AN12839

RT600 TrustZone®

Rev. 1 — 25 June 2020

Application Note

Contents

1 Introduction

This application note explains how TrustZone® technology offers an efficient, systemwide approach to security with hardware-enforced isolation built into the CPU. It also demonstrates how to configure the TrustZone to set secure and non-secure state on RT600 and how to switch between these states and to handle different secure faults.

2 Overview

The TrustZone for ARMv8M in combination with the Platform Security Architecture (PSA) offers a comprehensive security foundation. This subsystem includes the Secure Bus controller, NXP implemented Device Attribution Unit (IDAU), Security Attribution Unit (SAU), and secure GPIO. Built into the core platform, the TrustZone has both secure and non-secure Memory Protection Units(MPUs).

- The TrustZone for Cortex-M33 plus Platform Security Architecture include:
  - ARMV8-M addition states
    - Secure and non-secure stack pointers
    - Dual stack limit checking
    - Private SysTick timer for each state
  - Security Attribution Unit (SAU)
  - Memory Protection Unit (MPU) which has Secure and Non-Secure memories
  - NXP modules include
    - Defined Attribution Unit (using IDAU interface)
    - Secure Bus Control
    - Secure GPIO Controller
    - Secure DMA Controller

  Just having a combination of secure and non-secure code in the TrustZone decrease the attack surface of your software. Make sure that secure portion of your software is bullet-proof and protected.

Figure 1. TrustZone-M Sub-System

2.1 TrustZone for ARMv8M Cortex-M33

TrustZone is a technology used in all Cortex A to secure your smartphone, tablets, and smart TVs. TrustZone provides the means to implement separation and access control to isolate trusted software and resources to reduce the attack surface of key components. The created trusted enclave can protect trusted software and is ideal to store and run the critical security services. Best practices demand that this code be small, reviewed code with provisions of security services. The enclave can also protect trusted hardware to augment and fortify the trusted software. This includes the modules for hardware assists for cryptographic accelerators, random number generators, and secure storage.

Security needs be thought about from a system perspective – it needs to cover CPU, memory, peripherals and all the IP that connect these devices together.

Isolation is just the foundation – security is about layers of protection, adding in further HW and SW to add further layers.

2.2 RT600 Trusted Execution Environment

TrustZone for Armv8-M and trusted execution environment are available on all MIMXRT600 devices. Certain software must not have its execution flow compromised, it cannot have a possibility of diversion or modification in the middle of execution. Consider a case, such as payment validation where intermediate steps must not be modifiable with malicious code or be observable.

SW IP protection may require some routines to run without being readable by user.

Protection of the data/peripherals belonging to secure software IP which does not allow it to leak or be modified by an unauthorized source.

RT600 EdgeLock™ 400A Secure Execution Environment supports several on-chip security capabilities and is built on a foundation of secure boot, secure debug, and secure life cycle management designed to resist remote and local software attacks as shown below.

