AN14260

通过Overlay动态加载代码

第1.0版—2024年4月10日 应用笔记

文档信息

信息	内容
关键词	AN14260、overlay、链接器、性能优化、从RAM、GCC、EWEARM或Keil中执行代码
摘要	本应用笔记介绍了Overlay，表明它对现代微控制器性能的提升仍然是有效的。

1 介绍

Overlay是链接器的一项功能，可在同一地址加载不同的代码。这种技术在早期的家用计算机系统中非常普遍，主要是因为这些系统的内存容量有限，无法同时加载全部代码。

本应用笔记介绍了Overlay技术，表明它对现代微控制器性能的提升仍然是有效的。它甚至适用于没有任何操作系统的裸机。

1.1 概述

i.MX RT系列微控制器支持高内核频率，如表1所示。

表1. i.MX RT系列支持高内核频率

产品	最大内核频率
RT1010/1020	500MHz
RT1040/1050/1060	600MHz
RT1180	800MHz
RT1170	1000MHz

这些产品支持高内核频率。但由于很难按照这些内核相同的标准来小型化闪存，它们并未配备内部闪存。也就是说，内部闪存无法采用与内核相同的工艺生产。因此，对于i.MX RT系列来说，必须依赖外部存储器来存储其代码。

虽然外部存储器支持XIP，但其速度仍然低于片上RAM[1]。因此在性能方面，ITCM是存储代码的最佳位置，因为读取访问有望在一个周期内[2]完成。

遗憾的是，如果RAM的大小不足以满足应用程序的需求，那就别无选择，只能将代码存放在外部存储器中。然而在高内核频率下，外部存储器的带宽比指令获取的带宽要慢。因此在小区域引用的情况下，即使启用了缓存，也可能导致性能下降。此外，TCM的空间相对较小，这是因为更大的TCM区域会增加信号延迟，所以高内核频率通常会牺牲TCM的大小。

因此，通过动态加载代码，可以最大限度地利用有限的TCM资源。因为内核无需每次都从外部存储器获取代码，从而提高了性能。如果某个应用程序对ITCM的需求不高，还可以在FlexRAM中扩展DTCM，这样进一步提升性能。

在所附的示例项目中，代码通过MCUXpresso IDE (GCC)、IAR Embedded Workbench for Arm或Keil µVision IDE从外部闪存动态加载到RT1170-EVKB中的ITCM中。

具体操作如下：

- 如图1所示，`segment0`和`segment1`区段会从外部闪存动态加载到ITCM。
  - 根据需求调用`load_code()`函数来加载某个区段。
  - 加载`segment0`时，`segment1`中的任何函数都不能调用，反之亦然。
  - `segment0`中的任何函数都不能调用`segment1`中的任何函数，反之亦然。
- `CodeQuickAccess`区段始终位于ITCM中。
- `.text*`区段中的其他代码则始终位于外部闪存中。

1 *i.MX RT系列性能优化*（文档AN12437）
2 *使用i.MX RT FlexRAM*（文档AN12077）

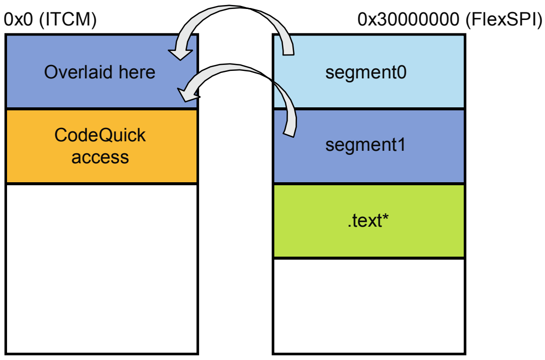

图1. 内存映射图

2 软件实现概述

本节介绍了在MCUXpresso IDE (GCC)、Keil µVision IDE和IAR Embedded Workbench for Arm中的软件实现。

