...

Table 506. SPI instructions (continued)

Instructions	Description
	Clock phase selection is automatically updated after executing learn instructions. See Data learning for more details. The operand value in learn instructions is the byte number. FlexSPI checks the same data pattern on each data line. The byte order is byte 0, byte 1, byte 2, byte 3, byte0, byte 1, and so on. • Byte 0 is DLPR register bits 7–0. • Byte 1 is DLPR register bits 15–8. • Byte 2 is DLPR register bits 23–16. • Byte 3 is DLPR register bit 31–24. The bit order is from high to low in each byte. See Flash memory access sequence examples.

25.3.6.2 Flash memory access sequence examples

Diagrams below all assume MCR0[SERCLKDIV] = 0.

Figure 108 shows an example SDR single I/O read sequence (Cypress Serial Nor Flash S25FS512S) in individual mode.

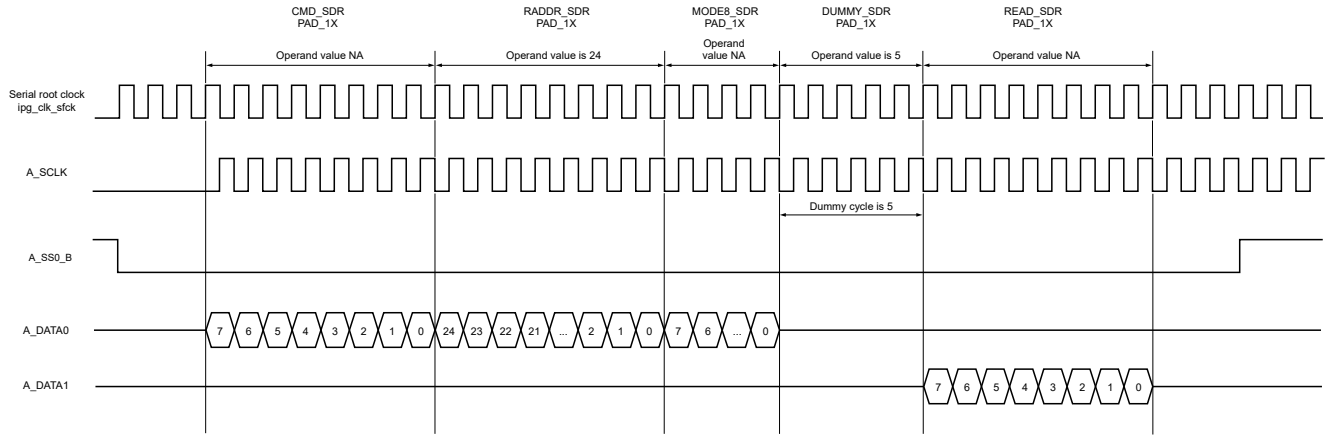


Figure 108. Flash memory access sequence example (SDR single I/O read sequence)

NOTE

- FlexSPI dummy instruction starts and ends at the rising edge of serial root clock.
- Device dummy cycle starts and ends at the falling edge of SCLK (on Cypress S25FS512S data sheet).

Figure 109 shows an example SDR quad I/O read sequence (Cypress Serial Nor Flash S25FS512S) in individual mode.

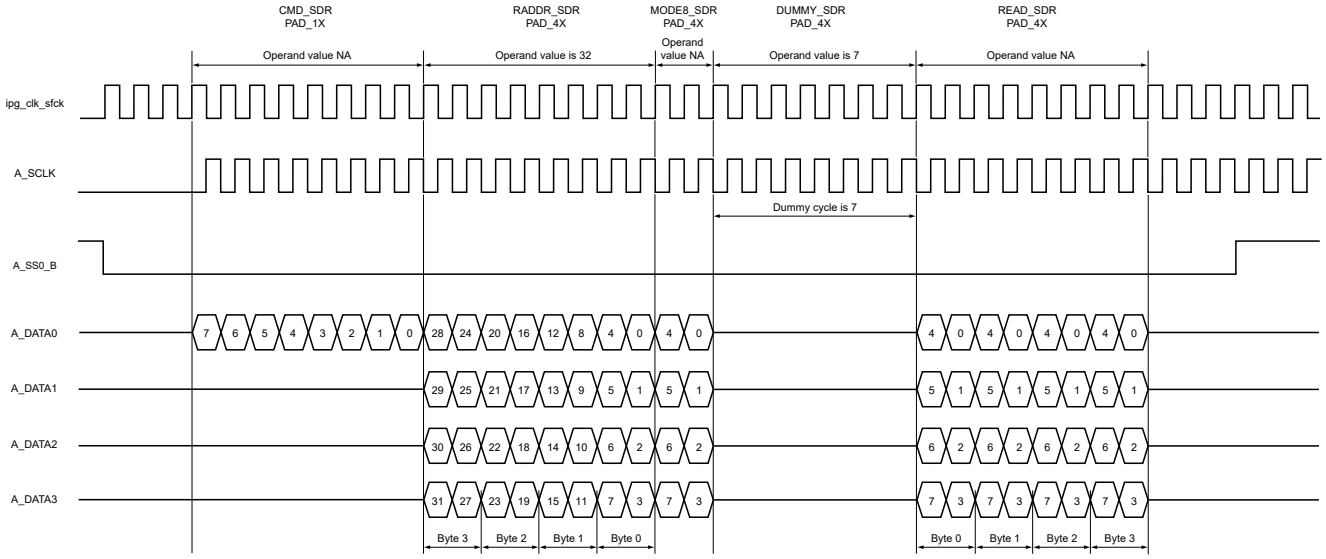


Figure 109. Flash memory access sequence example (SDR quad I/O read sequence)

NOTE

- FlexSPI dummy instruction starts and ends at the rising edge of serial root clock.
- Device dummy cycle starts and ends at the falling edge of SCLK (on Cypress S25FS512S data sheet).

Figure 110 shows an example DDR quad I/O read sequence (Cypress Serial Nor Flash S25FS512S) in parallel mode.

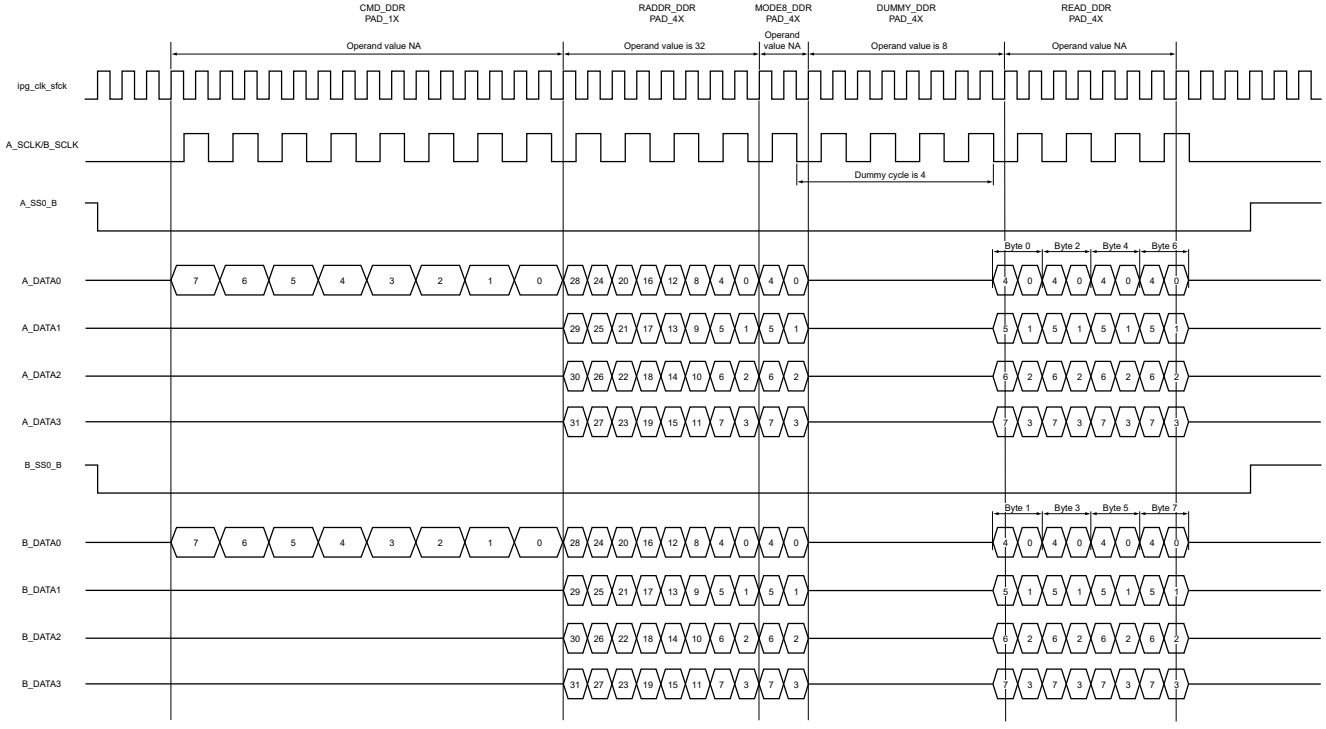


Figure 110. Flash memory access sequence example (DDR quad I/O read sequence)

Figure 111 shows an example learn instruction (not for a specified flash device) in individual mode.

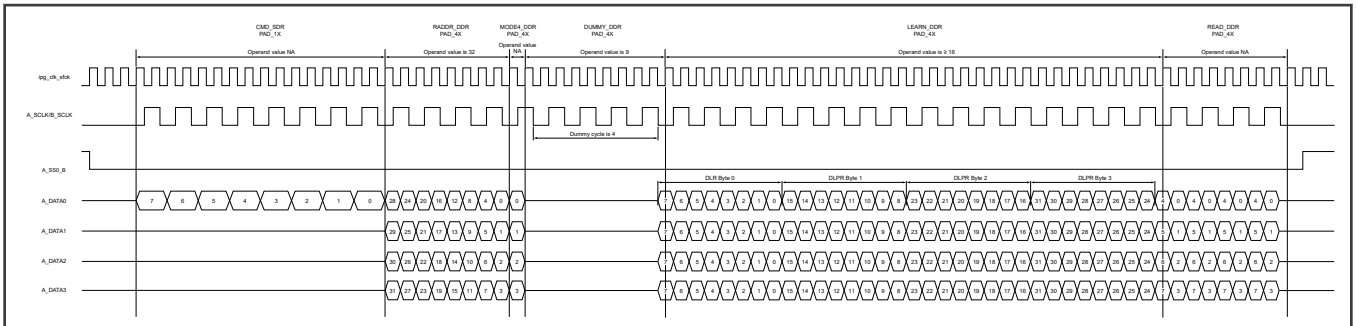


Figure 111. Flash memory access sequence example (data learning)

NOTE

- FlexSPI dummy instruction starts and ends at the rising edge of serial root clock.
- Device dummy cycle starts and ends at the falling edge of SCLK.
- The operand value of DUMMY_DDR instruction is odd because the total cycle number before DUMMY_DDR cycle is odd.

Figure 112 shows an example HyperBus device read transaction (single latency count) in individual mode.

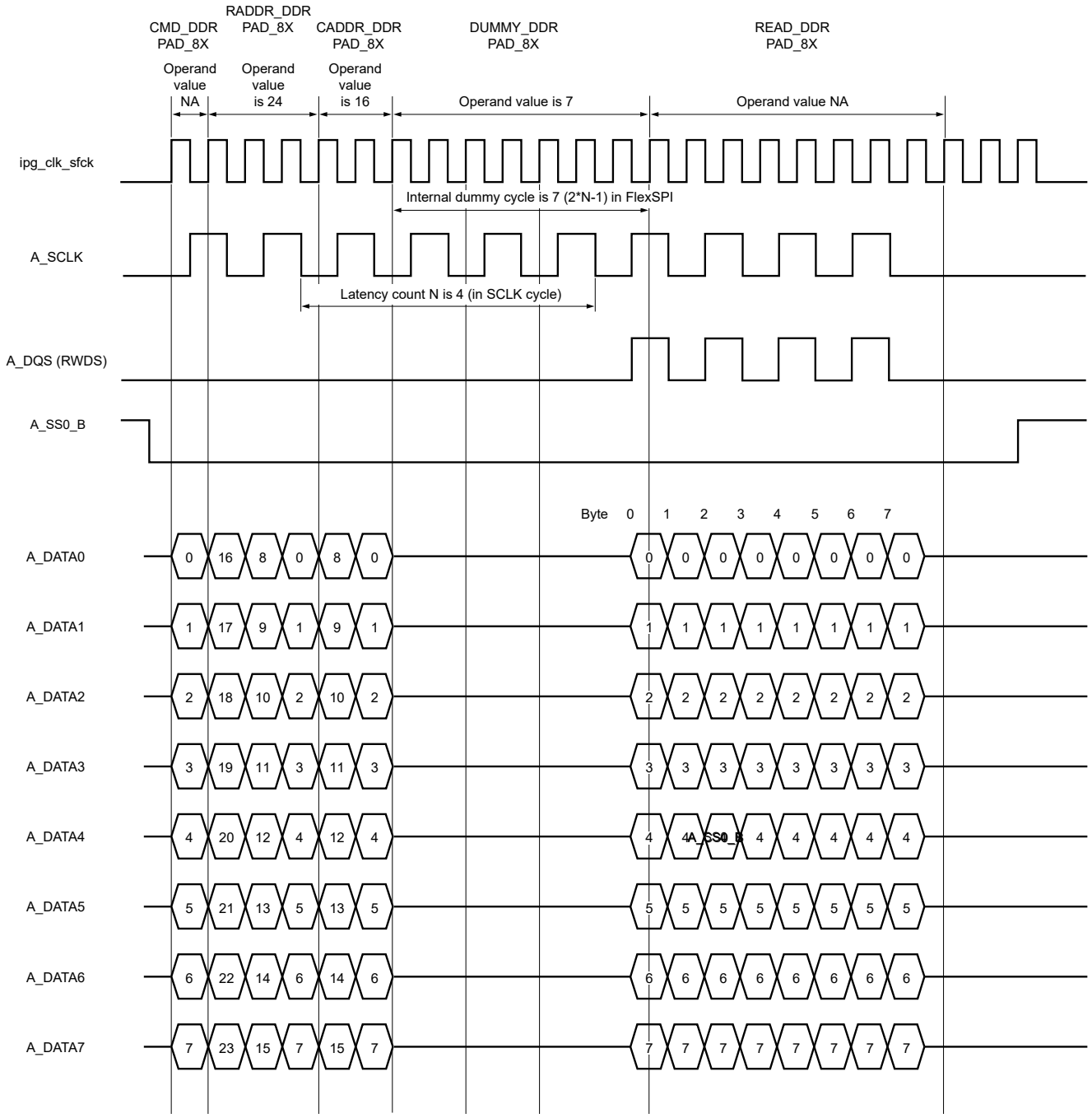


Figure 112. HyperBus device read transaction with single latency count

NOTE

FlexSPI continues to drive the clock until it has latched all of the read data it is expecting. On-chip delay for the data to reach FlexSPI can increase the number of SCK clock pulses.

Figure 113 shows an example HyperBus device read transaction (additional latency count) in individual mode.

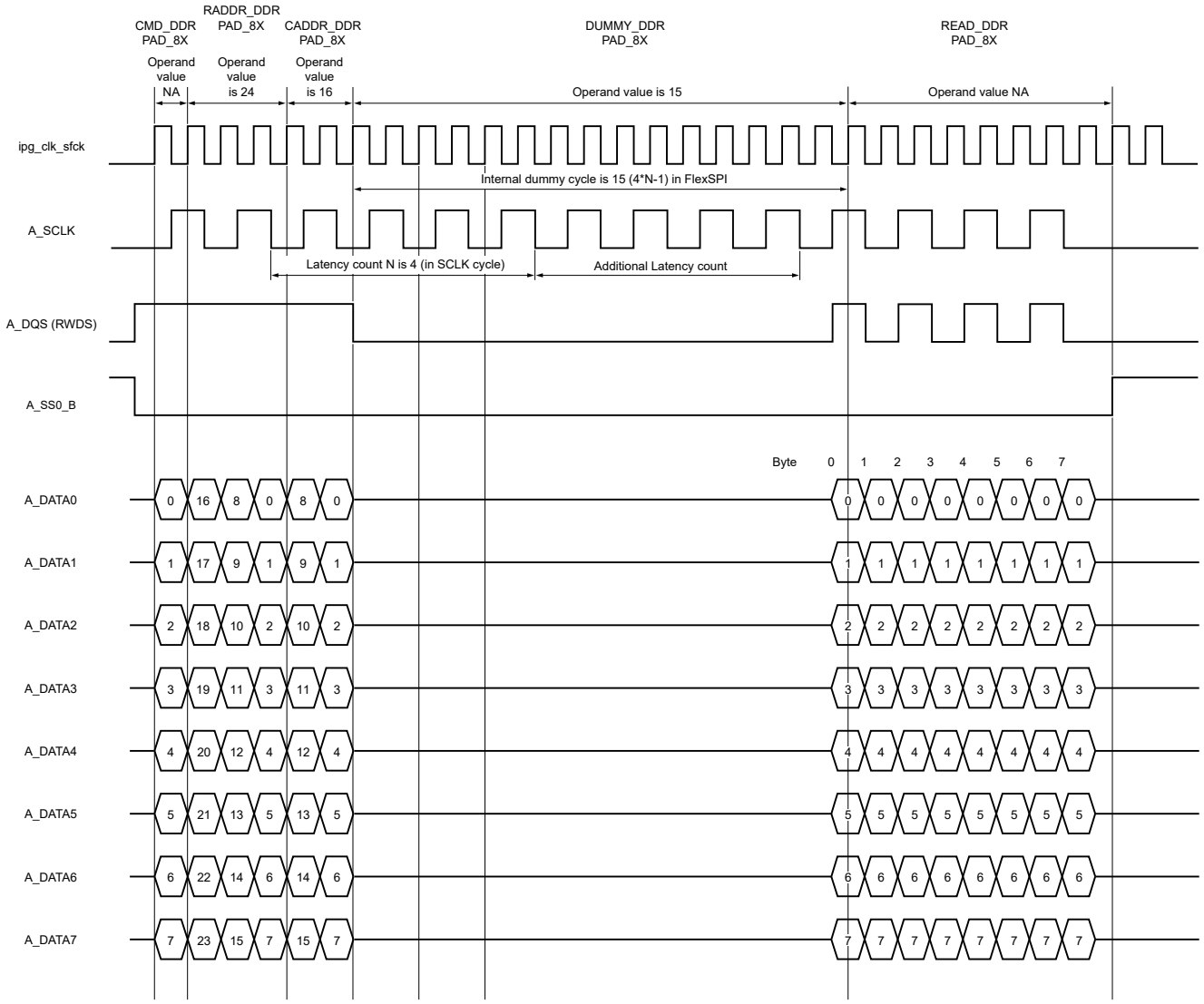


Figure 113. HyperBus device read transaction with additional latency count

Figure 114 shows an example HyperBus device write transaction (single latency count) in individual mode.

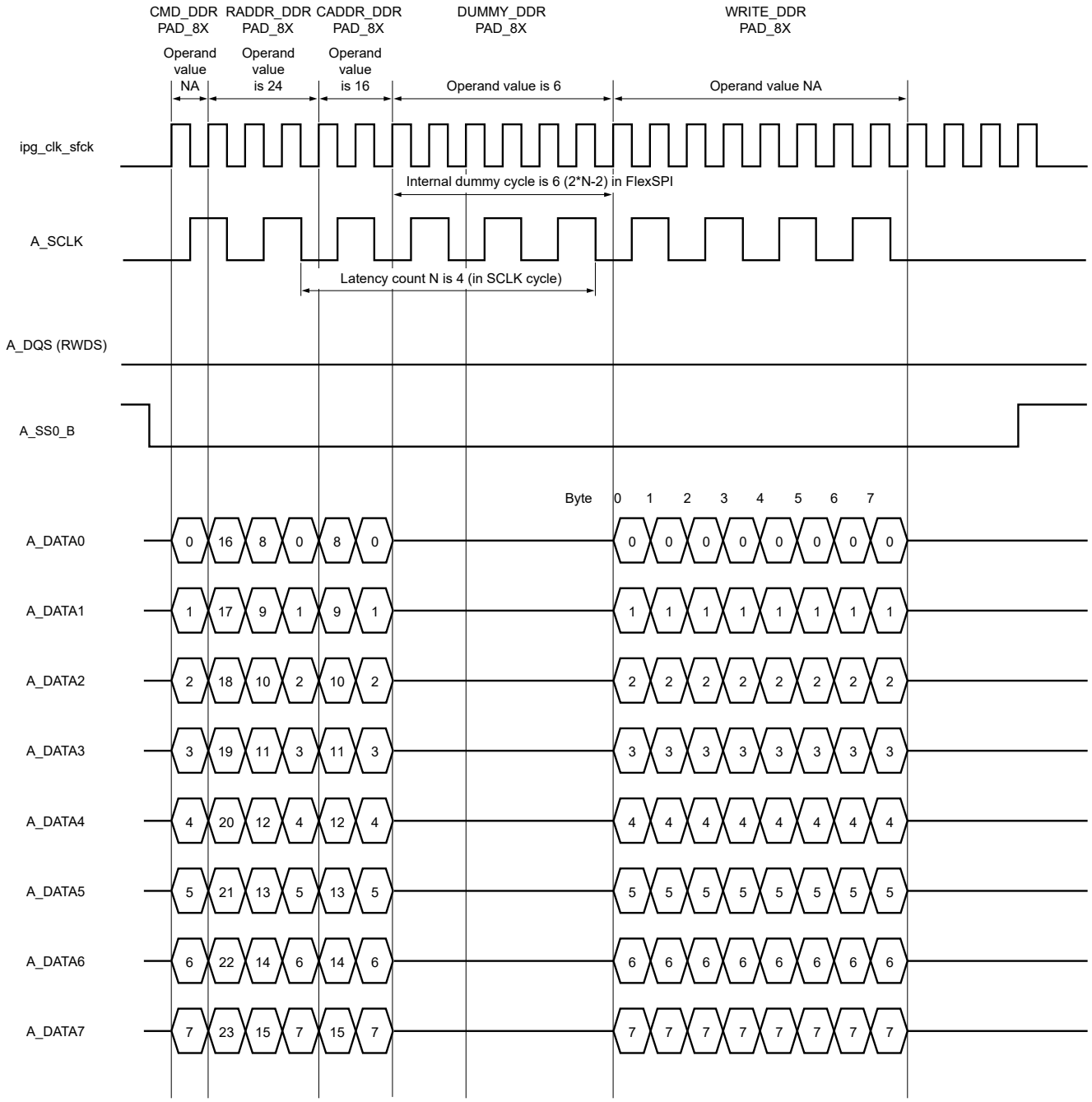


Figure 114. HyperBus device write transaction with single latency count

Figure 115 shows an example HyperBus device write transaction (additional latency count) in individual mode.

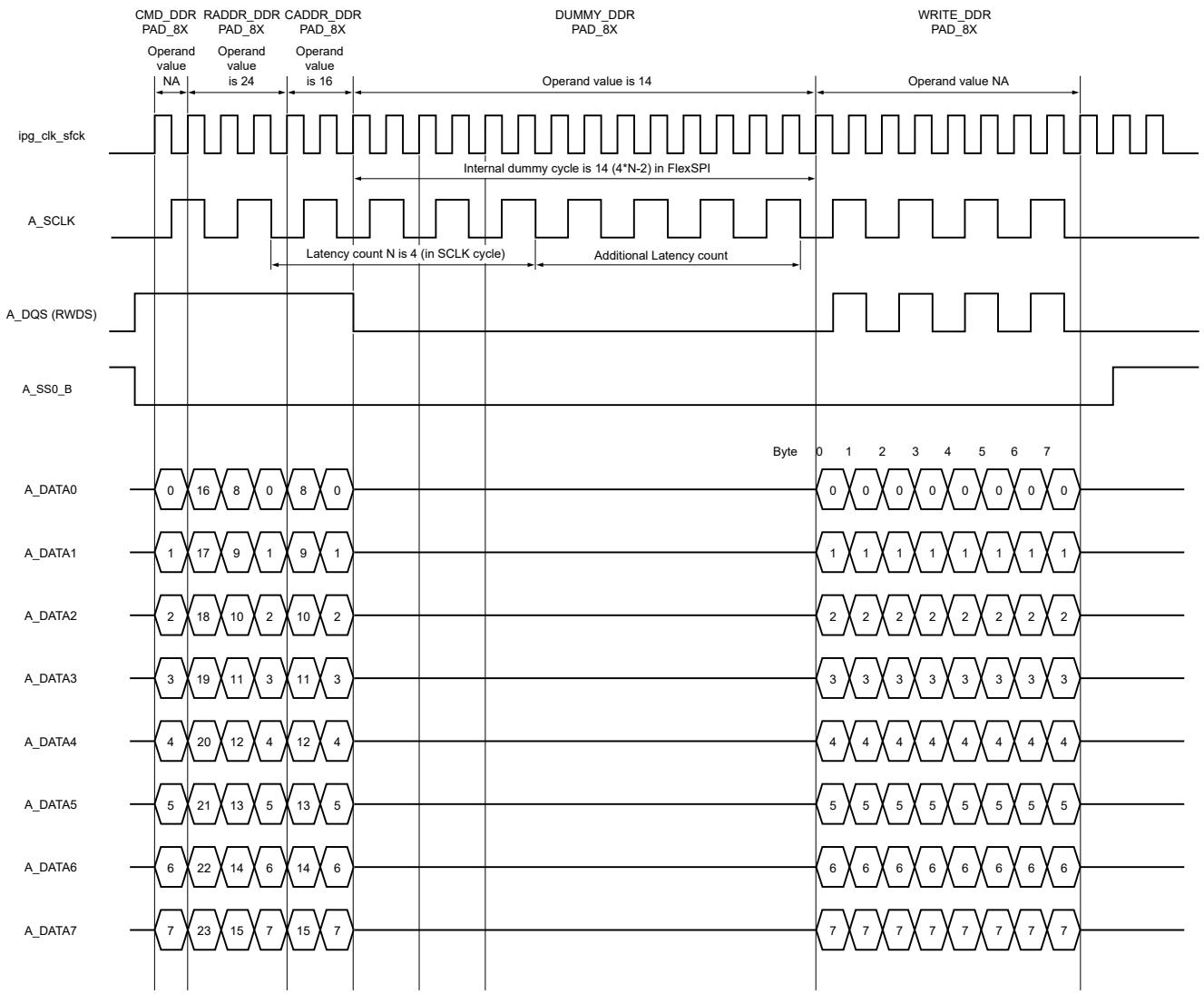


Figure 115. HyperBus device write transaction with additional latency count

25.3.7 Clocking

This section describes FlexSPI clocks and special clocking requirements.

Table 507. Clock usage

Clock name	Description	Comment
serial clock root (ipg_clk_sfck)	Root clock for Serial domain	FlexSPI core clock, used to generate the serial clock output. The clock is optionally divided using MCR0[SERCLKDIV] to obtain the SCLK.

Table continues on the next page...

Table 507. Clock usage (continued)

Clock name	Description	Comment
		Its frequency is same as SPI SCK frequency in SDR mode if MCR0[SERCLKDIV] = 0.
ahb clock (hclk)	AHB bus clock	hclk frequency is higher than 1/4 of serial root clock, so internal async FIFO is never full. This clock is used for internal logic in FlexSPI.
ipg clock (ipg_clk)	IPS bus clock	Register access clock. ipg_clk must be equal to hclk or an integer division of hclk. For example, ipg_clk = hclk/2.
SCLK	FlexSPI serial clock. Output clock on SCLK pin	Half clock frequency of serial clock root in DDR mode, and same frequency as serial clock root in SDR mode. Clock output toggles during the entire flash memory access sequence.
DQS_OUT	Dummy Read Strobe output	Same frequency as SCLK. Clock output toggles during READ and LEARN instructions. The DQS ensures that the data can be strobed correctly. It is only used for data strobe, instead of command or address strobe. The data includes useful read data and data learning patterns.
DQS_IN	Sample clock for receive data	Same frequency as SCLK. Sample clock comes from looped-back dummy read strobe, looped-back SCLK, or flash-memory-provided read strobe.

25.3.8 Interrupts

Table 508 describes all interrupts that FlexSPI generates.

Table 508. Flash memory address, row address, and column address

Interrupt	Enable	Condition
IP command done	INTEN[IPCMDDONEEN]	An IP command is finished.
IP command grant error	INTEN[IPCMDGEEN]	An IP command grant timeout occurs (bus not granted after MCR0[IPGRANTWAIT] × 1024 AHB clock cycles). See Overview of error categories, flags, and triggered sources.
AHB command grant error	INTEN[AHBCMDGEEN]	An AHB command grant timeout occurs (bus not granted after MCR0[AHBGRANTWAIT] × 1024 AHB

Table continues on the next page...

Table 508. Flash memory address, row address, and column address (continued)

		clock cycles). See Overview of error categories, flags, and triggered sources.
IP command error	INTEN[IPCMDERREN]	A command check error or a command execution error for an IP command occurs. See Overview of error categories, flags, and triggered sources.
AHB command error	INTEN[AHBCMDERREN]	A command check error or a command execution error for AHB command occurs. See Overview of error categories, flags, and triggered sources.
IP_RX_FIFO watermark available	INTEN[IPRXWAEN]	The fill level of IP_RX_FIFO reaches the watermark level (IPRXFCR[RXWMRK]).
IP_TX_FIFO watermark empty	INTEN[IPTXWEEN]	The empty level of IP_TX_FIFO reaches the watermark level (IPTXFCR[TXWMRK]).
Data learning failed	INTEN[DATALEARNFAILEN]	No valid sample clock phase found after LEARN instruction is executed. See Overview of error categories, flags, and triggered sources.
Sequence execution timeout	INTEN[SEQTIMEOUTEN]	A sequence execution time exceeds the timeout wait time (MCR1[SEQWAIT]). For example, the following flash memory read command sequence lasts about 800_0000h cycles (ipg_clk_sfck). If SEQWAIT is set to FFFFh, a sequence timeout interrupt is generated. • IP command triggers timeout. • Flash memory read data size is 100_0000h bytes. • Flash memory accesses in single mode and SDR mode.
AHB bus timeout	INTEN[AHBBUSTIMEOUTEN]	AHB bus response timeout occurs. For example, AHB read sequence is not configured properly in LUT (such as without a READ instruction). No data is read from an external device. As a result, FlexSPI cannot reach the read data in AHB_RX_Buffers for the AHB read command. See Overview of error categories, flags, and triggered sources.
Write command stops SCLK	INTEN[SCKSTOPBYWREN]	IP_TX_FIFO is empty while executing a write command sequence. FlexSPI stops SCLK output clock toggling and waits for write data filling.

Table continues on the next page...

Table 508. Flash memory address, row address, and column address (continued)

Read command stops SCLK	INTEN[SCKSTOPBYRDEN]	IP_RX_FIFO is full while executing a read command sequence. FlexSPI stops SCLK output clock toggling and waits to read the data from IP_RX_FIFO.

25.3.9 AHB transaction split

When writing or reading AHB in a GCM region, and FLSHxCR0[SPLITWREN] = 1 or FLSHxCR0[SPLITRDEN] = 1, and the corresponding flash memory is accessed, all read and write transactions to external memory devices are split into subtransactions. This feature is used for RAM devices with a self-refresh limit that has a maximum chip select assertion time interval requirement. If FlexSPI operates at a low frequency, a single AHB burst can violate this timing requirement.

Consider an example where a DDR RAM with SCK = 20 MHz has only one data line. If an AHB burst is 128 bytes long, FlexSPI requires about 26 µs to finish the whole data reading. If the timing interval is 6 µs, this read operation violates the RAM refresh limitation. The RAM cannot refresh the data during the read operation, which may cause data corruption. The situation is similar for AHB write operations. This feature can be enabled to reduce the chip select assertion time.

NOTE

This feature is transparent to software and it significantly impacts performance, so enable it only when necessary. Base your decision on the maximum supported AHB burst size of the current chip, the clock frequency, and the refresh limitation of external RAM.

25.3.10 Inline PRINCE encryption and decryption (IPED) CTR

IPED is a PRINCE-based encryption-decryption IP with a 64-bit block cipher and 128-bit key. FlexSPI supports automatic encryption on IP and AHB write commands, and automatic decryption on AHB read commands.

NOTE

The read or write access address sent to IPED.i_data is always 64 bits (eight bytes).

- For read operations, the start address and fetch size are always eight-byte-aligned. FlexSPI guarantees this condition.
- For write operations, software should manage any partial write that is not eight-byte-aligned. For example, before performing page erase and page program operations, the eight-byte boundary data should be read back.

Changing the register setting of IPEDCTRL changes the behavior of IPED.

To use IPED, apply these settings:

- AHBCR[READADDROPT] = 1
- AHBCR[PREFETCHEN] = 1 or AHBCR[READSZALIGN] = 1

FlexSPI supports a maximum of 7 address ranges for different 64-bit IPED.i_iv to encrypt or decrypt data.

The IPEDCTXnSTART and IPEDCTXnEND registers control the 7 ranges. There must be no overlap among these 7 ranges. Otherwise behavior of IP is undefined. Any single read or write request to flash memory must not cross one context boundary. For AHB read and write operations, hardware can automatically keep the boundary by writing 1 to AHBCR[ALIGNMENT]. Software must maintain the IP command write boundary. Any read or write request across context boundary leads to wrong data to or from the external device.

IPED context control 0 (IPEDCTXCTRL0) and IPED context control 1 (IPEDCTXCTRL1) control the write ability of IPEDCTXnSTART and IPEDCTXnEND. By default, IPEDCTXnSTART and IPEDCTXnEND cannot be modified after reset is enabled. Table 509 lists details.

Table 509. IPEDCTXnSTART and IPEDCTXnEND write ability

IPEDCTXCTRL0[CTXn_FREEZE0]	IPEDCTXCTRL1[CTXn_FREEZE1]	CTXn_FREEZE0 and CTXn_FREEZE1 writable	IPEDCTXnSTART and IPEDCTXnEND writable
10b	10b	yes	yes
01b (default)	10b (default)	yes	no
don't care	00b or 01b or 11b	no	no
00b or 11b	don't care	no	no

25.3.11 PRINCE IP GCM function support

7 regions of PRINCE have separate IV and AAD defined by start address and end address. IPED context control 0 (IPEDCTXCTRL0) and IPED context control 1 (IPEDCTXCTRL1) define the read and write control of each region, as before.

25.3.11.1 GCM read operation

Every AHB read request from the chip is aligned to a 32-byte boundary automatically by PRINCE IP. The read fetch size is also aligned to 32 bytes.

If prefetch is enabled, software must guarantee that the AHB_RX_Buffer size of the controller initiating the GCM region read operation is a multiple of 32 bytes.

Any valid read operation must be in 32-byte units, because the authentication check of PRINCE IP is based on 32 bytes of data.

PRINCE IP triggers the converted read address and read size based this rule: every 32 bytes of raw data = 32 bytes of encrypted data + an eight-byte tag.

The software setting and sequences to execute an AHB GCM read operation are:

1. Configure these fields:
   - Write 1 to IPEDCTRL[AHBGCMRD].
   - Write 1 to IPEDCTRL[IPED_EN].
   - Write 1 or 0 to IPEDCTRL[CONFIG], depending on test vectors.
2. Write the lower 32 bits of IV to IPED Context0 IV0 (IPEDCTX0IV0), and write the upper 32 bits of IV to IPED Context0 IV1 (IPEDCTX0IV1).
3. Write the start address to IPEDCTX0START[start_address]. Write 1 to IPEDCTX0START[GCM].
4. Write the end address to End Address of Region (IPEDCTX0END - IPEDCTX6END).
5. Write the lower 32 bits of AAD to IPED Contexta Additional Authenticated Datac (IPEDCTX0AAD0 - IPEDCTX6AAD1), and write the upper 32 bits of AAD to IPED Contexta Additional Authenticated Datac (IPEDCTX0AAD0 - IPEDCTX6AAD1).
6. Write 1 to AHBCR[PREFETCHEN] or AHBCR[READSZALIGN]. The prefetch buffer size must be a multiple of 32 bytes.

   If the current external device has boundary limitation (for instance pSRAM has 1 kB limitation), below additional setting is needed
   - AHBCR[ALIGNMENT] = 01 for a 1 kB boundary.
   - FLSHA1[31], FLSHA2[31], FLSHB1[31], or B2CR0[31] must be 1 depending on which chip select connects APMEM.

If PRINCE IP indicates an authentication check failure, all 32 bytes are discarded, and a parameter-defined value is put onto the AHB bus. An AHB bus error flag is set to indicate that the current read value is invalid. If INTEN[AHBGCMERREN] = 1, an interrupt is also generated.

25.3.11.2 GCM AHB Write

PRINCE IP supports AHB GCM write operations. The write start address must be 32-byte-aligned. The data size must be a multiple of 32 bytes. PRINCE IP does not maintain this requirement.

Any GCM write request violating this requirement leads to undefined behavior.

All software settings are the same as they are for GCM AHB read, except IPEDCTRL[AHGCMWR] must be 1.

Current AHB GCM write operations only support pSRAM. If the pSRAM has a boundary limitation, similar to AHB read operations, the following settings are needed:

- AHBCR[ALIGNMENT] = 01 for a 1 KB boundary.
- FLSHA1[30], FLSHA2[30], FLSHB1[30], or B2CR0[30] must be 1 depending on which Chip Select connects APMEM.

25.3.11.3 GCM IP command write operation

FlexSPI supports IP command write operations with PRINCE IP GCM.

During the address conversion, software controls the number of IP commands. Hardware performs no size or address alignment.

The list below shows a software sequence with a program size of 256 bytes as an example.

1. Configure these fields:
   - Write 1 to IPEDCTRL[IPED_EN].
   - Write 1 or 0 to IPEDCTRL[CONFIG], depending on test vectors.
2. Write the lower 32 bits of IV to IPED Context0 IV0 (IPEDCTX0IV0), and write the upper 32 bits of IV to IPED Context0 IV1 (IPEDCTX0IV1).
3. Write start address | bit0 GCM = 1 to IPEDCTX0START.
4. Write the end address to End Address of Region (IPEDCTX0END - IPEDCTX6END).
5. Write the lower 32 bits of AAD to IPED Contexta Additional Authenticated Datac (IPEDCTX0AAD0 - IPEDCTX6AAD1), and write the upper 32 bits of AAD to IPED Contexta Additional Authenticated Datac (IPEDCTX0AAD0 - IPEDCTX6AAD1).
6. Write the target system address to IPCR0[SFAR].
7. Write 1_0000_0000h to IPCR1[IDATSZ] to set the size to 256 bytes.
8. Configure IPTXFCR[TXWMRK] to be a multiple of 32 bytes.
9. Write 1 to IPEDCTRL[IPGCMWR].
10. If it is not same as previous programmed system address, write the target flash memory address to IPCR0[SFAR].
11. If INTR[IPTXWE] = 1, fill raw data into the IP transmit FIFO up to the IPTXFCR[TXWMRK] level.
12. Write 1 to INTR[IPTXWE] to clear it, then wait for INTR[IPTXWE] to become 1 again, back to one step before. It only shows interrupt way to fill the FIFO. DMA way is same as before. See Writing data to IP transmit FIFO.
13. Each time you write 256 bytes of raw data into the IP transmit FIFO, trigger an IP command by:
    a. Writing 1 to IPCMD[TRG]
    b. Changing the IPCR0 address to the next 256 bytes.