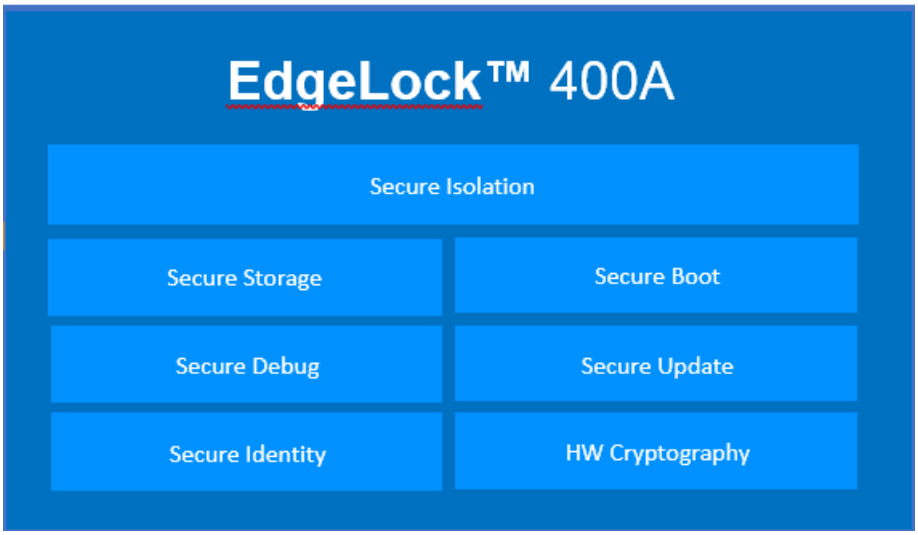

Figure 2. EdgeLock™ 400A

2.3 Secure and Non-Secure Memory Attribution

Memory can be Secure, Non-Secure (NS) or non-secure callable (NSC). These are defined by the Security Attribution Unit (SAU) is programmable or the Implementation Defined Attribution Unit (IDAU) is fixed by NXP. Secure data can only be read by secure code. Secure code can only be executed by a CPU in secure mode. NS data can be accessed by both secure state and non-secure state CPU. NS code cannot be executed by secure code. The NSC is a special region for NS code to branch into and execute a Secure Gateway (SG) opcode. This is the only way for NS code to call an S function. If SG is executed in NSC region and the CPU is in NS state, then the CPU moves to S state.

NOTE

Only SG instructions can be legally executed out of NSC region, so the branching to the actual S function is done in Secure mode.

The CPU states can be Secure privilege, secure non-privilege, privilege (Handler), or non-privilege (Thread). The NSC region of S-memory provides a veneer for S-application code to access function in S-memory without divulging the specific address of the secure function. On executing the SG instruction, the CPU changes from CPU-NS to CPU-S, then execute the veneered call to a secure function in S-memory. If the CPU-NS calls into an address in the NSC region that is not an SG instruction an exception fault is created. The exception fault results in the CPU entering secure state.

The secure application code developer creates function calls inside the NSC region to S-application code, allowing the NS-application the capability use functions inside S-memory.

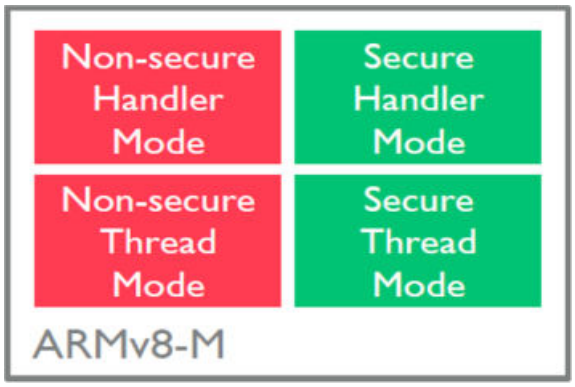

Figure 3. Memory Attributes

2.4 ARMv8M additional CPU States

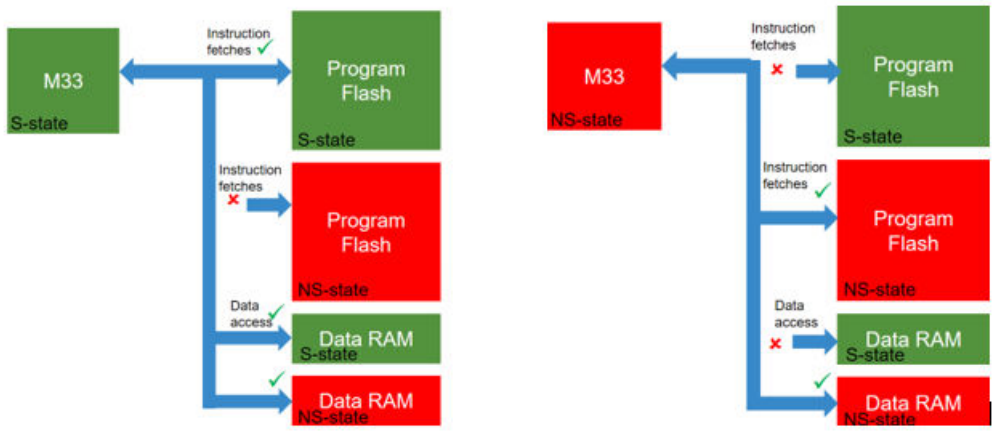

Figure 4. CPU Secure and Non-Secure State

Secure and non-secure code runs on a single CPU for efficient embedded implementation.

CPU in non-secure state can only execute from non-secure program memory. CPU in non-secure state can access data from both NS memory only.

For the secure, trusted code there is a new secure stack pointer and stack-limit checking. There are separate Memory Protection Units (MPUs) for S and NS regions and private SysTick timers for each state. The secure side can configure the target domain of interrupts.

2.5 Secure Bus

The secure bus includes a PPC (Peripheral Protection Checker), MPC (Memory Protection Checker) and the MSW (Master Security Wrapper). This is not the same as the secure bus between the PUF and PRINCE and AES.