2.1 链接器脚本

链接器脚本描述了对象所在的位置以及它们在执行时要加载到的位置。必须对链接器脚本进行定制以满足应用程序的特定要求。

2.1.1 MCUXpresso IDE (GCC)

假设目录结构如下。关键点是源代码目录下有`segment0`和`segment1`两个子目录。

```
├── CMSIS
├── board
├── component
├── device
├── doc
├── drivers
├── linker_script
├── startup
├── utilities
├── xip
└── source
    ├── segment0
    └── segment1
```

`source/segment0`和`source/segment1`中的所有代码都被定义为`segment0/1`，并通过以下三个步骤实现覆盖。链接器脚本是基于`evkbmimxrt1170_hello_world_demo_cm7_Debug.ld`的，其将所有代码都放置到外部闪存中。

1. 通过使用`EXLUDE_FILE`指令，`source/segment0`和`source/segment1`下的`.text*`区段将不会与其他`.text*`区段匹配。

   ```
   *(EXCLUDE_FILE(
            ./source/segment0/*.o ./source/segment1/*.o
         ) .text*)
   ```

2. 通过使用`OVERLAY`指令，所有放置在`BOARD_FLASH`中的`source/segment0`和`source/segment1`下的代码，都将被加载到`SRAM_ITC_cm7`中，并定义为`segment0/1`。

当一个区段中的某个函数被调用时，系统会根据要求将一个区段从BOARD_FLASH复制到SRAM_ITC_cm7中。

```
OVERLAY : NOCROSSREFS
{
segment0 { ./source/segment0/*.o(.text*) }
segment1 { ./source/segment1/*.o(.text*) }
} > SRAM_ITC_cm7 AT>BOARD_FLASH
```

注：

segment0和segment1中的任何函数都不能互相调用。

启用NOCROSSREFS选项后，会在链接时会导致"禁止交叉引用"的错误。

3. 通过使用PROVIDE指令，定义了load_size_segment0和__load_size_segment1两个标号，以便开发人员了解各个区段的大小。
   __load_start_segment0和__load_start_segment1标号是自动定义的，用于确定每个区段在ROM中的位置，并在第3节中使用。__load_size_segment0/1的定义如下：

```
PROVIDE(_load_size_segment0 = SIZEOF(segment0));
PROVIDE(_load_size_segment1 = SIZEOF(segment1));
```

   ROM和RAM地址在链接时是已知的。在映射文件中会显示如下内容：

```
segment0        0x00000000        0x30 load address 0x300085a8
 ./source/segment0/*.o(SORT_BY_ALIGNMENT(.text*))
 .text.task0     0x00000000        0x28 ./source/segment0/task0.o
                 0x00000000                 task0
 .text.task0.__stub
                 0x00000028         0x8 linker stubs
                 0x300085a8                 PROVIDE (_load_start_segment0 = LOADADDR (segment0))
                 [!provide]                 PROVIDE (_load_stop_segment0 = (LOADADDR (segment0) +
 SIZEOF (segment0)))
segment1        0x00000000        0x30 load address 0x300085d8
 ./source/segment1/*.o(SORT_BY_ALIGNMENT(.text*))
 .text.task1     0x00000000        0x28 ./source/segment1/task1.o
                 0x00000000                 task1
 .text.task1.__stub
                 0x00000028         0x8 linker stubs
                 0x300085d8                 PROVIDE (_load_start_segment1 = LOADADDR (segment1))
                 [!provide]                 PROVIDE (_load_stop_segment1 = (LOADADDR (segment1) +
 SIZEOF (segment1)))
                 0x00000030                 PROVIDE (__load_size_segment0 = SIZEOF (segment0))
                 0x00000030                 PROVIDE (__load_size_segment1 = SIZEOF (segment1))
```

segment0、segment1和相关标号均已明确定义。代码段信息也可以从映射文件中获取，具体如表2所示。

表2. 代码段信息

代码段	RAM地址	ROM地址	大小
0	0x0	0x3000085A8	0x30
1	0x0	0x3000085D8	0x30

2.1.2 Keil µVision IDE

以下步骤覆盖了source/segment0和source/segment1中的所有代码。链接器脚本是基于MIMXRT1176xxxxx_cm7_flexspi_nor.scf的，其将所有代码放置在外部闪存中。

