AN12384

LPC5460x UART Secondary Bootloader using YModem

Rev. 0 — April 2019 Application Note

Contents

1 Abstract

A simple Secondary Bootloader (SBL) software is designed in this application note. It can optionally load the new image from user UART terminal. The YModem file transfer protocol is used in this design to transfer the image file from PC to the board. In-Application Programming (IAP) feature is used to download the firmware's image to on-chip FLASH memory. The demo project is developed based on LPCXpresso54608 board and MCUXpresso SDK Software Library.

2 Overview

SBL is a small piece of code immediately running after the hardware boot process. It works as the normal software, but only dealing with some customized tasks before the user software, for example to do some previous configuration for the user system. It exists like the normal bootloader which is kept in ROM and used to boot the system, but can execute the additional customized booting task following the normal (first) bootloader. The Secondary Bootloader is still a part of software. So, it is more flexible than the ROM bootloader which is stable and can not modified by customer.

In this application note, a simple SBL software is designed to optionally load the new image file from user UART terminal and keep it in the on-chip flash memory for next run. Therefore, the design is also called **UART Flashloader with IAP**. The YModel file transfer protocol is used in this design to send the binary image file from PC to the board. The demo project is developed based on LPCXpresso54608 board and MCUXpresso SDK Software Library.

3 Hardware platform based on LPC54608 MCU

3.1 LPC54608 MCU

The LPC54808 MCU includes one 180 MHz Arm® Cortex®-M4 core with multiple high-speed connectivity options, advanced timers, and analog features. It integrates 512 KB FLASH and and totally 200 KB SRAM on-chip and the external memory interfaces. In this application note, the FLASH memory is the primary target to be operated. Table 1. Flash memory summarization on page 1 summerizes the FLASH memory as the basic knowledge for the following design.

The Arm Cortex-M4 processor has a single 4 GB address space. The `0x0000_0000` to `0x3FFF_FFFF` is for the on-chip FLASH and SRAM

Table 1. Flash memory summarization

Adress range	General use	Address range details and description	
`0x0000 0000` to `0x1FFF FFFF`	On-chip non-volatile memory	`0x0000 0000-0x0007 FFFF`	Flash memory (512 KB)
	Boot ROM	`0x0300 0000-0x0300 FFFF`	Boot ROM with flash services in a 64 KB space

Table continues on the next page...

Table 1. Flash memory summarization (continued)

	SRAMX	`0x0400 0000-0x0400 7FFF`	I&D SRAM bank (32 KB)
	SPI Flash Interface (SPIFI)	`0x1000 0000-0x17FF FFFF`	SPIFI memory mapped access space (128 MB)
`0x2000 0000` to `0x3FFF FFFF`	SRAM bank	`0x2000 0000-0x2002 7FFF`	SRAM bank (160 KB)
	SRAM bit band alias addressing	`0x2200 0000-0x23FF FFFF`	SRAM bit band alias addressing (32 MB)

The 512-KB programmable FLASH is mapped to the address from start to `0x0007_FFFF`, and the SRAM are mapped to three parts:

- the 32 KB SRAMX from `0x0400_0000` to `0x0400_7FFF`,
- the 160 KB normal SRAM from `0x2000_0000` to `0x2002_7FFF`,
- the 8 KB USB SRAM from `0x4010_0000` to `0x4010_1FFC` (not listed in Table 1. Flash memory summarization on page 1).

For the on-chip FLASH memory, it can be performed erase and write operations directly by the end-user application through the IAP. Some IAP commands operate on sectors and specify sector numbers.The size of a sector is 32 KB (a normal size for an erase operation) and the size of a page is 256 Byte (the minimal size for a write operation). One sector contains 128 pages. Sector **0** and page **0** are located at address `0x0000_0000`. Table 2. Mapping between sector and address on page 2 descirbes the mapping between sector number and address.

Table 2. Mapping between sector and address

Sector number	Sector size	Page numbers	Address range	Total flash (including this sector)
0	32 KB	0 - 127	`0x0000 0000 - 0x0000 7FFF`	32 KB
1	32 KB	128 - 255	`0x0000 8000 - 0x0000 FFFF`	64 KB
2	32 KB	256 - 383	`0x0001 0000 - 0x0001 7FFF`	96 KB
3	32 KB	384 - 511	`0x0001 8000 - 0x0001 FFFF`	128 KB
4	32 KB	512 - 639	`0x0002 0000 - 0x0001 7FFF`	160 KB
5	32 KB	640 - 767	`0x0002 8000 - 0x0002 FFFF`	192 KB
6	32 KB	768 - 895	`0x0003 0000 - 0x0003 7FFF`	224 KB
7	32 KB	896 - 1023	`0x0003 8000 - 0x0003 FFFF`	256 KB
8	32 KB	1024 - 1151	`0x0004 0000 - 0x0004 7FFF`	288 KB
9	32 KB	1152 -1279	`0x0004 8000 - 0x0004 FFFF`	320 KB

Table continues on the next page...

Table 2. Mapping between sector and address (continued)

10	32 KB	1280 - 1407	`0x0005 0000` - `0x0005 7FFF`	352 KB
11	32 KB	1408 - 1535	`0x0005 8000` - `0x0005 FFFF`	384 KB
12	32 KB	1536 - 1663	`0x0006 0000` - `0x0006 7FFF`	416 KB
13	32 KB	1664 - 1791	`0x0006 8000` - `0x0006 FFFF`	448 KB
14	32 KB	1792 - 1919	`0x0007 0000` - `0x0007 7FFF`	480 KB
15	32 KB	1920 - 2047	`0x0007 8000` - `0x0007 FFFF`	512 KB

To erase and write the on-chip FLASH in user application code, NXP MCUXpresso SDK Software Library already provides the IAP driver. The IAP driver would be used to handle the on-chip FLASH in this application demo.

In this application demo, the LPCXpresso54608 Board with the LPC54608J512BD208 chip is used to run the executable image and verify the software.

4 Software design

4.1 Boot to new firmware in application

The most critical point in the development of SBL software is to implement the **boot** operation, which prepares the running environment and jumps to a new application within another separated image.

Actually, there are two images for SBL software and user software individually. The SBL software's image is downloaded to the chip previously, and the user software's image is sent to the MCU by the communication between PC and SBL software running on the MCU. Then, the SBL writes the user software's image to the on-chip flash, jumps to it and runs.

Fortunately, the boot environment for Arm MCU is simple, just with a vector table. The vector table includes entries for various routine, for example, the most important items are the stack pointer and the reset (boot) handler's entry. The first vector is to keep the initial value for stack pointer. The stack pointer is used to access the stack automatically determined by compiler. The second vector is for the reset handler. The reset handler points to the reset routine and leads to the user `main()` function. All the other items in the vector table are the entries to various exception/interrupt events's routine. By default, the base address of the vector table is `0x0000_0000`, so the images is downloaded to `0x0000_0000` as the start address in most cases. However, it can be remapped to other address in software, by setting the Arm core's `SCB->VTOR` register. If the vector table is remapped, the Arm core travels to the new base address with the item's offset to get the vector once the responding event occurs. The vector table is placed in the front of image file. In the chip's memory space, the first address of an image for the user software (or called firmware) is the base address of vector table.