2.6 Security defined by address

The NXP IDAU implementation of Arm TrustZone for CPU0 involves using address bit 28 to divide the address space into potential secure and non-secure regions. Address bit 28 is not decoded in memory access hardware, so each physical location appears in two places on whatever bus they are located on. Other hardware determines which kinds of accesses (including non-secure callable) are allowed for any address.

Table 1. TrustZone and System General Mapping

Start Address	End Address	TrustZone	CM-33 bus	CM-33 usage
0x0000 0000	0x0FFF FFFF	Non-Secure	Code	Shared RAM, Boot ROM, FlexSPI memory mapped region.
0x1000 0000	0x1FFF FFFF	Secure	Code	Same as above
0x2000 0000	0x2FFF FFFF	Non-Secure	Data	Shared RAM, CM33 access to HiFi4 TCMs via inbound PIF. Non-cacheable FlexSPI memory mapped region for DSP only
0x3000 0000	0x3FFF FFFF	Secure	Data	Same as above
0x4000 0000	0x4FFF FFFF	Non-Secure	Data	AHB and APB Peripherals
0x5000 0000	0x5FFF FFFF	Secure	Data	Same as above

2.7 Attribution Unit

All addresses are either secure or non-secure. The Security Attribution Unit (SAU) inside of the ARMV8M works with the MPUs. There are 8 SAU regions supported by RT600.

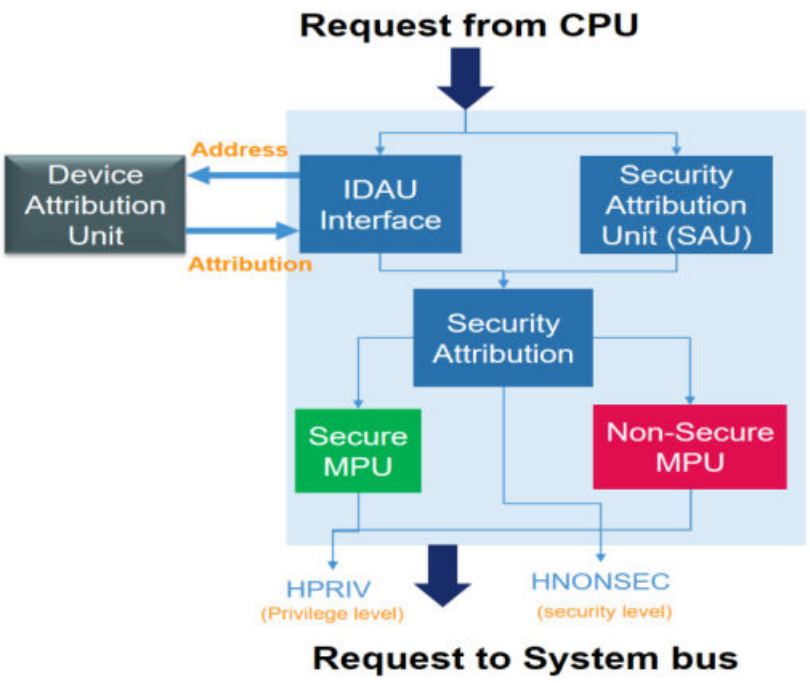

Figure 5. Security Attribution Unit

NXP incorporated an Implementation-specific Device Attribution Unit (IDAU) allowing a secure OS to be decoupled from the application.

2.8 RT600 IDAU and Security Attribution Unit

The IDAU is a simple design using address bit 28 to allow aliasing of the memories in two locations. If address bit 28 is = 0 the memory is Non-Secure. If address bit 28 = 1 the memory is Secure. The SAU allows 8 memory regions and allow the user to override the IDAU's fixed Map, to define the non-secure regions. By default, all memory is set to secure. At least one SAU descriptor should be used to make IDAU effective. If either IDAU or SAU marks a region that region is secure. NSC area can be defined in NS region of the IDAU.