1. 通过使用pragma指令，可以指定C源文件中的默认段。source/segment0/task0.c中默认段的定义如下：

```
#pragma clang section text="segment0"
```

2. 通过使用OVERLAY属性，RW_m_segment0/1被加载到同一地址。与GCC不同，这里使用的是代码段名(segment0/1)，而不是目录名。
   ScatterAssert函数可防止存放在ITCM中的代码超出ITCM的容量。具有OVERLAY属性的执行区域定义如下：

```
 RW_m_segment0 m_qacode_start OVERLAY
 {
   * (segment0)
 }
 RW_m_segment1 m_qacode_start OVERLAY
 {
   * (segment1)
 }
 RW_m_ram_text +0 { ;
   * (CodeQuickAccess)
 }
 ScatterAssert((LoadLength(RW_m_segment0) + LoadLength(RW_m_ram_text)) <
m_qacode_size)
 ScatterAssert((LoadLength(RW_m_segment1) + LoadLength(RW_m_ram_text)) <
m_qacode_size)
```

   注：
   Armlink中没有与GCC的NOCROSSREFS选项相对应的选项。
   为了检测无效的交叉引用，必须创建一个自定义脚本来解析Armlink生成的交叉引用信息。
3. ROM和RAM地址在链接时是已知的。在映射文件中会显示如下内容：

```
 Load Region LR_m_text (Base: 0x30000400, Size: 0x00005900, Max: 0x03fbfc00,
ABSOLUTE)
   Execution Region RW_m_segment0 (Exec base: 0x00000000, Load base: 0x30005c18,
Size: 0x0000004c, Max: 0xffffffff, OVERLAY)
   Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name
  Object
    0x00000000   0x30005c18   0x0000000a   Ven    RO          761    Veneer$$Code
  anon$$obj.o
    0x0000000a   0x30005c22   0x00000006   PAD
    0x00000010   0x30005c28   0x0000003c   Code   RO          623    segment0
  task0.o
   Execution Region RW_m_segment1 (Exec base: 0x00000000, Load base: 0x30005c68,
Size: 0x0000004c, Max: 0xffffffff, OVERLAY)
   Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name
  Object
    0x00000000   0x30005c68   0x0000000a   Ven    RO          762    Veneer$$Code
  anon$$obj.o
    0x0000000a   0x30005c72   0x00000006   PAD
   0x00000010   0x30005c78   0x0000003c   Code   RO          632    segment1
 task1.o
```

segment0和segment1均已明确定义。代码段信息也可以从映射文件中获取，具体如表3所示。

表3. 代码段信息

代码段	RAM地址	ROM地址	大小
0	0x10	0x300005C28	0x3C
1	0x10	0x300005C72	0x3C

2.1.3 IAR Embedded Workbench for Arm

以下步骤覆盖了source/segment0和source/segment1中的所有代码。链接器脚本是基于MIMXRT1176xxxxx_cm7_flexspi_nor.icf的，其将所有代码放置到外部闪存中。

1. 与Keil µVision IDE类似，通过使用`pragma`指令，可以指定C源文件中的默认段。
   `source/segment0/task0.c`中默认段的定义如下：

```
#pragma default_function_attributes = @ "segment0"
```

2. 通过使用`define overlay`指令，将`segment0/1`定义为`Overlay`。
   通过使用`initialize manually`指令，将此段分割为初始化程序段和初始化数据段两部分。这些部分在启动时不会自动处理初始化。更多信息，请参阅《IAR C/C++开发指南》。
   `Overlay`的命名定义如下：

```
define overlay Overlay { section segment0 };
define overlay Overlay { section segment1 };
initialize manually { section segment0, section segment1 };
```

