AN14260

Dynamic Loading of Code by Overlay

Rev. 1.0 — 10 April 2024 Application note

Document information

Information	Content
Keywords	AN14260, overlay, linker, performance optimization, code execution from RAM, GCC, EWEARM, Keil
Abstract	In this application note, overlay is introduced to show that it is still effective for modern microcontrollers to improve performance.

1 Introduction

Overlay is a feature of the linker that loads different code at the same address. It has been a popular technique in the early home computer systems, which lacked enough memory to load entire code in it at the same time.

In this application note, overlay is introduced to show that it is still effective for modern microcontrollers to improve performance. It is even applicable to BareMetal without any operating system.

1.1 Overview

i.MX RT series microcontrollers support high-core frequency as shown in Table 1.

Table 1. i.MX RT series supports high-core frequency

Products	Maximum core frequency
RT1010/1020	500 MHz
RT1040/1050/1060	600 MHz
RT1180	800 MHz
RT1170	1000 MHz

These products support high-core frequency. However, they do not have internal flash because it is difficult to miniaturize the flash at the same standard as the core. It means that the internal flash cannot be manufactured using the same process as the core. Therefore, external memory is required to store the code for the i.MX RT series.

While XIP is supported for external memory, it is slower than on-chip RAM[1]. Therefore, ITCM is the best location to fetch code in terms of performance because read access is expected to finish in one cycle[2].

Unfortunately, if RAM size is insufficient for an application, there is no choice but to locate code in the external memory. However, the external memory bandwidth is slower than the instruction fetch bandwidth at high-core frequency. Therefore, even if cache is enabled, under low locality of reference, it results in worse performance. Furthermore, TCM size is relatively small because high-core frequency sacrifices TCM size due to the longer signal delay by the bigger TCM area.

Therefore, by loading the code dynamically, limited TCM can be used at the best. The performance is improved because the core does not have to fetch code from external memory every time. Moreover, if less ITCM is required for an application, DTCM can be expanded in FlexRAM, which also leads to better performance.

For the sample project attached, code is dynamically loaded from external flash to ITCM on RT1170-EVKB by MCUXpresso IDE, IAR Embedded Workbench for Arm, or Keil µVision IDE.

The concrete specification is as follows:

- The section `segment0` and the section `segment1` are loaded dynamically from the external flash to ITCM, as shown in Figure 1.
  - Each segment is loaded by calling `load_code()` on demand.
  - When `segment0` is loaded, any function in `segment1` cannot be called and vice versa.
  - Any function in `segment0` cannot call any function in `segment1` and vice versa.
- The section `CodeQuickAccess` is statistically located in ITCM.
- Other code in the section `.text*` is statistically located in the external flash.

1 *i.MX RT Series Performance Optimization* (document AN12437)
2 *Using the i.MX RT FlexRAM* (document AN12077)

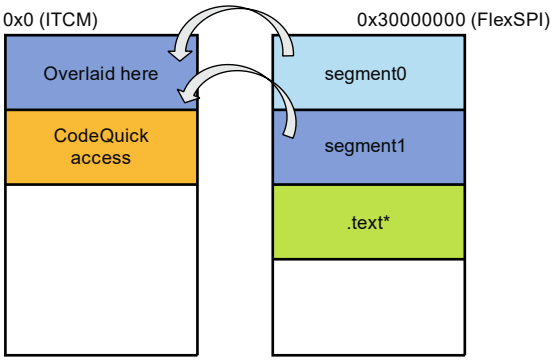

Figure 1. Memory map

2 Software implementation overview

This section describes the software implementation for MCUXpresso IDE (GCC), Keil µVision IDE, and IAR embedded workbench for Arm.

2.1 Linker script

A linker script describes where the object is located and where it is to be loaded at the execution time. The linker script must be customized to meet the requirements of the application.

2.1.1 MCUXpresso IDE (GCC)

Assume that the directory structure is as follows. The key point is that the source directory has `segment0` and `segment1` directories.

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

All the code in `source/segment0` and `source/segment1` is defined as `segment0/1` and overlaid by the following three steps. The linker script is based on `evkbmimxrt1170_hello_world_demo_cm7_Debug.ld`, which locates all the code in the external flash.

1. By using the `EXLUDE_FILE` directive, the `.text*` section under `source/segment0`, and `source/segment1` are excluded from matching with the `.text*` section.