    Each time you program 1 kB (four times of IPCMD[TRG]), write 1 to IPCMD[TRG] one additional time to trigger an extra IP command.

Repeat the above steps until all data are written into flash memory.

NOTE

- Do not use IPTXFCR[CLRIPTXF] in the middle of GCM page program.
- Do not clear IPEDCTRL[IPGCMWR] in the middle of programming.
- Do not use a write command in LUT to write a flash register in the middle of programming.
- If there is no region assigned to the data, region (6) is the default for encryption. If region 6 cannot be used for encryption, software should disable IPEDCTRL[IPGCMWR].
- If GCM IP command write is enabled but the data does not hit any region, region 6 is the default region for encryption. In this case, if region 6 is not used for encryption, software must disable IPEDCTRL[IPGCMWR].

25.3.12 Flash memory access via IP command

Follow these steps to trigger a flash memory access via IP command.

1. If this command is a programing command, fill the IP transmit FIFO with programming data (for instance, programming flash data and flash memory status registers.)
2. Write the flash memory access start address to IPCR0[SFAR].
3. Write the read or program data size to IPCR1[IDATSZ]. Write the sequence index to IPCR1[ISEQID]. Write the sequence number to IPCR1[ISEQNUM].
4. Write 1 to IPCMD[TRG] to trigger a flash memory access command.
5. Wait for the INTR[IPCMDDONE] flag to set or for the IP command done interrupt to fire, indicating the command has completed on the FlexSPI interface.

NOTE

- The IP transmit FIFO can be filled before or after writing the IPCR0, IPCR1, and IPCMD registers. If the SFM command starts with the IP transmit FIFO empty, FlexSPI stops SCLK toggling to wait for transmit data ready automatically.
- The IPCMD register must be written after writing the IPCR0 and IPCR1 registers.
- One IP command can issue up to eight command sequences.
- You cannot issue another IP command before the previous IP command is finished. Performing this action results in unknown behavior.

If this command is a read command, all read data from flash memory is put into the IP receive FIFO. Software must read data from the IP receive FIFO via AHB bus or IP bus. When the IP receive FIFO is full and the flash device still contains data to read, FlexSPI stops SCLK toggling until the FIFO has free space. See SCLK stop.

A triggered serial flash command contains:

- A flash memory access start address. IPCR0[SFAR] determines this address.
- A flash chip select. The flash memory access address and flash memory size setting (FLSHxCR0[FLSHSZ]) determine the chip select.
- A flash command sequence index and sequence number. FlexSPI sequentially executes the sequences indexed from IPCR1[ISEQID] to (IPCR1[ISEQID] + IPCR1[ISEQNUM]) in LUT.
- Flash individual/parallel memory access mode:

  Determined by IPCR1[IPAREN].
- The flash memory read or program data size, in bytes.
  - If the value of IPCR1[IDATSZ] is nonzero, the data size is IPCR1[IDATSZ].
  - If the value of IPCR1[IDATSZ] is zero, the operand value in the read or write instruction determines the data size.

NOTE

- Software must ensure that the last sequence index never exceeds the LUT sequence number. That is, IPCR1[ISEQID] + IPCR1[ISEQNUM] must be less than 16.
- For sequence numbers greater than one, the data size is applied to every command sequence.
- If there is no write or read instruction in the command sequence, the data size is ignored.

The IP command request is sent to the arbitrator after software triggers it. It is not executed on the FlexSPI interface until the arbitrator grants it. See Command arbitration.

25.3.12.1 Reading data from IP receive FIFO

FlexSPI puts read data from the external device into the IP receive FIFO for IP commands.

The data stored in the IP receive FIFO can be read using IPS and register or system and AHB bus transactions:

- 400C8100h–400C817Ch (by IPS bus)
- 10000000h–10000400h (by AHB bus)

If MCR0[ARDFEN] = 1, only the AHB bus can read the read data in the IP receive FIFO. Read access to the IP receive FIFO using IP bus always returns with data zero, and no bus error occurs.

If MCR0[ARDFEN] = 0, only the IPS bus can read the read data in the IP receive FIFO. Read access to the IP receive FIFO using the AHB bus triggers a bus error.

FlexSPI pushes read data into the IP receive FIFO in sets of 64 bits each time it receives 64 bits of data from an external device. When the number of read data bits is not 64-bit-aligned, FlexSPI pushes additional zero bits into the IP receive FIFO for the last push.

The processor or DMA can read the IP receive FIFO.

25.3.12.1.1 Reading via processor

To read data from the IP receive FIFO via processor, configure the items below.

- Write 0 to IPRXFCR[RXDMAEN].
- Write the watermark level to IPRXFCR[RXWMRK]. The watermark level is (IPRXFCR[RXWMRK] + 1) × 8 bytes.
- Write 1 to INTEN[IPRXWAEN] to enable the IP receive FIFO watermark available interrupt (optional).

The processor must poll INTR[IPRXWA] or wait for the IP receive FIFO watermark available interrupt before reading data from the FIFO. Waiting ensures that a watermark level amount of data is filled in the IP receive FIFO before reading.

After reading a watermark level amount of data from the FIFO, software must set the INTR[IPRXWA] flag to pop that data from the FIFO.

Figure 116 shows the reading flow from the IP receive FIFO via processor.

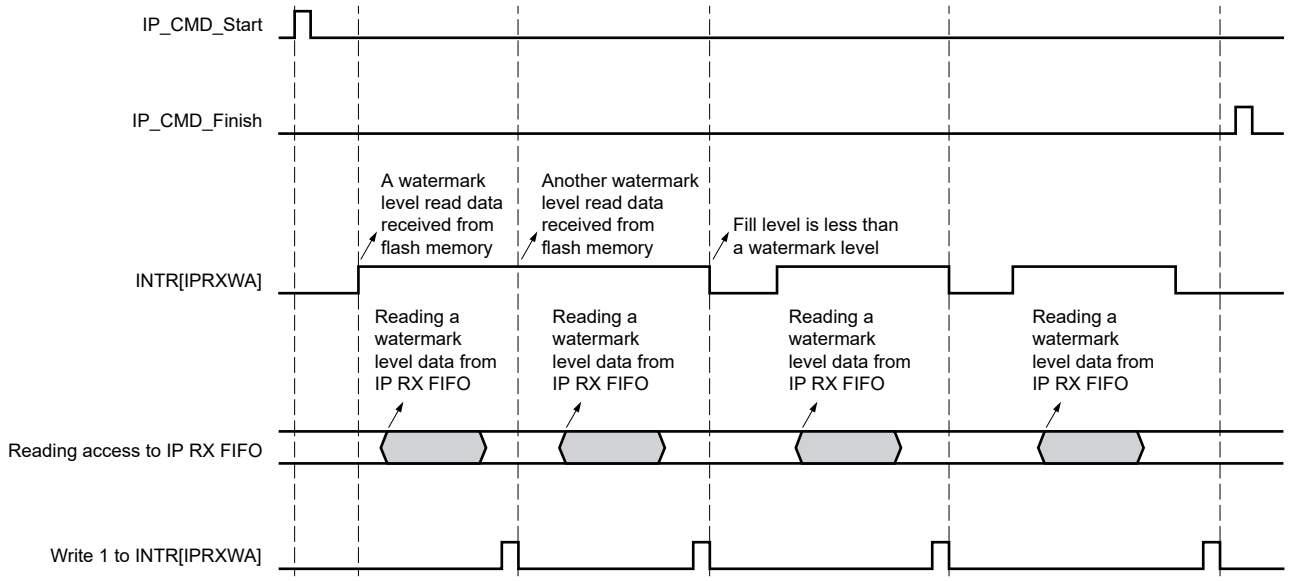


Figure 116. Reading IP receive FIFO via processor

NOTE

- The processor must read a watermark level of data from the IP receive FIFO before setting the INTR[IPRXWA] flag.
- The total flash read or program data size is not required to be a multiple of the watermark level. When the reading data size from the FIFO is less than a watermark level for the last time, software polls IPRXFSTS[FILL], not INTR[IPRXWA]. After all data from the FIFO are copied and all command sequences to flash memory are finished (INTR[IPCMDDONE] = 1), software sets IPRXFCR[CLRIPRXF] to clear the FIFO. If the FIFO is not cleared, the reading data will be incorrect for the next reading command to flash memory.
- Setting the INTR[IPRXWA] flag pops the IP receive FIFO data. Each read access to the FIFO does not pop the data.

25.3.12.1.2 Reading via DMA

To read data from the IP receive FIFO via DMA, configure the items below.

- Write 1 to IPRXFCR[RXDMAEN].
- Write the watermark level to IPRXFCR[RXWMRK]. The watermark level is (IPRXFCR[RXWMRK] + 1) × 8 bytes.
- Configure the DMA transfer minor loop size to match the watermark level.

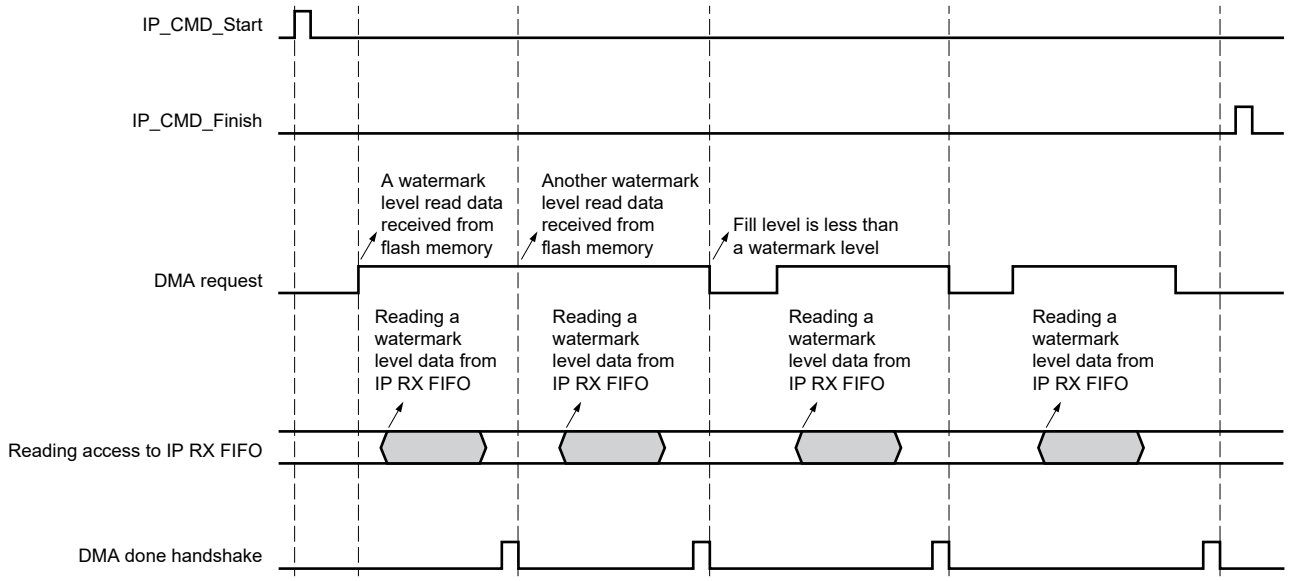


Figure 117. Reading from IP receive FIFO via DMA

NOTE

- When the fill level of the FIFO meets or exceeds the watermark level, a DMA request is generated. This request is level-valid, not pulse-valid.
- DMA reads a watermark level amount of data from the IP receive FIFO each time. (Configure the minor loop size to match the watermark level).
- DMA must return a done handshake (pulse valid) to FlexSPI each time it finishes reading a watermark level amount of data.
- The DMA done handshake (not each read access) pops the IP receive FIFO data.
- The total read and write data size (major loop size) must be a multiple of the watermark level. The DMA does not know when the data is ready for the last reading. When the data size is not a multiple of the watermark level, it is too complex for the DMA driver to poll IPRXFSTS[FILL].

25.3.12.2 Writing data to IP transmit FIFO

The programming data is put into the IP transmit FIFO, then FlexSPI transmits it to the flash memory. IPS or AHB access can write it:

- 400C_8180h–400C_81FCh (via IP bus)
- 11000000h–11000400h (via AHB bus)

To write the programming data into the IP transmit FIFO via AHB bus, write 1 to MCR0[ATDFEN]. The IP bus write access to the FIFO is ignored, but no bus error occurs.

To write the programming data into the IP transmit FIFO via IP bus, write 0 to MCR0[ATDFEN]. The AHB bus write access to the FIFO returns a bus error.

When FlexSPI fetches data to transmit, 64 bits of data from the IP transmit FIFO are popped. If the programming data size is not a multiple of 64 bits, the number of last popped valid bits is less than 64 bits. No error occurs; these invalid bits are not transmitted to the flash memory.

The processor or DMA can fill the IP transmit FIFO with programming data.

25.3.12.2.1 Writing via processor

To write data into the IP transmit FIFO via processor, configure the items below.

- Write 0 to IPTXFCR[TXDMAEN].
- Write the watermark level to IPTXFCR[TXWMRK]. The watermark level is (IPTXFCR[TXWMRK] + 1) × 8 bytes.
- Write 1 to INTEN[IPTXWEEN] to enable the IP transmit FIFO watermark available interrupt (optional).

The processor must poll INTR[IPTXWE] or wait for the IP transmit FIFO watermark empty interrupt before writing data to the FIFO. Waiting ensures that a watermark level amount of data is available in the IP transmit FIFO before writing.

After writing a watermark level amount of data to the FIFO, software must set the INTR[IPTXWE] flag to push that data into the FIFO. (The write pointer is incremented.)

NOTE

Setting the INTR[IPTXWE] flag pushes the IP transmit FIFO data. Each write access to the FIFO does not push the data.

Figure 118 shows the writing flow to the IP transmit FIFO via processor.

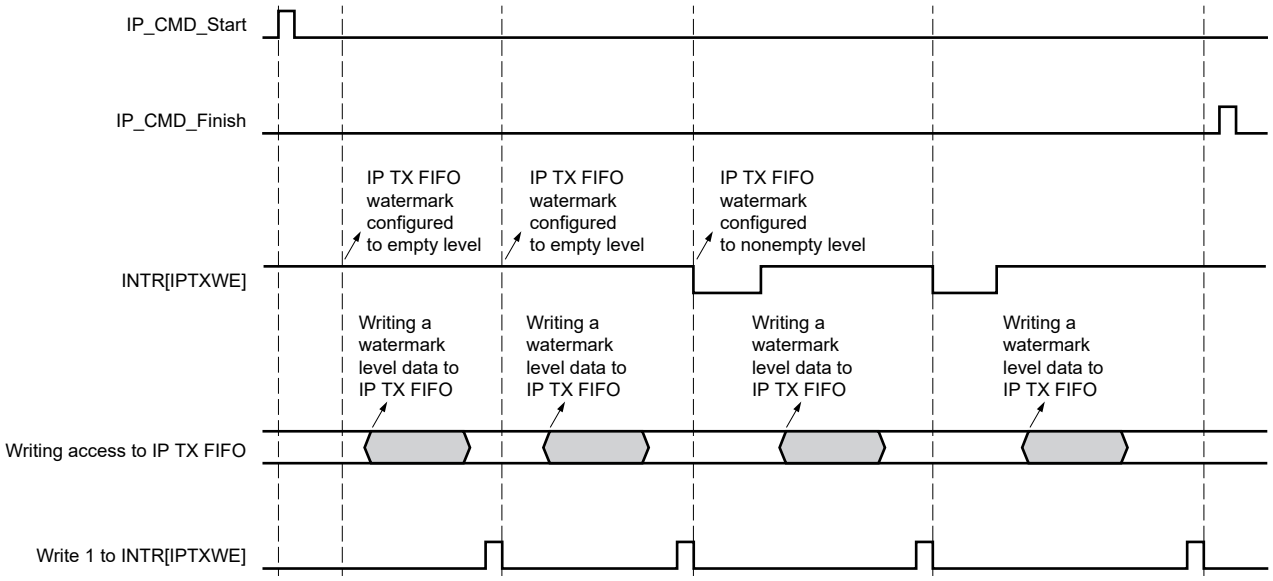


Figure 118. Writing to IP transmit FIFO via processor

NOTE

- The processor must write a watermark level amount of data into the IP transmit FIFO each time.
- The total flash write data size is not required to be a multiple of the watermark level. In this case, the writing data size to the FIFO is less than a watermark level for the last time. After all data to the FIFO are written and all command sequences to flash memory are finished (INTR[IPCMDDONE] = 1), the processor sets IPTXFCR[CLRIPTXF] to clear the FIFO. If the FIFO is not cleared, the programming data will be incorrect for the next programming command to flash memory.

25.3.12.2.2 Writing via DMA

To write data into the IP transmit FIFO via DMA, configure the items below.

- Write 1 to IPTXFCR[TXDMAEN].
- Write the watermark level to IPTXFCR[TXWMRK]. The watermark level is (IPTXFCR[TXWMRK] + 1) × 8 bytes.

- Configure the DMA transfer minor loop size to match the watermark level.

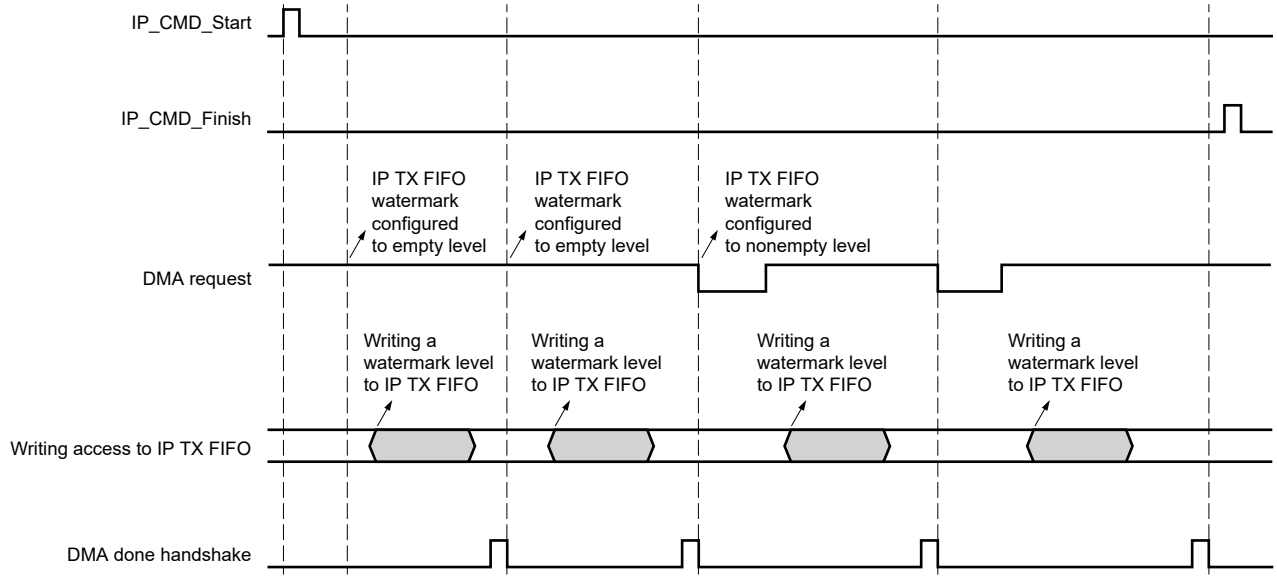


Figure 119. Writing into IP transfer FIFO via DMA

NOTE

- When there is empty space in the IP transmit FIFO greater than the watermark level, a DMA request is generated. This request is level-valid, not pulse-valid.
- DMA writes a watermark level amount of data into the FIFO each time. (Configure the minor loop size to match the watermark level.
- DMA returns a done handshake (pulse valid) to FlexSPI each time it finishes writing a watermark level amount of data.
- The DMA done handshake (not each write access) pushes the IP transmit FIFO data.
- The total program data size (major loop size) is not required to be a multiple of the watermark level. After writing all data into the IP transmit FIFO and all command sequences to flash memory are finished (INTR[IPCMDDONE] = 1), write 1 to IPTXFCR[CLRIPTXF] to clear the FIFO. Failure to clear the FIFO results in incorrect programming data for the next programming command to flash memory.

25.3.13 Flash memory access via AHB command

Flash memory can be accessed via AHB bus directly on the AHB address space. See chip-specific section for details about the AHB address space. This address space is mapped to serial flash memory in FlexSPI. AHB bus accesses to this address space trigger flash memory access command sequences as needed.

For AHB read access to serial flash memory, FlexSPI fetches data from flash memory into the AHB receive buffer, then returns the data on the AHB bus. For AHB write access to serial flash memory, FlexSPI buffers AHB bus write data into the AHB transmit buffer, then transmits the data to serial flash memory.

No software configuration or polling is needed for AHB commands, except for FlexSPI initialization. As with a normal AHB target, the AHB controller accesses the external flash device transparently.

AHB commands are normally used to access serial flash memory space. IP commands must be used to access the control and status registers or other spaces, such as a one-time programmable (OTP) space in an external flash device.

The AHB commands for read and write access are discussed in detail below.

25.3.13.1 AHB write access to flash memory

For AHB write access to flash memory, FlexSPI buffers the write data from the AHB bus into the internal AHB transmit buffer. FlexSPI then transmits the data to flash memory. FlexSPI only buffers write data for one AHB burst. Figure 120 shows the hardware operation in response to an AHB write access to flash memory.

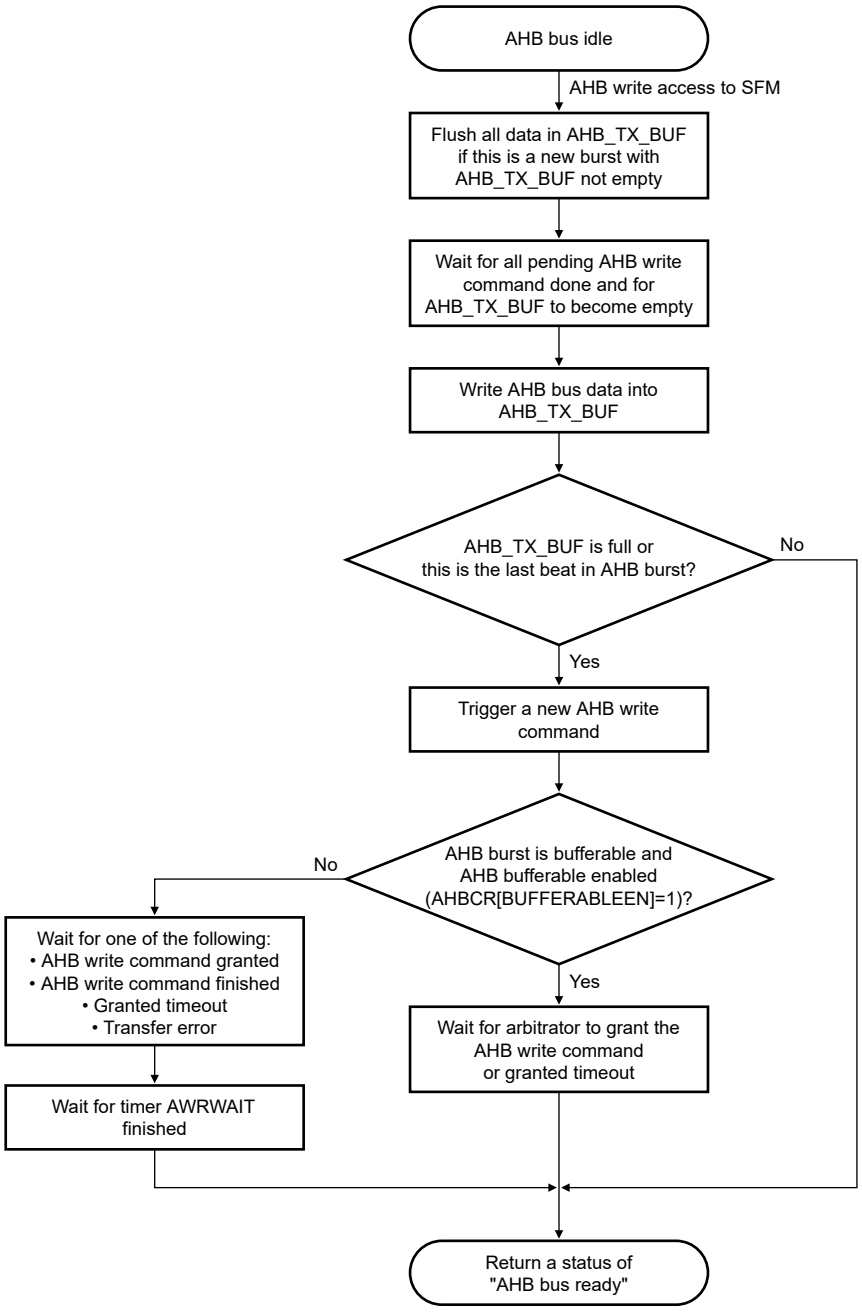


Figure 120. Hardware operation in response to AHB write access to flash memory

NOTE

1. If previous AHB burst is INCR write burst, AHB transmit buffer may not be empty when AHB bus is idle or new burst arrives. A new AHB write command is triggered to flush all data in AHB transmit buffer to external flash device. The new burst is held until this AHB write command finishes.
2. If the current access is bufferable, when the AHB write command is triggered, AHB ready is returned after the write command is granted. If the current access is non-bufferable, AHB ready is returned after the write command finishes.

FlexSPI triggers a new AHB write command in these cases:

- This beat is the last one in the current AHB burst, for any burst type except INCR.
- AHB transmit buffer is full after the current beat data is buffered.
- AHB bus becomes idle or a new burst arrives with the AHB transmit buffer not empty.

Table 510 describes the details of a triggered AHB write command.

Table 510. Triggered AHB write command

Item	Description
Flash memory access start address	Determined by AHB burst address. FlexSPI records the start address for the data in AHB transmit buffer, which is used as flash memory access start address.
Flash memory chip select	FlexSPI uses the settings of the flash memory access start address and flash memory size to determine the chip selection.
Flash memory command sequence index	Determined by FLSHxCR2[AWRSEQID].
Flash memory command sequence number	Determined by FLSHxCR2[AWRSEQNUM]. If AWRSEQNUM is not zero, multiple flash memory access command sequences are triggered every time for an AHB write command. The sequences indexed from AWRSEQID to (AWRSEQID + AWRSEQNUM) in LUT are executed sequentially.
Flash memory access mode (Individual/ Parallel)	Determined by AHBCR[APAREN].
Flash memory data size	Determined by byte number of buffered data in AHB transmit buffer.

The following examples indicate internal logic for AHB write access to flash memory, where AHB_TX_BUF is 64 bytes :

- AHB INCR 64-bit bufferable burst with address sequence 8h, 10h, 18h, 20h, ..., 50h (10 beats total).

  Two AHB write commands are triggered: the first command with flash memory start address 8h and data size 40h; the second command with flash memory start address 48h and data size 10h. See Figure 121.

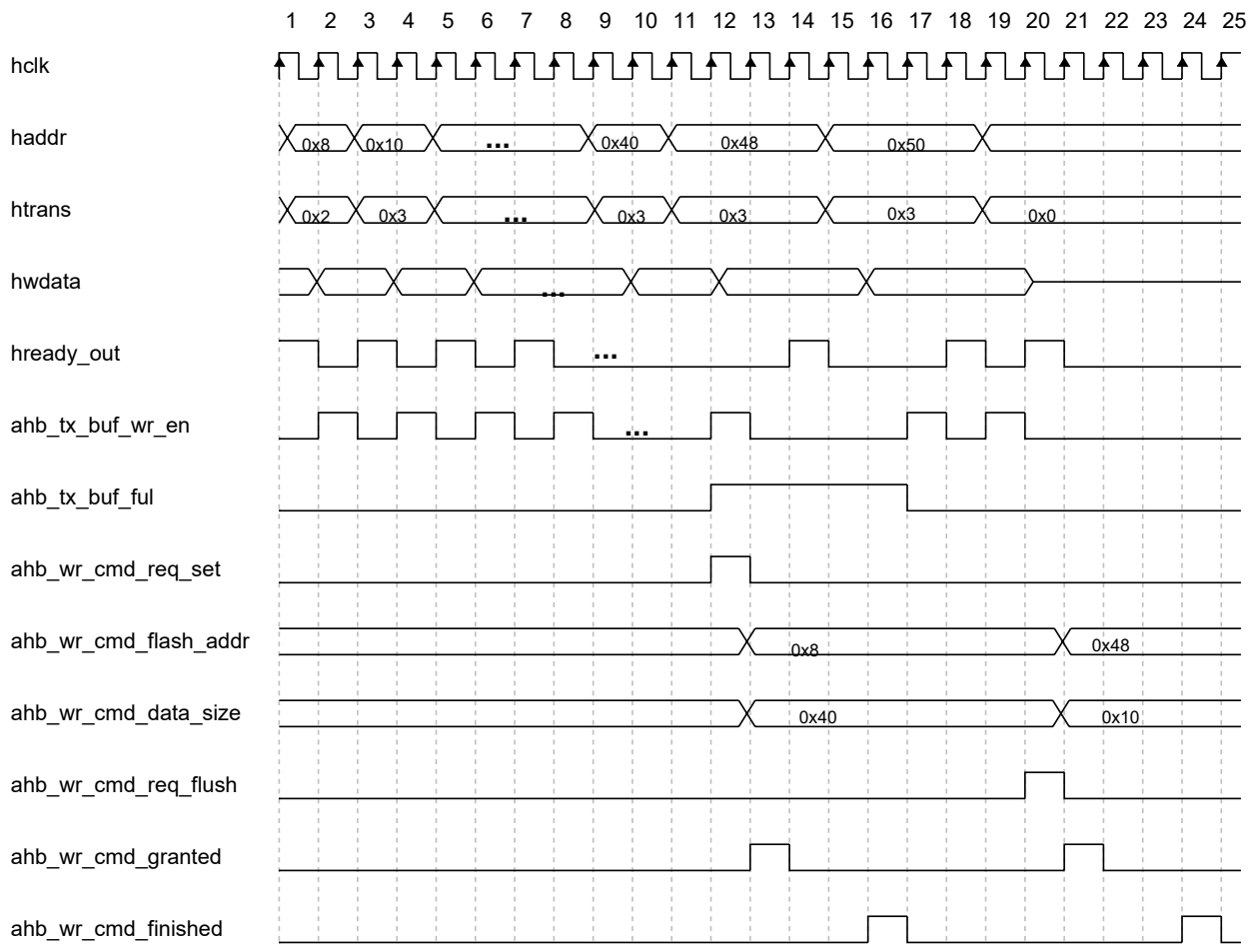


Figure 121. AHB write access (INCR, 64-bit, bufferable)

- AHB WRAP8 64-bit bufferable burst with address sequence 28h, 30h, 38h, 0h, 8h, 10h, 18h, 20h.

  One AHB write command is triggered with flash memory start address 0h and data size 40h. See Figure 122.

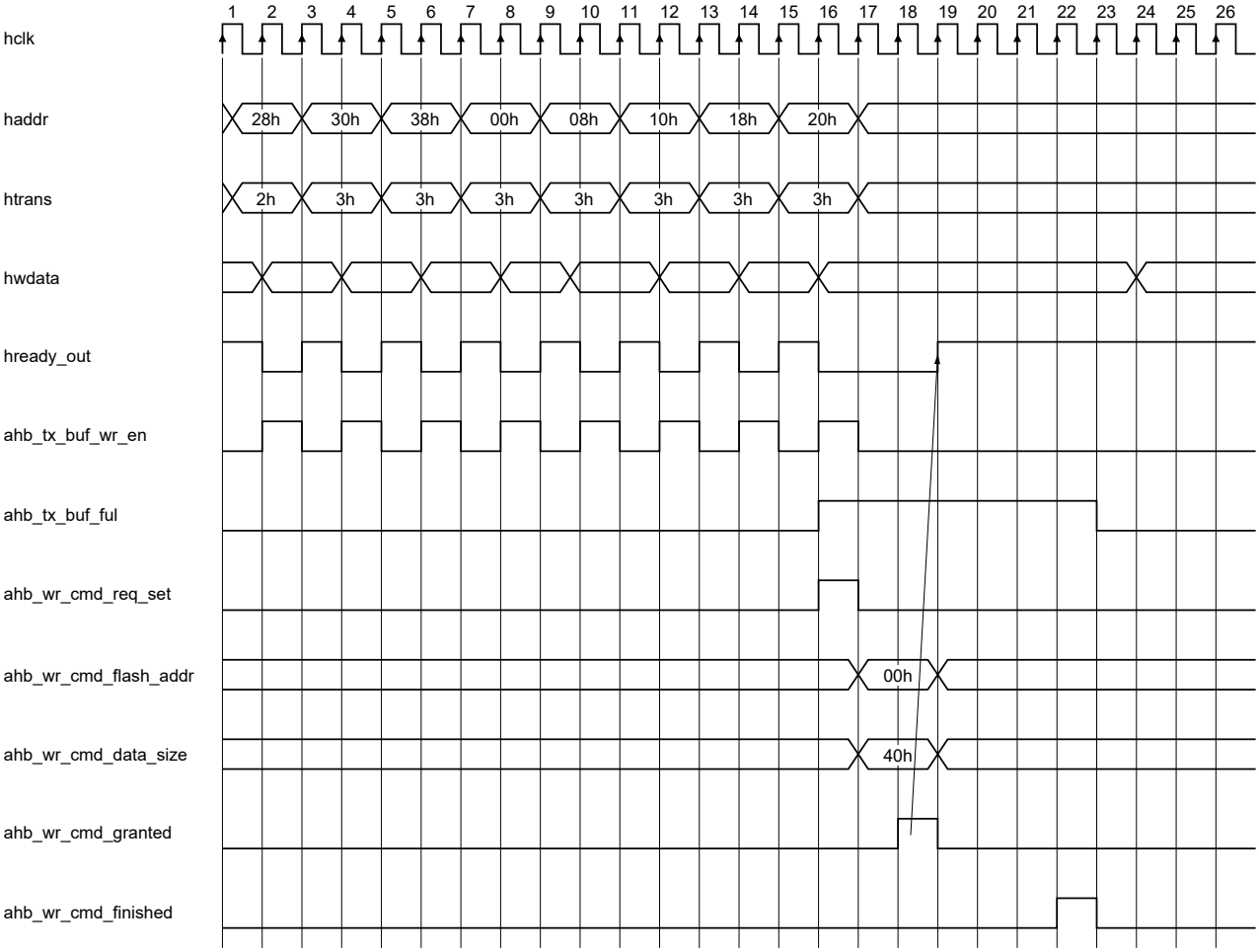


Figure 122. AHB write access (WRAP8, 64-bit, bufferable)

- AHB WRAP8 64-bit non-bufferable burst with address sequence 28h, 30h, 38h, 0h, 8h, 10h, 18h, 20h.

  One AHB write command is triggered with flash memory start address 0h and data size 40h.

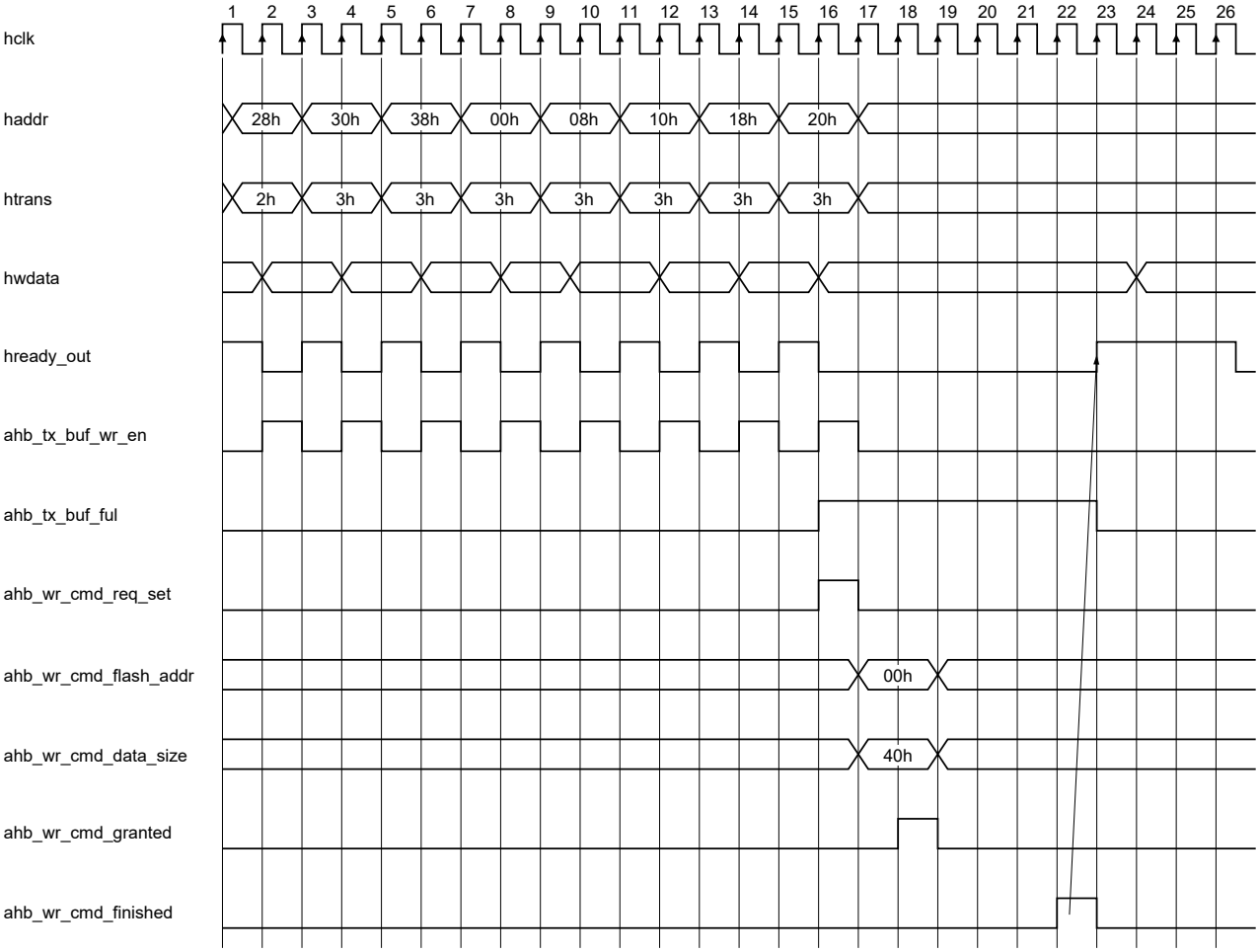


Figure 123. AHB write access (WRAP8, 64-bit, non-bufferable)

NOTE

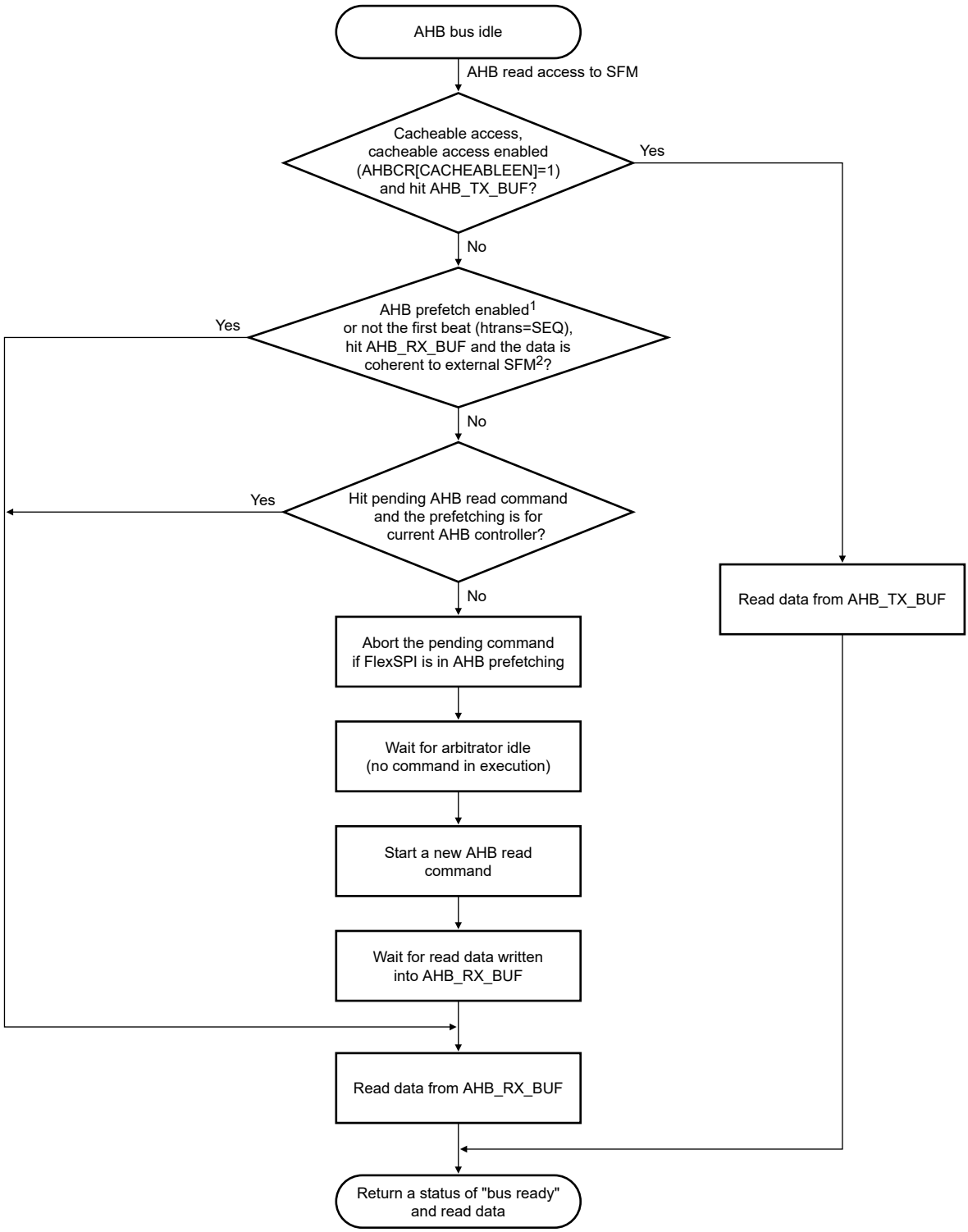
If the burst data size (in bytes) is greater than the AHB transmit buffer size, wrapper burst is not supported. For example, if AHB_TX_BUF is 64 bytes, AHB WRAP16 (16 × 64 bits = 128 bytes) write burst access is not supported.