3. 通过使用`place`指令，将已命名的`Overlay`放置在`QACODE_region` (ITCM)中。`Overlay`的具体位置如下：

```
place in QACODE_region              { overlay Overlay, block QACCESS_CODE };
```

   ROM和RAM地址在链接时是已知的。在映射文件中会显示如下内容：

```
  Section                Kind           Address    Size  Object
  -------                ----           -------    ----  ------
"P8":                                              0x48
  Overlay                                   0x0    0x24  <Overlay>
    part 1:
      Overlay:1-1                           0x0    0x24  <Init block>
        Veneer           inited             0x0     0x8  - Linker created -
        segment0         inited             0x8    0x1c  task0.o [7]part 2:
      Overlay:2-1                           0x0    0x24  <Init block>
        Veneer           inited             0x0     0x8  - Linker created
        -segment1        inited             0x8    0x1c  task1.o [8]
............
...............
  segment0 init                     0x3000'6520    0x24  <Block>
    Initializer bytes    const      0x3000'6520    0x24  <for Overlay:1-1>
  segment1_init                     0x3000'6544    0x24  <Block>
    Initializer bytes    const      0x3000'6544    0x24  <for Overlay:2-1>
```

`segment0`和`segment1`均已明确定义。段信息也可以从映射文件中获取，具体如表4所示。

表4. 代码段信息

代码段	RAM地址	ROM地址	大小
0	0x0	0x300006520	0x24
1	0x0	0x300006544	0x24

3 编程界面

在第2节中，已经从映射文件中获取了RAM地址、ROM地址及其大小。开发人员可以利用链接器定义的标号来引用代码段信息。虽然这些定义的标号会根据所用的工具链而有所不同，但其基本概念是相同的。

代码段信息表的定义如下：

```
typedef struct _segment_table_t
{
```

```
    uint32_t* ram_addr;
    uint32_t* rom_addr;
    uint32_t size;
} segment_table_t;

#if defined(_CC_ARM) || defined(_ARMCC_VERSION)
extern uint32_t Image$$RW_m_segment0$$Base[];
extern uint32_t Load$$RW_m_segment0$$Base[];
extern uint32_t Image$$RW_m_segment0$$Length[];
extern uint32_t Image$$RW_m_segment1$$Base[];
extern uint32_t Load$$RW_m_segment1$$Base[];
extern uint32_t Image$$RW_m_segment1$$Length[];
#define SEGMENT0_RAM_ADDR Image$$RW_m_segment0$$Base
#define SEGMENT0_ROM_ADDR Load$$RW_m_segment0$$Base
#define SEGMENT0_SIZE     (uint32_t)Image$$RW_m_segment0$$Length
#define SEGMENT1_RAM_ADDR Image$$RW_m_segment1$$Base
#define SEGMENT1_ROM_ADDR Load$$RW_m_segment1$$Base
#define   SEGMENT1_SIZE   (uint32_t)Image$$RW_m_segment1$$Length
#elif defined(__MCUXPRESSO)
extern  uint32_t___base_SRAM_ITC_cm7[];
extern uint32_t__load_start_segment0[];
extern uint32_t___load_stop_segment0[];
extern uint32_t___load_size_segment0[];
extern uint32_t__load_start_segment1[];
extern uint32_t___load_stop_segment1[];
extern uint32_t__load_size_segment1[];
#define SEGMENT0_RAM_ADDR__base_SRAM_ITC_cm7
#define SEGMENT0_ROM_ADDR__load_start_segment0
#define SEGMENT0_SIZE     (uint32_t)_load_size_segment0
#define SEGMENT1_RAM_ADDR__base_SRAM_ITC_cm7
#define SEGMENT1_ROM_ADDR__load_start_segment1
#define SEGMENT1_SIZE     (uint32_t)_load_size_segment1
#elif defined(_ICCARM_) || defined(_GNUC__)
#pragma section = "Overlay"
#pragma section = "segment0_init"
#pragma section = "segment1_init"
#define SEGMENT0_RAM_ADDR__section_begin("Overlay")
#define SEGMENT0_ROM_ADDR___section_begin("segment0_init")
#define SEGMENT0_SIZE     _section_size ("segment0_init")
#define SEGMENT1_RAM_ADDR__section_begin("Overlay")
#define SEGMENT1_ROM_ADDR___section_begin("segment1_init")
#define SEGMENT1_SIZE     __section_size ("segment1_init")
#endif


segment_table_t segment_table[SEGMENTNUM] =
{
    {SEGMENT0_RAM_ADDR, SEGMENT0_ROM_ADDR, SEGMENT0_SIZE},
    {SEGMENT1_RAM_ADDR, SEGMENT1_ROM_ADDR, SEGMENT1_SIZE},
};
```