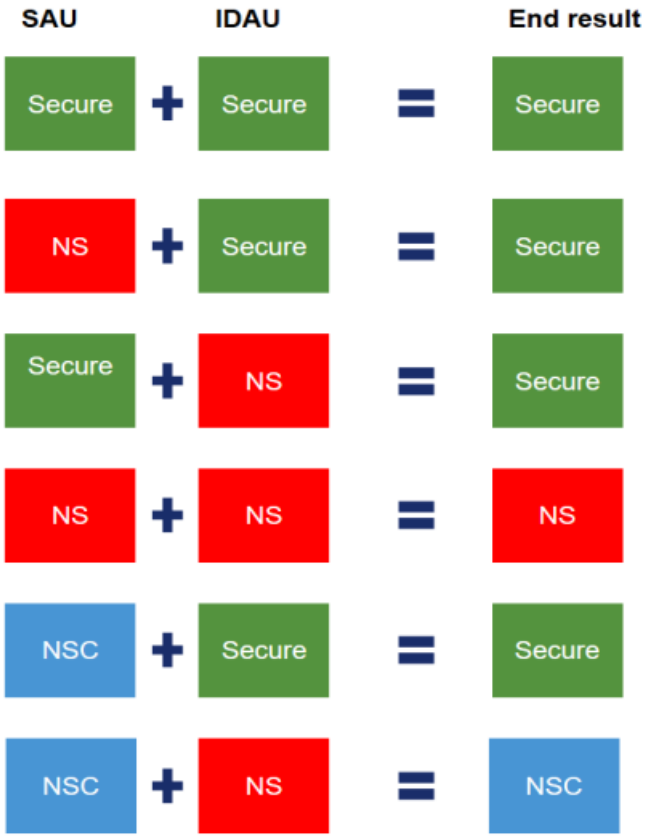

Figure 6. Security Attribution Unit

Configuration of SAU:

1. Configure SAU region 0 for non-secure RAM for code execution.

```
/* Configure SAU region 0 - Non-secure RAM for CODE execution*/
/* Set SAU region number */
SAU->RNR = 0;
/* Region base address */
SAU->RBAR = (CODE_START_NS & SAU_RBAR_BADDR_Msk);
/* Region end address */
SAU->RLAR = ((CODE_START_NS + CODE_SIZE_NS - 1) & SAU_RLAR_LADDR_Msk) |
/* Region memory attribute index */
((0U << SAU_RLAR_NSC_Pos) & SAU_RLAR_NSC_Msk) |
/* Enable region */
((1U << SAU_RLAR_ENABLE_Pos) & SAU_RLAR_ENABLE_Msk);
```

2. Configure SAU region 1 for non-secure RAM for Data.

```
/* Configure SAU region 1 - Non-secure RAM for DATA */
/* Set SAU region number */
SAU->RNR = 1;
/* Region base address */
SAU->RBAR = (DATA_START_NS & SAU_RBAR_BADDR_Msk);
/* Region end address */
SAU->RLAR = ((DATA_START_NS + DATA_SIZE_NS - 1) & SAU_RLAR_LADDR_Msk) |
/* Region memory attribute index */
((0U << SAU_RLAR_NSC_Pos) & SAU_RLAR_NSC_Msk) |
```

```
/* Enable region */
((1U << SAU_RLAR_ENABLE_Pos) & SAU_RLAR_ENABLE_Msk);
```

3. Configure SAU region 2 for non-secure callable RAM for code veneer table.

```
/* Configure SAU region 2 - Non-secure callable RAM for CODE veneer table*/
/* Set SAU region number */
SAU->RNR = 2;
/* Region base address */
SAU->RBAR = (CODE_START_NSC & SAU_RBAR_BADDR_Msk);
/* Region end address */
SAU->RLAR = ((CODE_START_NSC + CODE_SIZE_NSC - 1) & SAU_RLAR_LADDR_Msk) |
/* Region memory attribute index */
((1U << SAU_RLAR_NSC_Pos) & SAU_RLAR_NSC_Msk) |
/* Enable region */
((1U << SAU_RLAR_ENABLE_Pos) & SAU_RLAR_ENABLE_Msk);
```

4. Configure SAU region 3 for non-secure Flash for Code.

```
/* Configure SAU region 3 - Non-secure FLASH for CODE execution*/
/* Set SAU region number */
SAU->RNR = 3;
/* Region base address */
SAU->RBAR = (CODE_FLASH_START_NS & SAU_RBAR_BADDR_Msk);
/* Region end address */
SAU->RLAR = ((CODE_FLASH_START_NS + CODE_FLASH_SIZE_NS - 1) & SAU_RLAR_LADDR_Msk) |
/* Region memory attribute index */
((0U << SAU_RLAR_NSC_Pos) & SAU_RLAR_NSC_Msk) |
/* Enable region */
((1U << SAU_RLAR_ENABLE_Pos) & SAU_RLAR_ENABLE_Msk);
```