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

2. By using the `OVERLAY` directive, all the code under `source/segment0` and `source/segment1` is in `BOARD_FLASH` and is to be loaded in `SRAM_ITC_cm7` and are defined as `segment0/1`.

In each segment, for a function to be called, software must copy a segment from BOARD_FLASH to SRAM_ITC_cm7 on demand.

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

Note:
Any function in segment0 cannot be called from any function in segment1 and vice versa.
With the NOCROSSREFS option enabled, it causes an error "prohibited cross-reference" at linking time.

3. By using the PROVIDE directive, the two symbols, __load_size_segment0, and __load_size_segment1 are defined to let the programmer know the section size. __load_start_segment0 and __load_start_segment1 are automatically defined to determine the location of each segment in the ROM and are used in the Section 3. The definition of the __load_size_segment0/1 is as follows:

```
PROVIDE(__load_size_segment0 = SIZEOF(segment0));
PROVIDE(__load_size_segment1 = SIZEOF(segment1));
```

ROM and RAM addresses are known at linking time. It can be seen in the map file as follows:

```
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))
```

The segment0, segment1, and symbols are defined properly. Segment information can be retrieved from the map file, as shown in Table 2.

Table 2. Segment information

Segment	RAM address	ROM address	Size
0	0x0	0x3000085A8	0x30
1	0x0	0x3000085D8	0x30

2.1.2 Keil µVision IDE

The following steps overlay all the code in source/segment0 and source/segment1. The linker script is based on MIMXRT1176xxxxx_cm7_flexspi_nor.scf, which locates all the code in the external flash.

1. By using the pragma directive, a default section in the C source file can be specified. The definition of the default section in source/segment0/task0.c is as follows:

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

2. By using the `OVERLAY` attribute, `RW_m_segment0/1` are loaded at the same address. The section name (`segment0/1`) is used instead of the directory name in contrast with GCC.
   The `ScatterAssert` function prevents the code placed to the ITCM from exceeding the size of the ITCM.
   The definition of the execution region with the `OVERLAY` attribute is as follows:

```
 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)
```

   Note:
   Armlink does not have a counterpart of the NOCROSSREFS option in GCC.
   A custom script to interpret cross-reference information from the armlink must be created to detect invalid cross-references.

3. ROM and RAM addresses are known at linking time. It can be seen in the map file as follows:

```
  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
```

The `segment0` and `segment1` are defined properly. Segment information can be retrieved from the map file, as shown in Table 3.

Table 3. Segment information

Segment	RAM address	ROM address	Size
0	0x10	0x300005C28	0x3C
1	0x10	0x300005C72	0x3C

2.1.3 IAR embedded workbench for Arm

The following steps overlay all the code in `source/segment0` and `source/segment1`. The linker script is based on `MIMXRT1176xxxxx_cm7_flexspi_nor.icf`, which locates all the code in the external flash.

1. Similar to Keil µVision IDE, by using a `pragma` directive, a default section in the C source file can be specified.
   The definition of the default section in `source/segment0/task0.c` is as follows:

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

2. By using the `define overlay` directive, `segment0/1` is defined as `Overlay`.
   By using the `initialize manually` directive, the section is split into sections for initializers and initialized data. The initialization is not handled automatically at the startup. For more information, refer IAR C/C++ Development Guide.
   The definition of the named `Overlay` is as follows:

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

3. By using the `place` directive, the named `Overlay` is in the `QACODE_region` (ITCM). The placement of the `Overlay` is as follows:

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

   ROM and RAM addresses are known at linking time. It can be seen in the map file as follows:

```
  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>
```

The `segment0` and `segment1` are defined properly. Segment information can be retrieved from the map file, as shown in Table 4.

Table 4. Segment information

Segment	RAM address	ROM address	Size
0	0x0	0x300006520	0x24
1	0x0	0x300006544	0x24

3 Programming interface

In Section 2, RAM address, ROM address, and its size have been retrieved from the map file. For a programmer, linker-defined symbols can be used to refer to the segment information. The defined symbols depend on the tool chain, but the basic concept is common.

The definition of the segment information table is as follows:

```
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},
};
```

The modern computer is based on von Neumann architecture. In other words, code is data. Therefore, code can be easily copied from ROM to RAM by using the `memcpy()` function. Also, DSB and ISB instructions must be called to avoid unexpected behavior caused due to the old code prefetched. If the core is busy for other tasks, DMA can be used to reduce the core loading.