In the application demo, an API `FwBoot_BootToFwImage()` is created to jump to new firmware.

```
void FwBoot_BootToFwImage(uint32_t fwImageBaseAddr)
{
    void (*firmwareFunc)(void);
    uint32_t fwStackVal = *((uint32_t *)(fwImageBaseAddr));      /* the first word is for the stack 
pointer. */
    uint32_t fwEntryVal = *((uint32_t *)(fwImageBaseAddr+4U));  /* the second works is for the boot 
function. */
    firmwareFunc = (void (*)(void))fwEntryVal;
    SCB->VTOR = fwImageBaseAddr; /* The stack address is also the start address of vector. */
    __set_MSP(fwStackVal); /* setup the stack address for MSP. */
    __set_PSP(fwStackVal); /* setpu the stack address for PSP. */
    firmwareFunc();
}
```

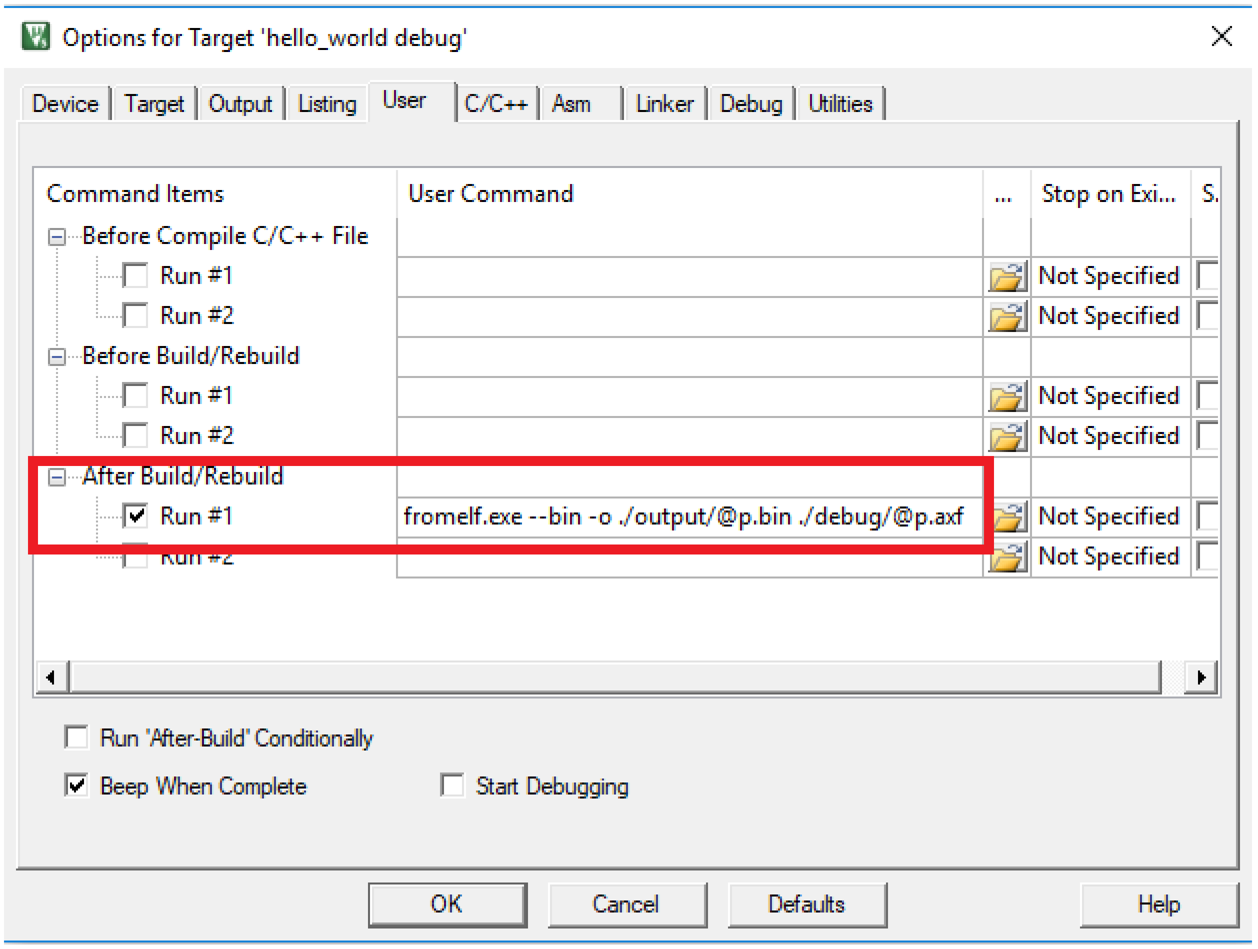
4.2 Format of Arm executable image file

The `*.bin` file is the simplest executable image file format for Arm. It is for the direct RAW binary code without the address tag. It can be written to flash memory without any additional translation. Other format, like `Hex`, `Elf`, or `Axf`, includes some complex information. To load these formats, the flashloader needs to translate the data into RAW binary one. In this application demo, the `bin` format is prefered.

Keil IDE does not output the bin file directly, but the user command in Keil can be processed within the **post build** action:

```
fromelf.exe --bin -o ./output/@p.bin ./debug/@p.axf
```

On the **User** tab of the **Options for Target** dialog box, fill the command into the **After Build/Rebuild** section and check the box to enable it, as shown in Figure 1. on page 4.


Options for Target 'hello_world debug'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Command Items	User Command	...	Stop on Exi...	S.
Before Compile C/C++ File				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
Before Build/Rebuild				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
After Build/Rebuild				
☑ Run #1	fromelf.exe --bin -o ./output/@p.bin ./debug/@p.axf		Not Specified	☐
☐ Run #2			Not Specified	☐

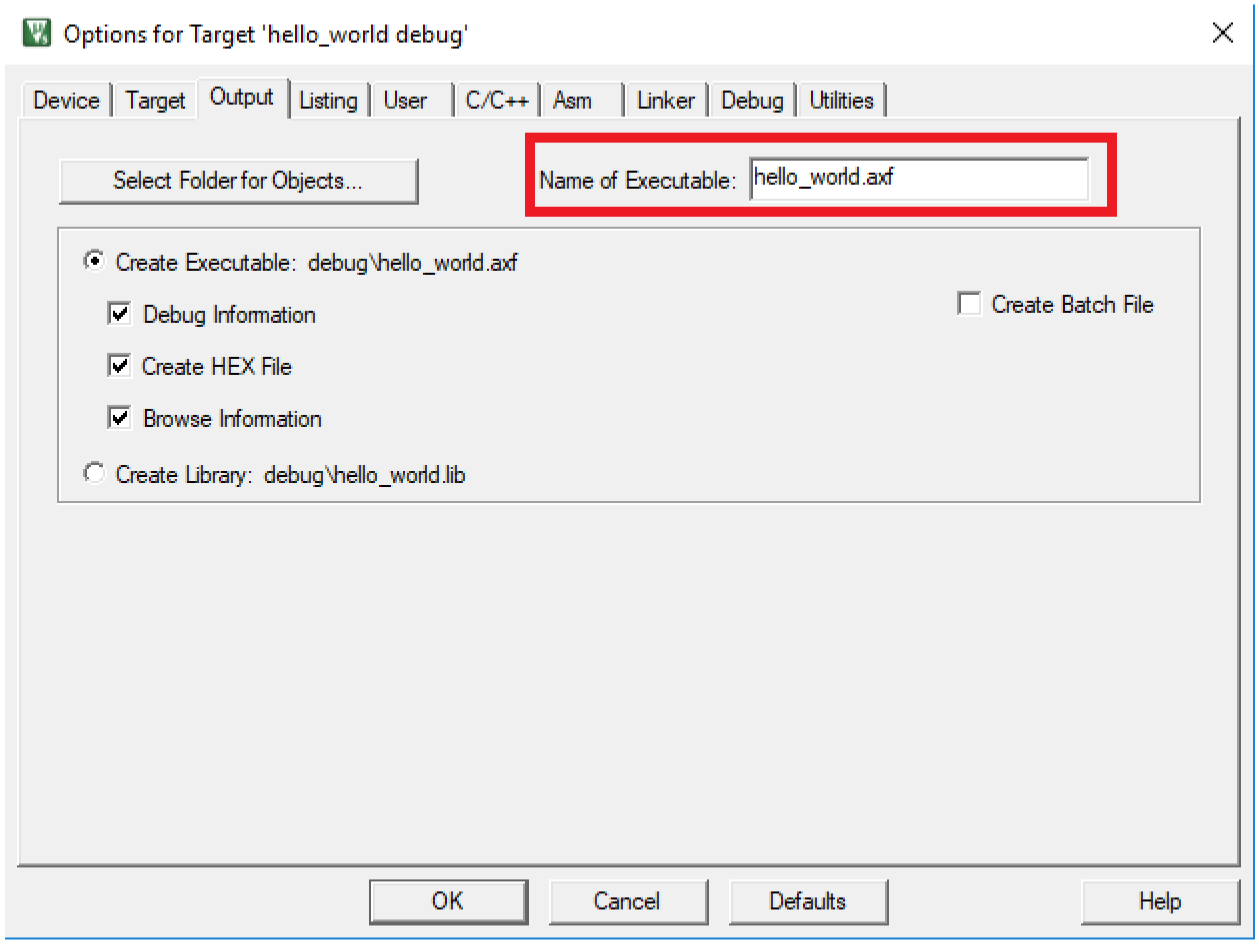
☐ Run 'After-Build' Conditionally

☑ Beep When Complete ☐ Start Debugging

OK | Cancel | Defaults | Help

Figure 1. Keil command for bin file 1

Do not forget to change the original generated image file's name with `axf` as post-fix in the **Output**tab, as shown in Figure 2. on page 5.