25.3.13.2 AHB read access to flash memory

For AHB read access to flash memory, FlexSPI checks the burst access type and register setting to determine whether to access these locations:

- The AHB transmit buffer
- The AHB receive buffer
- A pending AHB read command

If all of these options are missed, FlexSPI triggers a new AHB read command to fetch data from flash memory. Figure 124 shows the hardware operation in response to AHB read access to flash memory.



Notes:
1) AHB prefetch is enabled when both AHBCR[PREFETCHEN]=1 and AHBRXBUFxCR0[PREFETCHEN]=1 (x is the AHB RX buffer ID for the current AHB controller).
2) The AHB RX buffer holds the data read from flash memory. This data may become incoherent if AHB writes to the same flash-memory address.

Figure 124. Hardware operation in response to AHB read access to flash memory

Table 511 describes the details of a triggered AHB read command.

Table 511. Triggered AHB read command

Item	Description
Flash memory access start address and data size	Determined by AHB address, burst type, and burst size. FlexSPI fetches read data from the start address for the current burst or beat. See Table 512 for details.
Flash memory chip select	FlexSPI uses the settings of the flash memory access start address and flash memory size to determine the chip select.
Flash memory command sequence index	Determined by FLSHxCR2[AWRSEQID].
Flash memory command sequence number	Determined by FLSHxCR2[AWRSEQNUM]. If AWRSEQNUM is not zero, multiple flash memory access command sequences are triggered every time for an AHB read command. The sequences indexed from AWRSEQID to (AWRSEQID + AWRSEQNUM) in LUT are executed sequentially.
Flash memory access mode (Individual/ Parallel)	Determined by AHBCR[APAREN].

NOTE

- FlexSPI automatically determines which ARDSEQNUM and ARDSEQID fields that the flash device uses as sequence ID for chip selection. See MCR2[SAMEDEVICEEN].
- FlexSPI determines which AHB receive buffer to use for the current AHB read access by controller ID. For details about the AHB receive buffer ID and AHB controller ID mapping, see AHB receive buffer management.
- You cannot allocate the size of the AHB receive buffer to be less than the AHB Burst size. This allocation results in unknown behavior.

Table 512. AHB read command flash memory start address and data size

Prefetch enable	Cross flash boundary	Burst type	Flash memory start address [28:0]	Data size
0	Never cross flash boundary, because AHB burst never crosses 1 KB boundary.	SINGLE, INCR4, INCR8, INCR16	hbeat_start_address	(hburst_end_address-hbeat_start_address)
		INCR	hbeat_start_address	byte size of current beat For INCR burst with prefetch disabled, each beat is handled same as SINGLE burst.
		WRAP4, WRAP8, WRAP16	hburst_start_address	(hburst_end_address-hburst_start_address)
1	No	INCR, SINGLE, INCR4, INCR8, INCR16	hbeat_start_address	ahb_rx_buf_sz
	No	WRAP4, WRAP8, WRAP16	hburst_start_address	ahb_rx_buf_sz

Table continues on the next page...

Table 512. AHB read command flash memory start address and data size (continued)

Prefetch enable	Cross flash boundary	Burst type	Flash memory start address [28:0]	Data size
	Yes	INCR, SINGLE, INCR4, INCR8, INCR16	hbeat_start_address	(flash_top_address-hbeat_start_address)
	Yes	WRAP4, WRAP8, WRAP16	hburst_start_address	(flash_top_address-hburst_start_address)

NOTE

- hbeat_start_address is HADDR input from the AHB controller for current beat.
- hburst_start_address (for example 00h) is the lowest address for current burst. hburst_end_address (for example 10h) is the highest address in current burst plus 1. For example, WRAP4 burst with HADDR 8h, Ch, 0h, 4h.
- ahb_rx_buf_sz is the buffer size in bytes of AHB receive buffer that the current AHB controller uses.
- flash_top_address is the top address of currently accessed flash memory.

25.3.13.3 AHB receive buffer management

There are 8 buffers (Buffer 0–7) in the AHB receive buffer. These buffers are transparent to AHB controllers. FlexSPI fetches flash memory data and returns it on the AHB bus automatically. No status register polling is needed for AHB read access to Serial Flash Memory (SFM).

The total size of AHB receive buffers is 1 KB. Use AHBRXBUF0CR0[BUFSZ]–AHBRXBUFCR7[BUFSZ] to configure the buffer size of each AHB receive buffer. The buffer size for Buffer 0 to Buffer 7 could be set to 0. If the buffer size is set to 0, FlexSPI ignores the related setting of AHBRXBUFxCR0[MSTRID]. Buffer 7 is used for all AHB controllers not assigned to Buffer 0–6. FlexSPI ignores the Buffer 7 size setting field (AHBRXBUF7CR0[BUFSZ]). Its buffer size is: (AHB receive buffer total size) - (the sum of the sizes of Buffer 0–6).

When an AHB read access to serial flash memory occurs, FlexSPI determines which buffer to use via this method:

1. If the controller ID equals AHBRXBUF0CR0[MSTRID] and AHBRXBUF0CR0[BUFSZ] is not zero, Buffer 0 is used.
2. If the controller ID equals AHBRXBUF1CR0[MSTRID] and AHBRXBUF1CR0[BUFSZ] is not zero, Buffer 1 is used.
3. If the controller ID equals AHBRXBUF2CR0[MSTRID] and AHBRXBUF2CR0[BUFSZ] is not zero, Buffer 2 is used.
4. If the controller ID equals AHBRXBUF3CR0[MSTRID] and AHBRXBUF3CR0[BUFSZ] is not zero, Buffer 3 is used.
5. If the controller ID equals AHBRXBUF4CR0[MSTRID] and AHBRXBUF4CR0[BUFSZ] is not zero, Buffer 4 is used.
6. If the controller ID equals AHBRXBUF5CR0[MSTRID] and AHBRXBUF5CR0[BUFSZ] is not zero, Buffer 5 is used.
7. If the controller ID equals AHBRXBUF6CR0[MSTRID] and AHBRXBUF6CR0[BUFSZ] is not zero, Buffer 6 is used.
8. If all the above cases are not met, Buffer 7 is used.

NOTE

- Software must ensure that the size of each buffer is not less than the maximum burst size of an AHB read access from the AHB controller using this buffer. Otherwise, the behavior is undefined. The minimum buffer size request depends on the chip implementation. For example, if the capability of the chip to send out a maximum burst is 64 bytes, the minimum AHB receive buffer is 64 bytes.
- Assigning multiple buffers to a single AHB controller is not supported, unless the prefetch buffer enhancement feature is being used (see Prefetch buffer enhancement).
- When AHB read prefetch is enabled (AHBCR[PREFETCHEN] = 1), the size of the buffer determines the prefetch data size. If it does not cross the flash boundary, FlexSPI fetches data from the external flash device with that buffer size.
- The priority setting of AHB controller (AHBRXBUFxCR0[PRIORITY] = 1) is used only for suspending control of AHB prefetching. See Command abort and suspend.

25.3.13.3.1 Prefetch buffer enhancement

If a given controller frequently accesses memory at multiple address areas (non-sequential accesses), then the prefetch buffer can become less effective. For example, if the CPU is executing task A at a given address, then switches to task B at a different address, then when the CPU switches to task B (FLEXSPI receives request for an access that does not hit in the prefetch buffer), then the prefetch buffer will be flushed and a new prefetch will start at the task B address. When the CPU switches back to task A (task A addresses now miss in the prefetch buffer, because the buffer holds task B addresses instead), then the prefetch buffer will be flushed again and start prefetching at the task A address.

If the switches between task A and task B happen frequently, then FLEXSPI can end up re-reading items it had previously stored in the prefetch buffer as the buffer ends up being flushed often due to the task switches.

To increase the efficiency of FLEXSPI in situations like those described in the example above, the prefetch buffer enhancement feature can be enabled. When prefetch buffer enhancement is enabled (AHBRXBUFnCR0[REGIONEN]=1), then multiple prefetch buffers can be assigned to the same controller with an address range (AHBBUFREGIONSTARTn/ENDn) used as an additional filter to determine which buffer is used.

FLEXSPI compares the controller ID along with corresponding address range of each buffer to select one buffer. Other buffers are not impacted. This way, the AHB controller can prefetch the data from multiple address ranges in each of the buffers.

Program sequence:

- Configure AHBBUFREGIONSTARTn and AHBBUFREGIONENDn with the desired address ranges. Note that the addresses here use the AHB address including the system base address, not the serial flash address with the base address removed
- END must be larger than START
- Set AHBRXBUFnCR0[REGIONEN] = 1
- Set AHBRXBUFnCR0[MSTRID] with the MSTRID corresponding to the controller that will own multiple buffers, for example 4'b0011

NOTE

- This function is only meaningful when prefetch is enabled.
- Only Buffer 0 - 3 support this enhancement when REGIONEN=1, other buffers work like before. Buffer 0 - 3 with REGIONEN=0 works like before.

Prefetch buffer enhancement example:

The core in the system uses MSTRID = 0. The core will frequently switch between task A at 0x8000_1000-0x8000_1FFF and task B at 0x8000_2000-0x8000_2FFF, so dedicated prefetch buffers will be used for these tasks. The core will access other addresses in the FLEXSPI region using a third prefetch buffer.

To achieve this configuration:

- AHBBUFREGIONSTART0 = 0x8000_1000 and AHBBUFREGIONEND0 = 0x8000_2000. Configure Buffer 0 to the address range used by task A.
- AHBBUFREGIONSTART1 = 0x8000_2000 and AHBBUFREGIONEND1 = 0x8000_3000. Configure Buffer 1 to the address range used by task B.
- AHBRXBUF0CR0 = 0xC000_0004. Buffer 0 is a 256 bytes buffer assigned to the core (MSTRID = 0) with the prefetch buffer enhancement feature enabled (REGIONEN = 1), so only task A addresses will use buffer 0.
- AHBRXBUF1CR0 = 0xC000_0004. Buffer 1 is a second 256 bytes buffer assigned to the core (MSTRID = 0) with the prefetch buffer enhancement feature enabled (REGIONEN = 1), so only task B addresses will use buffer 1.
- AHBRXBUF4CR0 = 0x8000_0004. Buffer 4 is a third 256 bytes buffer assigned to the core (MSTRID = 0). The prefetch buffer enhancement feature will not be used for this buffer (and is not available for buffer 4 anyway). Core accesses that do not hit in Buffer 0 or Buffer 1 will use Buffer 4.

25.3.14 Command arbitration

There are four flash memory access command sources:

1. AHB command, that AHB write access to serial flash memory space triggers
2. AHB command, that AHB read access to serial flash memory space triggers
3. IP command, that writing IPCMD[TRG] triggers
4. Suspended command, an AHB read prefetch sequence which is suspended

An AHB bus access never triggers a write command and a read command at the same time.

The AHB prefetch sequence is an AHB command sequence triggered via AHB read access when AHB prefetch is enabled. After all read data is fetched for the current AHB read burst, FlexSPI prefetches more data to reduce read latency for the next AHB read access. An AHB command for a read operation is never aborted while fetching read data for the current AHB read burst. However, a new IP command or AHB command request while prefetching data (not for the current read burst) can abort an AHB read command.

When the arbitrator is idle (STS0[ARBIDLE] = 1), the granted priority of these command sources is:

1. AHB command (read or write)
2. IP command
3. Suspended command

NOTE

When the arbitrator is idle and there is no AHB or IP command request, a suspended command is not granted immediately. The arbitrator waits MCR2[RESUMEWAIT] AHB clock cycle idle states before resuming the suspended command to avoid AHB prefetch sequence being resumed and suspended frequently.

If the arbitrator is busy executing AHB or IP commands (not suspended commands), no new command requests are granted. If the grant times out, an AHB or IP command granted error occurs.

If a new AHB or IP command request comes while arbitrator is executing AHB read prefetch sequence, AHB read prefetch sequence is aborted (but not immediately). Arbitrator grants the AHB or IP command request after AHB read prefetch sequence is aborted on the FlexSPI interface and saved all internal data pointers.

25.3.14.1 Command abort and suspend

Table 513 describes the command abort and suspend mechanism.

Table 513. Command abort and suspend

Action	Description
Aborting a command	As mentioned above, if a new AHB or IP command request arrives, an AHB read prefetch sequence can be aborted.
Suspending a command	When an AHB read prefetch sequence is aborted on the FlexSPI interface, FlexSPI saves this suspended sequence in the cases listed below. FlexSPI resumes this sequence after the arbitrator has been idle for enough time. • There is no valid suspended command (AHBSPNDSTS[ACTIVE] = 0). This condition can occur when no command is suspended or when the suspended command is resumed. • The priority of aborted AHB read prefetch sequence is higher than the active suspend sequence. **NOTE** FlexSPI ignores and never resumes the original suspended sequence.
Activating a suspended command	The suspended command (internal status) becomes active when an AHB prefetch command is aborted and suspended. It becomes inactive in the following cases: • The arbitrator resumes the suspended command. • The AHB controller triggers a new AHB read command request using the same AHB receive buffer (buffer ID). **NOTE** MCR0[SWRESET] can be used to clear any suspended command. The saved suspended command is dropped and no resuming occurs. The suspended command must be cleared after the flash page program or erase. Otherwise, another command may lead to a period in which the flash memory cannot support a read command.

Figure 125 shows an example of the flow of command abort, suspend, or resume.

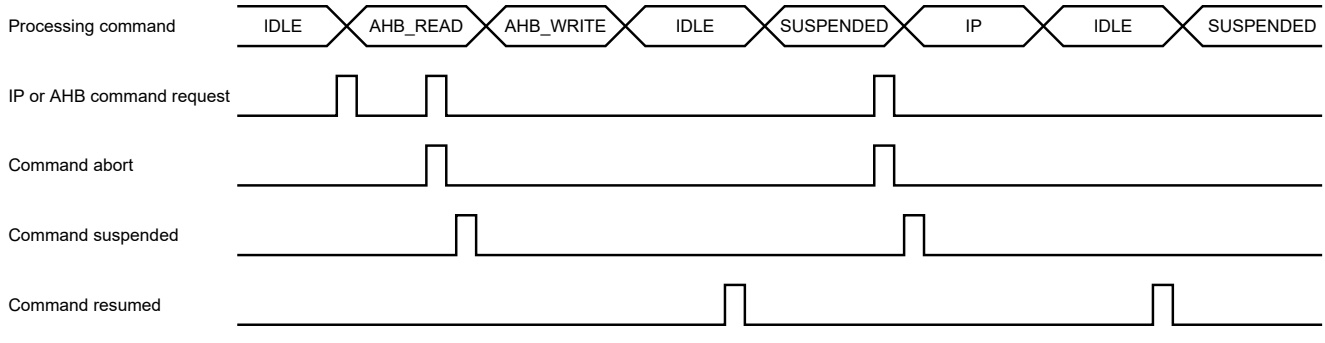


Figure 125. Command instruction execution on FlexSPI interface

NOTE

- The AHB prefetch sequence is aborted, suspended, or resumed twice in Figure 125. A new AHB write command request aborts the sequence the first time, and a new IP command request aborts the sequence the second time.
- The suspended sequence is resumed after a wait time of idle state. If there are frequent IP or AHB command requests and the wait time is not set reasonably, the suspend and resume activities may be triggered frequently.

25.3.15 SCLK stop

FlexSPI stops SCLK output toggling:

- When programming data is not ready for a programming command sequence.
- When an internal FIFO has no space to receive data for reading command sequence.

Certain devices may not support the stopping of SCLK during a command sequence (chip select is valid). SCLK stopping can be avoided via the methods described below.

- For flash memory reading that an IP command triggers:
  - Never trigger a read command with data size larger than IP receive FIFO size.
  - Internal asynchronous FIFO for flash memory reading should never be full.

    FlexSPI pops 64 bits of data per AHB clock cycle from this FIFO, and receives data from FlexSPI interface on the serial root clock. The flash memory access mode (Single, Dual, Quad, or Octal mode and Individual or Parallel mode) determines the receiving speed. For example, in Octal mode and Parallel mode, FlexSPI receives 16 bits per serial root clock cycle. If the AHB clock frequency is faster than 1/4 of the serial root clock, this asynchronous FIFO is never full.
- For flash memory programming that an IP command triggers:
  - Never trigger a program command with data size larger than the IP transmit FIFO size. Write all programming data into the IP transmit FIFO before triggering the IP command.
  - The internal asynchronous FIFO for flash memory programming must never be empty.

    FlexSPI fetches 64 bits of programming data per AHB clock cycle into this FIFO, and transmits data to the FlexSPI interface on the serial root clock. The flash memory access mode (Single, Dual, Quad, or Octal mode and Individual or Parallel mode) determines the transmitting speed. For example, in Octal mode and Parallel mode, FlexSPI transmits 16 bits per serial root clock cycle. If the AHB clock frequency is faster than 1/4 of the serial root clock, this asynchronous FIFO is never empty.
- For flash memory reading or programming that an AHB command triggers:
  - The internal asynchronous FIFO for flash memory reading and programming must never be full or empty. The frequency ratio limitation is same as it is for flash memory reading or flash memory programming that an IP command triggers.

NOTE

- FlexSPI never triggers an AHB read command with data size larger than the internal AHB receive buffer size.
- FlexSPI never triggers an AHB program command with data size larger than the internal AHB transmit buffer size.
- All programming data is buffered into the AHB transmit buffer before triggering an AHB program command in FlexSPI.

25.3.16 FlexSPI output timing

25.3.16.1 Output timing between data and SCLK

This section describes the output timing relationship of data (on A_DATA and B_DATA) and SCLK. There are three cases.

- **SDR instruction in SDR sequence**

  An SDR sequence contains only SDR instructions. In this case, all data bits last one serial root clock cycle on the FlexSPI interface. Figure 126 shows the relationship between the serial root clock, data, and SCLK:

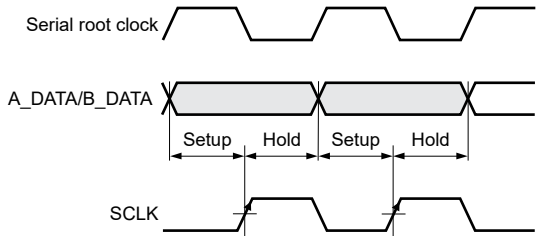


Figure 126. SDR instruction in SDR sequence

- **SDR instructions in a DDR sequence**

  A DDR sequence is a flash memory access command sequence that contains DDR instructions other than DUMMY_DDR. It may contain SDR instructions. If there are SDR instructions in a DDR sequence, all data bits last for two serial root clock cycles on the FlexSPI interface. Figure 127 shows the relationships between the serial root clock, data, and SCLK.

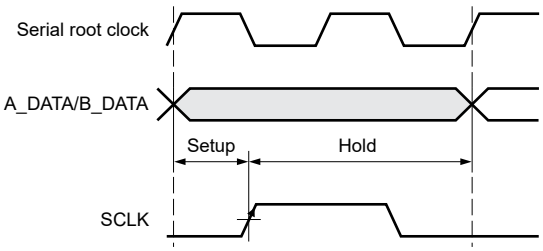


Figure 127. SDR instruction in DDR sequence

- **DDR instructions in a DDR sequence**

  For DDR instructions in a DDR sequence, all data bits last one serial root clock cycle on the FlexSPI interface. Figure 128 shows the relationships between the serial root clock, data, and SCLK.

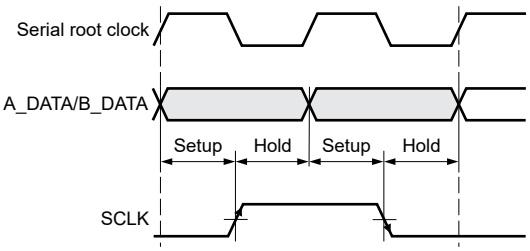


Figure 128. DDR instruction in DDR sequence

25.3.16.2 Output timing between chip select and SCLK

This section describes the output timing relationship between chip select (on A_SS0_B , A_SS1_B, B_SS0_B, and B_SS1_B) and SCLK. The timing relationship for the SDR sequence differs a little from the timing relationship for the DDR sequence.

- **Chip select timing in SDR sequence**

For an SDR sequence, the delay from chip select assertion to the SCLK rising edge is (FLSHxCR1[TCSS] + 0.5) cycles of the serial root clock. The delay from SCLK falling edge to chip select deassertion is FLSHxCR1[TCSH] cycles of serial root clock. Figure 129 shows the timing relationship between the chip select and SCLK.

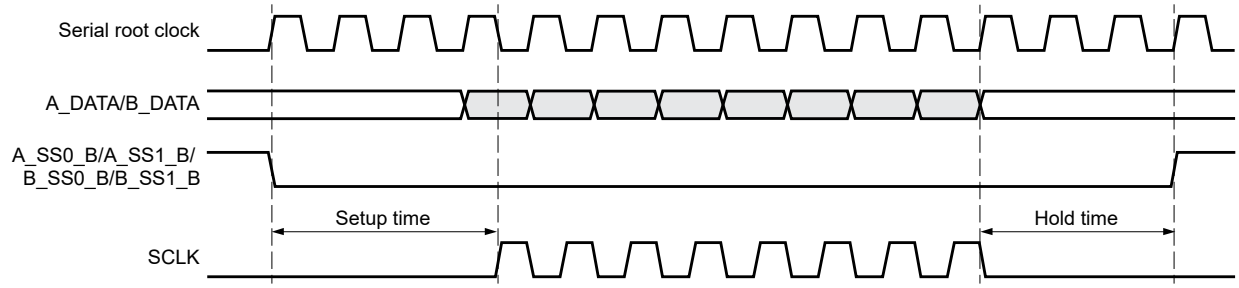


Figure 129. Chip select output timing for SDR sequence

- **Chip select timing in DDR sequence**

  For a DDR sequence, the delay from chip select assertion to the SCLK rising edge is **(FLSHxCR1[TCSS] + 0.5)** cycles of the serial root clock. The delay from the SCLK falling edge to chip select deassertion is **(FLSHxCR1[TCSH] + 0.5)** cycles of serial root clock.

  NOTE

  When AHB receive prefetch is enabled, and prefetch can be aborted, configure TCSH to 1 or greater. Configuring TCSH in this way guarantees a positive chip select hold time after the SCLK falling edge.

  Figure 130 shows the timing relationship between chip select and SCLK.

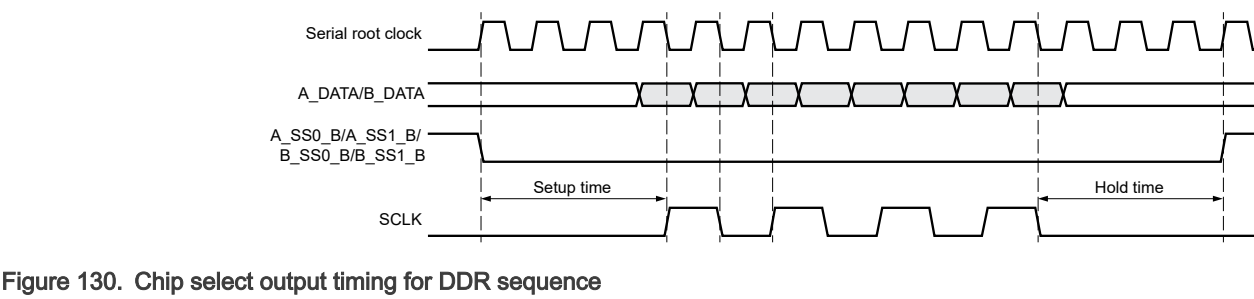


Figure 130. Chip select output timing for DDR sequence

For devices such as an FPGA device, a minimum chip select negation time (time between two chip select assertions) may be required. If the value of FLSHxCR1[CSINTERVAL] is nonzero, FlexSPI ensures a delay between chip select valid assertions. The delay time is CSINTERVAL × 256 cycles of serial root clock for an SDR or DDR sequence. If the external device has no limitation, configure the value of this field to zero. Figure 131 shows the timing of the chip select interval.

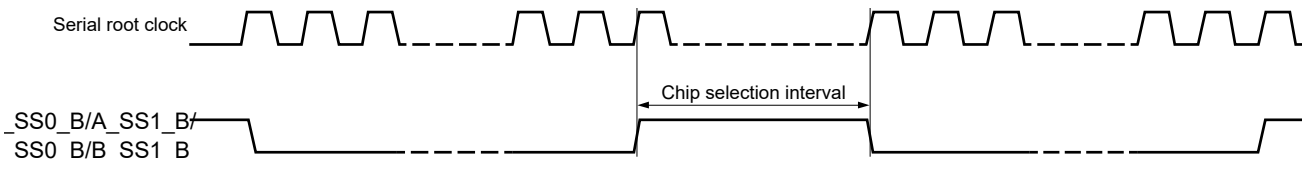


Figure 131. Chip select valid interval

25.3.17 FlexSPI input timing

This section describes the input timing of FlexSPI.

25.3.17.1 Receiving block

This section describes the receiving block in FlexSPI.

FlexSPI decodes instruction code to determine SDR mode or DDR mode. It also determines whether both the rising and falling edges are used for flash data sampling. FlexSPI samples the data line for learn instructions and read instructions.

MCR0[RXCLKSRC] selects the clock source from these four sources for the sampling clock:

- Internal dummy read strobe and looped back internally
- Internal dummy read strobe and looped back from DQS pad
- SCK output and looped back from SCK pad
- Flash-memory-provided read strobe

For details about DLL configuration, see DLL configuration for sampling.

Figure 132 shows the sampling block in FlexSPI.

FlexSPI uses both rising and falling edges to sample data from flash_A and flash_B. Two muxes with four clock source inputs generate the signals ipp_ind_io_fa, ipp_ind_io_fb. See Receive clock source features. ipp_ind_io_fa_int and ipp_ind_io_fb_int are reversed from ipp_ind_io_fa and ipp_ind_io_fb. Both sampling flip-flops are used when FlexSPI is in DDR mode. Only the falling edge sampling flip-flop is used in SDR mode.

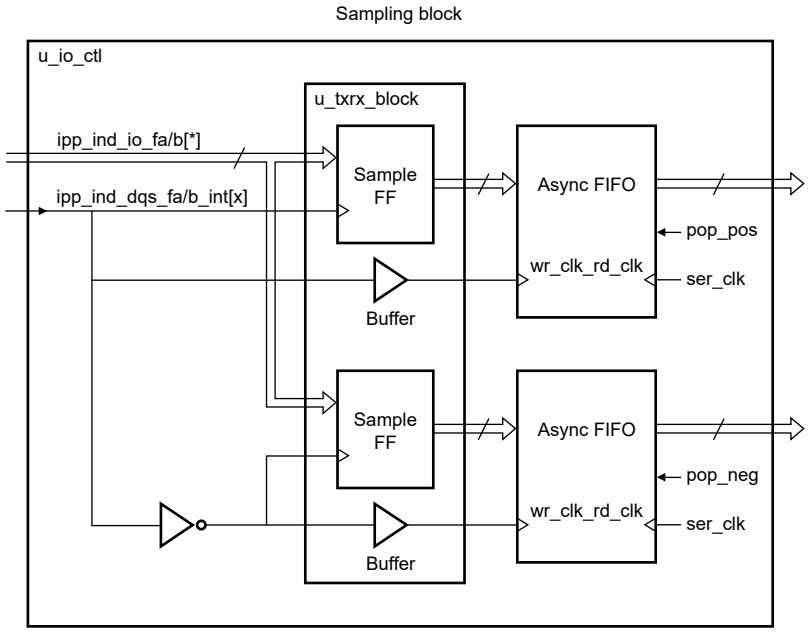


Figure 132. Sampling block in FlexSPI

Figure 133 is the receiving block diagram for data learning.

There are 16 clock phases for sampling data. An internal delay chain delays the clock phases.

See Data learning.

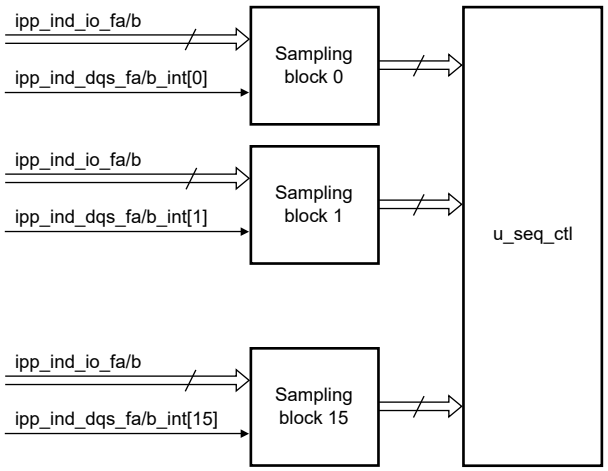


Figure 133. Sampling blocks for data learning

FlexSPI does not support read operations with a zero-turnaround cycle for direction switching on I/O pins. This read operation often occurs in QuadSPI configuration register reads. In Figure 134 and Figure 135, SIO[3:0] is an output from FlexSPI in the CMD phase, and is an input to FlexSPI in the DATA phase. No time remains for FlexSPI to switch the direction. This kind of read is not supported. FlexSPI and flash memory drive the I/O pins at the same time, which may cause a problem. To avoid this behavior on I/O pins, using SPI mode or reading with dummy cycles is required.

NOTE

The cycle numbers are not accurate in the examples below. See the device spec for accurate cycle numbers.

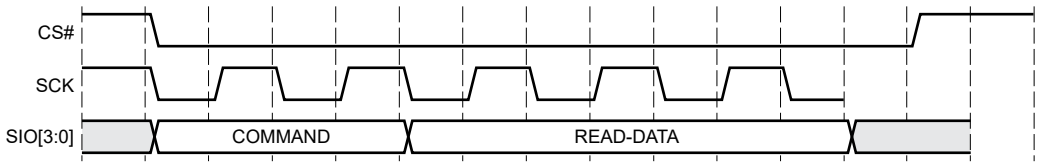


Figure 134. Unsupported read behavior

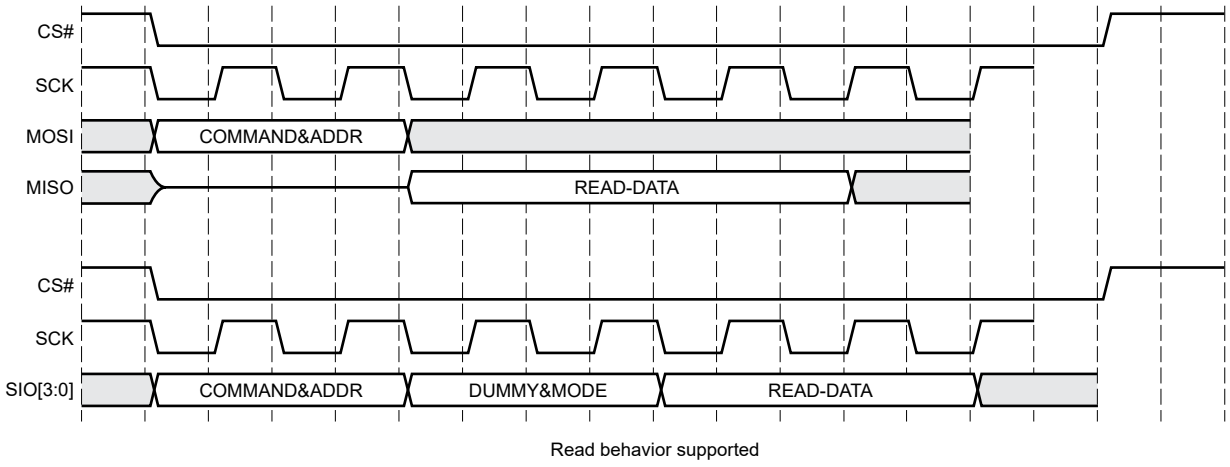


Figure 135. Supported read behavior

25.3.17.2 Receive clock source features

FlexSPI uses a read strobe to sample incoming data. There are several options for generating the read clock strobe that MCR0[RXCLKSRC] selects. Using the receive clock source method has implications for the board connections and can affect the maximum frequency that the interface supports. See the device data sheet for details on the maximum frequency supported for each RXCLKSRC option.

Table 514. RXCLKSRC options

MCR0[RXCLKSRC]	Description	Board connection	Max Frequency
0	Internal dummy read strobe and internal loopback	DQS pin is not used. The DQS pin can be configured for an alternate signal function.	Lowest
1	Internal dummy read strobe and loopback from DQS pad	FlexSPI uses the DQS pin, and it must be configured for its FlexSPI function. The internally generated read strobe is sent to the DQS pin and is sampled at the pin to match more closely the data pin timings. The DQS pin is typically left floating. External capacitance can be added to adjust timing.	Medium
2	SCK output and loopback from SCK pad	SCK is used as the read strobe. FlexSPI uses the signal looped back from the SCK pad as the read strobe to match more closely the data pin timings. DQS pin is not used and can be configured for an alternate signal function.	Medium
3	Flash-memory-provided read strobe	DQS pin is connected to the DQS or RWDS signal that the memory provides.	Highest

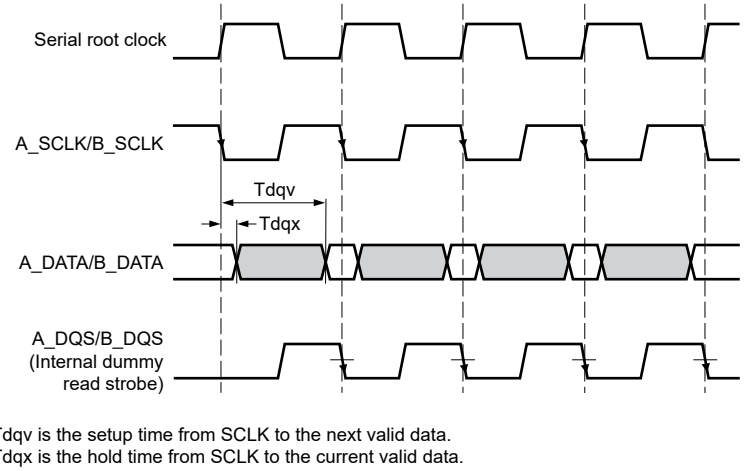
25.3.17.3 Input timing for sampling with dummy read strobe

This section describes the input timing when sampling with the internal dummy read strobe (MCR0[RXCLKSRC] = 0 or 1). The timing for sampling with an internal dummy read strobe loopback is very similar to the timing for loopback from pad. Sampling with a dummy read strobe loopback from the DQS pad can achieve a higher read frequency. It compensates for the delay of the SCLK output path and data pin input path.

The input timing is different for SDR mode and DDR mode.

- **Input timing for sampling with dummy read strobe in SDR mode**

  For SDR read or learn instructions, FlexSPI samples input data pins with the falling edge of the dummy read strobe. Figure 136 shows the input timing for sampling with dummy read strobe in SDR mode.



Tdqv is the setup time from SCLK to the next valid data.
Tdqx is the hold time from SCLK to the current valid data.

Figure 136. Input timing for sampling with dummy read strobe in SDR mode

- **Input timing for sampling with dummy read strobe in DDR mode**

  For DDR read or learn instructions, FlexSPI samples input data pins on rising and falling edges of the dummy read strobe. Figure 137 shows the input timing for sampling with dummy read strobe in DDR mode.

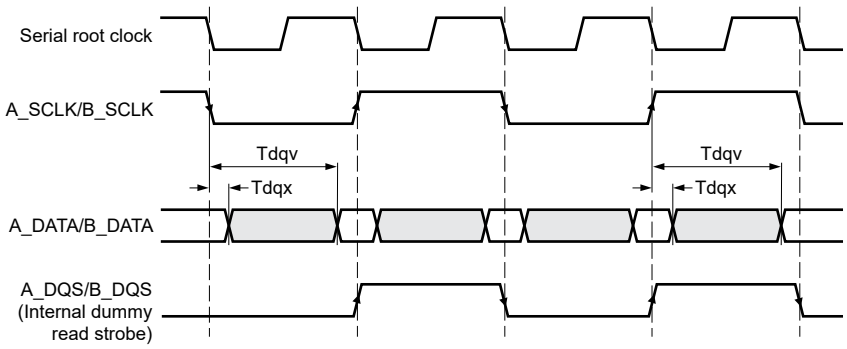


Figure 137. Input timing for sampling with dummy read strobe in DDR mode

25.3.17.4 Input timing for sampling with SCK output looped back from SCK pad

This section describes the input timing when sampling with the SCK output looped back from the SCK pad (MCR0[RXCLKSRC] = 2). The input timing is similar to sampling with a dummy read strobe. SCK output toggles for all types of instructions, but internal dummy strobe only toggles for read and learn instructions. In this case, FlexSPI receives more data bits when sampling with SCK output. FlexSPI automatically ignores the redundant data bits sampled.