5. Configure SAU region 4 for non-secure callable Flash for Code veneer table.

```
/* Configure SAU region 4 - Non-secure callable FLASH for CODE veneer table*/
/* Set SAU region number */
SAU->RNR = 4;
/* Region base address */
#if defined(__MCUXPRESSO)
SAU->RBAR = ((uint32_t)&_start_sg & SAU_RBAR_BADDR_Msk);
#else
SAU->RBAR = (CODE_FLASH_START_NSC & SAU_RBAR_BADDR_Msk);
#endif
/* Region end address */
#if defined(__MCUXPRESSO)
SAU->RLAR = (((uint32_t)&_start_sg + CODE_FLASH_SIZE_NSC - 1) & SAU_RLAR_LADDR_Msk) |
/* Region memory attribute index */
((1U << SAU_RLAR_NSC_Pos) & SAU_RLAR_NSC_Msk) |
/* Enable region */
((1U << SAU_RLAR_ENABLE_Pos) & SAU_RLAR_ENABLE_Msk);
#else
SAU->RLAR = ((CODE_FLASH_START_NSC + CODE_FLASH_SIZE_NSC - 1) & SAU_RLAR_LADDR_Msk) |
/* Region memory attribute index */
((1U << SAU_RLAR_NSC_Pos) & SAU_RLAR_NSC_Msk) |
/* Enable region */
((1U << SAU_RLAR_ENABLE_Pos) & SAU_RLAR_ENABLE_Msk);
#endif
```

2.9 Security Bus Controller

The RT600 uses a matrix of secure bus controller modules to manage the data flow in the MCU. A combination of a Peripheral Protection Checker (PPC), the Memory Protection Checker (MPC), the Master Security Wrapper (MSW) along with security locking and error log, secure interrupt masking, Hypervisor Interrupt, and GPIO masking.