Name of Executable: hello_world.axf

(●) Create Executable: debug\hello_world.axf

[✓] Debug Information

[✓] Create HEX File

[✓] Browse Information

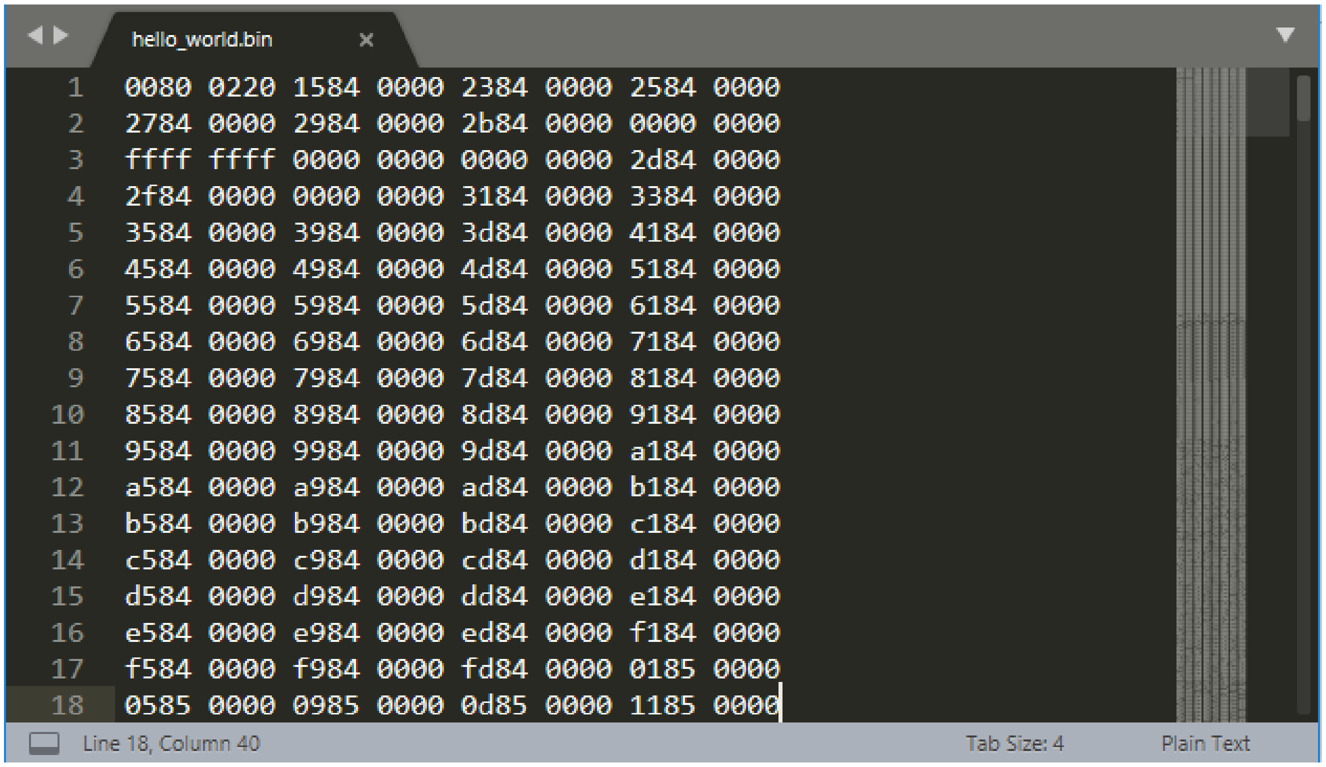
[ ] Create Batch File

( ) Create Library: debug\hello_world.lib

Figure 2. Keil command for bin file 2

Then, once the project is built, an executable bin file is generated and updated automatically for every rebuild..

Taking the `hello_world` project as an example, its executable bin file is as shown in Figure 3. on page 6.



```
0080 0220 1584 0000 2384 0000 2584 0000
2784 0000 2984 0000 2b84 0000 0000 0000
ffff ffff 0000 0000 0000 0000 2d84 0000
2f84 0000 0000 0000 3184 0000 3384 0000
3584 0000 3984 0000 3d84 0000 4184 0000
4584 0000 4984 0000 4d84 0000 5184 0000
5584 0000 5984 0000 5d84 0000 6184 0000
6584 0000 6984 0000 6d84 0000 7184 0000
7584 0000 7984 0000 7d84 0000 8184 0000
8584 0000 8984 0000 8d84 0000 9184 0000
9584 0000 9984 0000 9d84 0000 a184 0000
a584 0000 a984 0000 ad84 0000 b184 0000
b584 0000 b984 0000 bd84 0000 c184 0000
c584 0000 c984 0000 cd84 0000 d184 0000
d584 0000 d984 0000 dd84 0000 e184 0000
e584 0000 e984 0000 ed84 0000 f184 0000
f584 0000 f984 0000 fd84 0000 0185 0000
0585 0000 0985 0000 0d85 0000 1185 0000
```

Figure 3. `hello_world` **bin file**

4.3 Flash memory arrangement

4.3.1 On-chip flash memory for SBL and firmware

The SBL and user software are both running inside the MCU, so the arrangement of these two parts of software is necessary.

SBL starts following the chip hardware booting. Its code is placed from the address `0x0000_0000`, so the chip can recognize and boot it automatically by default. Then, to facilitate the operation to FLASH, a whole sector of 32 KB memory is reserved for the SBL. All the other memory of FLASH is for the user firmware.