25.3.17.5 Input timing for sampling with flash-memory-provided read strobe

This section describes the input timing when sampling with flash-memory-provided read strobe (MCR0[RXCLKSRC] = 3). The input timing is different for SDR mode and DDR mode.

NOTE

There are no known devices that provide a read strobe and support SDR mode operation.

- **Flash-memory-provided read strobe with SCLK**

  Certain flash devices provide both read data and read strobes with SCLK. Then the read strobe edge is aligned with the read data change. The FlexSPI controller delays the read strobe for one half cycle of the serial root clock (with DLL), then samples

read data with the delayed strobe. See DLL configuration for sampling. Figure 138 and Figure 139 show the input timing for sampling with flash memory read strobe in SDR mode and DDR mode.

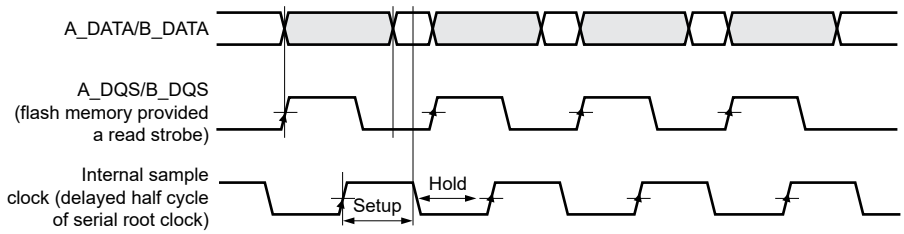


Figure 138. Input timing 2 for flash-memory-provided read strobe in SDR mode

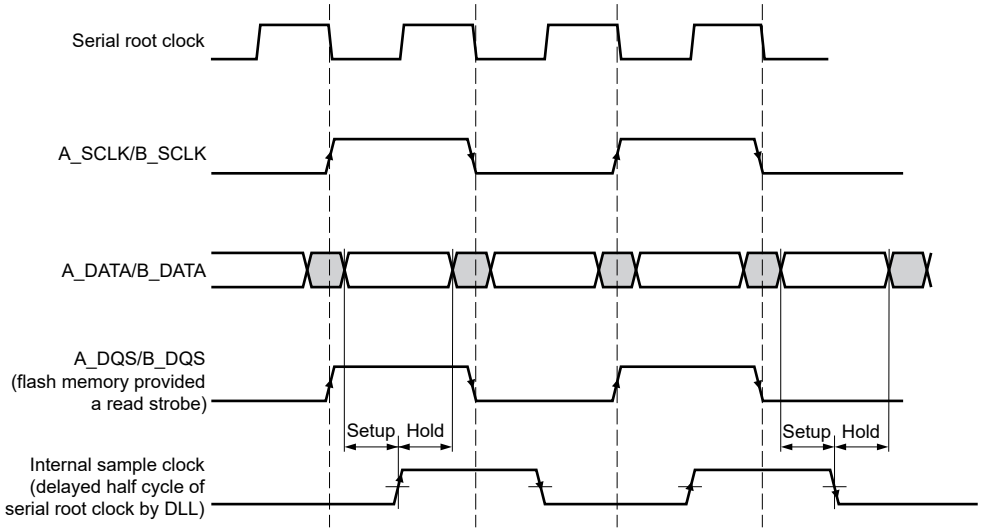


Figure 139. Input timing 2 for flash-memory-provided read strobe in DDR mode

25.3.17.6 DLL configuration for sampling

For the four sampling clock sources described previously, the input timing differs depending on the DLLxCR register configuration, according to the sampling clock source mode. The DLL is a delay-line chain that can be set to a fixed number of delay cells. It can also be auto-adjusted to lock on a certain phase delay to the reference clock.

- DLLxCR must be set to 0000_0100h (one fixed delay cell in DLL delay chain) in the following cases:
  - Sampling data with dummy read strobe looped back internally (MCR0[RXCLKSRC] = 0h)
  - Sampling data with dummy read strobe looped back from DQS pad (MCR0[RXCLKSRC] = 1h)
  - Sampling data with SCLK output looped back from SCLK pad (MCR0[RXCLKSRC] = 2h)
- When data is sampled with a flash-memory-provided read strobe (MCR0[RXCLKSRC] = 3h) and flash memory provides the read strobe with SCLK, DLL must be configured as follows (when serial root clock is over 100 Mhz. These settings ensure a lock of the reference clock (serial root clock).
  - DLLxCR[SLVDLYTARGET] is up to flash parameters used.
  - DLLxCR[REFPHASEGAP] = 2h
  - DLLxCR[DLLEN] = 1h
  - DLLxCR[OVRDEN] = 0h
  - Other fields in DLLxCR must be kept at their reset values (all zeroes).

After the register settings above are completed, set the DLL with following steps.

1. Reset the DLL(DLLxCR[DLLRESET] or MCR0[SWRESET]).
2. Wait for STS2[xREFLOCK] and STS2[xSLVLOCK] being asserted.
3. Wait for equal or more than 512 serial root clock cycles.
4. Start read or write operation.

NOTE

If the serial root clock is slower than 100 MHz, DLL cannot lock on a half cycle of serial root clock. The delay cell number is limited in the delay chain. DLL must be configured as follows instead:

- DLLxCR[OVRDEN] = 1
- DLLxCR[OVRDVAL] = N

  Each delay cell in DLL is about 75 ps to 225 ps. The delay of DLL delay chain is (N × Delay_cell_delay). N is set based on the maximum DDR frequency (N ≤ equation). This value is calculated based on the MAX_DDR_FREQ parameter. N = 34. This value is a recommended value. If a failure occurs, the value might require adjustment in a real application.

  You can calculate the delay of one delay cell and obtain the value of N. If the serial root clock is lower than 100 MHz, the delay chain can't lock with DLLxCR[SLVDLYTARGET] = 0xF (half cycle of root clock delay). You need to set DLLxCR[SLVDLYTARGET] as 0xE (15/32 cycle delay) or less. When DLL is locked (read STS2[AREFLOCK] and STS2[ASLVLOCK]), the delay cell number = STS2[ASLVSEL] +1 and the delay value (one delay cell) = (total delay)/(delay cell number). Based on delay value, the N can be easily calculated.
- Other fields in DLLxCR must be kept at their reset values (all zeroes).

25.3.18 Data learning

The FlexSPI controller generates 16 clock phases (Phase 0–15) via delay cell line (DCL) with the selected sample clock. Clock phase 0 is the selected sample clock (DQS_IN) with no delay cell. 16 sampling blocks are implemented for both Port A and 16 sampling blocks are implemented for Port B.

During a learn instruction, FlexSPI compares the sampled data bits with the internal data learn pattern in the Data Learning Pattern (DLPR) register and determines the correct sampling clock phase. The phase selection is automatically updated after a learn instruction and applied to subsequent read instructions or sequences.

When data learning is disabled (MCR0[LEARNEN] = 0), FlexSPI always uses clock phase 0 to sample FlexSPI data lines. Attempting to execute a learn instruction results in an error flag (INTR[IPCMDERR] or INTR[AHBCMDERR]).

FlexSPI supports data learning in both SDR mode and DDR mode. Data learning is not supported if the sampling clock source is a flash-memory-provided read strobe (MCR0[RXCLKSRC] = 3h).

Data learning can achieve high read frequency, but the flash memory input timing still limits the highest frequency. Flash memory must receive the command code and flash address bits.

When data learning is enabled, FlexSPI uses clock phase 0 after reset and before executing a learn instruction. The clock phase selection is updated after FlexSPI executes the learn instruction. The selected sample clock phase can be polled via STS0[DATALEARNPHASEA] and STS0[DATALEARNPHASEB]. When data learning fails, the INTR[DATALEARNFAIL] interrupt flag is set, and the previous clock phase selection is kept. You can write 1 to MCR2[CLRLEARNPHASE] to reset the internal clock phase selection to clock phase 0.

25.3.18.1 Data learning with flash-memory-provided preamble bit

Certain flash devices support driving with preamble bits before driving read data in each read command sequence. Preamble bits are also called the data-learning pattern (DLP). The DLP is programmable via configuration register in the flash device.

For the specified read command sequence, flash memory returns preamble bits after dummy cycles and before returning read data.

For these flash devices, the operation flow with data learning is:

1. Configure the data learning pattern in Data Learning Pattern (DLPR) register.
2. Configure the same data learning pattern in the flash device by triggering an IP command.
3. Enable data learning in the flash device by triggering an IP command.
4. Configure LUT sequence with valid read command sequence containing the learn instruction.
5. Trigger a flash memory read command via AHB or IP command as normal.

NOTE

The first four steps are executed only once. They are not needed for every flash memory read command.

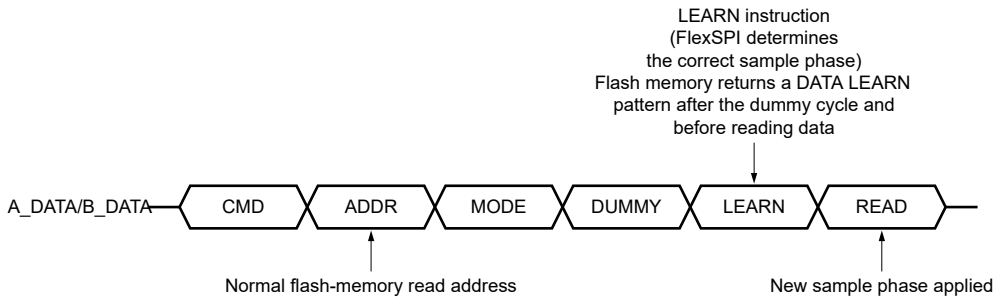


Figure 140. Data learning flow with flash-memory-provided preamble bit

25.3.18.2 Data learning without flash-memory-provided preamble bit

For flash devices that do not provide a preamble bit, the DLP cannot be returned automatically for each read command sequence. For these flash devices, the data learning sequence is:

1. Set data learning pattern in Data Learning Pattern (DLPR) register.
2. Reserve a certain flash memory area and program data (according to data learning pattern and flash memory access mode) to this area via IP command.
3. Configure LUT sequence with read sequence using a learn instruction instead of a read instruction. Trigger the read sequence to the reserved flash memory area via IP command.
4. Trigger the flash memory read command via AHB or IP command as normal. No learn instruction is needed in this read sequence.

NOTE

- The first three steps are executed only once. They are not needed for every flash memory read command.
- Step 4 should be executed with a certain interval to ensure that the internal sampling clock phase is adjusted in time.

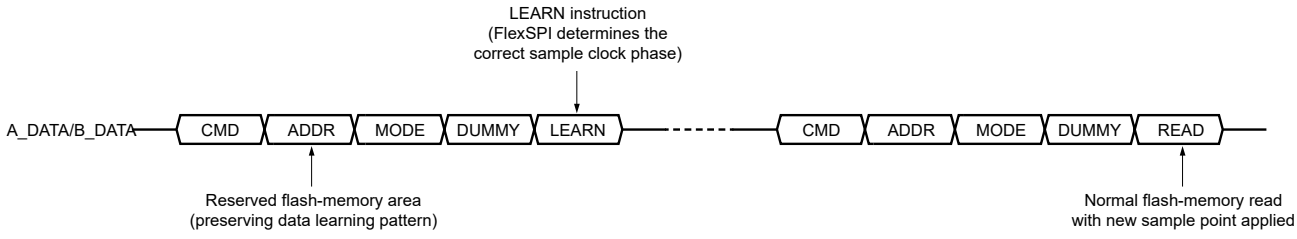


Figure 141. Data learning flow without flash-memory-provided preamble bit

NOTE

- For flash devices that do not provide a preamble bit, software overhead is required to read the reserved flash memory area at the interval.

These items determine the preprogramming data:

- Data Learning Pattern (DLPR register setting)
- DLP bit length
- Individual or Parallel mode for read command
- Octal, Quad, Dual, or Single mode for read command

Figure 142 shows a preprogramming data example where the DLPR value is 43h, eight bits, and flash memory is read in Quad mode and Parallel mode.

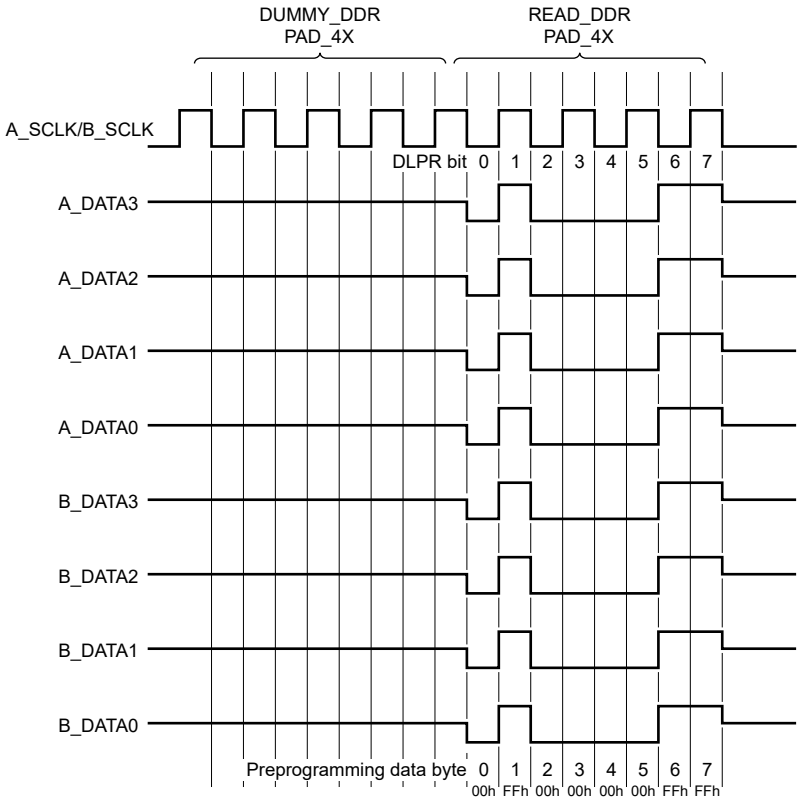


Figure 142. Preprogramming data for data learning

25.3.19 Execute-In-Place (XIP) enhanced mode

FlexSPI always supports XIP, regardless of whether external devices provide XIP enhanced mode. To support XIP, put program codes on an external device, then read or execute these codes directly via AHB read access to serial flash memory space. There is no configuration or status polling needed during AHB read access to the memory of the external device. The AHB receive buffer is fully transparent to software.

Certain devices provide XIP enhanced mode to improve code execution. In this mode, the command code is only sent in the first read instruction. It saves many cycles for command instructions and improves code execution. Devices enter or exit XIP enhanced mode via a special sequence that external devices specify. See the external device data sheet for more details.

Normally, a device enters XIP enhanced mode via this sequence:

1. Enable XIP enhanced mode in external flash device via IP command.

2. Send the first read sequence to external flash device with correct mode bits. Command code is included in this read sequence.
3. Send the subsequent read sequences to external flash device with correct mode bits. Command code is not included in these read sequences. Mode bits must be sent according to the flash specified; otherwise, the flash exits XIP enhanced mode.

The instruction JMP_ON_CS in FlexSPI must be used to support XIP enhanced mode in external devices. This instruction is only allowed in the AHB read command; otherwise, the instruction generates an IPCMDERR or AHBCDMERR interrupt (when the interrupt is enabled). When external devices do not support XIP enhanced mode, JMP_ON_CS must never be used. To support XIP enhanced mode:

- The first instruction in the read sequence must be the command instruction.
- The last valid instruction must be JMP_ON_CS (with operand 1h).

For the first AHB read command triggered, FlexSPI executes the instructions from the instruction pointer 0 in the sequence (which is the command instruction). After this sequence is executed, FlexSPI saves the operand in the JMP_ON_CS instruction as the start pointer for the next command to current device internally. For the next AHB read command triggered, FlexSPI executes from the instruction pointer 1h, bypassing the command instruction.

After XIP enhanced mode is started, it is possible to access flash via other commands like program commands. You must, however, use a special sequence to exit enhanced XIP mode before you can run a different command. Figure 143 indicates XIP operation with Flash XIP enhanced mode.

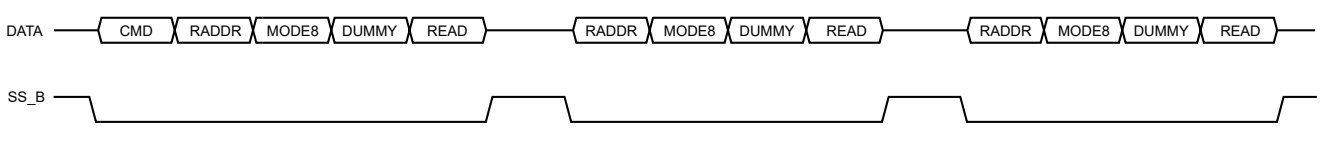


Figure 143. XIP Enhanced Mode Operation

25.3.20 AHB bus blocking

FlexSPI can temporarily block some or all AHB transactions. This feature is useful when IP commands are in progress that could cause a potential conflict with an AHB request. Examples of situations where IP bus blocking would be used are:

- Avoiding RWW (read while write). While programming or erasing flash, reads are not allowed. AHB bus blocking can prevent the AHB bus from requesting a read while IP write commands are in progress.
- Decrypting while encrypting. The IPED of FlexSPI cannot support simultaneous encryption and decryption. Partial AHB bus blocking can prevent AHB reads from IPED regions while IP writes to encrypted block are in progress.

To block all AHB transactions:

1. Configure blocking for all AHB transactions by writing 1 to IPCR2[IPBLKALLAHB].
2. Request AHB bus blocking by writing 1 to IPCR2[IPBLKAHBREQ].
3. Wait for the AHB bus blocking request to be acknowledged (IPCR2[IPBLKAHBACK] = 1).
4. Perform IP write commands as needed.
5. Request normal operation by writing 0 to IPCR2[IPBLKAHBREQ].
6. Wait for AHB blocking request clear to be acknowledged (IPCR2[IPBLKAHBACK] = 0).

To block AHB transactions to IPED regions only, while allowing AHB transactions to unencrypted areas:

1. Configure blocking for AHB transactions to IPED areas by writing 0 to IPCR2[IPBLKALLAHB].
2. Request AHB bus blocking by writing 1 to IPCR2[IPBLKAHBREQ].
3. Wait for the AHB bus blocking request to be acknowledged (IPCR2[IPBLKAHBACK] = 1).

4. Perform IP write commands as needed.
5. Request normal operation by writing 0 to IPCR2[IPBLKAHBREQ].
6. Wait for AHB blocking request clear to be acknowledged (IPCR2[IPBLKAHBACK] = 0).

NOTE

AHB bus blocking can only be used to prevent conflicts between AHB and IP transactions. Software must handle conflicts between multiple AHB transactions.

25.3.21 Domain control

FLEXSPI supports two kinds of protection modes to manage nonsecure controller IP command access:

- IP command trigger
- LUT write

The chip must maintain domain control of the AHB region.

25.3.21.1 LUT write protection

LUTCR[PROTECT] controls LUT write protection. By default, no protection is applied on the LUT Control (LUTCR) register and LUT memory, and any controller can access the LUT freely.

When LUTCR[PROTECT] = 1, only a secure controller can change the value of LUTCR and the content of the LUT table. Unless hardware resets FlexSPI to return to the LUT write-protection-disabled state, nonsecure controllers cannot write to the LUT table or LUTCR register.

No error or interrupt occurs if a non-secure controller attempts to write to the protected LUT; the write operation is ignored.

25.4 External signals

This section provides the external signal information for the FlexSPI module.

Table 515. External Signal List

Signal name	Function	Direction	Description
A_SS0_B	Peripheral Chip Select Flash A1	O	Chip select for the serial flash device A1
A_SS1_B	Peripheral Chip Select Flash A2	O	Chip select for the serial flash device A2
B_SS0_B	Peripheral Chip Select Flash B1	O	Chip select for the serial flash device B1
B_SS1_B	Peripheral Chip Select Flash B2	O	Chip select for the serial flash device B2
A_SCLK	Serial Clock Flash A	O	Serial clock output to the serial flash device A. This clock runs at half the frequency of serial clock root in DDR mode, and at the same frequency as serial clock root in SDR mode. Clock output toggles during the entire flash memory access sequence.
B_SCLK	Serial Clock Flash B	O	Serial clock output to the serial flash device B. This clock runs at half the frequency of serial clock root in DDR mode, and at the same

Table continues on the next page...

Table 515. External Signal List (continued)

Signal name	Function	Direction	Description
			frequency as serial clock root in SDR mode. Clock output toggles during the entire flash memory access sequence. **NOTE** B_SCLK may be used as the differential clock output of A_SCLK by writing 1 to MCR2[SCKBDIFFOPT].
A_DATAn	Serial I/O Flash A	I/O	Data I/O lines to and from the serial flash device A
B_DATAn	Serial I/O Flash B	I/O	Data I/O lines to and from the serial flash device B
A_DQS	Data Strobe Signal Flash A	I/O	Data strobe signal for port A. This signal has three functions: • Driven with read strobe by external device. Some flash devices provide the read strobe signal together with read data. In this case, if the external device drives this pad only when reading flash memory data, this pad may require a **pull-down** resistor. • Driven with latency information by external device. Some devices use this pin to indicate the dummy cycles needed (before program or read data transfer) such as HyperRAM or HyperFlash. • Loopback dummy read strobe. The FlexSPI controller provides internal dummy read strobe for flash memory read data. Higher read frequency can be achieved by looping back this dummy read strobe from the pad. This pin can be floated or have some capacitive load added at the board level to compensate for load on DATA or SCLK pins.
B_DQS	Data Strobe Signal Flash B	I/O	Similar to A_DQS

25.5 Initialization

The FlexSPI controller initialization sequence is:

1. Enable controller clocks (AHB clock, IP bus clock, or serial root clock) at system level.
2. To enter module stop mode, write 1 to MCR0[MDIS].
3. Configure module control registers: MCR0, MCR1, and MCR2. Do not change MCR0[MDIS].
4. When AHB commands are used, configure AHB Bus Control (AHBCR) and AHB Receive Buffer Control (AHBRXBUFxCR0) registers.

5. Configure flash control registers (FLSHxCR0, FLSHxCR1, or FLSHxCR2) according to external device type.
6. Configure DLL control register (DLLxCR) according to sample clock source selection.
7. To exit module stop mode, write 0 to MCR0[MDIS].
8. Configure LUT as needed for AHB command or IP command.
9. Optionally, to reset controller, write 1 to MCR0[SWRESET].

External devices must be configured via IP command normally after initialization. For example, the WRITE STATUS command performs device configuration for most serial NOR flash memory.

25.6 Application information

This section describes applications that FlexSPI supports.

25.6.1 Application on serial NOR flash device

This section provides the example LUT instruction sequences for serial NOR flash device (Cypress Flash S25FS128S).

25.6.1.1 Write enable command

Table 516 shows the WRITE ENABLE command sequence.

Table 516. WRITE ENABLE command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h	06h	Command name: WREN
1-7	STOP (00h)	0h	00h	

25.6.1.2 Write registers command

Table 517 shows the Write Registers command sequence.

Table 517. Write Registers command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	01h	Command name: WRR
1	WRITE_SDR	0h (Single)	1h or 2h	Data size is 1 byte or 2 bytes. Byte 0 is write data for Status Register 1; Byte 1 is write data for Configuration Register 1. IPCR1[IDATSZ] can override this value.
2-7	STOP (00h)	0h	00h	

25.6.1.3 Page Program command

Table 518 shows Page Program command sequence.

Table 518. PAGE PROGRAM command (Cypress serial NOR flash)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	02h or 12h	Command name: PP or 4PP PP is 3-byte address mode. 4PP is 4-byte address mode.
1	ADDR_SDR	0h (Single)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	WRITE_SDR	0h (Single)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default programming data size. This value is ignored for AHB command.
3-7	STOP (00h)	0h	00h	

Table 519 shows Page Program command sequence (QPI mode).

Table 519. PAGE PROGRAM (QPI mode) command (Cypress serial NOR flash)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	2h (Quad)	02h or 12h	Command name: PP or 4PP PP is 3-byte address mode. 4PP is 4-byte address mode.
1	ADDR_SDR	2h (Quad)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	WRITE_SDR	2h (Quad)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default programming data size. This value is ignored for AHB command.
3-7	STOP (00h)	0h	00h	

25.6.1.4 Read Status 1 command

Table 520 shows READ STATUS 1 command sequence.

Table 520. READ STATUS 1 command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	05h	Command name: RDSR1
1	READ_SDR	0h (Single)	1h	1 byte for Status register 1
2-7	STOP (00h)	0h	00h	

25.6.1.5 Read command

Table 521 shows READ command sequence.

Table 521. READ command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	03h or 13h	Command name: READ or 4READ READ is 3-byte address mode. 4READ is 4-byte address mode.
1	ADDR_SDR	0h (Single)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	READ_SDR	0h (Single)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.
3-7	STOP (00h)	0h	00h	

25.6.1.6 Fast Read command

Table 522 shows Fast Read command sequence. This table shows Cypress SPI Configuration Register bits CR2V[7] = 0, CR2V[3:0] = 8h.

Table 522. FAST_READ command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	0Bh or 0Ch	Command name: FAST_READ or 4FAST_READ FAST_READ is 3-byte address mode. 4FAST_READ is 4-byte address mode.
1	ADDR_SDR	0h (Single)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	DUMMY_SDR	0h (Single)	08h	Dummy cycle is 8 (in serial root clock) as CR2V[3:0] = 8h.
3	READ_SDR	0h (Single)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.
4-7	STOP (00h)	0h	00h	

25.6.1.7 Dual IO Fast Read command

Table 523 shows Dual IO FAST_READ command sequence. This table shows Cypress SPI Configuration Register bits CR2V[7] = 0, CR2V[3:0] = 8h, Continuous Read mode.

Table 523. Dual IO FAST_READ command (Continuous Read mode)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	BBh or BCh	Command name: DIOR or 4DIOR DIOR is 3-byte address mode. 4DIOR is 4-byte address mode.
1	ADDR_SDR	1h (Dual)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	MODE8_SDR	1h (Dual)	Axh	Enter continuous read mode or remain in continuous read mode.
3	DUMMY_SDR	1h (Dual)	08h	Dummy cycle is 8 (in serial root clock) as CR2V[3:0] = 8h.
4	READ_SDR	1h (Dual)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.
5	JMP_ON_CS	0h (or Don't care)	01h	CMD instruction is bypassed after the first read access.
6-7	STOP (00h)	0h	00h	

Table 524 shows Dual IO FAST_READ command sequence. This table shows Cypress SPI Configuration Register bits CR2V[7] = 0, CR2V[3:0] = 8h, Noncontinuous Read mode.

Table 524. Dual IO FAST_READ command (Noncontinuous Read mode)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	BBh or BCh	Command name: DIOR or 4DIOR DIOR is 3-byte address mode. 4DIOR is 4-byte address mode.
1	ADDR_SDR	1h (Dual)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	MODE8_SDR	1h (Dual)	Any value other than Axh	Exit continuous read mode or remain in noncontinuous read mode.
3	DUMMY_SDR	1h (Dual)	08h	Dummy cycle is 8 (in serial root clock) as CR2V[3:0] = 8h.
4	READ_SDR	1h (Dual)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.
5-7	STOP (00h)	0h	00h	

25.6.1.8 Quad IO Fast Read command

Table 525 shows Quad IO FAST_READ command sequence. This table shows Cypress SPI Configuration Register bits CR2V[7] = 0, CR2V[6] = 0, CR2V[3:0] = 8h, Non-QPI mode, Noncontinuous read mode.

Table 525. Quad IO FAST_READ command (Non-QPI mode, Noncontinuous read mode)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	EBh or ECh	Command name: QIOR or 4QIOR QIOR is 3-byte address mode. 4QIOR is 4-byte address mode.
1	ADDR_SDR	2h (Quad)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	MODE8_SDR	2h (Quad)	Any value other than Axh	Exit continuous read mode or remain in noncontinuous read mode.
3	DUMMY_SDR	2h (Quad)	08h	Dummy cycle is 8 (in serial root clock) as CR2V[3:0] = 8h.
4	READ_SDR	2h (Quad)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.
5-7	STOP (00h)	0h	00h	

Table 526 shows Quad IO FAST_READ command sequence. This table shows Cypress SPI Configuration Register bits CR2V[7] = 0, CR2V[6] = 0, CR2V[3:0] = 8h, Non-QPI mode, Continuous read mode.

Table 526. Quad IO FAST_READ command (Non-QPI mode, Continuous read mode)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	EBh or ECh	Command name: QIOR or 4QIOR QIOR is 3-byte address mode. 4QIOR is 4-byte address mode.
1	ADDR_SDR	2h (Quad)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	MODE8_SDR	2h (Quad)	A0h	Enter continuous read mode or remain in continuous read mode.
3	DUMMY_SDR	2h (Quad)	08h	Dummy cycle is 8 (in serial root clock) as CR2V[3:0] = 8h.
4	READ_SDR	2h (Quad)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.

Table continues on the next page...

Table 526. Quad IO FAST_READ command (Non-QPI mode, Continuous read mode) (continued)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
5	JMP_ON_CS	0h (or Don't care)	01h	CMD instruction is bypassed after the first read access.
6-7	STOP (00h)	0h	00h	

Table 527 shows Quad IO FAST_READ command sequence. This table shows Cypress SPI Configuration Register bits CR2V[7] = 0, CR2V[6] = 1, CR2V[3:0] = 8h, QPI mode, Continuous read mode.

Table 527. Quad IO FAST_READ command (QPI mode, Continuous read mode)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	2h (Quad)	EBh or ECh	Command name: QIOR or 4QIOR QIOR is 3-byte address mode. 4QIOR is 4-byte address mode.
1	ADDR_SDR	2h (Quad)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	MODE8_SDR	2h (Quad)	A0h	Enter continuous read mode or remain in continuous read mode.
3	DUMMY_SDR	2h (Quad)	08h	Dummy cycle is 8 (in serial root clock) as CR2V[3:0] = 8h.
4	READ_SDR	2h (Quad)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.
5	JMP_ON_CS	0h (or Don't care)	01h	CMD instruction is bypassed after the first read access.
6-7	STOP (00h)	0h	00h	

25.6.1.9 DDR Quad IO Fast Read command

Table 528 shows DDR Quad IO FAST_READ command sequence. This table shows Cypress SPI Configuration Register bits CR2V[7] = 0, CR2V[6] = 0, CR2V[3:0] = 8h, Non-QPI mode, Non-continuous read mode.

Table 528. DDR Quad IO FAST_READ command (Non-QPI mode, Noncontinuous read mode)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	EDh or EEh	Command name: QIOR_DDR or 4QIOR_DDR QIOR_DDR is 3-byte address mode. 4QIOR_DDR is 4-byte address mode.

Table continues on the next page...

Table 528. DDR Quad IO FAST_READ command (Non-QPI mode, Noncontinuous read mode) (continued)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
1	ADDR_DDR	2h (Quad)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	MODE8_DDR	2h (Quad)	Any value other than Axh	Exit continuous read mode or keep in noncontinuous read mode.
3	DUMMY_DDR	2h (Quad)	08h	Dummy cycle is 8 (in serial root clock) as CR2V[3:0] = 8h.
4	LEARN_DDR	2h (Quad)	1h	DLP is 8 bits.
5	READ_DDR	2h (Quad)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.
6-7	STOP (00h)	0h	00h	

Table 529 shows DDR Quad IO FAST_READ command sequence. This table shows Cypress SPI Configuration register bits CR2V[7] = 0, CR2V[6] = 0, CR2V[3:0] = 8h, Non-QPI mode, Continuous read mode.

Table 529. DDR Quad IO FAST_READ command (Non-QPI mode, Continuous read mode)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	EDh or EEh	Command name: QIOR_DDR or 4QIOR_DDR QIOR_DDR is 3-byte address mode. 4QIOR_DDR is 4-byte address mode.
1	ADDR_DDR	2h (Quad)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	MODE8_DDR	2h (Quad)	A0h	Enter continuous read mode or remain in continuous read mode.
3	DUMMY_DDR	2h (Quad)	08h	Dummy cycle is 8 (in serial root clock) as CR2V[3:0] = 8h.
4	LEARN_DDR	2h (Quad)	1h	DLP is 8 bits.
5	READ_DDR	2h (Quad)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.
6	JMP_ON_CS	0h (or Don't care)	01h	CMD instruction is bypassed after the first read access.
7	STOP (00h)	0h	00h	

Table 530 shows DDR Quad IO FAST_READ command sequence. This table shows Cypress SPI Configuration Register bits CR2V[7] = 0, CR2V[6] = 1, CR2V[3:0] = 8h, QPI mode, Continuous read mode.

Table 530. DDR Quad IO FAST_READ command (QPI mode, Continuous read mode)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	2h (Quad)	EDh or EEh	Command name: QIOR_DDR or 4QIOR_DDR QIOR_DDR is 3-byte address mode. 4QIOR_DDR is 4-byte address mode.
1	ADDR_DDR	2h (Quad)	18h or 20h	Address bit number (operand value) should be 24 in 3-byte address mode and 32 in 4-byte address mode.
2	MODE8_DDR	2h (Quad)	A0h	Enter continuous read mode or remain in continuous read mode.
3	DUMMY_DDR	2h (Quad)	08h	Dummy cycle is 8 (in serial root clock) as CR2V[3:0] = 8h.
4	LEARN_DDR	2h (Quad)	1h	DLP is 8 bits.
5	READ_DDR	2h (Quad)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.
6	JMP_ON_CS	0h (or Don't care)	01h	CMD instruction is bypassed after the first read access.
7	STOP (00h)	0h	00h	

25.6.2 Application on HyperBus device

This section provides the example LUT instruction sequences for HyperBus device (Cypress RPC flash, HyperRam, or HyperFlash).

25.6.2.1 HyperFlash

This section provides example sequences for HyperFlash devices (Cypress S26KS series).

Table 531 shows the read status command sequence.

Table 531. Read status command

Seq num	Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0 (Write - Addr = 555h, Data = 70h)	0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W# = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	CMD_DDR	3h (Octal)	00h	Row Address: 0000AAh (24 bit)
	2	CMD_DDR	3h (Octal)	00h	
	3	CMD_DDR	3h (Octal)	AAh	

Table continues on the next page...

Table 531. Read status command (continued)

Seq num	Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
	4	CMD_DDR	3h (Octal)	00h	Column Address: 05h (13 zero bits + 3 valid bits)
	5	CMD_DDR	3h (Octal)	05h	
	6	CMD_DDR	3h (Octal)	00h	Write Data: 0070h
	7	CMD_DDR	3h (Octal)	70h	
1 (Read - Addr = xxx, Data = Status register data)	0	CMD_DDR	3h (Octal)	A0h	CA bit 47: (R/W#) = 1h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	RADDR_DDR	3h (Octal)	18h	Row Address: 24 bits
	2	CADDR_DDR	3h (Octal)	10h	Column Address: (13 zero bits + 3 valid bits)
	3	DUMMY_RWDS_DDR	3h (Octal)	0Bh	When latency count = 11
	4	READ_DDR	0h (Octal)	4h	4-Byte read
	5-7	STOP (00h)	0h	00h	

Table 532 shows the read (memory) command sequence.

Table 532. Read (memory) command

Seq num	Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0 (Read - Addr = xxx, Data = memory data)	0	CMD_DDR	3h (Octal)	A0h	CA bit 47: (R/W#) = 1h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	RADDR_DDR	3h (Octal)	18h	Row Address: 24 bits
	2	CADDR_DDR	3h (Octal)	10h	Column Address: (13 zero bits + 3 valid bits)
	3	DUMMY_RWDS_DDR	3h (Octal)	0Bh	When latency count = 11
	4	READ_DDR	3h (Octal)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size. This value is ignored for AHB command.
	5-7	STOP (00h)	0h	00h	

Table 533 shows the word program command sequence.

Table 533. Word program command

Seq num	Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0 (Write - Addr = 555h, Data = AAh)	0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	CMD_DDR	3h (Octal)	00h	Row Address: 0000AAh (24 bit)
	2	CMD_DDR	3h (Octal)	00h	
	3	CMD_DDR	3h (Octal)	AAh	
	4	CMD_DDR	3h (Octal)	00h	Column Address: 05h (13 zero bits + 3 valid bits)
	5	CMD_DDR	3h (Octal)	05h	
	6	CMD_DDR	3h (Octal)	00h	Write Data: 00AAh
	7	CMD_DDR	3h (Octal)	AAh	
1 (Write - Addr = 2AAh, Data = 55h)	0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	CMD_DDR	3h (Octal)	00h	Row Address: 000055h (24 bit)
	2	CMD_DDR	3h (Octal)	00h	
	3	CMD_DDR	3h (Octal)	55h	
	4	CMD_DDR	3h (Octal)	00h	Column Address: 02h (13 zero bits + 3 valid bits)
	5	CMD_DDR	3h (Octal)	02h	
	6	CMD_DDR	3h (Octal)	00h	Write Data: 0055h
	7	CMD_DDR	3h (Octal)	55h	
2 (Write - Addr = 555h, Data = A0h)	0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	CMD_DDR	3h (Octal)	00h	Row Address: 0000AAh (24 bit)
	2	CMD_DDR	3h (Octal)	00h	
	3	CMD_DDR	3h (Octal)	AAh	
	4	CMD_DDR	3h (Octal)	00h	Column Address: 05h (13 zero bits + 3 valid bits)
	5	CMD_DDR	3h (Octal)	55h	
	6	CMD_DDR	3h (Octal)	00h	Write Data: 00A0h

Table continues on the next page...

Table 533. Word program command (continued)

Seq num	Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
	7	CMD_DDR	3h (Octal)	A0h	
3 (Word Program)	0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	RADDR_DDR	3h (Octal)	18h	Row Address: 24 bits
	2	CADDR_DDR	3h (Octal)	10h	Column Address: (13 zero bits + 3 valid bits)
	3	WRITE_DDR	3h (Octal)	02h	2-Byte written data
	4-7	STOP (0h)	0h	00h	

Table 534 shows Write-to-Buffer and Program-Buffer-to-Flash command sequence.

Table 534. Write-to-Buffer and Program-Buffer-to-Flash command

Seq num	Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0 (Write - Addr = 555h, Data = AAh)	0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	CMD_DDR	3h (Octal)	00h	Row Address: 0000AAh (24 bit)
	2	CMD_DDR	3h (Octal)	00h	
	3	CMD_DDR	3h (Octal)	AAh	
	4	CMD_DDR	3h (Octal)	00h	Column Address: 05h (13 zero bits + 3 valid bits)
	5	CMD_DDR	3h (Octal)	05h	
	6	CMD_DDR	3h (Octal)	00h	Write Data: 00AAh
	7	CMD_DDR	3h (Octal)	AAh	
1 (Write - Addr = 2AAh, Data = 55h)	0	CMD_DDR	0h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	CMD_DDR	3h (Octal)	00h	Row Address: 000055h (24 bit)
	2	CMD_DDR	3h (Octal)	00h	
	3	CMD_DDR	3h (Octal)	55h	

Table continues on the next page...