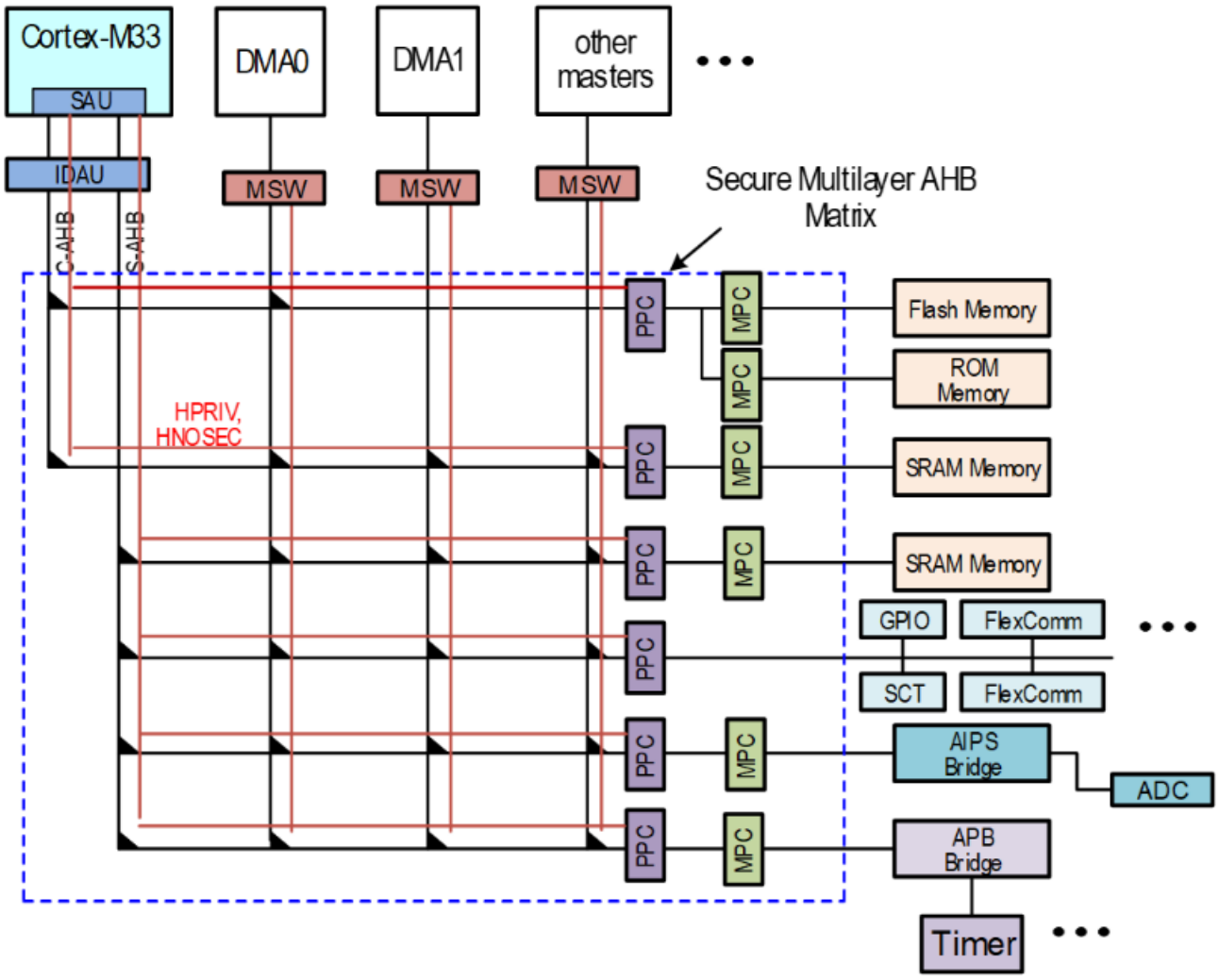

Figure 7. Secure Bus Matrix

The Bus matrix between bus masters like the M33 core or DMA engines are wrapped with the MSW and have security side band signals used for tamper detection. There is a PPC for each bus slave port. The MPCs are used for memories and bus bridges.

2.10 Memory Protection Checkers

The MPC is used with all memory devices, on-chip Flash, and SRAM as well as external memory devices. Memory blocks have one checker setting per ‘sector’ where typically, memory is divided into 32 sectors. For example, a 128 KB memory would have a granularity of 4 kB per sector.

All rules are set in the Secure Control register bank. A user must have the highest level “Secure Privileged” to set rules. The Privilege level is ignored if left in default states. By default, only the security level is checked.

2.11 Master Security Wrapper

The MSW wraps three types of bus masters, TrustZone aware Cortex M33 with security extension, simple masters such as SDIO, PowerQuad, DMA0, DMA1, Hash-AES and smart masters such as bus master's that can perform data and/or instruction access.

2.12 Security Locking

The secure bus controller allows locking of the following configurations:

- All PPC & MPC checkers settings
- All master security level (MSW) settings
- SAU Settings
- Secure MPU settings
- Secure Vector offset address (S_VTOR) for CM-33
- Non-secure MPU settings
- Non-secure Vector offset address (NS_VTOR) for CM-33

3 Demo Application

As part of this, we run SDK example to understand how to configure the TrustZone to set secure and non-secure state and how to switch between these states as well as how to handle different secure faults.

3.1 SDK Example 1

This application demonstrates following techniques for TrustZone applications development:

1. Application separation between secure and non-secure part
2. TrustZone environment configuration
3. Exporting secure function to non-secure world
4. Calling non-secure function from secure world
5. Creating veneer table.

3.1.1 Environment

3.1.2 Hardware environment

- Board
  - MIMXRT685EVK
- Debugger
  - Integrated CMSIS-DAP debugger on the board
- Miscellaneous
  - 1 Micro USB cable
  - PC
- Board Setup
  - Connect the micro USB cable between PC and J5 link on the board for loading and running a demo.

3.1.3 Software environment

- Tool chain
  - IAR embedded workbench 8.50.1 or MCUXpresso IDE 11.1.1 or Keil 5.29
- Software package
  - SDK_2.7.0_EVK-MIMXRT685

3.1.4 Steps and result

The basic steps are as follows:

1. Follow the Getting Started with MCUXpresso SDK for RT600 (can be found inside SDK->docs) in order to go through the steps for running hello_world demo (SDK\boards\evkmimxrt685\trustzone_examples\hello_world) using MCUXpresso, IAR, or Keil.

   NOTE

   The instruction for a TrustZone based application is a little different as compared to other application. Follow steps for TrustZone based application in getting started guide. Select UART from the SDK debug console option while importing trustzone based project (hello_world & secure_fault) for MCUXpresso.