The following shows how to load the code from ROM to 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();
}
```

Note: *Instruction cache must be invalidated after copy if OCRAM is used instead of ITCM when cache is enabled.*

The `main()` function is given below. The `segment0` has `task0()` function and `segment1` has `task1()` function and `CodeQuickAccess` has `task2()`.

```
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();
    }
}
```

Each task prints its address and static variable value to check whether the values are properly held even if the other code overrides the code. The following shows the `task0` prints its address and static variable value:

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

4 Running the demo

This demo runs on RT1170-EVKB and the MCUXpresso IDE is used for testing.

To run the demo, perform the following steps:

1. Connect a USB cable between the host PC and the OpenSDA USB port on the target board.
2. Open a serial terminal with the following settings:
   - 115,200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. To begin running the demo, either press the reset button on the board or launch the debugger in the IDE.

Figure 2 shows a serial terminal output. The `task0` and `task1` are fetched from the same address in ITCM. The `task2` is also fetched from ITCM. Static variables are properly held even if the code is dynamically loaded or unloaded.

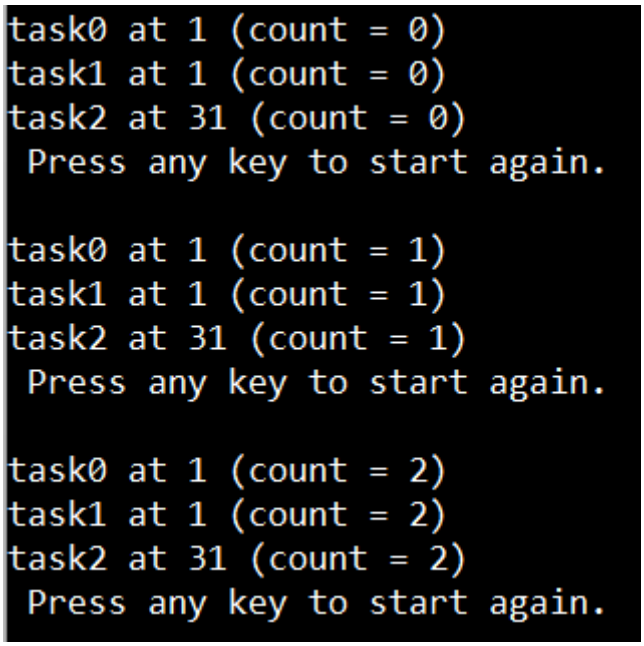

Figure 2. Serial terminal on the demo

Note: *The value "`task0 at 1`" indicates that the function pointer has a value of 0x1. However, Table 2 requires the physical address to be 0x0. This mismatch occurs because the least significant bit (LSB) of the function pointer is set to indicate that it points to a Thumb instruction.*

5 Benchmark test

Table 5 shows the CoreMark result in each section on the condition that the data is in DTCM. The code in `segment0/1` is as fast as the code in `CodeQuickAccess`. Code in `.text*` is about four times slower than other section.

Table 5. CoreMark in each section

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

Note: *ICACHE is disabled when CoreMark is measured in `.text*` section.*

It is improved by enabling ICACHE, but it depends on the locality of reference.

6 Conclusion

The i.MX RT series has a high-performance core. However, if code is fetched from external memory under low locality of reference, the core performance is not the best.

For some cases, overlay is effective to improve performance and is such a simple method that it is applicable even to BareMetal without any operating system.

On the other hand, it can get complex for software to manage many segments because the programmer must pay attention to which function is in which segment. Otherwise, it causes a runtime error or cross-reference error at linking time.

7 References

The references used to supplement this document are as follows:

- *Placement of sections with overlays*: ARM Compiler armlink User Guide Version 6.01
- Overlays
- *Overlay Code with GCC*: Overlay Code with GCC
- *Overlay and manual initialization example*: IAR C/C++ Development Guide

8 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

9 Revision history

Table 6 summarizes the revisions to this document.

Table 6. Revision history

Document ID	Release date	Description
AN14260 v.1.0	10 April 2024	Initial public release

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.

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.

Contents

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

Date of release: 10 April 2024
Document identifier: AN14260