Table 534. Write-to-Buffer and Program-Buffer-to-Flash command (continued)

Seq num	Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
	4	CMD_DDR	3h (Octal)	00h	Column Address: 02h (13 zero bits + 3 valid bits)
	5	CMD_DDR	3h (Octal)	02h	
	6	CMD_DDR	3h (Octal)	00h	Write Data: 0055h
	7	CMD_DDR	3h (Octal)	55h	
2 (Write - Addr = SA, Data = 25h)	0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	RADDR_DDR	3h (Octal)	18h	Row Address: SA (24 bit) SA is sector address. Write SA to IPCR0[SFAR].
	2	CADDR_DDR	3h (Octal)	10h	Column Address: 13 zero bits + 3 valid bits
	3	CMD_DDR	3h (Octal)	00h	Write Data: 0025h
	4	CMD_DDR	3h (Octal)	25h	
2 (Write - Addr = SA, Data = WC)	0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	RADDR_DDR	3h (Octal)	18h	Row Address: SA (24 bit) SA is sector address. Write SA to IPCR0[SFAR].
	2	CADDR_DDR	3h (Octal)	10h	Column Address: 13 zero bits + 3 valid bits
	3	CMD_DDR	3h (Octal)	WC	Write Data: WC WC is word count.
	4	CMD_DDR	3h (Octal)		
3 - N (Write - Addr = WBL, Data = PD) N is the word count + 2	0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	RADDR_DDR	3h (Octal)	18h	Row Address: WBL (24 bit) WBL is write buffer location. Write WBL to IPCR0[SFAR].

Table continues on the next page...

Table 534. Write-to-Buffer and Program-Buffer-to-Flash command (continued)

Seq num	Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
	2	CADDR_DDR	3h (Octal)	10h	Column Address: 13 zero bits + 3 valid bits
	3	WRITE_DDR	3h (Octal)	02h	2-Byte write data
	4-7	STOP (0h)	0h	00h	
N + 1 (Write - Addr = SA, Data = 29) Program Buffer to Flash	0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
	1	RADDR_DDR	3h (Octal)	18h	Row Address: SA (24 bit) SA is sector address. Write SA to IPCR0[SFAR].
	2	CADDR_DDR	3h (Octal)	10h	Column Address: 13 zero bits + 3 valid bits
	3	CMD_DDR	3h (Octal)	00h	Write Data: 29h
	4	CMD_DDR	3h (Octal)	29h	
	5-7	STOP (0h)	0h	00h	

25.6.2.2 HyperRAM

This section provides example sequences for HyperRAM (Cypress S27KL series).

The Read (memory) command sequence is same as HyperFlash. Table 535 shows the Write (memory) command sequence.

Table 535. Write (memory) command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_DDR	3h (Octal)	20h	CA bit 47: (R/W#) = 0h CA bit 46: (Target) = 0h CA bit 45: (Burst Type) = 1h CA bit 44-40: All reserved = 0h
1	RADDR_DDR	3h (Octal)	18h	Row Address: 24 bits
2	CADDR_DDR	3h (Octal)	10h	Column Address: (13 zero bits + 3 valid bits)
3	DUMMY_RWDS_DDR	3h (Octal)	0Bh	When latency count = 11
4	WRITE_DDR	3h (Octal)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default writing

Table continues on the next page...

Table 535. Write (memory) command (continued)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
				data size. This value is ignored for AHB command.
5-7	STOP (00h)	0h	00h	

25.6.3 Application on Serial NAND flash device

This section provides example LUT instruction sequences for serial NAND flash device (Micron Flash MT29 series). The operation of serial NAND flash is similar to serial NOR flash.

The read operation sequence is:

- Page read. Transfer the data from the NAND flash array to the cache register.
- Get feature to read the status.
- Random data read

Table 536 shows the page read command sequence.

Table 536. Page Read command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	1h (Single)	13h	Command code: 13h
1	RADDR_SDR	1h (Single)	18h	Row address: 24 bit
2-7	STOP (0h)	0h	00h	

Table 537 shows Get Feature command sequence.

Table 537. Get Feature command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	0h (Single)	0Fh	Command code: 0Fh
1	CMD_SDR	1h (Single)	C0h	Status register address (C0h)
2	READ_SDR	0x1 (Single)	02h	2 bytes read data
3-7	STOP (0h)	0h	00h	

Table 538 shows Random Data Read command sequence.

Table 538. Read from Cache x4 command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	1h (Single)	6Bh	Command code: 6Bh
1	MODE4_SDR	1h (Single)	0h or 1h	If plane selection is one, software should decode the flash address and write 1h to mode bits. If plane selection

Table continues on the next page...

Table 538. Read from Cache x4 command (continued)

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
				is zero, software should write 0h to mode bits. Plane selection bit is 18th bit of flash address. If NAND flash size is less than 4 Gbit, plane selection is always zero.
2	CADDR_SDR	1h (Single)	0Ch	Column address: 12 bit
3	DUMMY_SDR	2h (Quad)	08h	Dummy cycle number: 8 (serial root clock)
4	READ_SDR	2h (Quad)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default reading data size.
5-7	STOP (0x0)	0x0	0x00	

The program operation sequence is:

- Write enable
- Program Load. Transfer the write data to the cache register.
- Program Execute
- Get Feature to read the status.

Table 539 shows Program Load command sequence.

Table 539. Program Load command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	1h (Single)	02h	Command code: 02h
1	MODE4_SDR	1h (Single)	0h or 1h	If plane selection is one, software should decode the flash address and write 1h to mode bits. If plane selection is zero, software should write 0h to mode bits. Plane selection bit is 18th bit of flash address. If NAND flash size is less than 4 Gbit, plane selection is always zero.
2	CADDR_SDR	1h (Single)	0Ch	Column address: 12 bit
3	WRITE_SDR	1h (Single)	Any nonzero value	If IPCR1[IDATSZ] is zero, this operand value can be used as default writing data size.
4-7	STOP (0h)	0h	00h	

Table 540 shows Program Execute command sequence.

Table 540. Program Execute command

Instruction number	Instruction opcode[5:0]	Instruction num_pads[1:0]	Instruction operand[7:0]	Comment
0	CMD_SDR	1h (Single)	10h	Command code: 10h
1	RADDR_SDR	1h (Single)	18h	Row Address: 24 bit
2-7	STOP (0h)	0h	00h	

25.6.4 Application on FPGA device

An FPGA device can be accessed via both AHB and IP commands. All AHB accesses to FPGA are transparent to the software driver (no software intervention). An FPGA device may have some special requirements.

Table 541. Special requirements for FPGA devices

Condition	How to manage the condition
Device type may be different on A1, A2, B1, or B2.	Write 0 to MCR2[SAMEDEVICEEN] and configure the FLSHxCR0 and FLSHxCR1 registers separately for up to four external devices.
Device requires different wait cycle for programming.	The AHB write wait cycle number can be set separately for these four external devices (via FLSHxCR2[AWRWAIT]). Software can configure the sequences in LUT with different dummy instructions (operand determines dummy cycle). FlexSPI holds the AHB bus ready for this wait time; AHB bus performance may decrease when this wait time is very long.
Device requires different wait cycle for reading.	The AHB read sequence index and sequence number can be set separately for these four external devices (via FLSHxCR2[ARDSEQID] and FLSHxCR2[ARDSEQNUM]). Software can configure the sequences in LUT with different dummy instructions (operand determines dummy cycle).
Device may be sensitive to read instruction clock cycle number.	If its internal memory is implemented similar to a FIFO, the device is sensitive to the read instruction clock cycle number. Software can send the data size information to the external device via DATSZ instruction. The FPGA device decodes the data size information and determines how many data bytes must be popped.
Device may require interval time between chip select valid.	This interval can be managed via FLSHxCR1[CSINTERVAL].
Device may use SCLK as reference clock for its internal PLL.	SCLK must be free-running and the clock frequency must be stable. To achieve this configuration, write 1 to MCR0[SCKFREERUNEN] and only use SDR sequences.

25.6.5 Overview of error categories, flags, and triggered sources

Table 542 provides an overview of the categories, flags, and triggered sources of errors in FlexSPI.

Table 542. Error category, triggered sources, and flags

Error Category	Triggered Sources	Description	Error Flags
Command grant error	AHB write command	Command grant timeout	INTR[AHBCMDGE] is set AHB bus error response

Table continues on the next page...

Table 542. Error category, triggered sources, and flags (continued)

Error Category	Triggered Sources	Description	Error Flags
	AHB read command		INTR[AHBCMDGE] is set AHB bus error response
	IP command		INTR[IPCMDGE] is set
Command check error	AHB write command	• AHB write command with JMP_ON_CS instruction used in the sequence • Unknown instruction opcode in the sequence • Instruction DUMMY_SDR or DUMMY_RWDS_SDR used in DDR sequence • Instruction DUMMY_DDR or DUMMY_RWDS_DDR used in SDR sequence	INTR[AHBCMDERR] is set Command is not executed when an error is detected in command check
	AHB read command	• Unknown instruction opcode in the sequence • Instruction DUMMY_SDR or DUMMY_RWDS_SDR used in DDR sequence • Instruction DUMMY_DDR or DUMMY_RWDS_DDR used in SDR sequence	INTR[AHBCMDERR] is set Command is not executed when an error is detected in command check
	IP command	• IP command with JMP_ON_CS instruction used in the sequence • Unknown instruction opcode in the sequence • Instruction DUMMY_SDR or DUMMY_RWDS_SDR used in DDR sequence • Instruction DUMMY_DDR or DUMMY_RWDS_DDR used in SDR sequence • Flash boundary across	INTR[IPCMDERR] is set Command is not executed when an error is detected in command check
Command execution error	AHB write command	Command timeout during execution	INTR[AHBCMDERR] is set INTR[SEQTIMEOUT] is set An AHB bus error response

Table continues on the next page...

Table 542. Error category, triggered sources, and flags (continued)

Error Category	Triggered Sources	Description	Error Flags
			occurs, except in following cases: • Flush triggers AHB write command (INCR burst ended with AHB_TX_BUF not empty) • AHB bufferable write access and bufferable enabled (AHBCR[BUFFERABLEEN] = 1)
	AHB read command		INTR[AHBCMDERR] is set INTR[SEQTIMEOUT] is set AHB bus error response
	IP command		INTR[IPCMDERR] is set INTR[SEQTIMEOUT] is set
AHB bus timeout	AHB write command	AHB bus timeout (no bus ready return)	INTR[AHBBUSTIMEOUT] is set AHB bus error response
	AHB read command		
Data learning failed	Any command	No valid sample clock phase found after learn instruction is executed	INTR[DATALEARNFAIL] is set
Security error	IP command	Nonsecure controller requests IP command access to a secure region	INTR[IPCMDSECUREVIO] is set
Security error	AHB command	IP command GCM data check error	INTR[AHBGCMERR] is set

25.7 Memory map and register definition

This section includes the FlexSPI module memory map and detailed descriptions of all registers.

25.7.1 Register access

All registers can be accessed with 8-bit, 16-bit, and 32-bit width operations. Never change the values of reserved fields in control registers. Changing the values of reserved fields may impact the normal functioning of the controller.

NOTE

When FlexSPI can access sensitive memory contents, the FlexSPI controller should protect those contents using resource isolation such as XRDC or TrustZone, or equivalent partition control via SCFW. Ensure that only trusted software is allowed to access the FlexSPI controller register interface.

25.7.2 FlexSPI register descriptions

25.7.2.1 FlexSPI memory map

FLEXSPI0 base address: 400C_8000h

Offset	Register	Width (In bits)	Access	Reset value
0h	Module Control 0 (MCR0)	32	RW	FFFF_8002h
4h	Module Control 1 (MCR1)	32	RW	FFFF_FFFFh
8h	Module Control 2 (MCR2)	32	RW	2000_81F7h
Ch	AHB Bus Control (AHBCR)	32	RW	0000_0018h
10h	Interrupt Enable (INTEN)	32	RW	0000_0000h
14h	Interrupt (INTR)	32	RW	0000_0000h
18h	LUT Key (LUTKEY)	32	RW	5AF0_5AF0h
1Ch	LUT Control (LUTCR)	32	RW	0000_0002h
20h	AHB Receive Buffer 0 Control 0 (AHBRXBUF0CR0)	32	RW	8000_0010h
24h	AHB Receive Buffer 1 Control 0 (AHBRXBUF1CR0)	32	RW	8001_0010h
28h	AHB Receive Buffer 2 Control 0 (AHBRXBUF2CR0)	32	RW	8002_0010h
2Ch	AHB Receive Buffer 3 Control 0 (AHBRXBUF3CR0)	32	RW	8003_0010h
30h	AHB Receive Buffer 4 Control 0 (AHBRXBUF4CR0)	32	RW	8004_0010h
34h	AHB Receive Buffer 5 Control 0 (AHBRXBUF5CR0)	32	RW	8005_0010h
38h	AHB Receive Buffer 6 Control 0 (AHBRXBUF6CR0)	32	RW	8006_0010h
3Ch	AHB Receive Buffer 7 Control 0 (AHBRXBUF7CR0)	32	RW	8007_0010h
60h	Flash Control 0 (FLSHA1CR0)	32	RW	0001_0000h
64h	Flash Control 0 (FLSHA2CR0)	32	RW	0001_0000h
68h	Flash Control 0 (FLSHB1CR0)	32	RW	0001_0000h
6Ch	Flash Control 0 (FLSHB2CR0)	32	RW	0001_0000h
70h	Flash Control 1 (FLSHA1CR1)	32	RW	0000_0063h
74h	Flash Control 1 (FLSHA2CR1)	32	RW	0000_0063h
78h	Flash Control 1 (FLSHB1CR1)	32	RW	0000_0063h
7Ch	Flash Control 1 (FLSHB2CR1)	32	RW	0000_0063h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
80h	Flash Control 2 (FLSHA1CR2)	32	RW	0000_0000h
84h	Flash Control 2 (FLSHA2CR2)	32	RW	0000_0000h
88h	Flash Control 2 (FLSHB1CR2)	32	RW	0000_0000h
8Ch	Flash Control 2 (FLSHB2CR2)	32	RW	0000_0000h
94h	Flash Control 4 (FLSHCR4)	32	RW	0000_0000h
A0h	IP Control 0 (IPCR0)	32	RW	0000_0000h
A4h	IP Control 1 (IPCR1)	32	RW	0000_0000h
A8h	IP Control 2 (IPCR2)	32	RW	0000_0000h
B0h	IP Command (IPCMD)	32	RW	0000_0000h
B4h	Data Learning Pattern (DLPR)	32	RW	0000_0000h
B8h	IP Receive FIFO Control (IPRXFCR)	32	RW	0000_0000h
BCh	IP Transmit FIFO Control (IPTXFCR)	32	RW	0000_0000h
C0h	DLL Control 0 (DLLACR)	32	RW	0000_0100h
C4h	DLL Control 0 (DLLBCR)	32	RW	0000_0100h
E0h	Status 0 (STS0)	32	R	0000_0002h
E4h	Status 1 (STS1)	32	R	0000_0000h
E8h	Status 2 (STS2)	32	R	0100_0100h
ECh	AHB Suspend Status (AHBSPNDSTS)	32	R	0000_0000h
F0h	IP Receive FIFO Status (IPRXFSTS)	32	R	0000_0000h
F4h	IP Transmit FIFO Status (IPTXFSTS)	32	R	0000_0000h
100h - 17Ch	IP Receive FIFO Data a (RFDR0 - RFDR31)	32	R	See section
180h - 1FCh	IP TX FIFO Data a (TFDR0 - TFDR31)	32	W	0000_0000h
200h - 2FCh	Lookup Table a (LUT0 - LUT63)	32	RW	See section
420h	HADDR REMAP Start Address (HADDRSTART)	32	RW	0000_0000h
424h	HADDR REMAP END ADDR (HADDREND)	32	RW	0000_0000h
428h	HADDR Remap Offset (HADDROFFSET)	32	RW	0000_0000h
42Ch	IPED Function Control (IPEDCTRL)	32	RW	0000_0000h
440h	Receive Buffer Start Address of Region 0 (AHBBUFREGIONSTART0)	32	RW	0000_0000h
444h	Receive Buffer Region 0 End Address (AHBBUFREGIONEND0)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
448h	Receive Buffer Start Address of Region 1 (AHBBUFREGIONSTART1)	32	RW	0000_0000h
44Ch	Receive Buffer Region 1 End Address (AHBBUFREGIONEND1)	32	RW	0000_0000h
450h	Receive Buffer Start Address of Region 2 (AHBBUFREGIONSTART2)	32	RW	0000_0000h
454h	Receive Buffer Region 2 End Address (AHBBUFREGIONEND2)	32	RW	0000_0000h
458h	Receive Buffer Start Address of Region 3 (AHBBUFREGIONSTART3)	32	RW	0000_0000h
45Ch	Receive Buffer Region 3 End Address (AHBBUFREGIONEND3)	32	RW	0000_0000h
500h	IPED context control 0 (IPEDCTXCTRL0)	32	RW	5555_5555h
504h	IPED context control 1 (IPEDCTXCTRL1)	32	RW	AAAA_AAAAh
520h	IPED Context0 IV0 (IPEDCTX0IV0)	32	RW	0000_0000h
524h	IPED Context0 IV1 (IPEDCTX0IV1)	32	RW	0000_0000h
528h	Start Address of Region (IPEDCTX0START)	32	RW	0000_0000h
52Ch	End Address of Region (IPEDCTX0END)	32	RW	0000_0000h
530h	IPED Context0 Additional Authenticated Data0 (IPEDCTX0AAD0)	32	RW	0000_0000h
534h	IPED Context0 Additional Authenticated Data1 (IPEDCTX0AAD1)	32	RW	0000_0000h
540h	IPED Context1 IV0 (IPEDCTX1IV0)	32	RW	0000_0000h
544h	IPED Context1 IV1 (IPEDCTX1IV1)	32	RW	0000_0000h
548h	Start Address of Region (IPEDCTX1START)	32	RW	0000_0000h
54Ch	End Address of Region (IPEDCTX1END)	32	RW	0000_0000h
550h	IPED Context1 Additional Authenticated Data0 (IPEDCTX1AAD0)	32	RW	0000_0000h
554h	IPED Context1 Additional Authenticated Data1 (IPEDCTX1AAD1)	32	RW	0000_0000h
560h	IPED Context2 IV0 (IPEDCTX2IV0)	32	RW	0000_0000h
564h	IPED Context2 IV1 (IPEDCTX2IV1)	32	RW	0000_0000h
568h	Start Address of Region (IPEDCTX2START)	32	RW	0000_0000h
56Ch	End Address of Region (IPEDCTX2END)	32	RW	0000_0000h
570h	IPED Context2 Additional Authenticated Data0 (IPEDCTX2AAD0)	32	RW	0000_0000h
574h	IPED Context2 Additional Authenticated Data1 (IPEDCTX2AAD1)	32	RW	0000_0000h
580h	IPED Context3 IV0 (IPEDCTX3IV0)	32	RW	0000_0000h
584h	IPED Context3 IV1 (IPEDCTX3IV1)	32	RW	0000_0000h
588h	Start Address of Region (IPEDCTX3START)	32	RW	0000_0000h

Table continues on the next page...

Table continued from the previous page...

Offset	Register	Width (In bits)	Access	Reset value
58Ch	End Address of Region (IPEDCTX3END)	32	RW	0000_0000h
590h	IPED Context3 Additional Authenticated Data0 (IPEDCTX3AAD0)	32	RW	0000_0000h
594h	IPED Context3 Additional Authenticated Data1 (IPEDCTX3AAD1)	32	RW	0000_0000h
5A0h	IPED Context4 IV0 (IPEDCTX4IV0)	32	RW	0000_0000h
5A4h	IPED Context4 IV1 (IPEDCTX4IV1)	32	RW	0000_0000h
5A8h	Start Address of Region (IPEDCTX4START)	32	RW	0000_0000h
5ACh	End Address of Region (IPEDCTX4END)	32	RW	0000_0000h
5B0h	IPED Context4 Additional Authenticated Data0 (IPEDCTX4AAD0)	32	RW	0000_0000h
5B4h	IPED Context4 Additional Authenticated Data1 (IPEDCTX4AAD1)	32	RW	0000_0000h
5C0h	IPED Context5 IV0 (IPEDCTX5IV0)	32	RW	0000_0000h
5C4h	IPED Context5 IV1 (IPEDCTX5IV1)	32	RW	0000_0000h
5C8h	Start Address of Region (IPEDCTX5START)	32	RW	0000_0000h
5CCh	End Address of Region (IPEDCTX5END)	32	RW	0000_0000h
5D0h	IPED Context5 Additional Authenticated Data0 (IPEDCTX5AAD0)	32	RW	0000_0000h
5D4h	IPED Context5 Additional Authenticated Data1 (IPEDCTX5AAD1)	32	RW	0000_0000h
5E0h	IPED Context6 IV0 (IPEDCTX6IV0)	32	RW	0000_0000h
5E4h	IPED Context6 IV1 (IPEDCTX6IV1)	32	RW	0000_0000h
5E8h	Start Address of Region (IPEDCTX6START)	32	RW	0000_0000h
5ECh	End Address of Region (IPEDCTX6END)	32	RW	0000_0000h
5F0h	IPED Context6 Additional Authenticated Data0 (IPEDCTX6AAD0)	32	RW	0000_0000h
5F4h	IPED Context6 Additional Authenticated Data1 (IPEDCTX6AAD1)	32	RW	0000_0000h

25.7.2.2 Module Control 0 (MCR0)

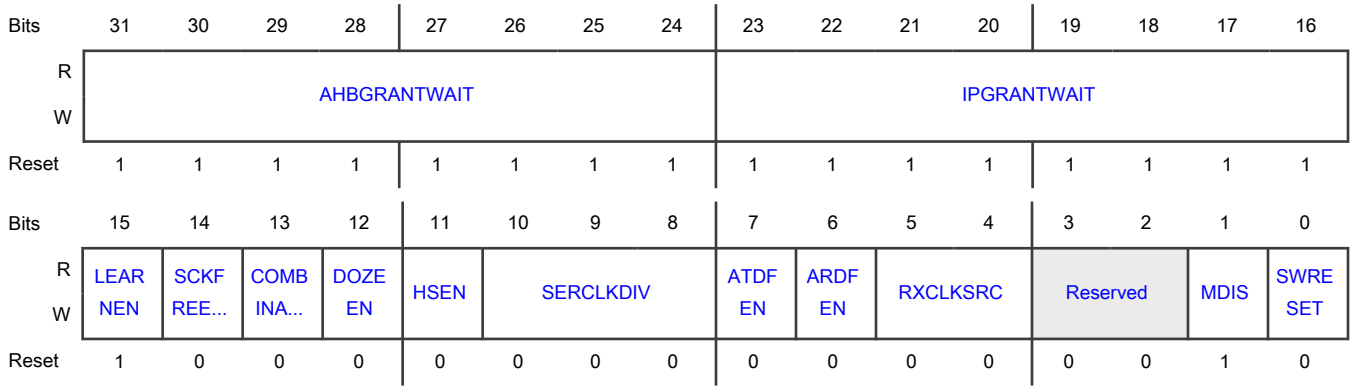
Offset

Register	Offset
MCR0	0h

Function

Controls basic functions of FlexSPI module

Diagram



Fields

Field	Function
31-24 AHBGRANTWAIT	Timeouts Wait Cycle for AHB command Grant Sets duration of timeout wait cycle for AHB commands. If the arbitrator does not grant the AHB-triggered command, it times out after (AHBGRANTTIMEOUT × 1024) AHB clock cycles. When the pending command sequence is IP-triggered and the read or write data size is too large, this grant timeout may occur. When an AHB command grant timeout occurs, an INTR[AHBCMDGE] interrupt is generated. When INTEN[AHBCMDGEEN] = 1, the arbitrator ignores AHB commands. **NOTE** This field is for debugging only. Do not change the value from its default.
23-16 IPGRANTWAIT	Timeout Wait Cycle for IP Command Grant Sets duration of timeout wait cycle for IP commands. If the arbitrator does not grant the IP-triggered command, it times out after (IPGRANTTIMEOUT × 1024) AHB clock cycles. When the pending command sequence is AHB-triggered and the read or write data size is too large, this grant timeout may occur. When IP command grant timeout occurs, an INTR[IPCMDGE] interrupt is generated. When INTEN[IPCMDGEEN] = 1, the arbitrator ignores IP commands. **NOTE** This field is for debugging only. Do not change the value from its default.
15 LEARNEN	Data Learning Enable Enables data learning feature. When data learning is disabled, the sampling clock phase 0 is always used for receive data sampling, even when a learn instruction is correctly executed. See Data learning. 0b - Disable 1b - Enable
14 SCKFREERUNEN	SCLK Free-running Enable Enables free-running SCLK output. For FPGA applications, the external device may use SCLK as a reference clock to its internal PLL. If SCLK free-running is enabled, data sampling with loopback clock from SCLK pad is not supported (MCR0[RXCLKSRC] = 2).

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Disable 1b - Enable
13 COMBINATIONEN	Combination Mode Enable Enables combination mode, which supports flash Octal mode access. When Port A and Port B data are four bits wide, this mode combines Port A and B Data pins (A_DATA[3:0] and B_DATA[3:0]). When Port A and Port B data are eight bits wide, combination mode is not supported; write 0 to this field in this case. 0b - Disable 1b - Enable
12 DOZEEN	Doze Mode Enable Enables Doze mode. When enabled, AHB clock and serial clock are gated off when there is a Doze-mode request from the system. 0b - Disable 1b - Enable
11 HSEN	Half Speed Serial Flash Memory Access Enable Enables the clock divider to provide a half-speed clock to external serial flash devices (A_SCLK/B_SCLK) for all commands in SDR and DDR mode. Write 1 to MCR0[MDIS] before changing the value of this field. Failure to do so may cause issues in the internal logic or state machine. 0b - Disable 1b - Enable
10-8 SERCLKDIV	Serial Root Clock Divider Sets divider value for serial root clock. The serial root clock can be divided inside FlexSPI . See Clocking. **NOTE** Do not modify this field while FlexSPI is active (MCR0[MDIS] = 0). 000b - Divided by 1 001b - Divided by 2 010b - Divided by 3 011b - Divided by 4 100b - Divided by 5 101b - Divided by 6 110b - Divided by 7 111b - Divided by 8
7	AHB Write Access to IP Transmit FIFO Enable

Table continues on the next page...

Table continued from the previous page...

Field	Function
ATDFEN	Enables AHB write access to IP transmit FIFO. See Writing data to IP transmit FIFO. 0b - AHB write access disabled. IP bus writes to IP transmit FIFO. AHB bus write access to IP transmit FIFO memory space produces bus error. 1b - AHB write access enabled. AHB bus writes to IP transmit FIFO. IP Bus write access to IP transmit FIFO memory space is ignored and causes no bus error.
6 ARDFEN	AHB Read Access to IP Receive FIFO Enable Enables AHB read access to IP receive FIFO. See Reading data from IP receive FIFO. 0b - AHB read access disabled. IP bus reads IP receive FIFO. AHB Bus read access to IP receive FIFO memory space produces bus error. 1b - AHB read access enabled. AHB bus reads IP receive FIFO. IP Bus read access to IP receive FIFO memory space returns data zero and causes no bus error.
5-4 RXCLKSRC	Sample Clock Source for Flash Reading Selects sampling clock source for flash memory reading. See Receive clock source features. 00b - Dummy Read strobe that FlexSPI generates, looped back internally 01b - Dummy Read strobe that FlexSPI generates, looped back from DQS pad 10b - SCLK output clock and looped back from SCLK pad 11b - Flash-memory-provided read strobe and input from DQS pad
3-2 —	Reserved
1 MDIS	Module Disable Disables FlexSPI module. When the module is disabled, AHB and serial clock are gated off internally to save power. Only register access is allowed, except for the LUT, IP receive FIFO, and IP transmit FIFO. 0b - No impact 1b - Module disable
0 SWRESET	Software Reset Resets the internal FlexSPI state machine and aborts any transactions in progress. Hardware automatically writes 0 to this field after software reset is done. Configuration registers are not reset. 0b - No impact 1b - Software reset

25.7.2.3 Module Control 1 (MCR1)

Offset

Register	Offset
MCR1	4h

Function
Controls basic functions of FlexSPI module

Diagram

Fields

Field	Function
31-16 SEQWAIT	Command Sequence Wait Sets wait time for command sequence. Command sequence execution times out and aborts after (SEQWAIT × 1024) serial root clock cycles. When this timeout occurs, if the interrupt is enabled (INTEN[SEQTIMEOUTEN] = 1), an INTR[SEQTIMEOUT] interrupt is generated. Also, the arbitrator ignores AHB commands. **NOTE** You cannot write 0 to this field.
15-0 AHBBUSWAIT	AHB Bus Wait Sets wait time for AHB bus. When data is not received from or transmitted to serial flash memory space after (AHBBUSWAIT × 1024) AHB clock cycles, the read or write access times out. When this timeout occurs, the AHB bus receives an error response, and an interrupt is generated (INTR[AHBBUSTIMEOUT]). When INTR[AHBBUSTIMEOUT] = 1, the arbitrator ignores AHB commands. **NOTE** You cannot write 0 to this field.

25.7.2.4 Module Control 2 (MCR2)

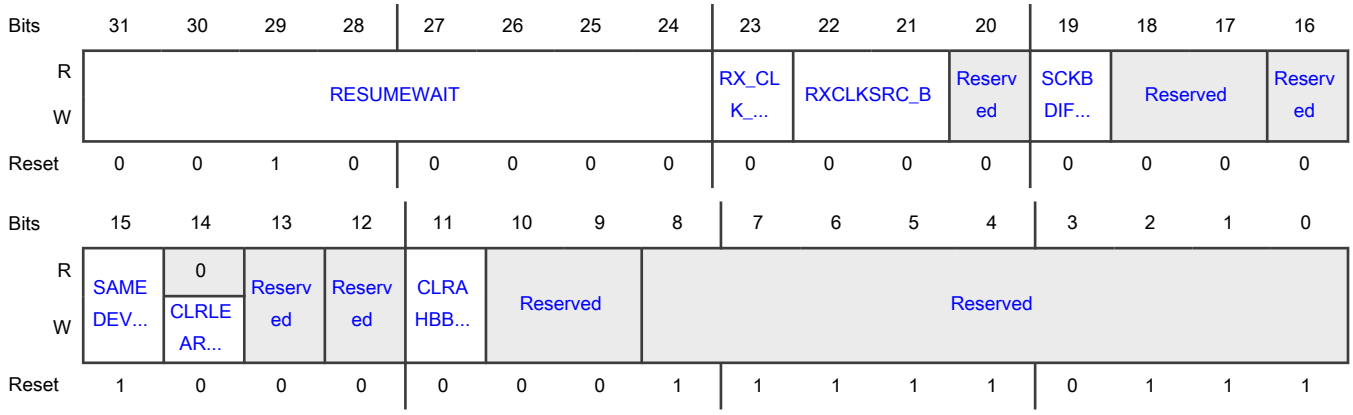
Offset

Register	Offset
MCR2	8h

Function

Controls basic functions of FlexSPI module.

Diagram



Fields

Field	Function
31-24 RESUMEWAIT	Resume Wait Duration Determines duration (in AHB clock cycles) to remain in idle state before suspended command sequence is resumed. See Command abort and suspend.
23 RX_CLK_SRC_DIFF	Sample Clock Source Different Selects whether to use the same clock source or a different clock source for Port A and Port B. 0b - Use MCR0[RXCLKSRC] for Port A and Port B. MCR2[RXCLKSRC_B] is ignored and MCR0[RXCLKSRC] selects the Sample Clock source for Flash Reading of both ports A and B. 1b - Use MCR0[RXCLKSRC] for Port A, and MCR2[RXCLKSRC_B] for Port B. MCR0[RXCLKSRC] selects the Sample Clock source for Flash Reading of port A (A_SCLK) and MCR2[RXCLKSRC_B] selects the Sample Clock source for Flash Reading of port B (B_SCLK).
22-21 RXCLKSRC_B	Port B Receiver Clock Source Selects Port B sample clock source selection for flash memory reading. 00b - Dummy read strobe that FlexSPI generates, looped back internally. 01b - Dummy read strobe that FlexSPI generates, looped back from DQS pad.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	10b - SCLK output clock and looped back from SCLK pad 11b - Flash-memory-provided read strobe and input from DQS pad
20 —	Reserved
19 SCKBDIFFOPT	SCLK Port B Differential Output Controls whether to use B_SCLK pad as the Port B SCLK output. Alternatively, B_SCLK pad can be used as A_SCLK differential clock output (inverted clock to A_SCLK). Before changing the value of this field, write 1 to MCR0[MDIS]. After changing the value of this field, write 1 to MCR0[SWRESET]. 0b - Use B_SCLK pad as port B SCLK clock output. Port B flash memory access is available. 1b - Use B_SCLK pad as port A SCLK inverted clock output (Differential clock to A_SCLK). Port B flash memory access is not available.
18-17 —	Reserved
16 —	Reserved
15 SAMEDEVICEEN	Same Device Enable Sets all external devices (A1, A2, B1, and B2) to be the same devices (in type and in size). 0b - In Individual mode, FLSHA1CRx and FLSHA2CRx, FLSHB1CRx and FLSHB2CRx settings are applied to Flash A1, A2, B1, B2 separately. In Parallel mode, FLSHA1CRx register setting is applied to Flash A1 and B1, FLSHA2CRx register setting is applied to Flash A2 and B2. FLSHB1CRx and FLSHB2CRx register settings are ignored. 1b - FLSHA1CR0, FLSHA1CR1, and FLSHA1CR2 register settings are applied to Flash A1, A2, B1, B2. FLSHA2CRx, FLSHB1CRx, and FLSHB2CRx settings are ignored.
14 CLRLEARNPHASE	Clear Learn Phase Selection Resets the sampling clock phase selection to 0. When 1 is written to this field, it becomes 0 again immediately. 0b - No impact 1b - Reset sample clock phase selection to 0
13 —	Reserved
12 —	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
11 CLRAHBBUFOPT	Clear AHB Buffer Determines whether AHB receive and transmit buffers are cleared automatically when FlexSPI returns Stop mode ACK. If an AHB receive buffer or transmit buffer will be powered off in Stop mode, software should write 1 to this field. Otherwise, an AHB read access after exiting Stop mode may hit either buffer, but their data entries are invalid. 0b - Not cleared automatically 1b - Cleared automatically
10-9 —	Reserved
8-0 —	Reserved

25.7.2.5 AHB Bus Control (AHBCR)

Offset

Register	Offset
AHBCR	Ch

Function

Controls AHB bus interface functions.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	AFLASHBASE				Reserved						ALIGNMENT		Reserved	Reserved	Reserved	Reserved
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved	Reserved		Reserved	Reserved	READSZA...	Reserved	Reserved	RESUMED...	READADD...	PREFETC...	BUFFERA...	CACHABL...	CLRAHBT...	0 CLRAHBR...	APAREN
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0

Fields

Field	Function
31-29 AFLASHBASE	AHB Memory-Mapped Flash Base Address
28-22 —	Reserved
21-20 ALIGNMENT	AHB Boundary Alignment Configures the size of AHB read and write boundary. All accesses that cross the boundary are divided into smaller sub accesses. 00b - No limit 01b - 1 KB 10b - 512 bytes 11b - 256 bytes
19 —	Reserved
18 —	Reserved
17 —	Reserved
16 —	Reserved
15 —	Reserved
14-13 —	Reserved
12 —	Reserved
11 —	Reserved
10 READSZALIGN	AHB Read Size Alignment Configures how AHB read size is determined. See Table 512. 0b - Register settings such as PREFETCH_EN determine AHB read size.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - AHB read size to up size to 8 bytes aligned, no prefetching
9 —	Reserved
8 —	Reserved
7 RESUMEDISABLE	AHB Read Resume Disable Disables the resumption of AHB read prefetch. 0b - Suspended AHB read prefetch resumes when AHB is IDLE. 1b - Suspended AHB read prefetch does not resume once aborted.
6 READADDROPT	AHB Read Address Option Removes AHB burst start address alignment limitation. When the FlexSPI controller is used for FPGA application, this field may be required for FlexSPI to fetch the exact byte number as an AHB burst. In this case, FPGA device should be non-word-addressable and this field must be 0. 0b - AHB read burst start address alignment is limited when flash memory is accessed in parallel mode or flash is word-addressable. 1b - AHB read burst start address alignment is not limited. FlexSPI fetches more data than the AHB burst requires for address alignment.
5 PREFETCHEN	AHB Read Prefetch Enable Enables AHB read prefetch. When enabled, FlexSPI fetches more flash read data than the current AHB burst requires. This action reduces the read latency for next AHB read access. **NOTE** AHB read prefetch is enabled only when both this field and AHBRXBUFnCR0[PREFETCHEN] are 1. 0b - Disable 1b - Enable
4 BUFFERABLEEN	Bufferable Write Access Enable Enables AHB bus bufferable write access. This field affects the last beat of an AHB write access. See AHB write access to flash memory. 0b - Disabled. For all AHB write accesses (bufferable or nonbufferable), FlexSPI returns AHB Bus Ready after transmitting all data and finishing command. 1b - Enabled. For AHB bufferable write access, FlexSPI returns AHB Bus Ready when the arbitrator grants the AHB command. FlexSPI does not wait for the AHB command to finish.
3 CACHABLEEN	Cacheable Read Access Enable Enables AHB bus cacheable read access.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Disabled. When an AHB bus cacheable read access occurs, FlexSPI does not check whether it hit the AHB transmit buffer. 1b - Enabled. When an AHB bus cacheable read access occurs, FlexSPI first checks whether the access hit the AHB transmit buffer.
2 CLRAHBTXBUF	Clear AHB Transmit Buffer Clears status and pointers of AHB transmit buffer. When the clear operation completes, this field clears automatically, so it always reads as 0. 0b - No impact. 1b - Enable clear operation.
1 CLRAHBRXBUF	Clear AHB Receive Buffer Clears status and pointers of AHB receive buffer. When the clear operation completes, this field clears automatically, so it always reads as 0. 0b - No impact. 1b - Enable clear operation.
0 APAREN	AHB Parallel Mode Enable Enables Parallel mode for AHB-triggered read and write commands. 0b - Flash is accessed in Individual mode. 1b - Flash is accessed in Parallel mode.

25.7.2.6 Interrupt Enable (INTEN)

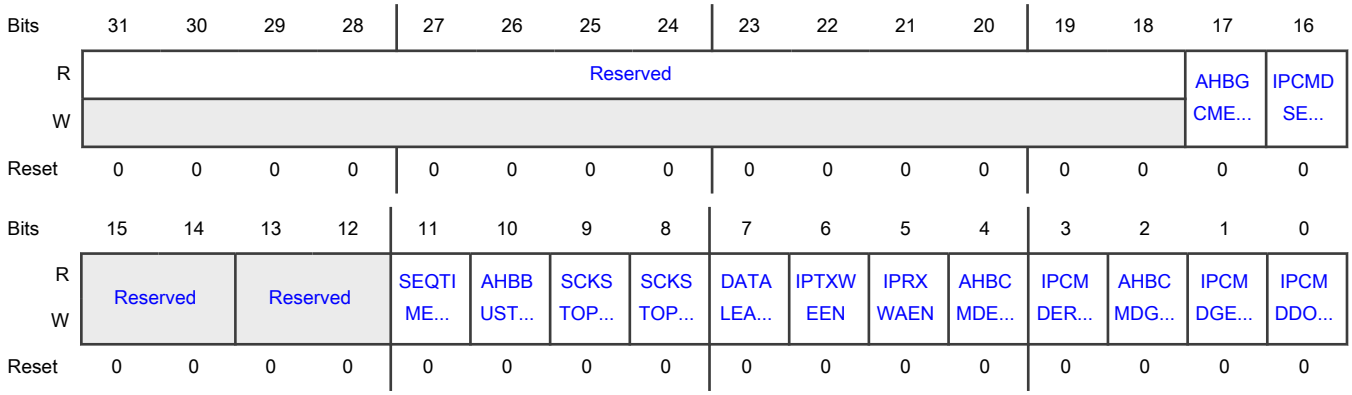
Offset

Register	Offset
INTEN	10h

Function

Includes AHB error response, IPS error response, and ECC error interrupt enables. See Interrupts.