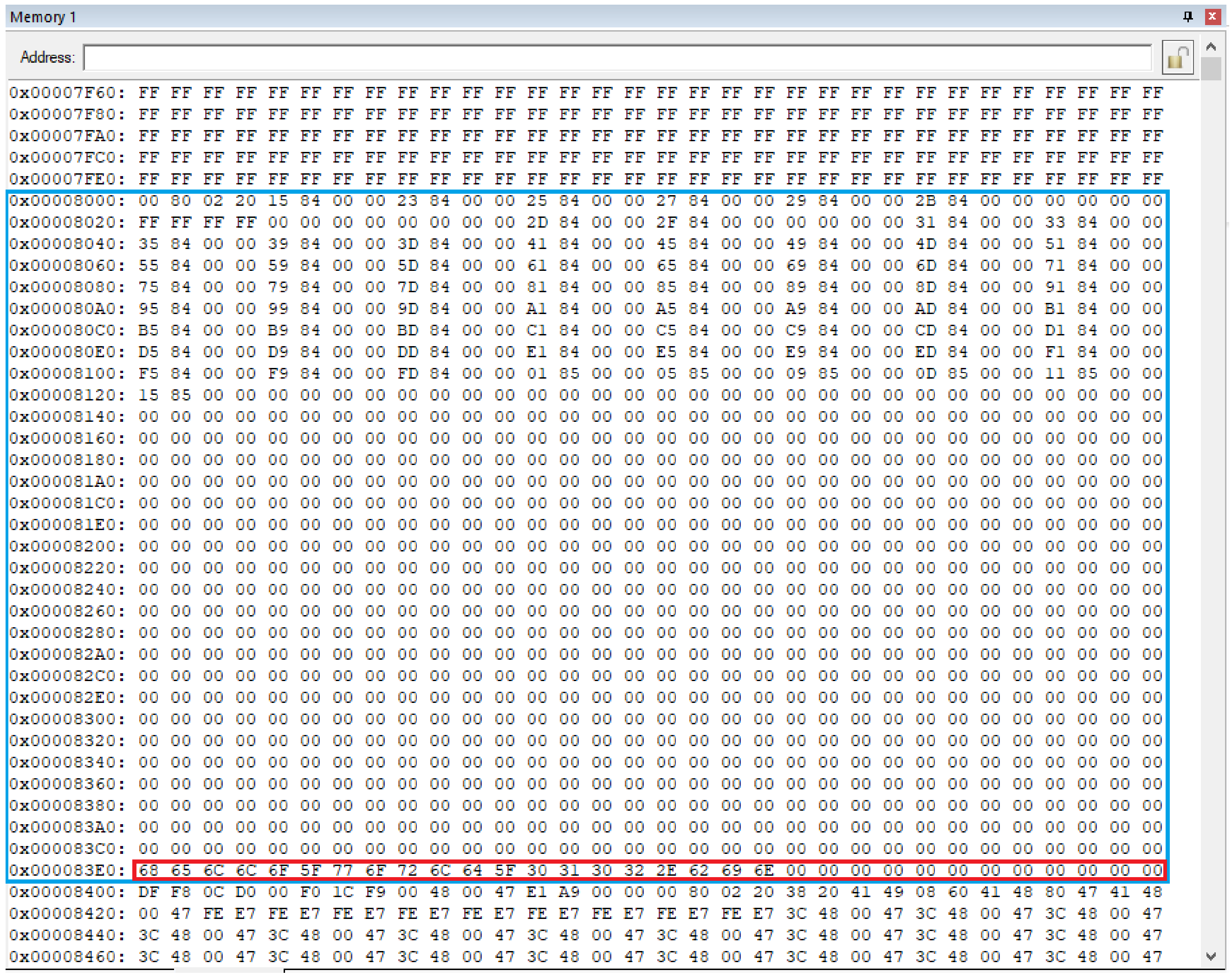
Table 3. On-chip flash memory

Memory address	FLASH sector	Usage
`0x0000_0000` - `0x0000_7FFF`	Sector 0	2nd Bootloader
`0x0000_8000` - `0x0007_FFFF`	Sector 1 - 15	User firmware

The arrangement of these address settings are defined in the demo code.

```
/* for user firmware. */
#define BOOT_FIRMWARE_BASE_ADDRESS (1U * FSL_FEATURE_SYSCON_FLASH_SECTOR_SIZE_BYTES) /*
(1024 * 32) Bytes. */
#define BOOT_FIRMWARE_INFO_OFFSET (1024U-32U) /* The last 32 bytes in
vector table. */
```

Keep the firmware version information in the unused vector items. Arm reserves 256 vectors for Cortex-M4's vector table with 1 KB memory space, but not all the 256 vectors are implemented for a specific chip. For the LPC54608 MCU, only 73 vectors are used, while the tailing 652 Byte are spare in the vector table. In this application demo, the tailing 32 bytes are used to record the file name of the user firmware.



Memory 1

Address:

```
0x00007F60: 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
0x00007F80: 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
0x00007FA0: 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
0x00007FC0: 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
0x00007FE0: 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
0x00008000: 00 80 02 20 15 84 00 00 23 84 00 00 25 84 00 00 27 84 00 00 29 84 00 00 2B 84 00 00 00 00 00 00
0x00008020: FF FF FF FF 00 00 00 00 00 00 00 00 2D 84 00 00 2F 84 00 00 00 00 00 00 31 84 00 00 33 84 00 00
0x00008040: 35 84 00 00 39 84 00 00 3D 84 00 00 41 84 00 00 45 84 00 00 49 84 00 00 4D 84 00 00 51 84 00 00
0x00008060: 55 84 00 00 59 84 00 00 5D 84 00 00 61 84 00 00 65 84 00 00 69 84 00 00 6D 84 00 00 71 84 00 00
0x00008080: 75 84 00 00 79 84 00 00 7D 84 00 00 81 84 00 00 85 84 00 00 89 84 00 00 8D 84 00 00 91 84 00 00
0x000080A0: 95 84 00 00 99 84 00 00 9D 84 00 00 A1 84 00 00 A5 84 00 00 A9 84 00 00 AD 84 00 00 B1 84 00 00
0x000080C0: B5 84 00 00 B9 84 00 00 BD 84 00 00 C1 84 00 00 C5 84 00 00 C9 84 00 00 CD 84 00 00 D1 84 00 00
0x000080E0: D5 84 00 00 D9 84 00 00 DD 84 00 00 E1 84 00 00 E5 84 00 00 E9 84 00 00 ED 84 00 00 F1 84 00 00
0x00008100: F5 84 00 00 F9 84 00 00 FD 84 00 00 01 85 00 00 05 85 00 00 09 85 00 00 0D 85 00 00 11 85 00 00
0x00008120: 15 85 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
0x00008140: 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
0x00008160: 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
0x00008180: 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
0x000081A0: 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
0x000081C0: 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
0x000081E0: 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
0x00008200: 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
0x00008220: 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
0x00008240: 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
0x00008260: 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
0x00008280: 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
0x000082A0: 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
0x000082C0: 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
0x000082E0: 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
0x00008300: 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
0x00008320: 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
0x00008340: 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
0x00008360: 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
0x00008380: 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
0x000083A0: 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
0x000083C0: 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
0x000083E0: 68 65 6C 6C 6F 5F 77 6F 72 6C 64 5F 30 31 30 32 2E 62 69 6E 00 00 00 00 00 00 00 00 00 00 00 00
0x00008400: DF F8 0C D0 00 F0 1C F9 00 48 00 47 E1 A9 00 00 00 80 02 20 38 20 41 49 08 60 41 48 80 47 41 48
0x00008420: 00 47 FE E7 FE E7 FE E7 FE E7 FE E7 FE E7 FE E7 FE E7 3C 48 00 47 3C 48 00 47 3C 48 00 47
0x00008440: 3C 48 00 47 3C 48 00 47 3C 48 00 47 3C 48 00 47 3C 48 00 47 3C 48 00 47 3C 48 00 47 3C 48 00 47
0x00008460: 3C 48 00 47 3C 48 00 47 3C 48 00 47 3C 48 00 47 3C 48 00 47 3C 48 00 47 3C 48 00 47 3C 48 00 47
```

Figure 4. Using spare space in FW vector table

Figure 4. on page 7 shows the binary image files generated from the SBL software and the modified `hello_world` demo project in MCUXpresso SDK software library.

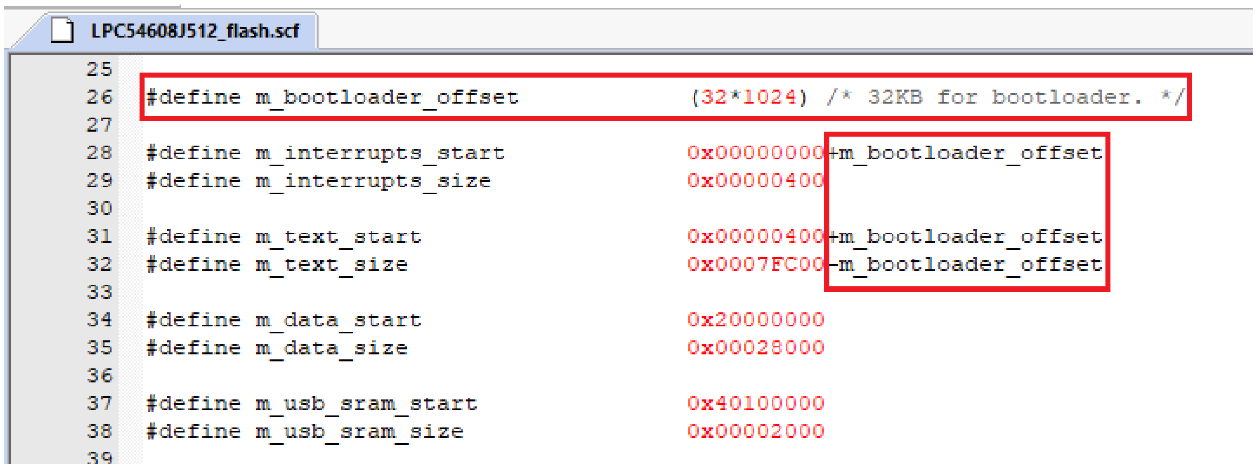
- The FLASH with the address lower than `0x0000_8000` is all `0xFF`. The memory is just erased for SBL space but not used.
- The FLASH with the address beginning from `0x0000_8000` is for the user firmware.
- The FLASH with the address between `0x0000_8000` and `0x0000_8400` is for user firmware's vector table with the length of 1 KB. Only the front items are used as the exeception/interrupt function's entries. The memory for unused vectors is all zero.
- The FLASH with the address between `0x0000_83E0` and `0x0000_8400` are the tailing 16 bytes in user firmware's vector table. In this application demo, the memory is used to keep the firmware's file name. As shown in Figure 4. on page 7, `0x68`, `0x65`, `0x6C`, `0x6C`, `0x6F`, `0x5F`, `0x77`, `0x6F`, `0x72`, `0x6C`, `0x64`, `0x5F`, `0x30`, `0x31`, `0x30`, `0x32`, `0x2E`, `0x62`, `0x69`, and `0x6E` are just the ASICC codes for the string of `hello_world_0102.bin`.