2. Connect the development platform to your PC via USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number). Configure the terminal with these settings:
   - 115200 baud rate
   - 8 data bits
   - No Parity
   - 1 Stop bit
   - No Flow control
4. Result:

The application shows how to use SAU to configure secure and non-secure memory and how to switch between them in two states. There are two projects in application: secure and non-secure. Only Secure can handle UART and entry to function PRINTF to UART as defined in NSC while the strcmp callback function is defined in normal mode. The application implements the function of calling non-secure code from secure code and calling secure code from non-secure code, print the execution results in the two states.

3.2 SDK Example 2

The Secure Faults demo application demonstrates handling of different secure faults. This application is based on application Hello World. In addition, user can invoke different secure faults by setting the value of variable testCaseNumber.

3.2.1 Environment

3.2.2 Hardware environment

- Board
  - MIMXRT685EVK
- Debugger
  - Integrated CMSIS-DAP debugger on the board
- Miscellaneous

- 1 Micro USB cable
- PC

- Board Setup
  - Connect the micro USB cable between PC and J5 link on the board for loading and running a demo.

3.2.3 Software environment

- Tool chain
  - IAR embedded workbench 8.50.1 or MCUXpresso IDE 11.1.1 or Keil 5.29
- Software package
  - SDK_2.7.0_EVK-MIMXRT685

3.2.4 Steps and result

All the steps remain same as defined in Section 3.1.4 except step 1 where one needs to import “secure_faults” example application instead of hello_world. To get the different secure fault errors, change the variable 'testcaseNumber' value from 1-5 in secure_faults_s.c file before compiling.

Results:

As of part of this application, user can invoke the following faults:

- Invalid transition from secure to normal world:

  In this example, direct address to non-secure RESET is used to jump into normal world. There are two issues related with this approach:
  - All core registers are not clear so there is potential data leak
  - The most LSB of address into normal world has to be cleared
  - As this is not met, therefore secure fault is generated.Both issues can be solved by using __cmse_nonsecure_call keyword attribute. If this attribute is used for a function call to normal world, the compiler will:
    1. Clear all used registers to avoid potential data leak
    2. Clear LSB address bit
    3. Jump to address using BXNS instruction
- Invalid entry point from normal to secure world:

  Calling function located in secure world without asm(SG) cause SAU to call HardFault .The PRINTF_NS entry point is intentionally increased by 4. Therefore, the Secure Gateway SG instruction is skipped causing secure fault event due to an illegal entry point to S world.
- Invalid data access from normal world, example 1:

  In this example, the pointer is set to address 0x30000000. This address has secure attribute (see SAU settings). If data is read from this address, the secure fault is generated as in NS world, the application does not has access to secure memory.
- Invalid input parameters in entry function:

  The input parameter is set to address 0x30000000. This address has secure attribute (see SAU settings). This secure violation is not detected by secure fault, since the input parameter is used by secure function in secure mode. So this function has access to whole memory. However every entry function should check source of all input data in order to avoid potential data leak from secure memory. The correctness of input data cannot be checked automatically. This has to be check by software using Test Target TT instructions.
- Invalid data access from normal world, example 2:

The pointer is set to address 0x00130000. This address has non-secure attribute in SAU but it has secure attribute in AHB secure controller. If data is read from this address, the data bus error is generated. Compared to test #3, this error is caught by the AHB secure controller, not by SAU, because in SAU this address is non-secure so the access from normal world is correct from SAU perspective.

4 Conclusion

TrustZone technology offers an efficient, systemwide approach to security with hardware-enforced isolation built into the CPU. It does this by running two domains side-by-side and sharing resources per set configuration. It is more efficient in hardware resources and overall design effort than implementing a dedicated security subsystem. It encourages separation of applications into security-critical aspects from general-purpose aspects: reducing the attack surface. The SDK examples show how to configure the TrustZone to set secure and non-secure state and how to switch between these states as well as how to handle different secure faults.

5 References

1. RT600 user manual.
2. RT600 data sheet.
3. MCUXpresso SDK Release Notes for EVK-MIMXRT685 (can be found inside SDK)
4. Getting Started with MCUXpresso SDK for EVK-MIMXRT685 (Can be found inside SDK).
5. MCUXpresso SDK API Reference Manual

6 Revision history

Revision number	Date	Substantive changes
0	05/2020	Initial release
1	06/2020	General fixes

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. 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 or registered trademarks of Arm Limited (or its subsidiaries) 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. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2020. **All rights reserved.**

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 25 June 2020

Document identifier: AN12839

arm