Diagram



Fields

Field	Function
31-18 —	Reserved
17 AHBGCMERREN	AHB Read GCM Error Interrupt Enable 0b - Disable interrupt or no impact 1b - Enable interrupt
16 IPCMDSECUREVIOEN	IP Command Security Violation Interrupt Enable 0b - Disable interrupt or no impact 1b - Enable interrupt
15-14 —	Reserved
13-12 —	Reserved
11 SEQTIMEOUTEN	Sequence execution Timeout Interrupt Enable 0b - Disable interrupt or no impact 1b - Enable interrupt
10 AHBBUSTIMEOUTEN	AHB Bus Timeout Interrupt Enable 0b - Disable interrupt or no impact 1b - Enable interrupt
9 SCKSTOPBYWREN	SCLK Stopped By Write Interrupt Enable Enables interrupt that indicates SCLK is stopped during command sequence because asynchronous transmit FIFO is empty.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Disable interrupt or no impact 1b - Enable interrupt
8 SCKSTOPBYRDEN	SCLK Stopped By Read Interrupt Enable Enables interrupt that indicates SCLK is stopped during command sequence because asynchronous receive FIFO is full. 0b - Disable interrupt or no impact 1b - Enable interrupt
7 DATALEARNFAILEN	Data Learning Failed Interrupt Enable 0b - Disable interrupt or no impact 1b - Enable interrupt
6 IPTXWEEN	IP Transmit FIFO Watermark Empty Interrupt Enable Enables interrupt that indicates IP transmit FIFO contains more empty space than watermark level. 0b - Disable interrupt or no impact 1b - Enable interrupt
5 IPRXWAEN	IP Receive FIFO Watermark Available Interrupt Enable Enables interrupt that indicates IP receive FIFO contains more valid data than the watermark level. 0b - Disable interrupt or no impact 1b - Enable interrupt
4 AHBCMDERREN	AHB-Triggered Command Sequences Error Detected Interrupt Enable 0b - Disable interrupt or no impact 1b - Enable interrupt
3 IPCMDERREN	IP-Triggered Command Sequences Error Detected Interrupt Enable 0b - Disable interrupt or no impact 1b - Enable interrupt
2 AHBCMDGEEN	AHB-Triggered Command Sequences Grant Timeout Interrupt Enable. 0b - Disable interrupt or no impact 1b - Enable interrupt
1 IPCMDGEEN	IP-Triggered Command Sequences Grant Timeout Interrupt Enable 0b - Disable interrupt or no impact 1b - Enable interrupt
0	IP-Triggered Command Sequences Execution Finished Interrupt Enable

Table continues on the next page...

Table continued from the previous page...

Field	Function
IPCMDDONEEN	0b - Disable interrupt or no impact 1b - Enable interrupt

25.7.2.7 Interrupt (INTR)

Offset

Register	Offset
INTR	14h

Function

Includes AHB error response, IPS error response, and ECC error interrupts. See Interrupts.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved														AHBGCME...	IPCMDSE...
W															W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		Reserved		SEQTIME...	AHBBUST...	SCKSTOP...	SCKSTOP...	DATALEA...	IPTXWE	IPRXWA	AHBCMDE...	IPCMDERR	AHBCMDGE	IPCMDGE	IPCMDDO...
W					W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-18 —	Reserved
17 AHBGCMERR	AHB Read GCM Error **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
16 IPCMDSECUREVIO	IP Command Security Violation **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
15-14 —	Reserved
13-12 —	Reserved
11 SEQTIMEOUT	Sequence Execution Timeout **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
10 AHBBUSTIMEOUT	AHB Bus Timeout **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
9 SCKSTOPBYWR	SCLK Stopped Due To Empty Transmit FIFO Generated when SCLK is stopped during command sequence because the asynchronous transmit FIFO is empty. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
8 SCKSTOPBYRD	SCLK Stopped Due To Full Receive FIFO Generated when SCLK is stopped during command sequence because the asynchronous receive FIFO is full. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
7 DATALEARNFAIL	Data Learning Failed **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - No effect 1b - Clear the flag
6 IPTXWE	IP Transmit FIFO Watermark Empty Generated when the IP transmit FIFO contains more empty space than the watermark level. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
5 IPRXWA	IP Receive FIFO Watermark Available Generated when the IP receive FIFO contains more valid data than the watermark level. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
4 AHBCMDERR	AHB-Triggered Command Sequences Error Generated upon an AHB-triggered command sequence error. When an error is detected for an AHB command, the command is ignored and not executed. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - Clear the flag
3 IPCMDERR	IP-Triggered Command Sequences Error Generated upon an IP-triggered command sequence error. When an error is detected for an IP command, the command is ignored and not executed. **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
2 AHBCMDGE	AHB-Triggered Command Sequences Grant Timeout **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
1 IPCMDGE	IP-Triggered Command Sequences Grant Timeout **NOTE** This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag
0 IPCMDDONE	IP-Triggered Command Sequences Execution Finished Generated upon completion of an IP-triggered command sequence. Also generated when IPCMDGE or IPCMDERR interrupt is generated

Table continues on the next page...

Table continued from the previous page...

Field	Function
	NOTE This field behaves differently for register reads and writes. When reading 0b - Interrupt condition has not occurred 1b - Interrupt condition has occurred When writing 0b - No effect 1b - Clear the flag

25.7.2.8 LUT Key (LUTKEY)

Offset

Register	Offset
LUTKEY	18h

Function

Contains the key to lock and unlock LUT. See Lookup table (LUT).

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	KEY															
Reset	0	1	0	1	1	0	1	0	1	1	1	1	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	KEY															
Reset	0	1	0	1	1	0	1	0	1	1	1	1	0	0	0	0

Fields

Field	Function
31-0 KEY	LUT Key Contains key to lock or unlock LUT. The key is 5AF05AF0h. Read value is always 5AF05AF0h.

25.7.2.9 LUT Control (LUTCR)

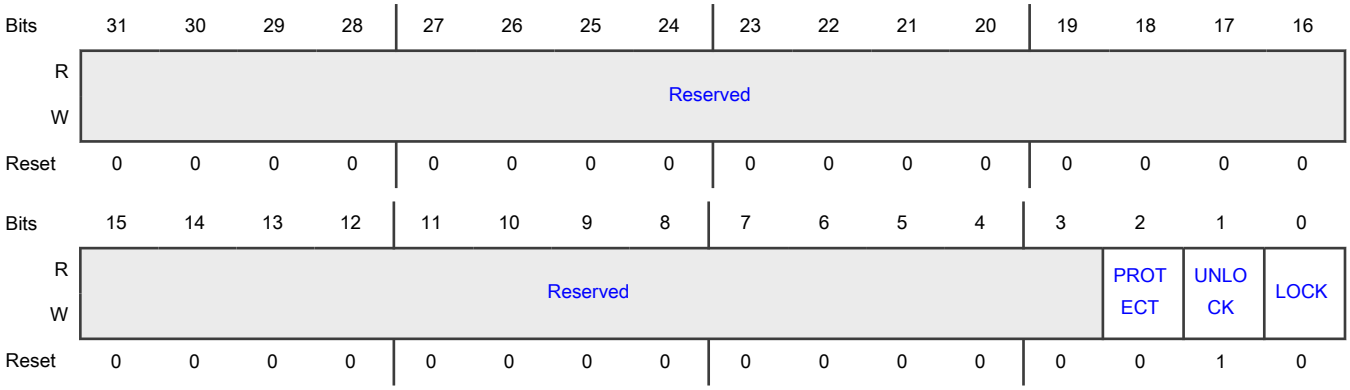
Offset

Register	Offset
LUTCR	1Ch

Function

Used with LUTKEY register to lock or unlock LUT. For the lock or unlock operation to be successful, this register must be written immediately after writing 5AF05AF0h to the LUTKEY register. See Lookup table (LUT) for details on locking and unlocking LUT. You cannot write 00 or 11 to the LOCK and UNLOCK fields.

Diagram



Fields

Field	Function
31-3 —	Reserved
2 PROTECT	LUT Protection Controls lookup table protection. 0b - Not protected. All IPS controllers can access LUTCR and LUT memory. 1b - Protected. Only secure IPS controller can change the value of LUTCR and write to LUT memory.
1 UNLOCK	Unlock LUT 0b - LUT is locked (LUTCR[LOCK] must be 1) 1b - LUT is unlocked and can be written
0 LOCK	Lock LUT 0b - LUT is unlocked (LUTCR[UNLOCK] must be 1) 1b - LUT is locked and cannot be written

25.7.2.10 AHB Receive Buffer n Control 0 (AHBRXBUF0CR0 - AHBRXBUF7CR0)

Offset

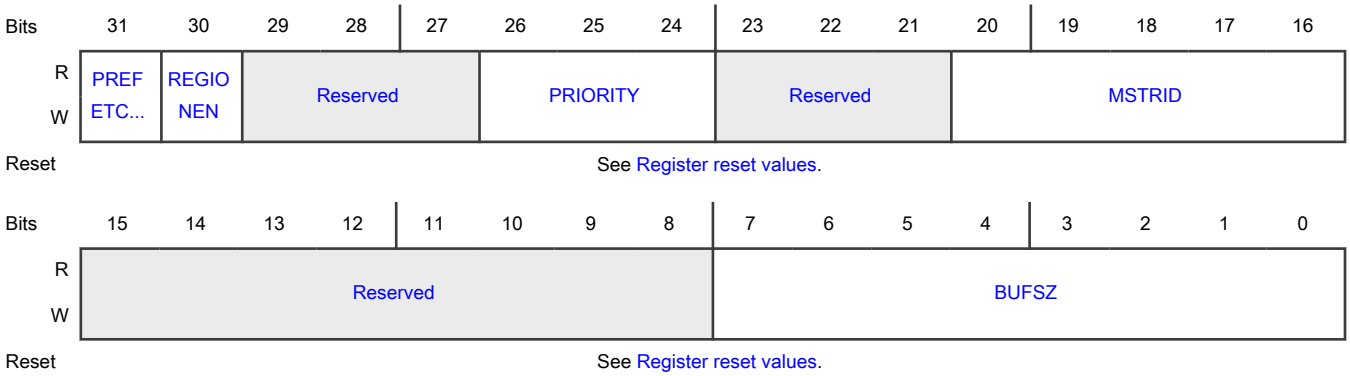
For n = 0 to 7:

Register	Offset
AHBRXBUFnCR0	20h + (n × 4h)

Function

Stores the read data from the SPI interface.

Diagram



Register reset values

Register	Reset value
AHBRXBUF0CR0	8000_0010h
AHBRXBUF1CR0	8001_0010h
AHBRXBUF2CR0	8002_0010h
AHBRXBUF3CR0	8003_0010h
AHBRXBUF4CR0	8004_0010h
AHBRXBUF5CR0	8005_0010h
AHBRXBUF6CR0	8006_0010h
AHBRXBUF7CR0	8007_0010h

Fields

Field	Function
31 PREFETCHEN	AHB Read Prefetch Enable

Table continues on the next page...

Table continued from the previous page...

Field	Function
	Enables AHB read prefetch for the controller corresponding to the current AHB receive buffer. The prefetch feature is disabled when AHBCR[PREFETCHEN] is 0. You can use this field to enable or disable prefetch separately for each controller. 0b - Disabled 1b - Enabled when AHBCR[PREFETCHEN] is enabled.
30 REGIONEN	AHB Receive Buffer Address Region Enable Enable prefetch buffer enhancement. See Prefetch buffer enhancement. **NOTE** Only AHBRXBUF 0-3 have this function. 0b - Disabled. The buffer hit is based on the value of MSTRID only. 1b - Enabled. The buffer hit is based on the value of MSTRID and the address within AHBBUFREGIONSTARTn and AHBREGIONENDn.
29-27 —	Reserved
26-24 PRIORITY	AHB Controller Read Priority Configure the priority for the AHB Controller Read to which this AHB receive buffer is assigned. 7 is the highest priority, 0 the lowest. See Command abort and suspend.
23-21 —	Reserved
20-16 MSTRID	AHB Controller ID Configures the ID of the AHB controller to which this AHB receive buffer is assigned. See AHB receive buffer management.
15-8 —	Reserved
7-0 BUFSZ	AHB Receive Buffer Size Configures the size of the AHB receive buffer in multiples of 64 bits. See AHB receive buffer management.

25.7.2.11 Flash Control 0 (FLSHA1CR0 - FLSHB2CR0)

Offset

Register	Offset
FLSHA1CR0	60h

Table continues on the next page...

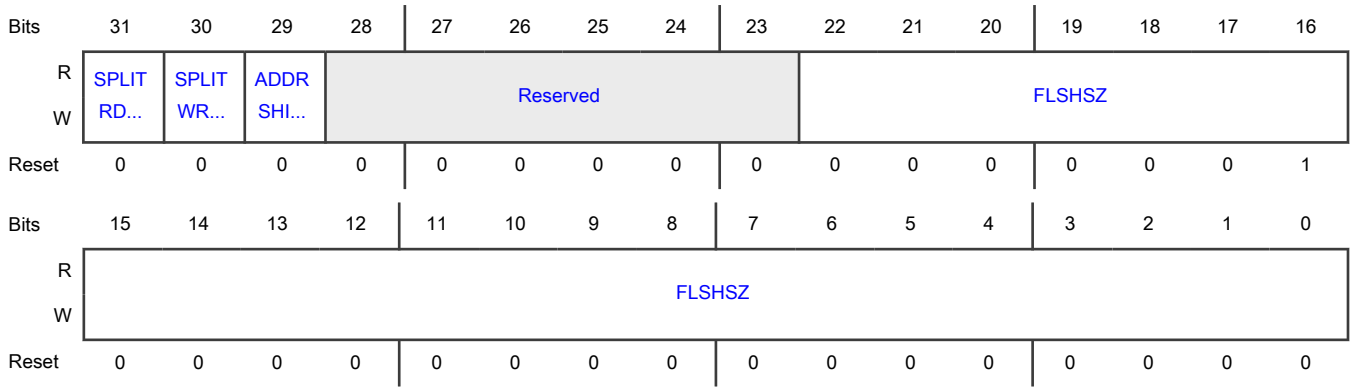
Table continued from the previous page...

Register	Offset
FLSHA2CR0	64h
FLSHB1CR0	68h
FLSHB2CR0	6Ch

Function

Contains flash memory size setting. FlexSPI determines which device is accessed (Chip Select) via this register.

Diagram



Fields

Field	Function
31 SPLITRDEN	AHB Read Access Split Function Enable When current access is inside of GCM region, used to control the split function enable and disable, based on which flash is accessed. See AHB transaction split. 0b - Disable 1b - Enable
30 SPLITWREN	AHB Write Access Split Function Enable When current access is inside of GCM region, used to control the split function enable and disable, based on which flash memory is accessed. See AHB transaction split 0b - Disable 1b - Enable
29 ADDRSHIFT	AHB Address Shift Function control Used to left-shift five bits for the flash address to support ISSI quad PSRAM.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Disabled 1b - Enabled
28-23 —	Reserved
22-0 FLSHSZ	Flash Size in KB Configures the maximum flash memory size. For AHB access: Maximum flash size supported for each device is 512 MB. When the value of this field is greater than the maximum size, the device flash size is interpreted as the maximum. The maximum flash size is the largest flash size supported for a single device. It is also the maximum total flash size supported for all devices (up to four). If the total flash size is larger than the maximum size, only the maximum flash size address space is accessible. For IPS access, The maximum flash size supported for each device is 4 GB. When the value of this field is greater than 400000h, the device flash size is taken as 4 GB. The max total flash size supported (for all 4 devices) is also 4 GB. If the total flash size is larger than 4 GB, only 4 GB of address space is accessible.

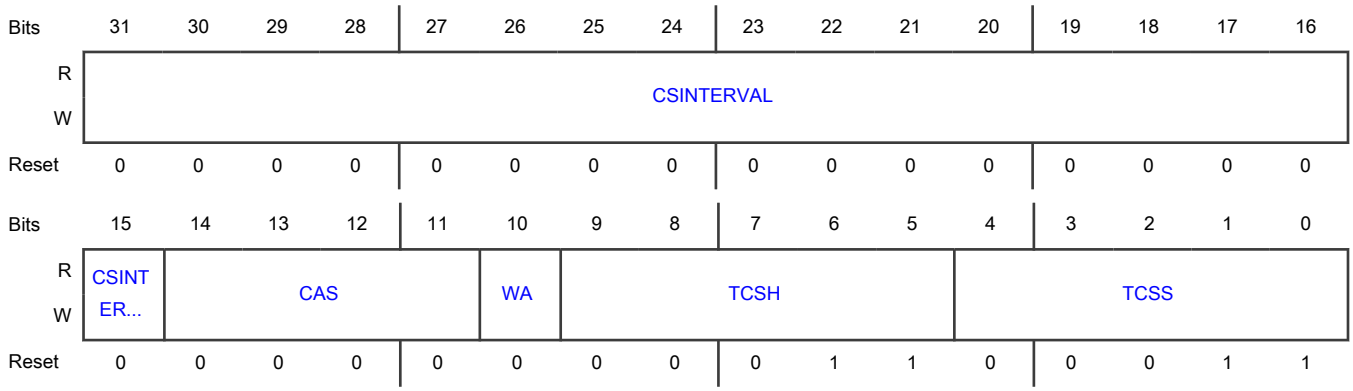
25.7.2.12 Flash Control 1 (FLSHA1CR1 - FLSHB2CR1)

Offset

Register	Offset
FLSHA1CR1	70h
FLSHA2CR1	74h
FLSHB1CR1	78h
FLSHB2CR1	7Ch

Function
Contains settings for flash device-specific timings and flash internal address space.

Diagram



Fields

Field	Function
31-16 CSINTERVAL	Chip Select Interval Configures the minimum interval between flash device chip select deassertion and chip select assertion. If the external flash device has a limitation on the interval between command sequences, configure this field accordingly. If there is no limitation, write 0h to this field. When CSINTERVALUNIT = 0, the chip select invalid interval is: CSINTERVAL × 1 serial clock cycle; When CSINTERVALUNIT = 1, the chip select invalid interval is: CSINTERVAL × 256 serial clock cycles. **NOTE** The minimum chip select interval is 2 cycles, even when the value of CSINTERVAL is less than 2.
15 CSINTERVALUNIT	Chip Select Interval Unit Configures the interval unit for chip select. 0b - 1 serial clock cycle 1b - 256 serial clock cycles
14-11 CAS	Column Address Size When external flash memory has a separate address field for rows and columns, this field configures the flash column address bit width. FlexSPI automatically splits a flash-mapped address into row address and column address according to the values of this field and the WA field. When the external flash memory does not support column address, write 0 to this field. FlexSPI transmits all flash address bits as Row address. For flash address mapping, see Flash address sent to flash memory devices.
10 WA	Word-Addressable Configures whether external flash memory is word-addressable or byte-addressable. If flash memory is word-addressable, it should be accessed in multiples of 16 bits. Currently, FlexSPI does not transmit flash address bit 0 to external flash memory. For flash address mapping, see Flash address sent to flash memory devices. 0b - Byte-addressable 1b - Word-addressable
9-5 TCSH	Serial Flash CS Hold Time Used to meet flash TCSH timing requirement. Serial flash CS Hold time that FlexSPI promises is: (TCSH + 1/2) serial clock cycles for DDR mode, and TCSH serial clock cycles for SDR mode. See Output timing between chip select and SCLK.
4-0 TCSS	Serial Flash CS Setup Time Used to meet flash TCSS timing requirement. Serial flash CS Setup time that FlexSPI promises is: (TCSS + 1/2) serial root clock cycles for SDR and DDR mode. See Output timing between chip select and SCLK.

25.7.2.13 Flash Control 2 (FLSHA1CR2 - FLSHB2CR2)

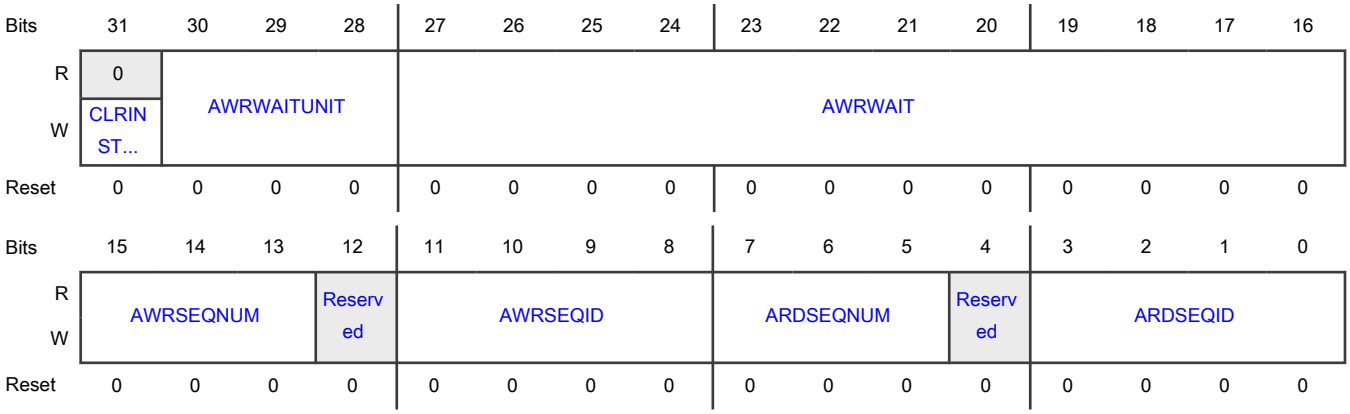
Offset

Register	Offset
FLSHA1CR2	80h
FLSHA2CR2	84h
FLSHB1CR2	88h
FLSHB2CR2	8Ch

Function

Contains fields to configure AHB bus access. If the four external devices are different types, AHB read and write commands may use different command sequences; AHB bus ready wait time may also differ.

Diagram



Fields

Field	Function
31 CLRINSTRPTR	Clear Instruction Pointer Clears the instruction pointer, which is the pointer that JMP_ON_CS saves internally. See Programmable sequence engine. This field is used for AHB read access to external flash memory supporting Execute-In-Place (XIP) mode.
30-28 AWRWAITUNIT	AWRWAIT Unit Configures the unit of AHB write wait time, as the value of AWRWAIT determines, in terms of AHB clock cycles. 000b - 2 001b - 8 010b - 32

Table continues on the next page...

Table continued from the previous page...

Field	Function
	011b - 128 100b - 512 101b - 2048 110b - 8192 111b - 32768
27-16 AWRWAIT	AHB Write Wait Configures AHB write wait time, with the AWRWAITUNIT field. Certain devices (such as FPGA) require time to write data into internal memory after the command sequences finished on the FlexSPI interface. If another read command sequence arrives before the current programming finishes internally, the read data may be wrong. This field is used to hold the AHB bus ready for AHB write access, waiting until the programming is finished in the external device. This hold ensures that no AHB read command is triggered before the programming finishes in the external device. The wait cycle between an AHB-triggered command sequence finishing on FlexSPI and the AHB return bus being ready is: AWRWAIT × AWRWAITUNIT.
15-13 AWRSEQNUM	Sequence Number for AHB Write-Triggered Command Configures the sequence number of an AHB read-triggered command. For certain flash devices (for example, HyperFlash, HyperRam, and Serial NAND flash), a flash programming access is done via several command sequences. An AHB write command triggers (AWRSEQNUM + 1) command sequences to external flash memory each time. FlexSPI executes the sequences in LUT incrementally. **NOTE** • Software should ensure that the last sequence index never exceeds LUT sequence numbers: AWRSEQID + AWRSEQNUM < 16 • Software must ensure that the AWRSEQNUM and LUT fields are configured correctly according to the external device specification. FlexSPI does not check the sequence; it executes the sequences one by one.
12 —	Reserved
11-8 AWRSEQID	Sequence Index for AHB Write-Triggered Command
7-5 ARDSEQNUM	Sequence Number for AHB Read-Triggered Command Configures the sequence number of an AHB read-triggered command in the lookup table. For certain flash devices (for example, HyperFlash, HyperRam, and Serial NAND flash), a flash read access is done via several command sequences. An AHB read command triggers (ARDSEQNUM + 1) command sequences to external flash memory each time. FlexSPI executes the sequences in LUT incrementally.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	NOTE • Software should ensure that the last sequence index never exceeds LUT sequence numbers: ARDSEQID + ARDSEQNUM ≤ 16. • Software must ensure that the ARDSEQNUM and LUT fields are configured correctly according to the external device specification. FlexSPI does not check the sequence; it executes the sequences one by one.
4 —	Reserved
3-0 ARDSEQID	Sequence Index for AHB Read-Triggered Command in LUT

25.7.2.14 Flash Control 4 (FLSHCR4)

Offset

Register	Offset
FLSHCR4	94h

Function
Provides configuration for all external devices.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved				Reserved				Reserved							
											Reserved	Reserved	WMENB	WMENA	Reserved	WMOPT1
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-12	Reserved

Table continues on the next page...

Table continued from the previous page...

Field	Function
—	
11-9 —	Reserved
8-6 —	Reserved
5 —	Reserved
4 —	Reserved
3 WMENB	Write Mask Enable for Port B Enables write mask for flash device on port B. 0b - Disabled. When writing to external device, DQS(RWDS) pin is not driven. 1b - Enabled. When writing to external device, FlexSPI drives DQS(RWDS) pin as write mask output.
2 WMENA	Write Mask Enable for Port A Enables write mask for flash device on port A. 0b - Disabled. When writing to external device, DQS(RWDS) pin is not driven. 1b - Enabled. When writing to external device, FlexSPI drives DQS(RWDS) pin as write mask output.
1 —	Reserved
0 WMOPT1	Write Mask Option 1 Used to remove AHB and IP write burst start address alignment limitation. 0b - When writing to an external device, DQS pin is used as write mask. When flash memory is accessed in individual mode, AHB or IP write burst start address alignment is not limited. 1b - When writing to an external device, DQS pin is not used as write mask. When flash memory is accessed in individual mode, AHB or IP write burst start address alignment is limited.

25.7.2.15 IP Control 0 (IPCR0)

Offset

Register	Offset
IPCR0	A0h

Function

Provides all configuration required for IP commands. Provides the start address for the flash device, instead of the chip address, to be accessed for IP command. FlexSPI determines the chip select automatically according to this start address.

NOTE

- You cannot issue an IP command that crosses flash device boundaries. If you do so, it generates an IPCMDERR interrupt.
- Configure this register before an IP command is triggered.
- Do not change the values in this register while an IP command is in progress.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SFAR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SFAR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 SFAR	Serial Flash Address Configures the serial flash address for IP commands. The address should be the address of the flash device without the base address.

25.7.2.16 IP Control 1 (IPCR1)

Offset

Register	Offset
IPCR1	A4h

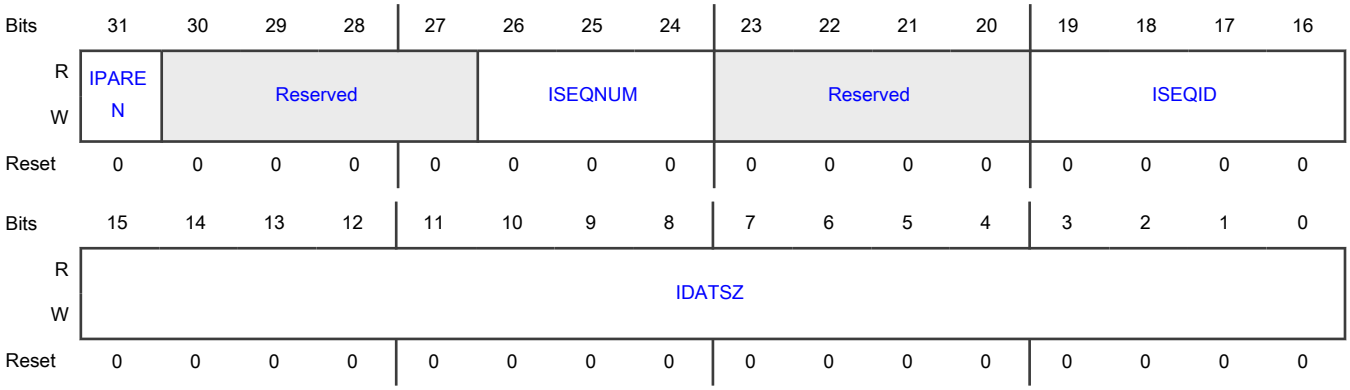
Function

Provides all configuration required for IP commands. Provides the flash read and program data size, sequence index in LUT, sequence number and individual/parallel mode settings for IP commands.

NOTE

- Configure this register before an IP command is triggered.
- Do not change the values in this register while an IP command is in progress.

Diagram



Fields

Field	Function
31 IPAREN	Parallel Mode Enable for IP Commands 0b - Disabled. Flash memory is accessed in Individual mode. 1b - Enabled. Flash memory is accessed in Parallel mode.
30-27 —	Reserved
26-24 ISEQNUM	Sequence Number for IP command: ISEQNUM+1.
23-20 —	Reserved
19-16 ISEQID	Sequence Index in LUT for IP command.
15-0 IDATSZ	Flash Read/Program Data Size (in bytes) for IP command.

25.7.2.17 IP Control 2 (IPCR2)

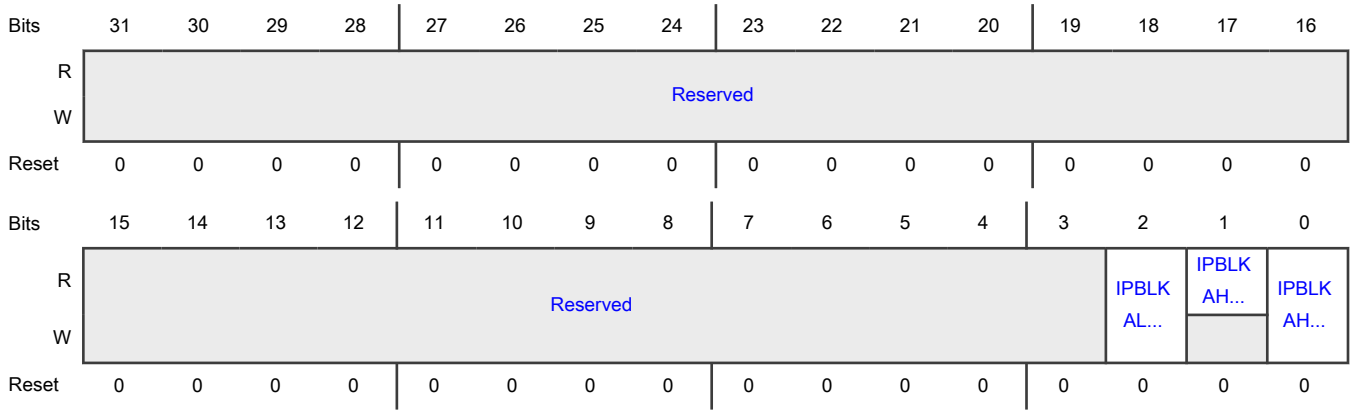
Offset

Register	Offset
IPCR2	A8h

Function

Configures whether IP commands can block AHB commands.

Diagram



Fields

Field	Function
31-3 —	Reserved
2 IPBLKALLAHB	IP Command Blocking All AHB Command Enable 0b - IP commands only block AHB commands that affect the IPED region. 1b - IP commands block all AHB commands.
1 IPBLKAHBACK	IP Command Blocking AHB Command Acknowledgment Enable 0b - IP commands do not block AHB command acknowledgment. 1b - IP commands block AHB command acknowledgment.
0 IPBLKAHBREQ	IP Command Blocking AHB Command Request Enable 0b - IP commands do not block AHB command requests. 1b - IP commands block AHB command requests.

25.7.2.18 IP Command (IPCMD)

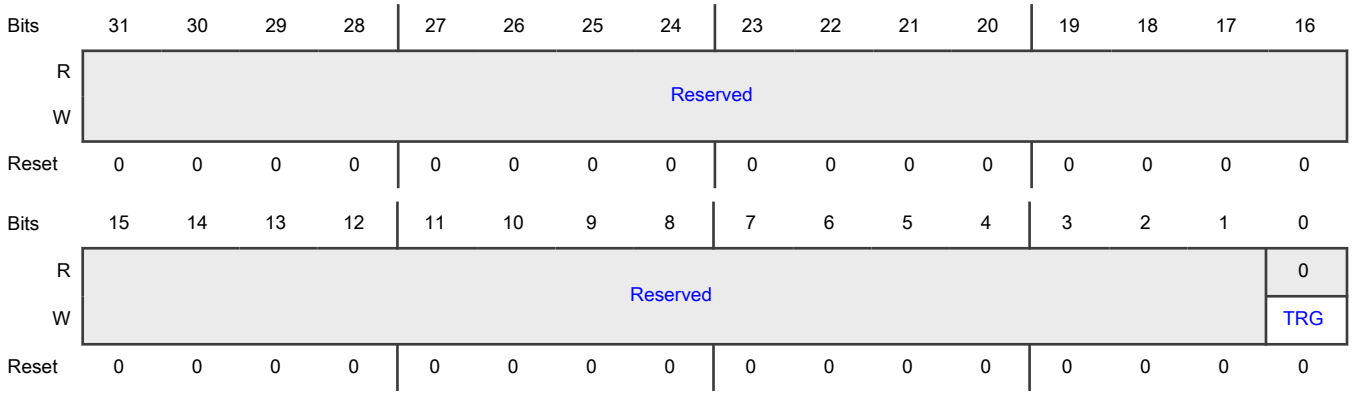
Offset

Register	Offset
IPCMD	B0h

Function

Used to trigger an IP command to access external flash device. When the arbitrator grants the IP command, the command is executed on the FlexSPI interface.

Diagram



Fields

Field	Function
31-1 —	Reserved
0 TRG	Command Trigger Triggers an IP command. This field clears automatically after it has been written and always reads as 0. **NOTE** You cannot trigger another IP command before the previous IP command finishes on the FlexSPI interface. Software must poll INTR[IPCMDDONE] or wait for this interrupt. 0b - No action 1b - Start the IP command that the IPCR0 and IPCR1 registers define.

25.7.2.19 Data Learning Pattern (DLPR)

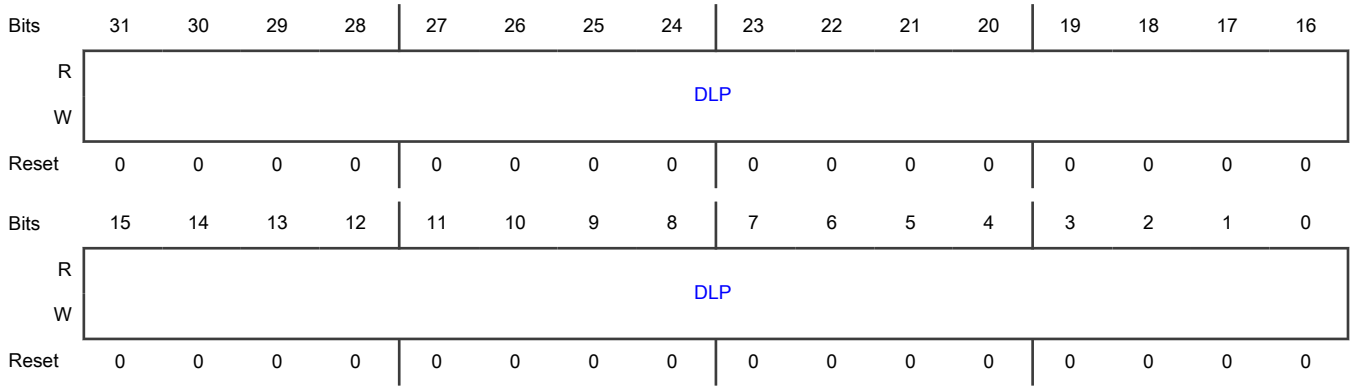
Offset

Register	Offset
DLPR	B4h

Function

Provides the pattern to be used during Data Learning in FlexSPI.

Diagram



Fields

Field	Function
31-0 DLP	Data Learning Pattern The operand in the LEARN_SDR or LEARN_DDR instruction code determines the data learning pattern bit number. This number never exceeds 32 bits. If the operand in the instruction code is greater than 32, a 32-bit pattern is used. See Data learning.

25.7.2.20 IP Receive FIFO Control (IPRXFCR)

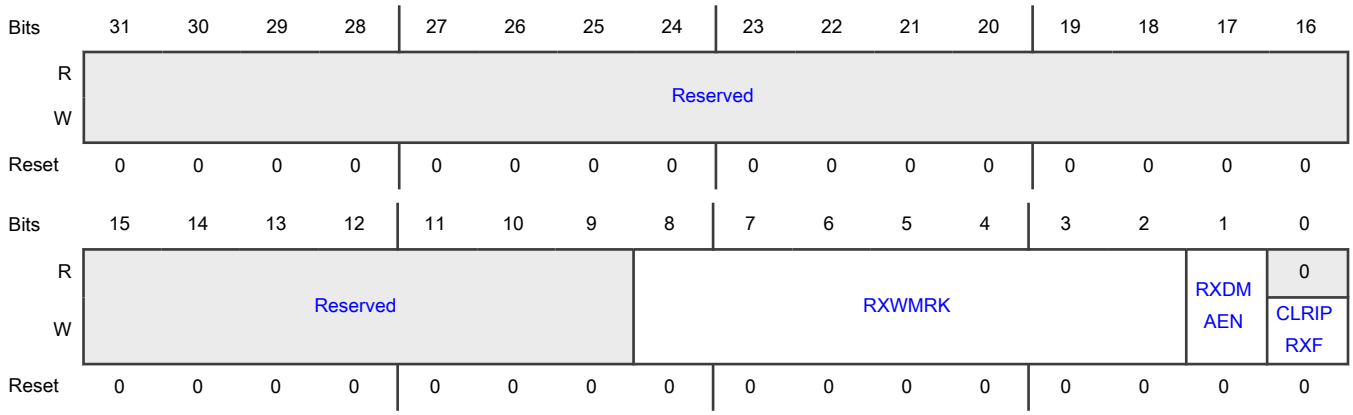
Offset

Register	Offset
IPRXFCR	B8h

Function

Provides the configuration fields for IP receive FIFO management.

Diagram



Fields

Field	Function
31-9 —	Reserved
8-2 RXWMRK	IP Receive FIFO Watermark Level Configures the watermark level for the IP receive FIFO. The watermark level is (RXWMRK + 1) × 64 bits. The INTR[IPRXWA] interrupt is set when FlexSPI fills the IP receive FIFO ≥ the watermark level. A DMA request occurs when the fill level is ≥ the watermark level and IPRXFCR[RXDMAEN] = 1. When the fill level is ≥ the watermark level and INTEN[IPRXWAEN] = 1, an INTR[IPRXWA] interrupt is generated. **NOTE** After writing 1 to INTR[IPRXWA] to clear it, the read address should be rolled back to the start address (memory mapped). If the IP bus reads the IP receive FIFO, the read address should roll back to RFDR0. If the AHB bus reads the IP receive FIFO, the read address should roll back to ARDF_BASE.
1 RXDMAEN	IP Receive FIFO Reading by DMA Enable 0b - Disabled. The processor reads the FIFO. 1b - Enabled. DMA reads the FIFO.
0 CLRIPRXF	Clear IP Receive FIFO Clears all valid data entries in IP receive FIFO. Resets the read and write pointers in IP receive FIFO. 0b - No function 1b - A clock cycle pulse clears all valid data entries in IP receive FIFO.