现代计算机以冯·诺依曼架构为基础。换句话说，代码本质上就是数据。因此，可以使用`memcpy()`函数轻松地将代码从ROM复制到RAM。此外，为了避免因预取的旧代码而导致的意外行为，必须调用DSB和ISB指令。如果内核忙于其他任务，可用DMA来减轻内核的负载。

下面展示了如何将代码从ROM加载到RAM：

```
static void load_code(segment_index_t index)
{
    memcpy(_segment_table[index].ram_addr,__segment_table[index].rom_addr,
 __segment_table[index].size);
    __DSB();
    __ISB();
}
```

***注：**在启用缓存时，如果使用了OCRAM而不是ITCM，则在复制操作后必须使指令缓存失效。*

`main()`函数如下所示。`segment0`包含`task0()`函数，`segment1`包含`task1()`函数，且`CodeQuickAccess`包含`task2()`函数。

```
int main(void) {
    /* Init board hardware. */
    BOARD_ConfigMPU();
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();

    while (1) {
        load_code(SEGMENT0);     // Dynamically load code in SEGMENT0
        task0();
        load_code(SEGMENT1);     // Dynamically load code in SEGMENT1
        task1();
        task2();
        PRINTF(" Press any key to start again.\r\n\r\n");
        GETCHAR();
    }
}
```

每个任务都会打印其地址和静态变量的值，以验证即使其他代码将此代码覆盖，这些值是否仍然被正确保存。以下显示了`task0`打印其地址和静态变量值的情况：

```
void task0(void) {
    static uint32_t count;
    PRINTF("task0 at %p (count = %d)\r\n", &task0, count++);
}
```

4 运行演示

本演示在RT1170-EVKB上运行，并使用MCUXpresso IDE进行测试。

要运行此演示，请执行以下步骤：

1. 将主机PC与目标板中的OpenSDA USB端口用一个USB线相连接。
2. 打开串行终端，配置如下：
   - 115,200波特率；
   - 8个数据位
   - 无奇偶校验位
   - 1个停止位
   - 无流量控制
3. 将程序下载到目标板。
4. 要开始运行演示时，请按下电路板上的复位按钮或在IDE中启动调试器。

图2所示为串行终端的输出。`task0`和`task1`是从ITCM中的同一地址获取的。`task2`也是从ITCM获取的。即使代码被动态加载或卸载，静态变量也能被准确保存下来。