4.3.2 Compiler link configuration

The SBL project's memory mapping is similar to the normal project, which starts the image address from `0x0000_0000`, and no additional setting is needed. As the image of the user firmware project starts from `0x0000_8000` (sector 1), perform the following steps to modify its linker configurations.

1. Update the linker file.

   Add an address offset of `m_bootloader_offset` to reserve the space for bootloader, as shown in Figure 5. on page 8.

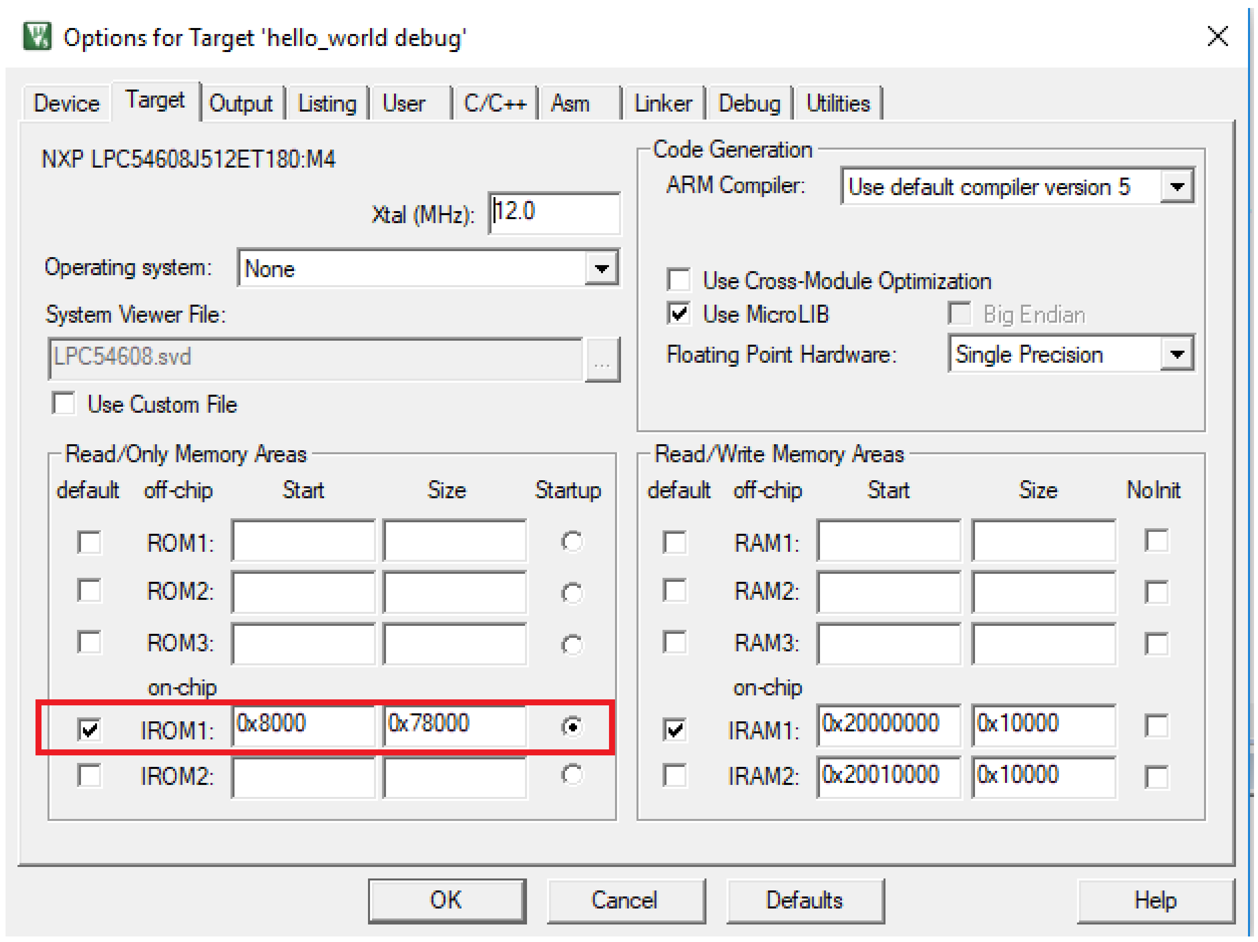


LPC54608J512_flash.scf

```
#define m_bootloader_offset             (32*1024) /* 32KB for bootloader. */

#define m_interrupts_start              0x00000000+m_bootloader_offset
#define m_interrupts_size               0x00000400

#define m_text_start                    0x00000400+m_bootloader_offset
#define m_text_size                     0x0007FC00-m_bootloader_offset

#define m_data_start                    0x20000000
#define m_data_size                     0x00028000

#define m_usb_sram_start                0x40100000
#define m_usb_sram_size                 0x00002000
```

Figure 5. Updating linker file

2. Update the memory setting in IDE.



Options for Target 'hello_world debug'

NXP LPC54608J512ET180:M4

Xtal (MHz): 12.0

Operating system: None

System Viewer File: LPC54608.svd

[ ] Use Custom File

Code Generation

ARM Compiler: Use default compiler version 5

[ ] Use Cross-Module Optimization

[✓] Use MicroLIB [ ] Big Endian

Floating Point Hardware: Single Precision

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
[ ]	ROM1:			○
[ ]	ROM2:			○
[ ]	ROM3:			○
	on-chip			
[✓]	IROM1:	0x8000	0x78000	●
[ ]	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
[ ]	RAM1:			[ ]
[ ]	RAM2:			[ ]
[ ]	RAM3:			[ ]
	on-chip			
[✓]	IRAM1:	0x20000000	0x10000	[ ]
[ ]	IRAM2:	0x20010000	0x10000	[ ]

Figure 6. Updating memory settings in IDE

After the modifications, the user firmware's image may not be downloaded into the chip with the IDE's debug tool. It cannot be debugged unless the 2nd bootloader exists inside the chip.

- If the 2nd bootloader is not ready yet, when entering the debug demo, the chip boots from the address of `0x0000_0000`, but the available code in current project starts from `0x0000_8000`, so the chip can run to current firmware.
- If the 2nd bootloader is ready, the chip boots from the address of `0x0000_0000`, the 2nd bootloader can help to jump to the current firmware, and then the debugger can monitor the statement and catch the break point for the current project.

4.4 YModem file transfer protocol

4.4.1 YModem overview

YModem is a general file transfer protocol for transferring files between PC and embedded system in the embedded development. It is fast and high-efficient to transfer the file with CRC check to make sure the data is right. Also, the receiver sends the ACK to the sender for next package when it is ready to catch that one, and the data stream is under control. It is a typical way to implement the bootloader for MCU as well.

YMODEM-1K uses a block size of one kilobyte instead of the standard 128 bytes. 1 K-blocks is an option in the original YMODEM standard, but this variant neglects the rest of the features, and is best described as a 1 k variant of XMODEM.

In this application demo, a YModem protocol component is created for general use cases. As this component is coded with a pure C language, it can be easily ported to other embedded system.

4.4.2 MCU: Port YModem receiver

In the YModem component, `xymodem.h/xymodem.c` files process the protocol interaction, and users implement the two functions based on the specific hardware platform in the `xymodem_port.c` file.

- `void XYModem_Uart_SendByte`(unsigned `char ch`)

  This function sends out a byte of `unsigned char ch` through the user USART channel. In the application demo code, this function is implemented with the polling method to send out the character through the **Terminal UART** (USART0).

- `int XYModem_Uart_RecvByteTimeout`(unsigned `char *ch`)

  This function receives a byte in an indicated time period (defined in the implementation but not as parameter). It returns **1** if this function runs timeout with no available data received, returns **0** if the new available data received in time and the received data is returned through pointer `unsigned char * ch`. In the application demo code, this receiving function is implemented with the a hardware timer (RIT) a USART receiver working in interrupt mode and a ring FIFO to buffer the receiving data. When using this function to read a data, the timer and the FIFO (connect to the receiver) are enabled together. Either for the event that the timer runs timeout or there is any available received data in the FIFO, this function will return and tell the return event.