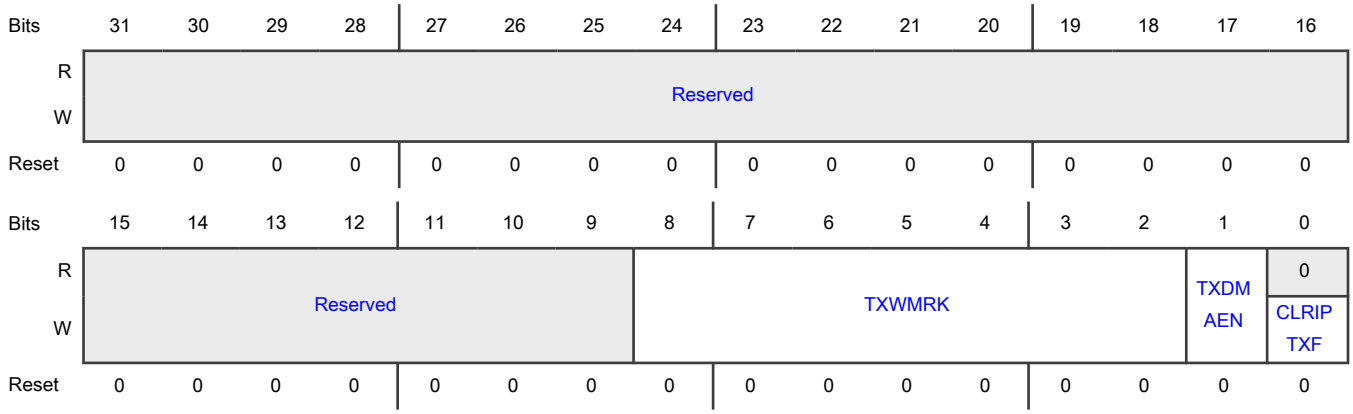
25.7.2.21 IP Transmit FIFO Control (IPTXFCR)

Offset

Register	Offset
IPTXFCR	BCh

Function
Provides the configuration fields for IP transmit FIFO management.

Diagram



Fields

Field	Function
31-9 —	Reserved
8-2 TXWMRK	Transmit Watermark Level Sets the transmit watermark level. The watermark level is (TXWMRK + 1) × 64 bits. INTR[IPTXWE] is set when FlexSPI empties the IP transmit FIFO ≥ the watermark level. When the empty level ≥ the watermark level and IPTXFCR[TXDMAEN] = 1, a DMA request occurs. When the empty level ≥ the watermark level and INTEN[IPTXWEEN] = 1, an INTR[IPTXWE] interrupt is generated. **NOTE** • The watermark level should not be larger than the write window. • The watermark level should not be larger than the IP transmit FIFO size. • The write address to the IP receive FIFO should roll back to the start address of the write window. After pushing the IP transmit fifo, you should write 1 to INTR[IPTXWE] to clear it, which will perform the rollback.
1 TXDMAEN	Transmit FIFO DMA Enable Selects whether DMA or processor fills IP transmit FIFO. 0b - Processor 1b - DMA
0 CLRIPTXF	Clear IP Transmit FIFO Clears all valid data entries in IP transmit FIFO. The read and write pointers in IP transmit FIFO are reset. 0b - No function 1b - A clock cycle pulse clears all valid data entries in the IP transmit FIFO.

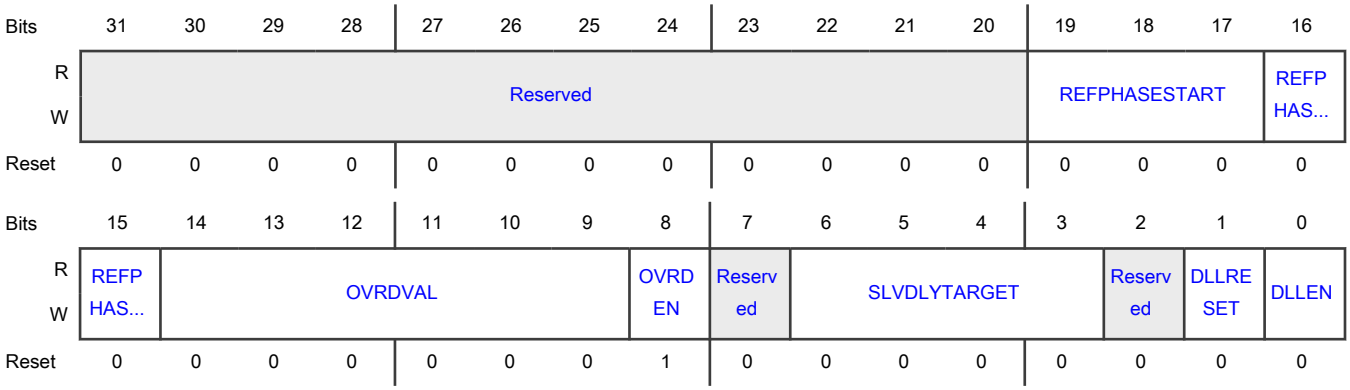
25.7.2.22 DLL Control 0 (DLLACR - DLLBCR)

Offset

Register	Offset
DLLACR	C0h
DLLBCR	C4h

Function
Configures Flash A and B sample clock DLL.

Diagram



Fields

Field	Function
31-20 —	Reserved
19-17 REFPHASESTART	Reference Clock Delay Line Start Phase If the DLL does not lock well when DLLxCR[DLLEN] is enabled, update the register value of REFPHASESTART. 7h is recommended value. Note that DLLxCR[DLLRESET] is required to be triggered after REFPHASESTART is updated.
16-15 REFPHASEGAP	Reference Clock Delay Line Phase Adjust Gap. REFPHASEGAP setting of 2h is recommended if DLLEN is set.
14-9 OVRDVAL	Target Clock Delay Line Override Value Configures the override value for the target clock line delay cell number. When OVRDEN = 1, the delay cell number in DLL is OVRDVAL + 1. See DLL configuration for sampling.
8 OVRDEN	Target Clock Delay Line Override Value Enable Enables the override value for the target clock line delay cell number.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Disable 1b - Enable
7 —	Reserved
6-3 SLVDLYTARGET	Target Delay Line Configures the target delay line for the target. The delay target for target delay line is: ((SLVDLYTARGET+1) × 1/32 × clock cycle of reference clock (serial root clock). If serial root clock is ≥ 100 MHz, DLLEN is set to 1, OVRDEN is set to 0, then SLVDLYTARGET setting is up to flash parameters used.
2 —	Reserved
1 DLLRESET	DLL reset Forces a DLL reset. The forced reset causes the DLL to lose lock and recalibrate to detect a ref_clock half-period phase shift. The reset action is edge-triggered, so software must write 0 to this field after writing 1 to it (no delay limitation). 0b - No function 1b - Force DLL reset.
0 DLLEN	DLL Calibration Enable Enables DLL calibration. When DLL calibration is disabled, the delay cell number in the target delay line is always 1. When DLLxCR[OVRDEN] = 1, the target delay line is overridden and this field is ignored. 0b - Disable 1b - Enable

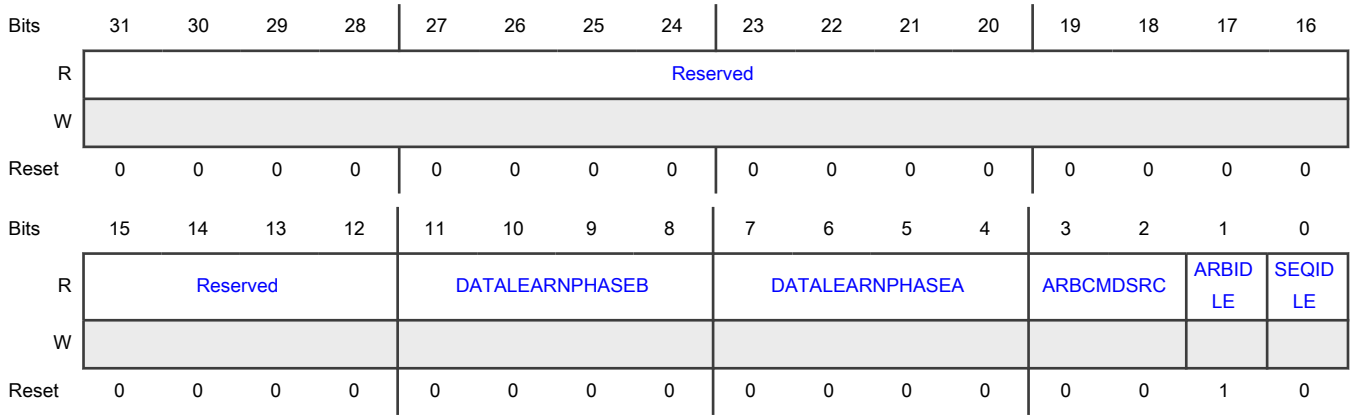
25.7.2.23 Status 0 (STS0)

Offset

Register	Offset
STS0	E0h

Function
Indicates the status of the internal state machine.

Diagram



Fields

Field	Function
31-12 —	Reserved
11-8 DATALEARNPHASEB	Data Learning Phase Selection on Port B Indicates the sampling clock phase selection on Port B after data learning. This field is similar to STS0[DATALEARNPHASEA]. The sampling clock phase is chosen separately for Port A and Port B.
7-4 DATALEARNPHASEA	Data Learning Phase Selection on Port A Indicates the sampling clock phase selection on Port A after data learning. There are 16 clock phases for the sampling clock that the delay cell line generates. When data learning is not enabled, the default clock phase 0 is used to sample flash read data. When data learning is enabled and a LEARN_SDR or LEARN_DDR instruction has executed correctly, FlexSPI determines the correct clock phase to sample read data. See Data learning.
3-2 ARBCMDSRC	ARB Command Source Indicates the trigger source of current command sequence that the arbitrator has granted. When ARB_CTL is not busy (STS0[ARBIDLE] = 1), the value of this field does not matter. 00b - Trigger source is AHB read command. 01b - Trigger source is AHB write command. 10b - Trigger source is IP command (by writing 1 to IPCMD[TRG]). 11b - Trigger source is a suspended command that has resumed.
1 ARBIDLE	ARB_CTL State Machine Idle Indicates whether the state machine in ARB_CTL is idle, or a command sequence that the arbitrator granted has not finished on the FlexSPI interface. When idle, no transaction is occurring on the FlexSPI interface (STS0[SEQIDLE] = 1). When waiting for the FlexSPI controller to become idle, poll this field instead of STS0[SEQIDLE]. 0b - Not idle

Table continues on the next page...

Table continued from the previous page...

Field	Function
	1b - Idle
0 SEQIDLE	SEQ_CTL State Machine Idle Indicates whether the state machine in SEQ_CTL is idle, or a command sequence is executing on the FlexSPI interface. 0b - Not idle 1b - Idle

25.7.2.24 Status 1 (STS1)

Offset

Register	Offset
STS1	E4h

Function
Indicates the error code of the AHB or IPS interface error response.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved				IPCMDERRCODE				Reserved				IPCMDERRID			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved				AHBCMDERRCODE				Reserved				AHBCMDERRID			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-28 —	Reserved
27-24 IPCMDERRCODE	IP Command Error Code When an IP command Error is detected, indicates the error code. This field returns to zero when INTR[IPCMDERR] is cleared.

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0000b - No error 0010b - IP command with JMP_ON_CS instruction used in the sequence 0011b - Unknown instruction opcode in the sequence 0100b - DUMMY_SDR or DUMMY_RWDS_SDR instruction used in DDR sequence 0101b - DUMMY_DDR or DUMMY_RWDS_DDR instruction used in SDR sequence 0110b - Flash memory access start address exceeds entire flash address range (A1, A2, B1, and B2) 1110b - Sequence execution timeout 1111b - Flash boundary crossed
23-20 —	Reserved
19-16 IPCMDERRID	IP Command Error ID When an IP command error is detected, indicates the sequence index. This field returns to zero when INTR[IPCMDERR] is cleared.
15-12 —	Reserved
11-8 AHBCMDERRCODE	AHB Command Error Code Contains the error code when an AHB command error is detected. This field returns to zero when INTR[AHBCMDERR] is cleared. 0000b - No error 0010b - AHB Write command with JMP_ON_CS instruction used in the sequence 0011b - Unknown instruction opcode in the sequence 0100b - DUMMY_SDR or DUMMY_RWDS_SDR instruction used in DDR sequence 0101b - DUMMY_DDR or DUMMY_RWDS_DDR instruction used in SDR sequence 1110b - Sequence execution timeout
7-4 —	Reserved
3-0 AHBCMDERRID	AHB Command Error ID When an AHB command error is detected, indicates the sequence index. This field returns to zero when INTR[AHBCMDERR] is cleared.

25.7.2.25 Status 2 (STS2)

Offset

Register	Offset
STS2	E8h

Function
Indicates the status of Flash A and B sample clock DLLs.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved		BREFSEL						BSLVSEL						BREFLOCK	BSLVLOCK
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved		AREFSEL						ASLVSEL						AREFLOCK	ASLVLOCK
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Fields

Field	Function
31-30 —	Reserved
29-24 BREFSEL	Flash B Sample Clock Reference Delay Line Delay Cell Number Indicates the sample clock reference delay line delay cell number for Flash B. There are 0-63 phases. • 000001b - Indicates lock phase 0. • 100000b - Indicates lock phase 63.
23-18 BSLVSEL	Flash B Sample Clock Target Delay Line Delay Cell Number Indicates the sample clock target delay line delay cell number for Flash B. There are 0-63 phases. • 000001b - Indicates lock phase 0. • 100000b - Indicates lock phase 63.
17 BREFLOCK	Flash B Sample Clock Reference Delay Line Locked 0b - Not locked 1b - Locked

Table continues on the next page...

Table continued from the previous page...

Field	Function
16 BSLVLOCK	Flash B Sample Target Reference Delay Line Locked 0b - Not locked 1b - Locked
15-14 —	Reserved
13-8 AREFSEL	Flash A Sample Clock Reference Delay Line Delay Cell Number Indicates the sample clock reference delay line delay cell number for Flash A. There are 0-63 phases. • 000001b - Indicates lock phase 0. • 100000b - Indicates lock phase 63.
7-2 ASLVSEL	Flash A Sample Clock Target Delay Line Delay Cell Number Indicates the sample clock target delay line delay cell number for Flash A. There are 0-63 phases. • 000001b - Indicates lock phase 0. • 100000b - Indicates lock phase 63.
1 AREFLOCK	Flash A Sample Clock Reference Delay Line Locked 0b - Not locked 1b - Locked
0 ASLVLOCK	Flash A Sample Target Delay Line Locked 0b - Not locked 1b - Locked

25.7.2.26 AHB Suspend Status (AHBSPNDSTS)

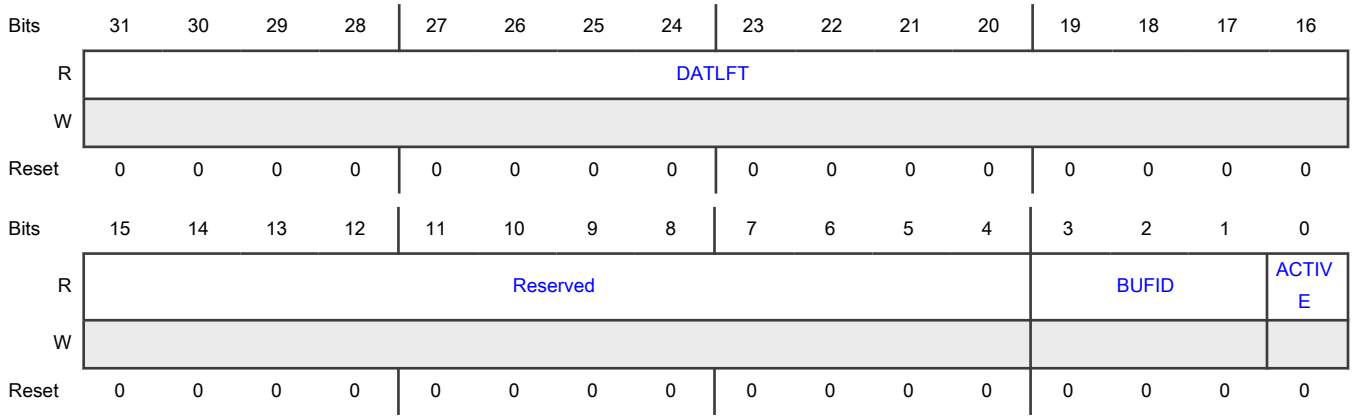
Offset

Register	Offset
AHBSPNDSTS	ECh

Function

Indicates the status of suspended AHB read prefetch command sequence. When an IP or AHB command is triggered while the arbitrator processes an AHB read sequence (prefetching additional data not for current AHB burst), the prefetch sequence is suspended. When there are no longer transactions on FlexSPI, the prefetch sequence may be resumed. FlexSPI saves only one AHB prefetch sequence. When a new prefetch sequence is suspended with an active sequence already suspended, the previous suspended sequence is removed and never resumed. See Command abort and suspend.

Diagram



Fields

Field	Function
31-16 DATLFT	Data Left Contains the data size remaining (in bytes) for a suspended command sequence.
15-4 —	Reserved
3-1 BUFID	AHB Receive Buffer ID for Suspended Command Sequence
0 ACTIVE	Active AHB Read Prefetch Suspended Indicates whether an AHB read prefetch command sequence has been suspended. 0b - No suspended AHB read prefetch command. 1b - An AHB read prefetch command sequence has been suspended.

25.7.2.27 IP Receive FIFO Status (IPRXFSTS)

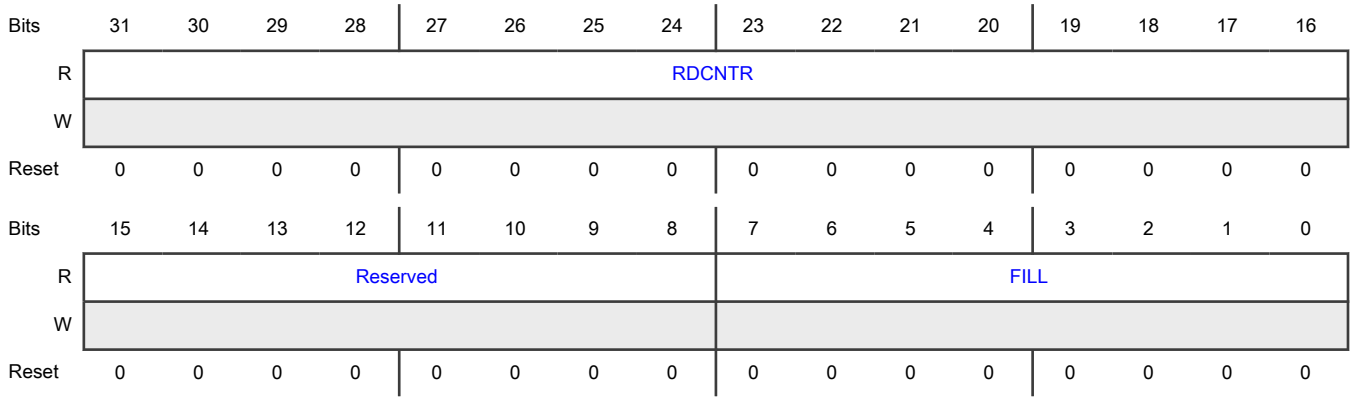
Offset

Register	Offset
IPRXFSTS	F0h

Function

Indicates the status of IP receive FIFO.

Diagram



Fields

Field	Function
31-16 RDCNTR	Read Data Counter Contains counter for total data read. The data read is the value of this field × 64 bits.
15-8 —	Reserved
7-0 FILL	Fill Level of IP Receive FIFO Indicates how full the IP receive FIFO is. The fill level is the value of this field × 64 bits.

25.7.2.28 IP Transmit FIFO Status (IPTXFSTS)

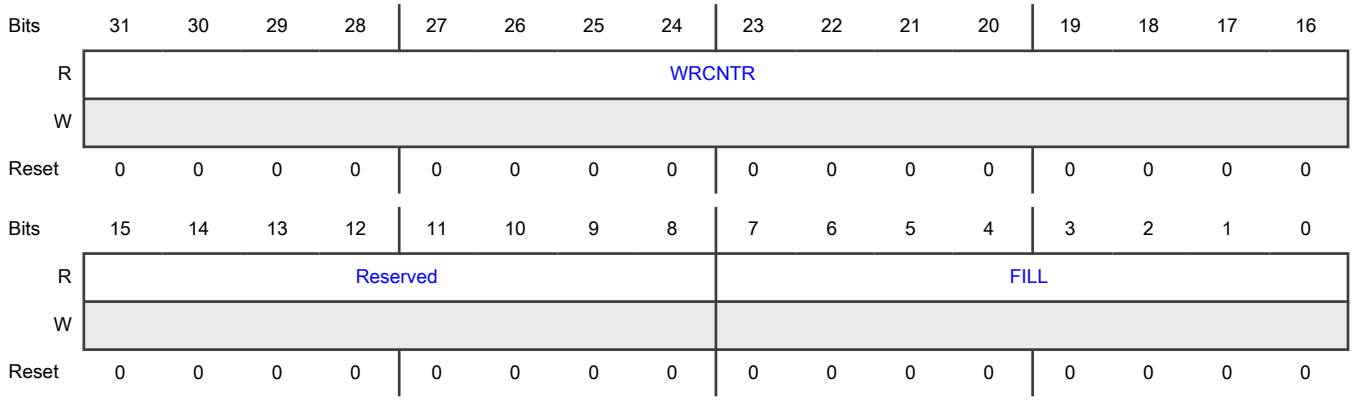
Offset

Register	Offset
IPTXFSTS	F4h

Function

Indicates the status of IP transmit FIFO.

Diagram



Fields

Field	Function
31-16 WRCNTR	Write Data Counter Contains counter for total data written. The data written is the value of this field × 64 bits.
15-8 —	Reserved
7-0 FILL	Fill Level of IP Transmit FIFO Indicates how full the IP transmit FIFO is. The fill level is the value of this field × 64 bits.

25.7.2.29 IP Receive FIFO Data a (RFDR0 - RFDR31)

Offset

For a = 0 to 31:

Register	Offset
RFDRa	100h + (a × 4h)

Function

Provides read access to IP receive FIFO via IPS bus. The read value is unknown for read access to invalid entries in the IP receive FIFO.

NOTE

RFDR is implemented as memory, so the reset value is unknown.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RXDATA															
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RXDATA															
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Fields

Field	Function
31-0 RXDATA	Receive Data Contains receive data. See Reading data from IP receive FIFO.

25.7.2.30 IP TX FIFO Data a (TFDR0 - TFDR31)

Offset

For a = 0 to 31:

Register	Offset
TFDRa	180h + (a × 4h)

Function

Provides write access to IP transmit FIFO via IPS bus.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 TXDATA	Transmit Data Contains transmit data. See Writing data to IP transmit FIFO.

25.7.2.31 Lookup Table a (LUT0 - LUT63)

Offset

For a = 0 to 63:

Register	Offset
LUTa	200h + (a × 4h)

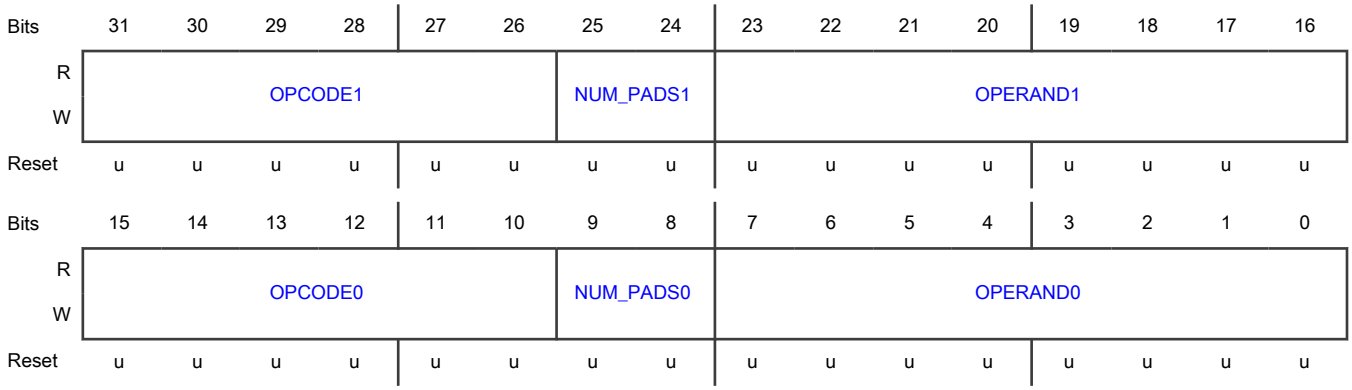
Function

Contains a lookup table for command sequences. Software should set the sequence index before triggering an IP command or AHB command. FlexSPI fetches the command sequence from LUT when an IP or AHB command is triggered. There are 16 command sequences in LUT. See Lookup table (LUT) and Programmable sequence engine.

NOTE

LUT is implemented as memory, so the reset value is unknown.

Diagram



Fields

Field	Function
31-26 OPCODE1	OPCODE1
25-24 NUM_PADS1	NUM_PADS1

Table continues on the next page...

Table continued from the previous page...

Field	Function
23-16 OPERAND1	OPERAND1
15-10 OPCODE0	OPCODE
9-8 NUM_PADS0	NUM_PADS0
7-0 OPERAND0	OPERAND0

25.7.2.32 HADDR REMAP Start Address (HADDRSTART)

Offset

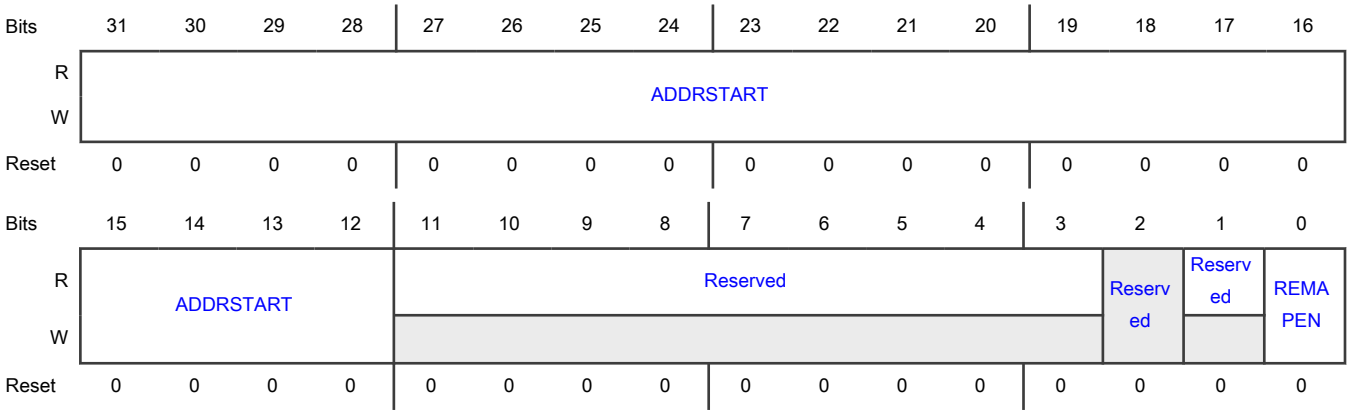
Register	Offset
HADDRSTART	420h

Function

Used to configure options for the dual-image remap feature.

See Dual image use case using HADDRSTART, HADDREND, and HADDROFFSET registers.

Diagram



Fields

Field	Function
31-12	HADDR Start Address

Table continues on the next page...

Table continued from the previous page...

Field	Function
ADDRSTART	Contains the start address of the HADDR remap range, aligned to 4 KB.
11-3 —	Reserved
2 —	Reserved
1 —	Reserved
0 REMAPEN	AHB Bus Address Remap Enable 0b - HADDR REMAP Disabled 1b - HADDR REMAP Enabled

25.7.2.33 HADDR REMAP END ADDR (HADDREND)

Offset

Register	Offset
HADDREND	424h

Function

Used to configure options for the dual-image remap feature.

See Dual image use case using HADDRSTART, HADDREND, and HADDROFFSET registers.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ENDSTART															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ENDSTART				Reserved											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-12 ENDSTART	End Address of HADDR Remap Range Contains the end address of the HADDR remap range, aligned to 4 KB.
11-0 —	Reserved

25.7.2.34 HADDR Remap Offset (HADDROFFSET)

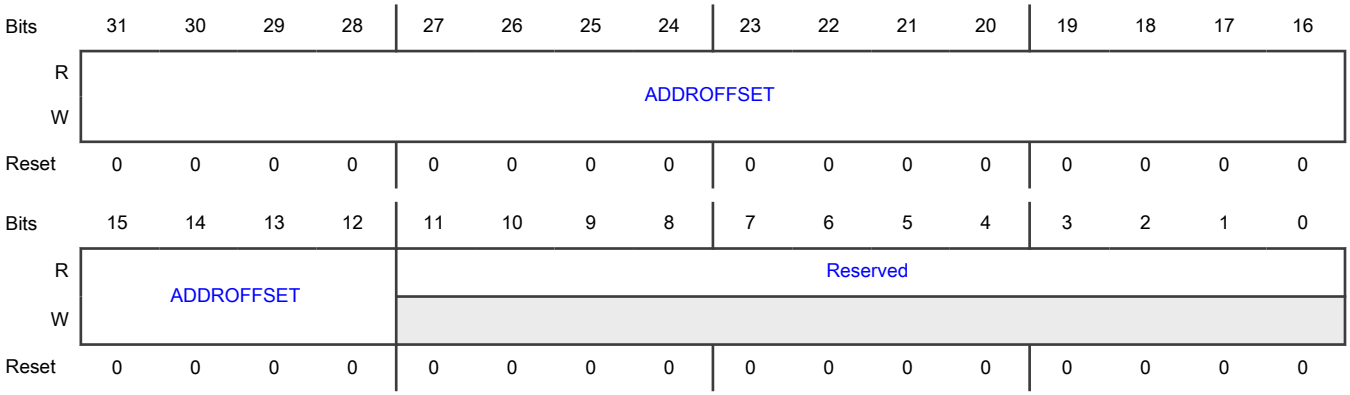
Offset

Register	Offset
HADDROFFSET	428h

Function
Used to configure options for the dual-image remap feature.

See Dual image use case using HADDRSTART, HADDREND, and HADDROFFSET registers.

Diagram



Fields

Field	Function
31-12 ADDROFFSET	HADDR Offset Contains the offset for HADDR. The remapped address is ADDR[31:12] = ADDR_original[31:12] + ADDROFFSET.
11-0 —	Reserved

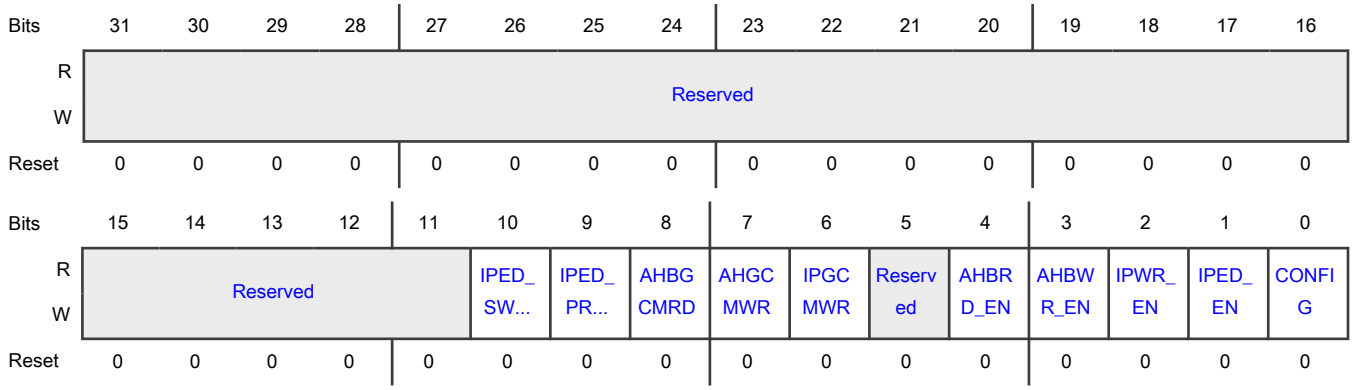
25.7.2.35 IPED Function Control (IPEDCTRL)

Offset

Register	Offset
IPEDCTRL	42Ch

Function
Selects IPED mode for encryption and decryption.

Diagram



Fields

Field	Function
31-11 —	Reserved
10 IPED_SWRESET	Abort Current Decryption or Encryption 0b - No function. 1b - Aborts current decryption or encryption and waits for the next start operation.
9 IPED_PROTECT	IPED Protection Can be used to limit access to IPED configuration registers. 0b - No restrictions 1b - Only privileged controllers can write IPED registers.
8 AHBGCMRD	AHB Read IPED GCM Mode Decryption Enable 0b - Disable 1b - Enable
7 AHGCMWR	AHB Write IPED GCM Mode Encryption Enable

Table continues on the next page...

Table continued from the previous page...

Field	Function
	0b - Disable 1b - Enable
6 IPGCMWR	IP Write GCM Mode Enable 0b - Disabled 1b - Enabled
5 —	Reserved
4 AHBRD_EN	AHB Read IPED CTR Mode Decryption Enable 0b - Disable 1b - Enable
3 AHBWR_EN	AHB Write IPED CTR Mode Encryption Enable. 0b - Disable 1b - Enable
2 IPWR_EN	IP Write IPED CTR Mode Encryption Enable 0b - Disable 1b - Enable
1 IPED_EN	IPED Encryption and Decryption Enable 0b - Disable 1b - Enable
0 CONFIG	IPED Mode Select Configures whether encryption and decryption are fully pipelined. 0b - Fully pipelined 1b - Not fully pipelined

25.7.2.36 Receive Buffer Start Address of Region a (AHBBUFREGIONSTART0 - AHBBUFREGIONSTART3)

Offset

Register	Offset
AHBBUFREGIONSTART0	440h
AHBBUFREGIONSTART1	448h

Table continues on the next page...

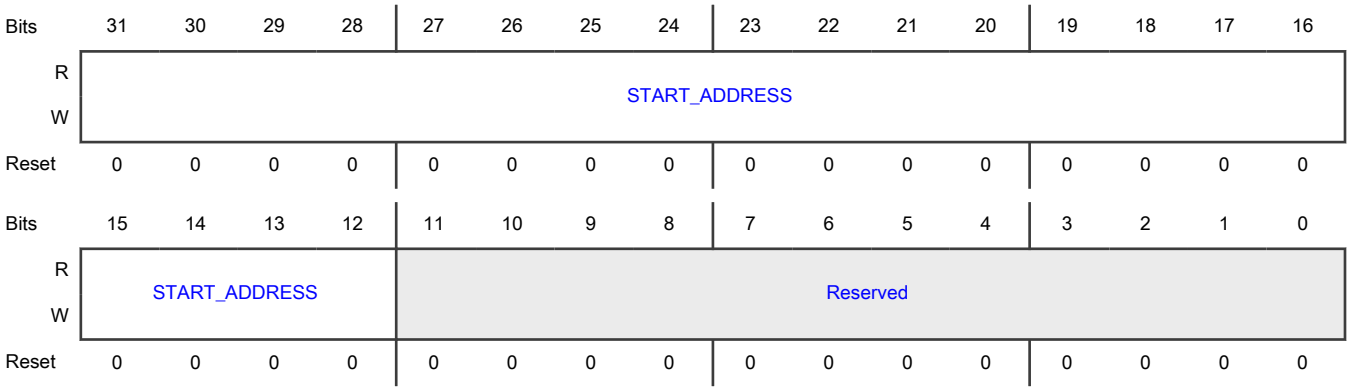
Table continued from the previous page...

Register	Offset
AHBBUFREGIONSTART2	450h
AHBBUFREGIONSTART3	458h

Function

Indicates the start address of AHB receive buffer address region.

Diagram



Fields

Field	Function
31-12 START_ADDRESS	Start Address of Prefetch Sub-Buffer Region Indicates start address. The start address must be at least four-KB aligned. The address used here is the AHB address that is compared with the system bus address to determine if an AHB read hits in the region. See Prefetch buffer enhancement.
11-0 —	Reserved

25.7.2.37 Receive Buffer Region a End Address (AHBBUFREGIONEND0 - AHBBUFREGIONEND3)

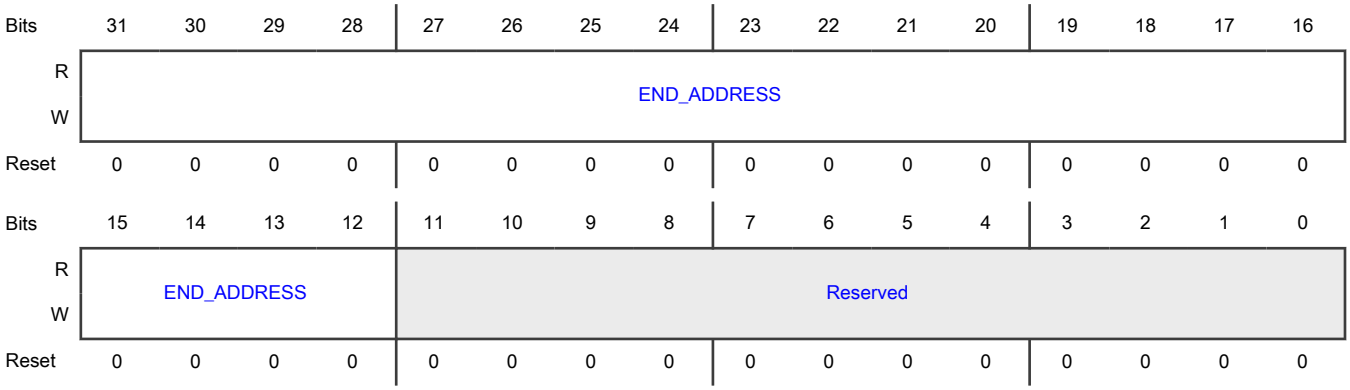
Offset

Register	Offset
AHBBUFREGIONEND0	444h
AHBBUFREGIONEND1	44Ch
AHBBUFREGIONEND2	454h
AHBBUFREGIONEND3	45Ch

Function

Indicates the end address of AHB receive buffer address region.