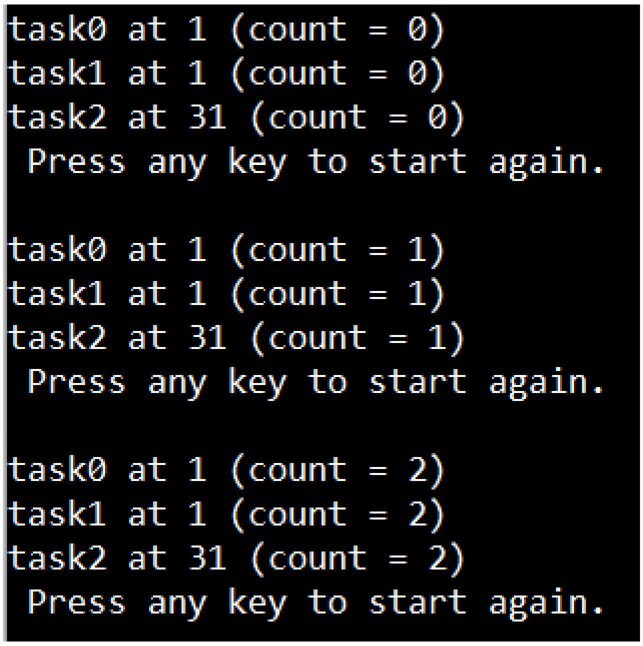

图2. 演示中的串行终端

注： *"`task0 at 1`"表示函数指针的值为0x1。然而，表2却要求物理地址为0x0。出现这种不匹配的原因是函数指针的最低有效位（LSB）被设置为指向Thumb指令。*

5 基准测试

表5所示为数据在DTCM中的情况下各区段的CoreMark测试结果。`segment0/1`中的代码执行速度与`CodeQuickAccess`一样快。而`.text*`中代码的执行速度比其他区段慢了大约四倍。

表5. 各个段的CoreMark

部分	CoreMark
`.text*`	1145
`segment0`	4024
`segment1`	4024
`CodeQuickAccess`	4049

注： *当在.text*区段进行CoreMark测试时，禁用CACHE。*

通过启用CACHE，CoreMark的测试结果会有所提高，但这取决于引用的区域性。

6 结论

i.MX RT系列拥有高性能的内核。然而，如果在小区域引用的情况下从外部存储器获取代码，则内核的性能不会是最佳的。

在某些情况下，使用Overlay技术可有效提升性能，且这种方法很简单，甚至适用于没有任何操作系统的裸机环境。

另一方面，软件管理多个代码段也可能变得复杂，因为程序开发人员必须注意哪个函数位于哪个代码段中。否则，可能会导致运行时错误或链接时的交叉引用错误。

7 参考资料

以下参考资料可用来完善本文档：

- *带Overlay部分的放置*：《Arm编译器armlink用户指南版本6.01》
- Overlay
- *GCC的Overlay代码*：《GCC的Overlay代码》
- *Overlay和手动初始化示例*：《IAR C/C++开发指南》

8 关于本文中源代码的说明

本文中所示的示例代码具有以下版权和BSD-3-Clause许可：

2024年恩智浦版权所有；在满足以下条件的情况下，可以源代码和二进制文件的形式重新分发和使用本源代码（无论是否经过修改）：

1. 重新分发源代码必须保留上述版权声明、这些条件和以下免责声明。
2. 以二进制文件形式重新分发时，必须在文档和/或随分发提供的其他材料中复制上述版权声明、这些条件和以下免责声明。
3. 未经事先书面许可，不得使用版权所有者的姓名或参与者的姓名为本软件的衍生产品进行背书或推广。

本软件由版权所有者和参与者“按原样”提供，不承担任何明示或暗示的担保责任，包括但不限于对适销性和特定用途适用性的暗示保证。在任何情况下，无论因何种原因或根据何种法律条例，版权所有者或参与者均不对因使用本软件而导致的任何直接、间接、偶然、特殊、惩戒性或后果性损害（包括但不限于采购替代商品或服务；使用损失、数据损失或利润损失或业务中断）承担责任，无论是因合同、严格责任还是侵权行为（包括疏忽或其他原因）造成的，即使事先被告知有此类损害的可能性也不例外。

9 修订历史

表6总结了本文档的修订情况。

表6. 修订历史

文档ID	发布日期	说明
AN14260 v.1.0	2024年4月10日	首次公开发布

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.cn/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.

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.

IAR — is a trademark of IAR Systems AB.

i.MX — is a trademark of NXP B.V.

目录

Date of release: 10 April 2024
Document identifier: AN14260