In the `xymodem_port.c` file:

```
/*
 * send byte "ch".
 */
void XYModem_Uart_SendByte(unsigned char ch)
{
    Terminal_PutChar(ch);
}

/*
 * return 1 if timeout without available received data.
 * return 0 if data is available before timeout.
 */
int XYModem_Uart_RecvByteTimeout(unsigned char *ch)
{
    return Terminal_GetCharTimeout(ch, 1000u) ? 0 : 1;
}
```

The detail implementation's code is in the `terminal_uart.c` file:

```
    #define APP_TERMINAL_UART_RX_BUF_LEN   32u
    static uint8_t        gAppTerUartRxBuf[APP_TERMINAL_UART_RX_BUF_LEN];
    static RBUF_Handler_T gAppTerUartRxFifoHandle;
    volatile bool         bAppRitTimeout = false;

    void Terminal_Init(uint32_t baudrate)
    {
        usart_config_t usartConfigStruct;
        rit_config_t ritConfigStruct;

        /* prepare the uart rx buffer and software flags. */
        RBUF_Init(&gAppTerUartRxFifoHandle, gAppTerUartRxBuf, APP_TERMINAL_UART_RX_BUF_LEN);
        bAppRitTimeout = false;

        /* setup uart. */
        USART_GetDefaultConfig(&usartConfigStruct);
        usartConfigStruct.baudRate_Bps = baudrate;
        usartConfigStruct.enableRx = true;
        usartConfigStruct.enableTx = true;
        usartConfigStruct.txWatermark = kUSART_TxFifo0;
        usartConfigStruct.rxWatermark = kUSART_RxFifo1;
        USART_Init(USART0, &usartConfigStruct, CLOCK_GetFreq(kCLOCK_Flexcomm0));

        /* enable uart rx interrupt. */
        USART_EnableInterrupts(USART0, kUSART_RxLevelInterruptEnable);
        NVIC_EnableIRQ(FLEXCOMM0_IRQn);

        /* setup rit timer. */
        ritConfigStruct.enableRunInDebug = true;
        RIT_Init(RIT, &ritConfigStruct);
        RIT_ClearCounter(RIT, true); /* Enable auto-clear when the counter reach to compare value.*/
        RIT_SetTimerCompare(RIT, CLOCK_GetFreq(kCLOCK_CoreSysClk)); /* Interval 1s. */
        NVIC_EnableIRQ(RIT_IRQn);
    }

    /* ISR entry for USART0. */
    void FLEXCOMM0_IRQHandler(void)
    {
        uint32_t flags = USART_GetStatusFlags(USART0);
        uint8_t  rxDat;

        /* rx available interrupt. */
        if (kUSART_RxFifoNotEmptyFlag == (kUSART_RxFifoNotEmptyFlag & flags) )
        {
            rxDat = USART_ReadByte(USART0);
            if ( !RBUF_IsFull(&gAppTerUartRxFifoHandle) )
            {
                RBUF_PutDataIn(&gAppTerUartRxFifoHandle, rxDat);
            }
        }
        USART_ClearStatusFlags(USART0, flags);
    }

    /* putchar through terminal uart. */
    void Terminal_PutChar(uint8_t ch)
    {
        USART_WriteBlocking(USART0, &ch, 1U);
    }

    /* ISR entry for RIT timer. */
    void RIT_IRQHandler(void)
    {
        uint32_t flags;

        flags = RIT_GetStatusFlags(RIT);

        bAppRitTimeout = true;

        RIT_ClearStatusFlags(RIT, flags);
        RIT_StopTimer(RIT); /* for one time trigger. */
    }

    /* getchar from terminal within the timeout period defined by "ms" */
    bool Terminal_GetCharTimeout(uint8_t *rxDat, uint32_t ms)
    {
        bool bRet = false;
        /* setup alert timer. */
        bAppRitTimeout = false;
        if (ms > 0u)
        {
            RIT_StopTimer(RIT);
            RIT_ClearStatusFlags(RIT, kRIT_TimerFlag);
            RIT->COUNTER = 0u; RIT->COMPVAL_H = 0u; /* clear counter. */
            RIT_SetTimerCompare(RIT, CLOCK_GetFreq(kCLOCK_CoreSysClk) / 1000 * ms ); /* setup timeout
period. */
            RIT_StartTimer(RIT);
        }
        while (1)
        {
            if ( !RBUF_IsEmpty(&gAppTerUartRxFifoHandle) )
            {
                *rxDat = RBUF_GetDataOut(&gAppTerUartRxFifoHandle);
                bRet = true;
                break; /* return with available rx data. */
            }
            if (bAppRitTimeout)
            {
                bRet = false;
                break; /* return without available rx data. */
            }
        }
        RIT_StopTimer(RIT); /* terminate trigger. */
        RIT->COUNTER = 0u; RIT->COMPVAL_H = 0u; /* clear counter. */
        return bRet;
    }
```

4.4.3 PC: Send file through YModem as host

Most terminal softwares integrate the YModem protocol, so there is no need to build a special desktop software for communicating with MCU. In this application demo, the **Tera Term** software is used as the desktop terminal software on PC. Select **File** -> **Transfer** -> **YMODEM** -> **Send...** to activate the window for transferring the given file, as shown in Figure 7. on page 13.

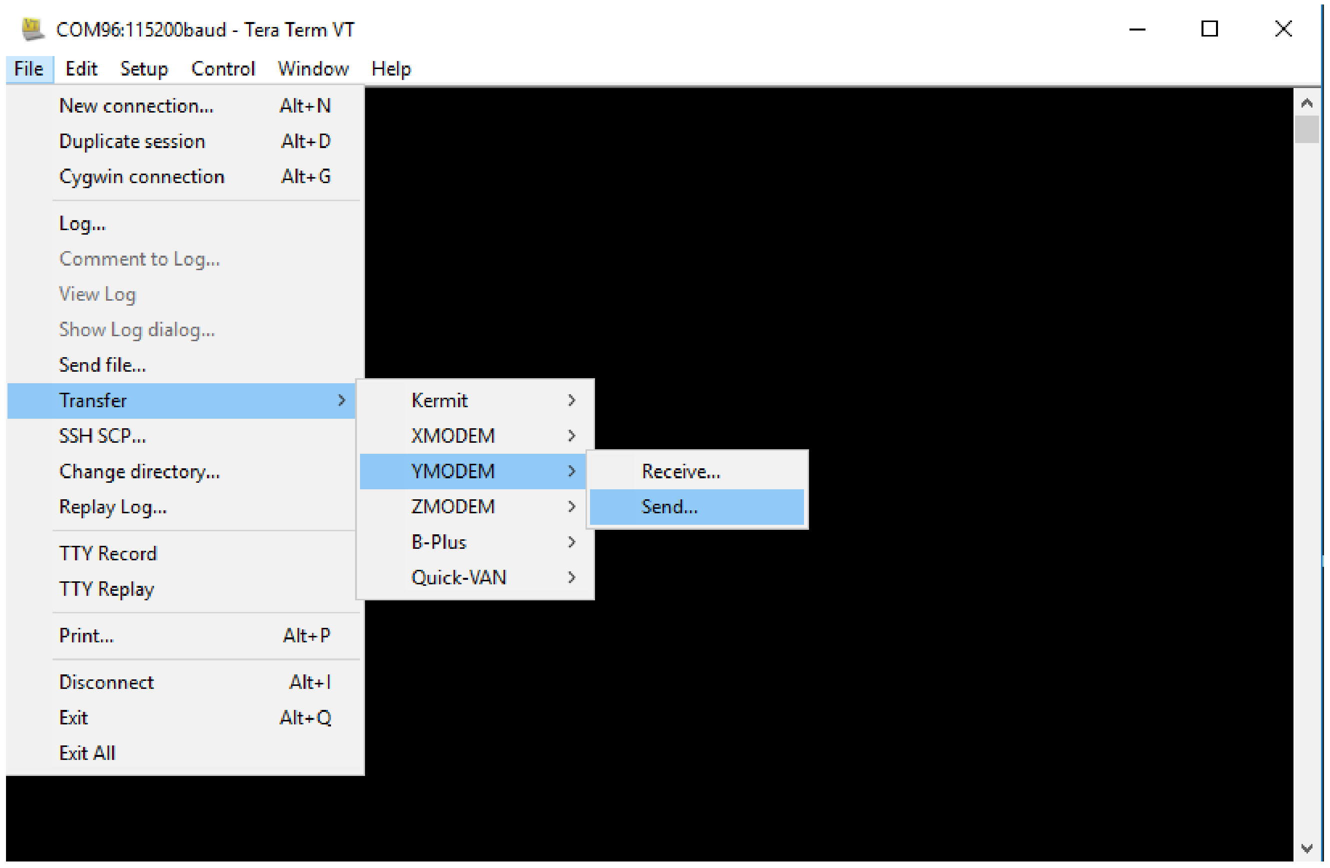


Figure 7. Tera Term YModem

4.5 MCUXPresso SDK driver for IAP

IAP function is used for MCU to program the the received image file to on-chip FLASH. NXP's MCUXpresso SDK software library provides the IAP driver to erase and write the FLASH.