Diagram



Fields

Field	Function
31-12 END_ADDRESS	End Address of Prefetch Sub-Buffer Region Indicates end address. The end address must be at least four-KB aligned. The address used here is the AHB address that is compared with the system bus address to determine if an AHB read hits in the region. See Prefetch buffer enhancement.
11-0 —	Reserved

25.7.2.38 IPED context control 0 (IPEDCTXCTRL0)

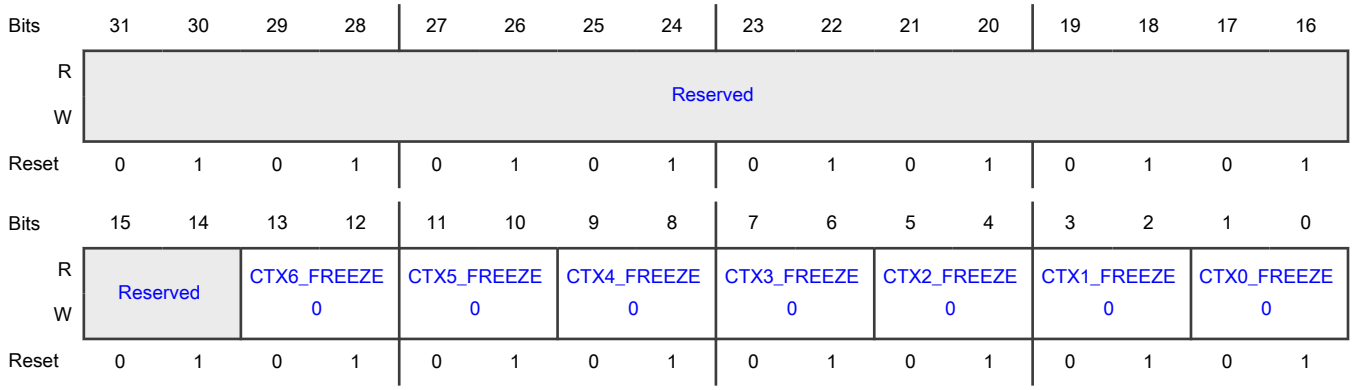
Offset

Register	Offset
IPEDCTXCTRL0	500h

Function

Freezes the IPEDCTXnSTART and IPEDCTXnEND registers. If bit1 and bit0 of the CTXn_FREEZE0 field are different, and IPEDCTXCTRL1[CTXn_FREEZE1] = 10b, CTXn_FREEZE0 is writable. See Inline PRINCE encryption and decryption (IPED) CTR.

Diagram



Fields

Field	Function
31-14 —	Reserved
13-12 CTX6_FREEZE0	Context Register Freeze for Region 6 Controls the read and write properties of this field and region 6 context registers (IPEDCTX6xxxx). This field freezes the IPEDCTX6START and IPEDCTX6END registers. If bit1 and bit0 of CTX6_FREEZE0 field are different, and IPEDCTXCTRL1[CTX6_FREEZE1] = 10b, CTX6_FREEZE0 is writable.
11-10 CTX5_FREEZE0	Context Register Freeze for Region 5 Controls the read and write properties of this field and region 5 context registers (IPEDCTX5xxxx). This field freezes the IPEDCTX5START and IPEDCTX5END registers. If bit1 and bit0 of CTX5_FREEZE0 field are different, and IPEDCTXCTRL1[CTX5_FREEZE1] = 10b, CTX5_FREEZE0 is writable.
9-8 CTX4_FREEZE0	Context Register Freeze for Region 4 Controls the read and write properties of this field and region 4 context registers (IPEDCTX4xxxx). This field freezes the IPEDCTX4START and IPEDCTX4END registers. If bit1 and bit0 of CTX4_FREEZE0 field are different, and IPEDCTXCTRL1[CTX4_FREEZE1] = 10b, CTX4_FREEZE0 is writable.
7-6 CTX3_FREEZE0	Context Register Freeze for Region 3 Controls the read and write properties of this field and region 3 context registers (IPEDCTX3xxxx). This field freezes the IPEDCTX3START and IPEDCTX3END registers. If bit1 and bit0 of CTX3_FREEZE0 field are different, and IPEDCTXCTRL1[CTX3_FREEZE1] = 10b, CTX3_FREEZE0 is writable.
5-4 CTX2_FREEZE0	Context Register Freeze for Region 2 Controls the read and write properties of this field and region 2 context registers (IPEDCTX2xxxx). This field freezes the IPEDCTX2START and IPEDCTX2END registers. If bit1 and bit0 of CTX2_FREEZE0 field are different, and IPEDCTXCTRL1[CTX2_FREEZE1] = 10b, CTX2_FREEZE0 is writable.
3-2 CTX1_FREEZE0	Context Register Freeze for Region 1 Controls the read and write properties of this field and region 1 context registers (IPEDCTX1xxxx). This field freezes the IPEDCTX1START and IPEDCTX1END registers. If bit1 and bit0 of CTX1_FREEZE0 field are different, and IPEDCTXCTRL1[CTX1_FREEZE1] = 10b, CTX1_FREEZE0 is writable.

Table continues on the next page...

Table continued from the previous page...

Field	Function
1-0 CTX0_FREEZE0	Context Register Freeze for Region 0 Controls the read and write properties of this field and region 0 context registers (IPEDCTX0xxxx). This field freezes the IPEDCTX0START and IPEDCTX0END registers. If bit1 and bit0 of CTX0_FREEZE0 field are different, and IPEDCTXCTRL1[CTX0_FREEZE1] = 10b, CTX0_FREEZE0 is writable.

25.7.2.39 IPED context control 1 (IPEDCTXCTRL1)

Offset

Register	Offset
IPEDCTXCTRL1	504h

Function

Freezes the IPEDCTXnSTART and IPEDCTXnEND registers. If bit1 and bit0 of the CTXn_FREEZE0 field are different, and IPEDCTXCTRL1[CTXn_FREEZE1] = 10b, CTXn_FREEZE1 is writable. See Inline PRINCE encryption and decryption (IPED) CTR.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved		CTX6_FREEZE1		CTX5_FREEZE1		CTX4_FREEZE1		CTX3_FREEZE1		CTX2_FREEZE1		CTX1_FREEZE1		CTX0_FREEZE1	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Fields

Field	Function
31-14 —	Reserved
13-12 CTX6_FREEZE1	Context Register Freeze for Region 6 Controls the read and write properties of this field and region 6 context registers (IPEDCTX6xxxx). This field freezes the IPEDCTX6START and IPEDCTX6END registers. If bit1 and bit0 of CTX6_FREEZE0 field are different, and IPEDCTXCTRL1[CTX6_FREEZE1] = 10b, CTX6_FREEZE1 is writable.
11-10	Context Register Freeze for Region 5

Table continues on the next page...

Table continued from the previous page...

Field	Function
CTX5_FREEZE1	Controls the read and write properties of this field and region 5 context registers (IPEDCTX5xxxx). This field freezes the IPEDCTX5START and IPEDCTX5END registers. If bit1 and bit0 of CTX5_FREEZE0 field are different, and IPEDCTXCTRL1[CTX5_FREEZE1] = 10b, CTX5_FREEZE1 is writable.
9-8 CTX4_FREEZE1	Context Register Freeze for Region 4 Controls the read and write properties of this field and region 4 context registers (IPEDCTX4xxxx). This field freezes the IPEDCTX4START and IPEDCTX4END registers. If bit1 and bit0 of CTX4_FREEZE0 field are different, and IPEDCTXCTRL1[CTX4_FREEZE1] = 10b, CTX4_FREEZE1 is writable.
7-6 CTX3_FREEZE1	Context Register Freeze for Region 3 Controls the read and write properties of this field and region 3 context registers (IPEDCTX3xxxx). This field freezes the IPEDCTX3START and IPEDCTX3END registers. If bit1 and bit0 of CTX3_FREEZE0 field are different, and IPEDCTXCTRL1[CTX3_FREEZE1] = 10b, CTX3_FREEZE1 is writable.
5-4 CTX2_FREEZE1	Context Register Freeze for Region 2 Controls the read and write properties of this field and region 2 context registers (IPEDCTX2xxxx). This field freezes the IPEDCTX2START and IPEDCTX2END registers. If bit1 and bit0 of CTX2_FREEZE0 field are different, and IPEDCTXCTRL1[CTX2_FREEZE1] = 10b, CTX2_FREEZE1 is writable.
3-2 CTX1_FREEZE1	Context Register Freeze for Region 1 Controls the read and write properties of this field and region 1 context registers (IPEDCTX1xxxx). This field freezes the IPEDCTX1START and IPEDCTX1END registers. If bit1 and bit0 of CTX1_FREEZE0 field are different, and IPEDCTXCTRL1[CTX1_FREEZE1] = 10b, CTX1_FREEZE1 is writable.
1-0 CTX0_FREEZE1	Context Register Freeze for Region 0 Controls the read and write properties of this field and region 0 context registers (IPEDCTX0xxxx). This field freezes the IPEDCTX0START and IPEDCTX0END registers. If bit1 and bit0 of CTX0_FREEZE0 field are different, and IPEDCTXCTRL1[CTX0_FREEZE1] = 10b, CTX0_FREEZE1 is writable.

25.7.2.40 IPED Context0 IV0 (IPEDCTX0IV0)

Offset

Register	Offset
IPEDCTX0IV0	520h

Function

Sets the IPED context IV for encryption and decryption.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CTX0_IV0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CTX0_IV0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 CTX0_IV0	Lowest 32 bits of IV for region 0.

25.7.2.41 IPED Context0 IV1 (IPEDCTX0IV1)

Offset

Register	Offset
IPEDCTX0IV1	524h

Function

Sets the IPED context IV for encryption and decryption.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CTX0_IV1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CTX0_IV1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 CTX0_IV1	Highest 32 bits of IV for region 0.

25.7.2.42 Start Address of Region (IPEDCTX0START - IPEDCTX6START)

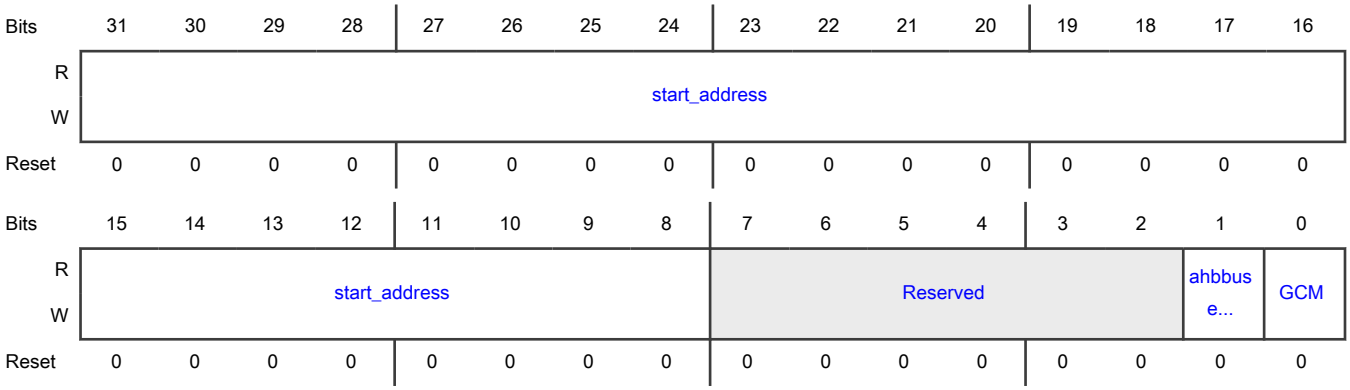
Offset

Register	Offset
IPEDCTX0START	528h
IPEDCTX1START	548h
IPEDCTX2START	568h
IPEDCTX3START	588h
IPEDCTX4START	5A8h
IPEDCTX5START	5C8h
IPEDCTX6START	5E8h

Function

Indicates start address of region. If the CTXn_FREEZE0 fields of the IPEDCTXCTRL0 register are 10b and CTXn_FREEZE1 fields of IPEDCTXCTRL1 register are 10b, then IPEDCTXnSTART is writable.

Diagram



Fields

Field	Function
31-8 start_address	Start Address

Table continues on the next page...

Table continued from the previous page...

Field	Function
	Indicates start address of IPED region. The start address must be at least 256-byte aligned. The address used here is the AHB address that is compared with the system bus address.
7-2 —	Reserved
1 ahbbuserror_dis	AHB Bus Error Disable Used to configure whether IPED errors generate bus errors on an AHB access. 0b - AHB bus errors enabled 1b - AHB bus errors disabled
0 GCM	GCM Mode Enable 0b - Disabled. CTR mode is used. 1b - Enabled. GCM mode is used.

25.7.2.43 End Address of Region (IPEDCTX0END - IPEDCTX6END)

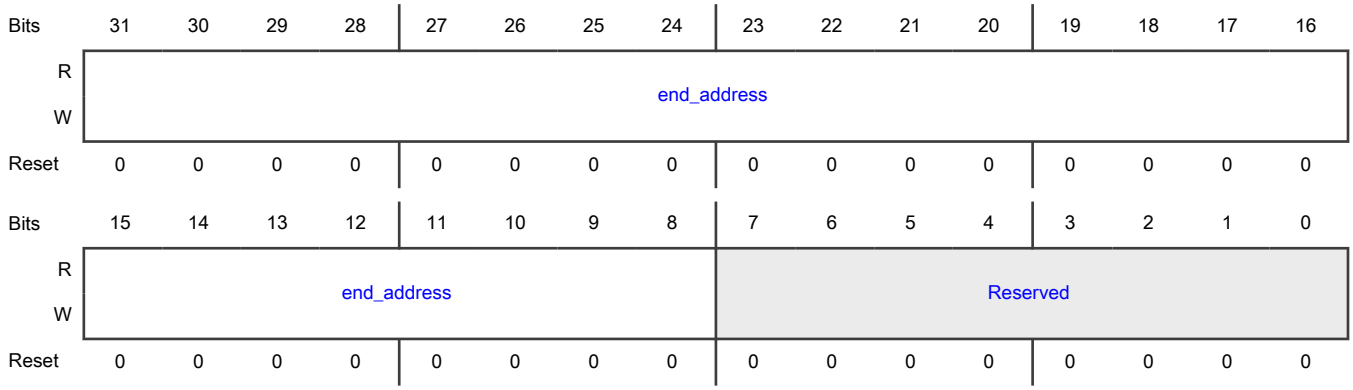
Offset

Register	Offset
IPEDCTX0END	52Ch
IPEDCTX1END	54Ch
IPEDCTX2END	56Ch
IPEDCTX3END	58Ch
IPEDCTX4END	5ACh
IPEDCTX5END	5CCh
IPEDCTX6END	5ECh

Function

Indicates end address of region. If IPEDCTXCTRL0[CTXn_FREEZE0] = 10b and IPEDCTXCTRL1[CTXn_FREEZE1] = 10b, IPEDCTXnEND is writable.

Diagram



Fields

Field	Function
31-8 end_address	End Address of IPED Region Contains the end address of the IPED region. Must be at least 256-byte aligned. This address is the AHB address that is compared with the system bus address.
7-0 —	Reserved

25.7.2.44 IPED Contexta Additional Authenticated Datac (IPEDCTX0AAD0 - IPEDCTX6AAD1)

Offset

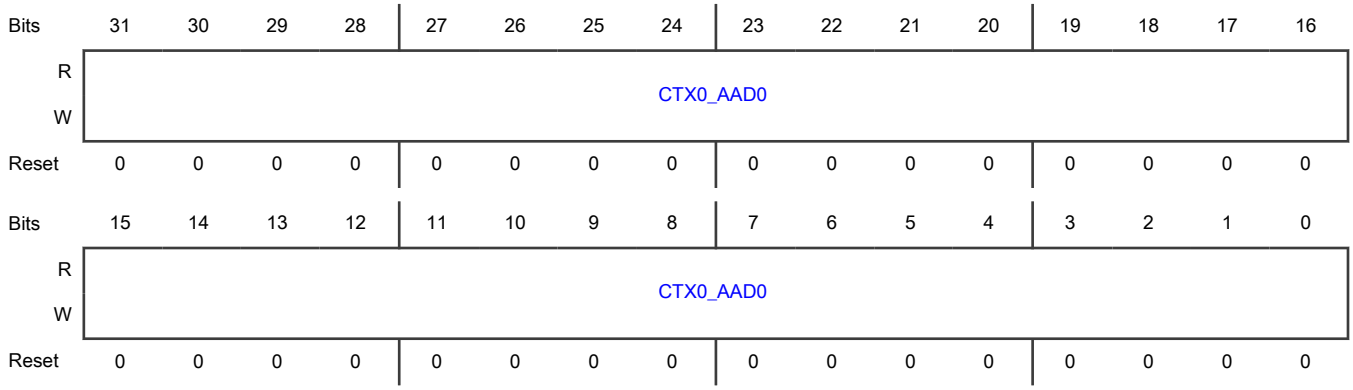
For a = 0 to 6; c = 0 to 1:

Register	Offset
IPEDCTXaAADc	530h + (a × 20h) + (c × 4h)

Function

Sets the IPED context Additional Authenticated Data (AAD) for encryption and decryption.

Diagram



Fields

Field	Function
31-0 CTX0_AAD0	CTX AAD Contains Lowest 32 bits of AAD. Only used for IPED GCM mode, not for IPED CTR mode.

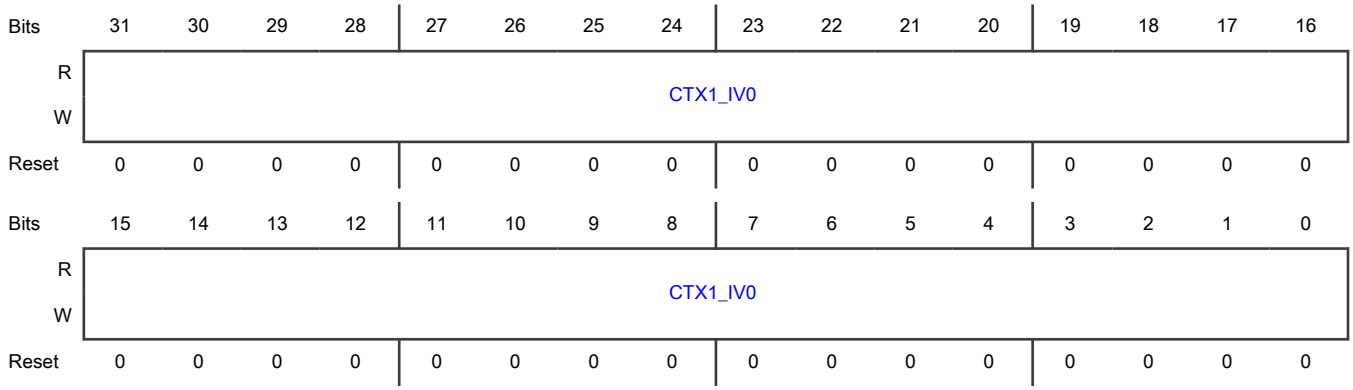
25.7.2.45 IPED Context1 IV0 (IPEDCTX1IV0)

Offset

Register	Offset
IPEDCTX1IV0	540h

Function
Sets the IPED context IV for encryption and decryption.

Diagram



Fields

Field	Function
31-0 CTX1_IV0	Lowest 32 bits of IV for region 1.

25.7.2.46 IPED Context1 IV1 (IPEDCTX1IV1)

Offset

Register	Offset
IPEDCTX1IV1	544h

Function

Sets the IPED context IV for encryption and decryption.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CTX1_IV1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CTX1_IV1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 CTX1_IV1	Highest 32 bits of IV for region 1.

25.7.2.47 IPED Context2 IV0 (IPEDCTX2IV0)

Offset

Register	Offset
IPEDCTX2IV0	560h

Function

Sets the IPED context IV for encryption and decryption.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CTX2_IV0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CTX2_IV0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 CTX2_IV0	Lowest 32 bits of IV for region 2.

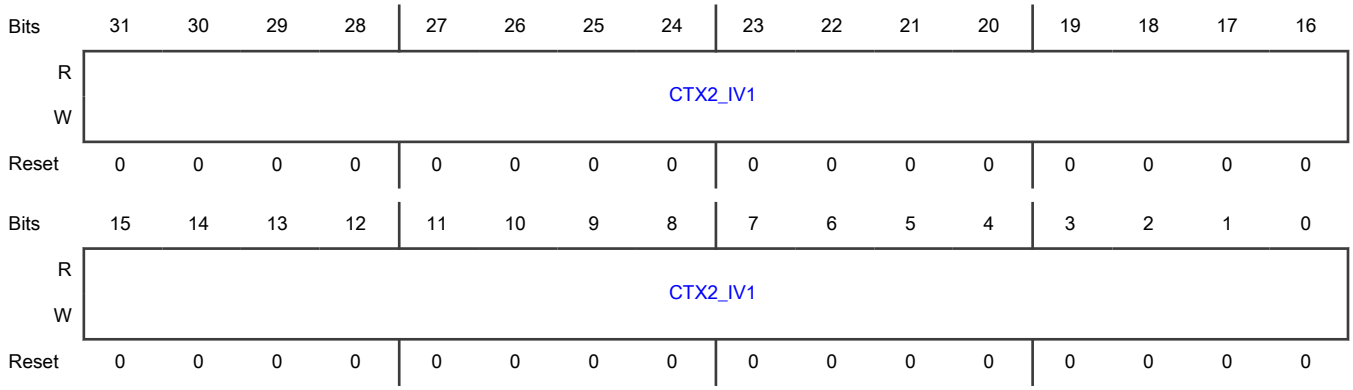
25.7.2.48 IPED Context2 IV1 (IPEDCTX2IV1)

Offset

Register	Offset
IPEDCTX2IV1	564h

Function
Sets the IPED context IV for encryption and decryption.

Diagram



Fields

Field	Function
31-0 CTX2_IV1	Highest 32 bits of IV for region 2.

25.7.2.49 IPED Context3 IV0 (IPEDCTX3IV0)

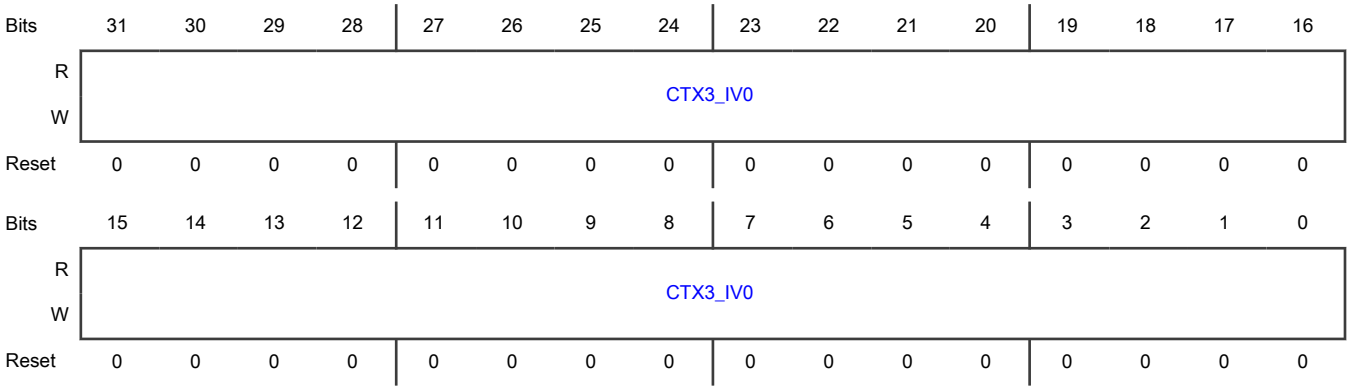
Offset

Register	Offset
IPEDCTX3IV0	580h

Function

Sets the IPED context IV for encryption and decryption.

Diagram



Fields

Field	Function
31-0 CTX3_IV0	Lowest 32 bits of IV for region 3.

25.7.2.50 IPED Context3 IV1 (IPEDCTX3IV1)

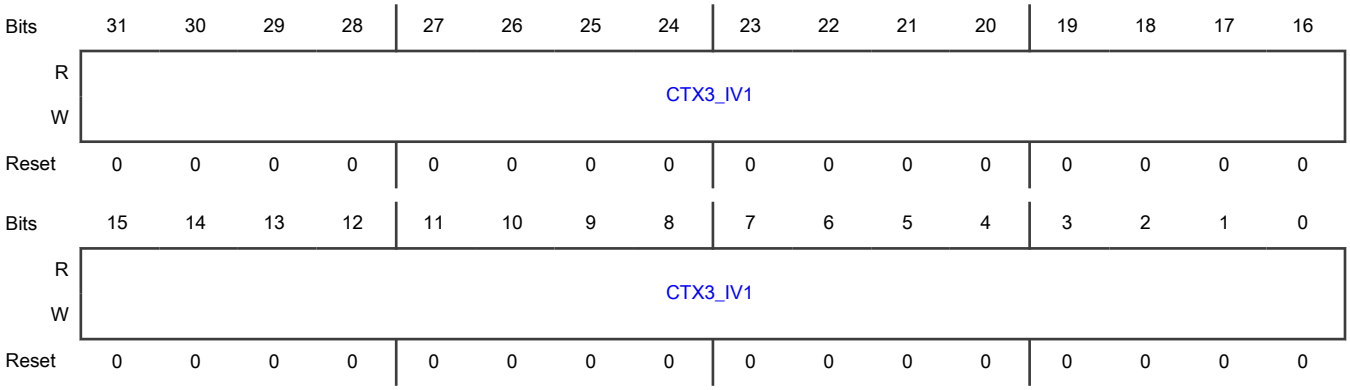
Offset

Register	Offset
IPEDCTX3IV1	584h

Function

Sets the IPED context IV for encryption and decryption.

Diagram



Fields

Field	Function
31-0 CTX3_IV1	Highest 32 bits of IV for region 3.

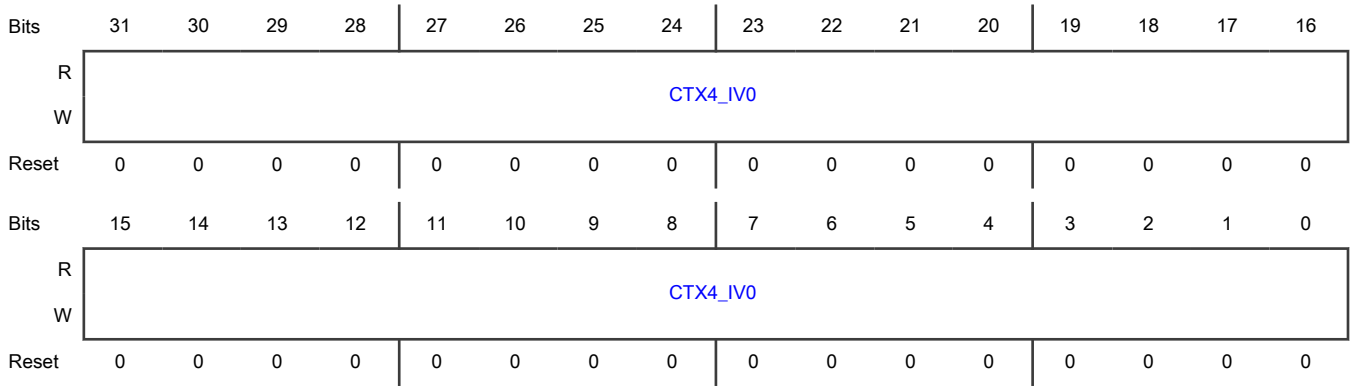
25.7.2.51 IPED Context4 IV0 (IPEDCTX4IV0)

Offset

Register	Offset
IPEDCTX4IV0	5A0h

Function
Sets the IPED context IV for encryption and decryption.

Diagram



Fields

Field	Function
31-0 CTX4_IV0	Lowest 32 bits of IV for region 4.

25.7.2.52 IPED Context4 IV1 (IPEDCTX4IV1)

Offset

Register	Offset
IPEDCTX4IV1	5A4h

Function

Sets the IPED context IV for encryption and decryption.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CTX4_IV1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CTX4_IV1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 CTX4_IV1	Highest 32 bits of IV for region 4.

25.7.2.53 IPED Context5 IV0 (IPEDCTX5IV0)

Offset

Register	Offset
IPEDCTX5IV0	5C0h

Function

Sets the IPED context IV for encryption and decryption.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CTX5_IV0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CTX5_IV0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 CTX5_IV0	Lowest 32 bits of IV for region 5.

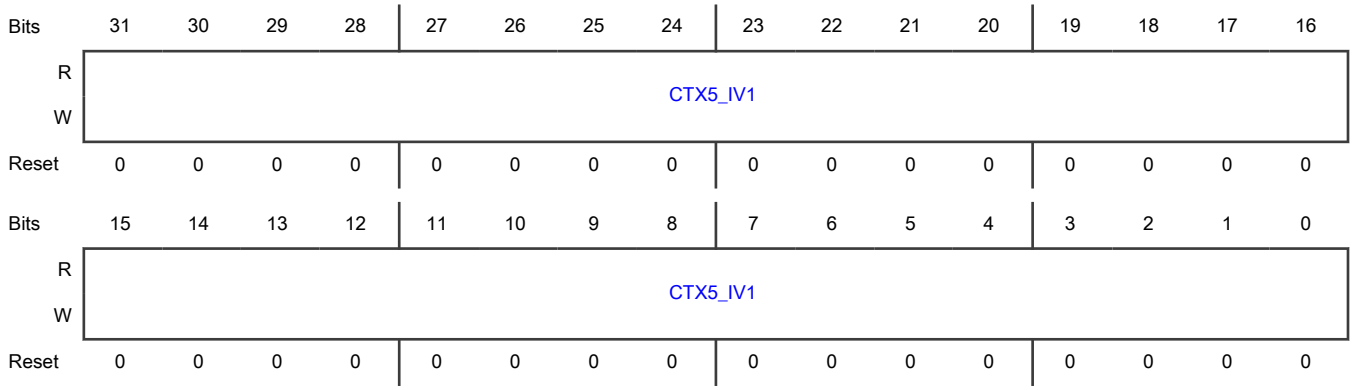
25.7.2.54 IPED Context5 IV1 (IPEDCTX5IV1)

Offset

Register	Offset
IPEDCTX5IV1	5C4h

Function
Sets the IPED context IV for encryption and decryption.

Diagram



Fields

Field	Function
31-0 CTX5_IV1	Highest 32 bits of IV for region 5.

25.7.2.55 IPED Context6 IV0 (IPEDCTX6IV0)

Offset

Register	Offset
IPEDCTX6IV0	5E0h

Function

Sets the IPED context IV for encryption and decryption.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CTX6_IV0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CTX6_IV0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 CTX6_IV0	Lowest 32 bits of IV for region 6.

25.7.2.56 IPED Context6 IV1 (IPEDCTX6IV1)

Offset

Register	Offset
IPEDCTX6IV1	5E4h

Function

Sets the IPED context IV for encryption and decryption.

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CTX6_IV1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CTX6_IV1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fields

Field	Function
31-0 CTX6_IV1	Highest 32 bits of IV for region 6.

25.8 AHB memory map definition

This section describes FlexSPI module AHB memory map in detail.

25.8.1 AHB memory map for serial flash memory and RAM access

AHB read and write access for serial flash and RAM memory are mapped to a specific address range. See the system memory map for specific address ranges supported.

AHB bus features supported for serial flash and RAM memory reading:

- Cacheable and non-cacheable access
- Prefetch enable and disable
- Burst size: 8, 16, 32, 64 bits
- All burst types: types: SINGLE, INCR, WRAP4, INCR4, WRAP8, INCR8, WRAP16, INCR16

AHB bus features for Serial RAM writing:

- Bufferable and Non-Bufferable access
- Burst size: 8, 16, 32, 64 bits
- All burst types: SINGLE, INCR, WRAP4, INCR4, WRAP8, INCR8, WRAP16, INCR16

See Flash memory access via AHB command for details about AHB access to serial flash memory and RAM.

25.8.2 AHB Memory Map for IP RX FIFO read access

The following address range is mapped for AHB read access to IP RX FIFO: 0x10000000 - 0x10000400.

AHB Bus feature supported for IP RX FIFO reading:

- Burst size: 8/16/32/64 bits
- All burst type: SINGLE/INCR/WRAP4/INCR4/WRAP8/INCR8/WRAP16/INCR16

Refer Reading data from IP receive FIFO for more details about IP RX FIFO reading.

25.8.3 AHB Memory Map for IP TX FIFO write access

The following address range is mapped for AHB write access to IP TX FIFO: 0x11000000 - 0x11000400.

AHB Bus feature supported for IP TX FIFO writing:

- Burst size: 8/16/32/64 bits
- All burst type: SINGLE/INCR/WRAP4/INCR4/WRAP8/INCR8/WRAP16/INCR16

Refer Writing data to IP transmit FIFO for more details about IP TX FIFO filling.

Appendix A
Release notes

A.1 General changes

- The IFR and Fusemap excel sheet attachments are now collated in one attachment, *MCXNx4x_IFR_Fusemap.xlsx*.
- The IFR sheet of "*MCXNx4x_IFR_Fusemap.xlsx*" is updated.

A.2 About This Manual chapter changes

No substantial content changes.

A.3 Introduction changes

- Minor update in Figure 3.
- Note added in Figure 2.
- Updated the "Power-efficient operating modes" row of table "Key features" in Overview.

- Updated the features block diagram.
- Updated the Power-efficient operating modes in Key features table.
- Updated the Target applications list.

A.4 Security overview changes

No substantial content changes.

A.5 Core Overview changes

- Peripheral acronym is corrected from ENC to QDC, in the table "Interrupt Vector Assignments".
- Added a note "IFR and PFR refer to "Protected Flash Regions" in this document."

A.6 Life Cycle States chapter changes

- Added a note "IFR and PFR refer to "Protected Flash Regions" in this document."

- Updated Life cycle states
- Updated Table "FlexSPI NOR FCB" for serialClkFreq

A.7 ROM API chapter changes

- Removed FlexSPI parallel mode mentions.

A.8 ISP chapter changes

No substantial content changes.

A.9 Non-secure Boot ROM changes

- Updated XOM access control settings.
- Updated Secondary bootloader mode.
- Updated Top-level boot flow.
- Updated IFR region definitions.
- Added a note "IFR and PFR refer to "Protected Flash Regions" in this document."

A.10 Secure Boot ROM module changes

- Updated the Miscellaneous functions section
- Updated the TrustZone Preset Data Structure section
- Added a note "IFR and PFR refer to "Protected Flash Regions" in this document."

A.11 Debug Mailbox (DBGMB)

A.11.1 Chip-specific DBGMB information changes

No substantial content changes.

A.11.2 DBGMB module changes

- Updated Block diagram.
- Updated Ports.
- Updated Credential Constraints (DCFG_CC_SOCU).

- Updated DCFG_VENDOR_USAGE .
- Updated DM-AP commands.

A.12 System Controller (SYSCON)

A.12.1 Chip-specific SYSCON information changes

No substantial content changes.

A.12.2 SYSCON module changes

- Peripheral naming ENC is corrected as QDC.

Table continues on the next page...

• Updated Signals.
• Updated the description of the SWD_ACCESS_CPU0 register.

A.13 VBAT

A.13.1 Chip-specific VBAT information changes

• Added Clearing the CLOCK_DET flag when the CLKMON is disabled.

A.13.2 VBAT module changes

• In Overview, text revised. • In FRO 16.384 kHz clock, text added. • [TIMCFG], bitfield description updated.
• Added note in STATUSA[VOLT_DET].

A.14 EdgeLock Secure Subsystem (ELS)

A.14.1 Chip-specific ELS information changes

• Updated the ELS subsystem diagram.

A.14.2 ELS module changes

• Minor updates in Keys

A.15 Physically Unclonable Function (PUF)

A.15.1 Chip-specific PUF information changes

No substantial content changes.

A.15.2 PUF module changes

• Modified Hardware Information (HW_INFO) for its reset value.

A.16 PUF Key Context Management

A.16.1 Chip-specific PUF CTRL information changes

No substantial content changes.

A.16.2 PUF Context Key Management module changes

No substantial content changes.

A.17 Public-key Cryptography Accelerator (PKC)

A.17.1 Chip-specific PKC information changes

No substantial content changes.

A.17.2 PKC module changes

No substantial content changes.

A.18 OTP Controller (OTPC)

A.18.1 Chip-specific OTPC information changes

No substantial content changes.

A.18.2 OTPC changes

No substantial content changes.

A.19 Code Watchdog Timer (CDOG)

A.19.1 Chip-specific WWDT information changes

No substantial content changes.

A.19.2 CDOG module changes

- Updated Fault types
- Updated Interrupts
- Updated Secure counter
- Updated Clocking
- Updated Reset
- Updated field description for the following fields:
  - CONTROL[DEBUG_HALT_CTRL]
  - CONTROL[IRQ_PAUSE]
  - CONTROL[ADDRESS_CTRL]
  - CONTROL[STATE_CTRL]
  - CONTROL[SEQUENCE_CTRL]

- CONTROL[MISCOMPARE_CTRL]
- CONTROL[TIMEOUT_CTRL]

- Updated Function description of the following registers:
  - STATUS
  - START
  - STOP
  - RESTART
  - ADD
  - ADD16
  - ADD256
  - SUB
  - SUB1
  - SUB16
  - SUB256
  - ASSERT16

A.20 Glitch Detect (GDET)

A.20.1 Chip-specific GDET information changes

No substantial content changes.

A.20.2 GDET module changes

- Added a footnote in GDET voltage mode change.

A.21 Intrusion and Tamper Response Controller (ITRC)

A.21.1 Chip-specific ITRC information changes

No substantial content changes.

A.21.2 ITRC module changes

No substantial content changes.

A.22 Memory Block Checker (MBC)

A.22.1 Chip-specific MBC information changes

No substantial content changes.

A.22.2 MBC module changes

• Updated Memory block access evaluation.
• CR occurrences replaced with TRDC_CR in: • TRDC_CR[VLD] register.

A.23 Digital Tamper (TDET)

A.23.1 Chip-specific Digital Tamper information changes

No substantial content changes.

A.23.2 Digital Tamper Detect (TDET) module changes

No substantial content changes.

A.24 Secure AHB bus and AHB Controller (AHBSC)

A.24.1 Chip-specific Secure AHB Controller information changes

• Added Secure Interrupt Masking.

A.24.2 AHBSC module changes

• Updated the Master Security Wrapper (MSW) section • Removed "Interrupt, DMA and GPIO: Secure instance and masking." • Peripheral name is corrected from ENC to QDC.
• Updated the Master Security Wrapper (MSW) section • Removed "Interrupt, DMA and GPIO: Secure instance and masking."

A.25 Flash Controller (FMC)

A.25.1 Chip-specific FMC information changes

No substantial content changes.

A.25.2 Flash Memory Controller (FMC-NPX) module changes

• In Config options, updated the description of the buffering controls to indicate that these controls are in other modules. Added note to see the chip-specific section regarding registers to configure FMC. • In Features, added references to the chip-specific section to the items describing input controls. • In Modes of operation, added "automatically" to statement about FMC becoming disabled.

- In Speculative reads, removed information about registers in other chapters and replaced it with note to see chip-specific section.
- In Initialization and application information, added note to see chip-specific section.

A.26 FlexSPI

A.26.1 Chip-specific FLEXSPI information changes

No substantial content changes.

A.26.2 FlexSPI module changes

No substantial content changes.

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Hazardous voltage — Although basic supply voltages of the product may be much lower, circuit voltages up to 60 V may appear when operating this product, depending on settings and application. Customers incorporating or otherwise using these products in applications where such high voltages may appear during operation, assembly, test etc. of such application, do so at their own risk. Customers agree to fully indemnify NXP Semiconductors for any damages resulting from or in connection with such high voltages. Furthermore, customers are drawn to safety standards (IEC 950, EN 60 950, CENELEC, ISO, etc.) and other (legal) requirements applying to such high voltages.

AEC unqualified products — This product has not been qualified to the appropriate Automotive Electronics Council (AEC) standard Q100 or Q101 and should not be used in automotive applications, including but not limited to applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

CoolFlux — is a trademark of NXP B.V.

CoolFlux DSP — is a trademark of NXP B.V.

EdgeLock — is a trademark of NXP B.V.

eIQ — is a trademark of NXP B.V.

Synopsys & Designware — are registered trademarks of Synopsys, Inc.

Synopsys — Portions Copyright © 2018-2022 Synopsys, Inc. Used with permission. All rights reserved.

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© NXP B.V. 2024. All rights reserved.

For more information, please visit: https://www.nxp.com

Date of release: 09/2024

Document identifier: MCXNx4xSRM