Five APIs are used in this application demo:

- `status_t IAP_PrepareSectorForWrite(uint32_t startSector, uint32_t endSector)`: to prepare for any FLASH operation.
- `status_t IAP_EraseSector(uint32_t startSector, uint32_t endSector, uint32_t systemCoreClock)`: to erase the FLASH.
- `status_t IAP_BlankCheckSector(uint32_t startSector, uint32_t endSector)`: to check whether the FLASH is really erased after erase operation.
- `status_t IAP_CopyRamToFlash(uint32_t dstAddr, uint32_t *srcAddr, uint32_t numOfBytes, uint32_t systemCoreClock)`: to write the FLASH.
- `status_t IAP_Compare(uint32_t dstAddr, uint32_t *srcAddr, uint32_t numOfBytes)`: to check whether the FLASH is really written after the write operation.

In the demo code, SBL software knows the length of firmware image according to the first package of YModem transfer, erases enough FLASH sectors for the coming image data, and then writes binary data to FLASH when receiving a new package.

```
    /* start the ymodem and get the first package. */
    err = ymodem_init(&gAppModemStruct);

    gAppFwWriteImageLen    = gAppModemStruct.filelen;
    gAppFwWriteSectorStart = BOOT_FIRMWARE_BASE_ADDRESS / FSL_FEATURE_SYSCON_FLASH_SECTOR_SIZE_BYTES;
    gAppFwWriteSectorCount = (gAppFwWriteImageLen + FSL_FEATURE_SYSCON_FLASH_SECTOR_SIZE_BYTES-1) /
FSL_FEATURE_SYSCON_FLASH_SECTOR_SIZE_BYTES;

    /* erase the flash sectors to be programmed. */
    IAP_PrepareSectorForWrite(gAppFwWriteSectorStart, gAppFwWriteSectorStart
+gAppFwWriteSectorCount-1);
    IAP_EraseSector(gAppFwWriteSectorStart, gAppFwWriteSectorStart+gAppFwWriteSectorCount-1,
SystemCoreClock);
    errIAP = IAP_BlankCheckSector(gAppFwWriteSectorStart, gAppFwWriteSectorStart
+gAppFwWriteSectorCount-1);
    if (errIAP != kStatus_IAP_Success)
    {
        modem_cancle();
        Terminal_PutString("Sector Erase failed.\r\n");
        FwBoot_Exit();
    }

    do /* recevie the firmware data and program the flash. */
    {
        err = modem_recvdata(&gAppModemStruct);
        if (err == 1)
        {
            break; /* done. */
        }

        if (gAppModemStruct.cur_num == 1u) /* the first package would include the firmware info. */
        {
            /* copy the file name of firmware into the image. */
            strcpy( (char *)(gAppModemStruct.buf+BOOT_FIRMWARE_INFO_OFFSET), (char
*)gAppModemStruct.filename);
        }

        /* get the following indexes according to package index gAppModemStruct.cur_num:
         * - gAppFwWriteSectorCurIdx
         * - gAppFwWriteCurAddr
         */
        gAppFwWriteCurAddr      = BOOT_FIRMWARE_BASE_ADDRESS
                                + (gAppModemStruct.cur_num-1) * MODEM_PACKAGE_BYTE_COUNT;
        gAppFwWriteSectorCurIdx = gAppFwWriteCurAddr / FSL_FEATURE_SYSCON_FLASH_SECTOR_SIZE_BYTES;
        /* program to flash. */
        IAP_PrepareSectorForWrite(gAppFwWriteSectorCurIdx, gAppFwWriteSectorCurIdx);
        IAP_CopyRamToFlash(
                gAppFwWriteCurAddr,                /* dstaddr. */
                (uint32_t *)(gAppModemStruct.buf), /* srcAddr. */
                MODEM_PACKAGE_BYTE_COUNT,          /* numOfBytes. */
                SystemCoreClock                    /* systemCoreClock. */
            );
        errIAP = IAP_Compare(
                gAppFwWriteCurAddr,                /* dstaddr. */
                (uint32_t *)(gAppModemStruct.buf), /* srcAddr. */
                MODEM_PACKAGE_BYTE_COUNT           /* numOfBytes. */
            );
        if (errIAP != kStatus_IAP_Success)
        {
            err = 10;
        }

    } while (err == 0u);
```

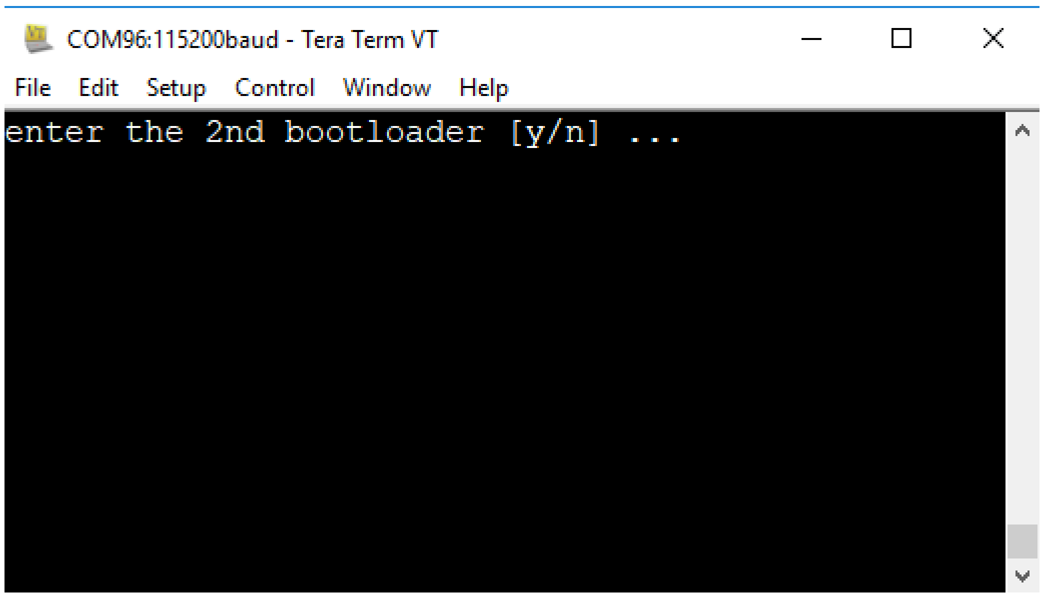
Considering the package is error and will be re-sent from the host, though the auto-creasing mode is simpler, the SBL uses the package index to get the writing sector index and the address instead.

Prepare the sector before the write operation, and **Compare** or **Check** the written data from FLASH after the write operation.

5 Demonstration

The SBL project and the modifed `hello_world` and `rit_example` demo projects from MCUXpresso SDK software library are packed alone with this application note. The projects are originally developed with Keil IDE, but can be easily ported to other IAR IDE like MCUXpresso IDE or IAR IDE. The demo projects are originally verified on the LPCXpresso54608 board. The following introduces how to run the demo.

1. Connect the board debug port to PC with a USB cable.
2. Prepare the user firmware's binary image file.
   - Any existing project for LPC54608 is OK to be a base project. In this application note, the `hello_world` demo is used as it is considerted as the simplest demo project in the software library and the `rit_example` demo is used as it enables the interrupt event.
   - Change their linker configurations by following the guide in Compiler link configuration on page 8.
   - Add the user command for generating the binary image file.
   - Build the project and get the binary image file.
3. Download the SBL project.
   - Build the SBL project and download it through IDE. Other downloading way, like debug operation or command line tool, is available.
4. Launch the YModem to download the image.
   - Execute the **Tera Term** to open the UART port to LPCXpresso54608 with 115200 baudrate, no parity, 1 stop bit.
   - Reset the board, and a question is printed to the UART terminal. The SBL will wait for the input for about five seconds, as shown in Figure 8. on page 16.
     - — For no input, it jumps to the per-downloaded user firmware automatically.
     - — For input **n**, it jumps to the pre-downloaded user firmware immediately.
     - — For input **y**, it starts the YModem communication immediately and wait to get a new firmware image file.

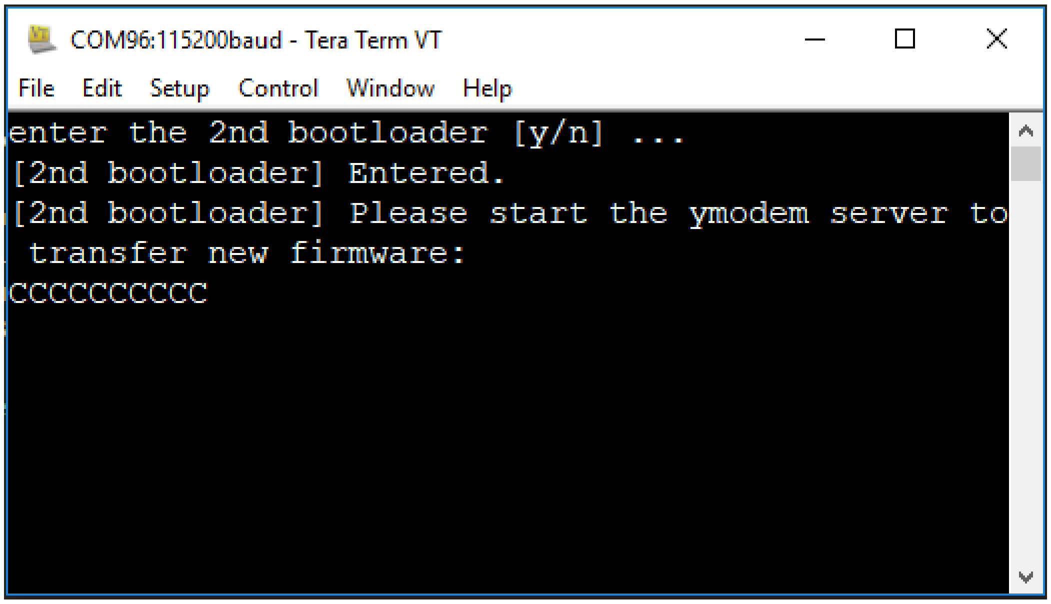

COM96:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
enter the 2nd bootloader [y/n] ...
```

Figure 8. Demo step 1

- Input **y** to start the YModem receiver for the first time.

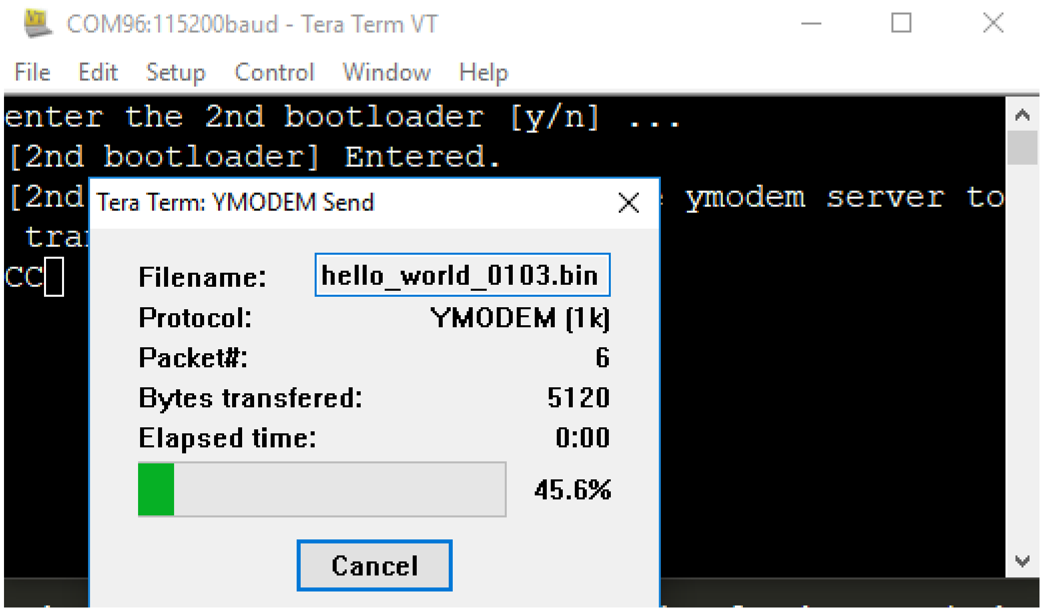
  Many **C** come, as shown in Figure 9. on page 16, because YModem is waiting for input and timeout. Feed it with the image file.


COM96:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
enter the 2nd bootloader [y/n] ...
[2nd bootloader] Entered.
[2nd bootloader] Please start the ymodem server to
 transfer new firmware:
CCCCCCCCCC
```

Figure 9. Demo step 2

- Send the prepared user firmware image file using the YModem tool of **Tera Term**, by following the guide in PC: Send file through YModem as host on page 12. Then the progress bar appears in the dialog window, as shown in Figure 10. on page 17.


COM96:115200baud - Tera Term VT
File Edit Setup Control Window Help
Tera Term: YMODEM Send
Cancel

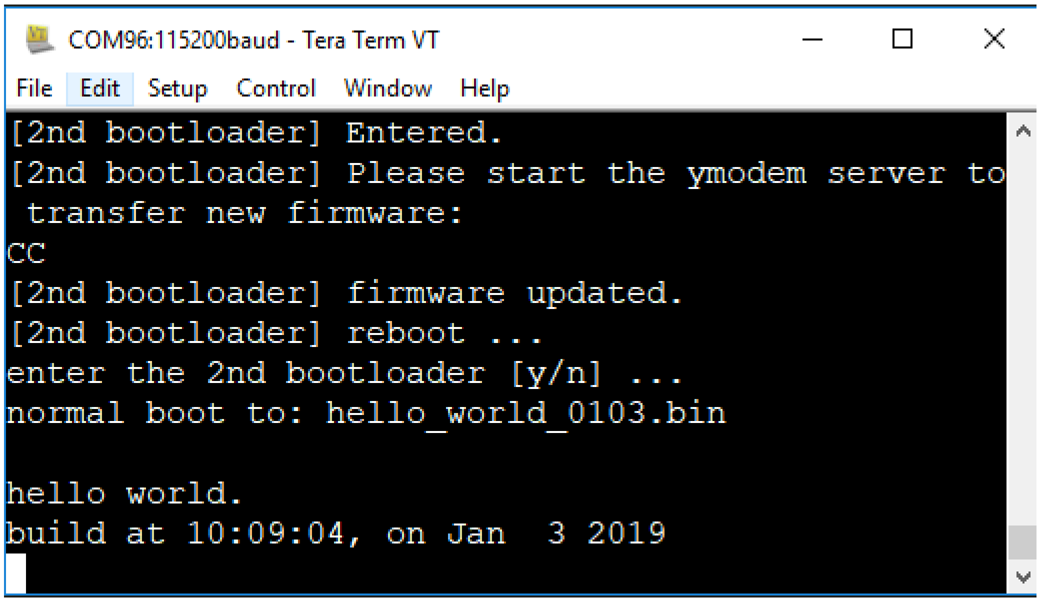
```
enter the 2nd bootloader [y/n] ...
[2nd bootloader] Entered.
[2nd                            ymodem server to
 tra
CC
```

Filename: hello_world_0103.bin
Protocol: YMODEM (1k)
Packet#: 6
Bytes transfered: 5120
Elapsed time: 0:00
45.6%

Figure 10. Demo step 3

5. Watch the result.

   After downloading the image, the SBL resets the MCU automatically. The question will come again, **enter the 2nd bootloader [y/n] ...**. Press **n** or just wait to timeout, and the SBL will jump to the new firmware with printing its file name, as shown in Figure 11. on page 17.


COM96:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
[2nd bootloader] Entered.
[2nd bootloader] Please start the ymodem server to
 transfer new firmware:
CC
[2nd bootloader] firmware updated.
[2nd bootloader] reboot ...
enter the 2nd bootloader [y/n] ...
normal boot to: hello_world_0103.bin

hello world.
build at 10:09:04, on Jan  3 2019
```

Figure 11. Demo step 4

You can try to download another prepared firmware (for example, using `rit_example.bin`) to see whether the new firmware can be executed. Of course, it works.

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, I2C BUS, 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, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS 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. 2019. 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: April 2019

Document identifier: AN12384