

1 Introduction

Although i.MX RT series already supports ROM boot by UART or USB to upgrade the firmware, but still there is customized requirement to upgrade firmware in On-The-Air(OTA) or other use case. It requires to implement second boot loader instead ROM boot loader. This document intends to introduce one example to illustrate how to implement second boot loader on i.MX RT1050, i.MX RT1020, and i.MX RT1060 part based on XIP flash.

2 Overview

Second boot loader manages the upgrade of the firmware. Generally it includes three parts, flash operation, communication, and security management. Second boot loader may be XIP(eXecute-In-Place) or NON-XIP depending upon customer application case and memory size on the system. As for NON-XIP, it means to run boot loader in internal memory or external memory; it must consume memory space for boot loader that is a limitation for the case with limited memory space.

This document introduces one common example to implement second boot loader based on XIP. It describes how to implement flash operation, communicate with host device by UART and USB. It also presents an example to show how to implement security management, which program the encrypted image and boot with BEE configuration to implement on-the-fly.

See the system diagram in [Figure 1](#) .

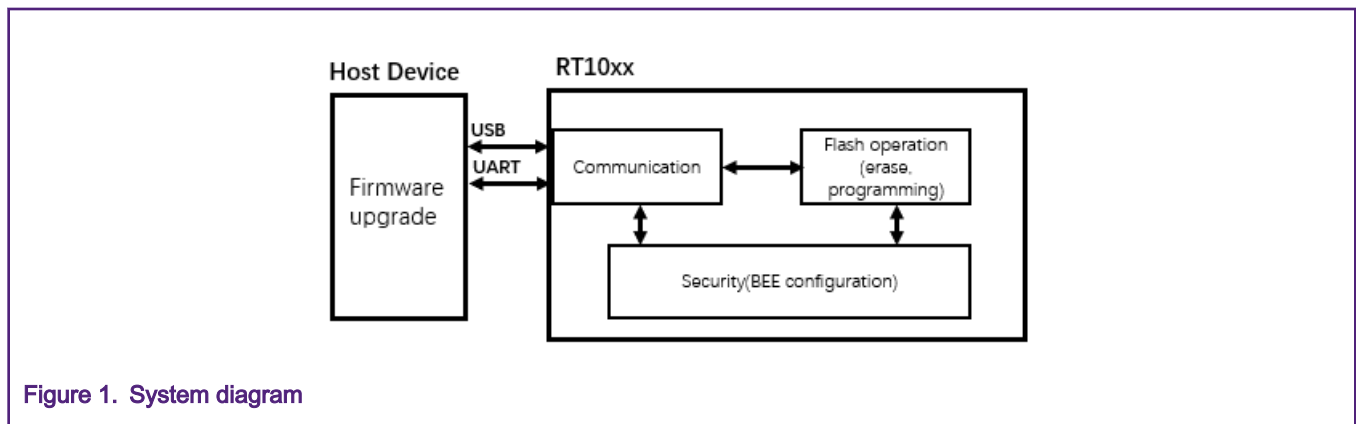


Figure 1. System diagram

3 Second Boot loader implement

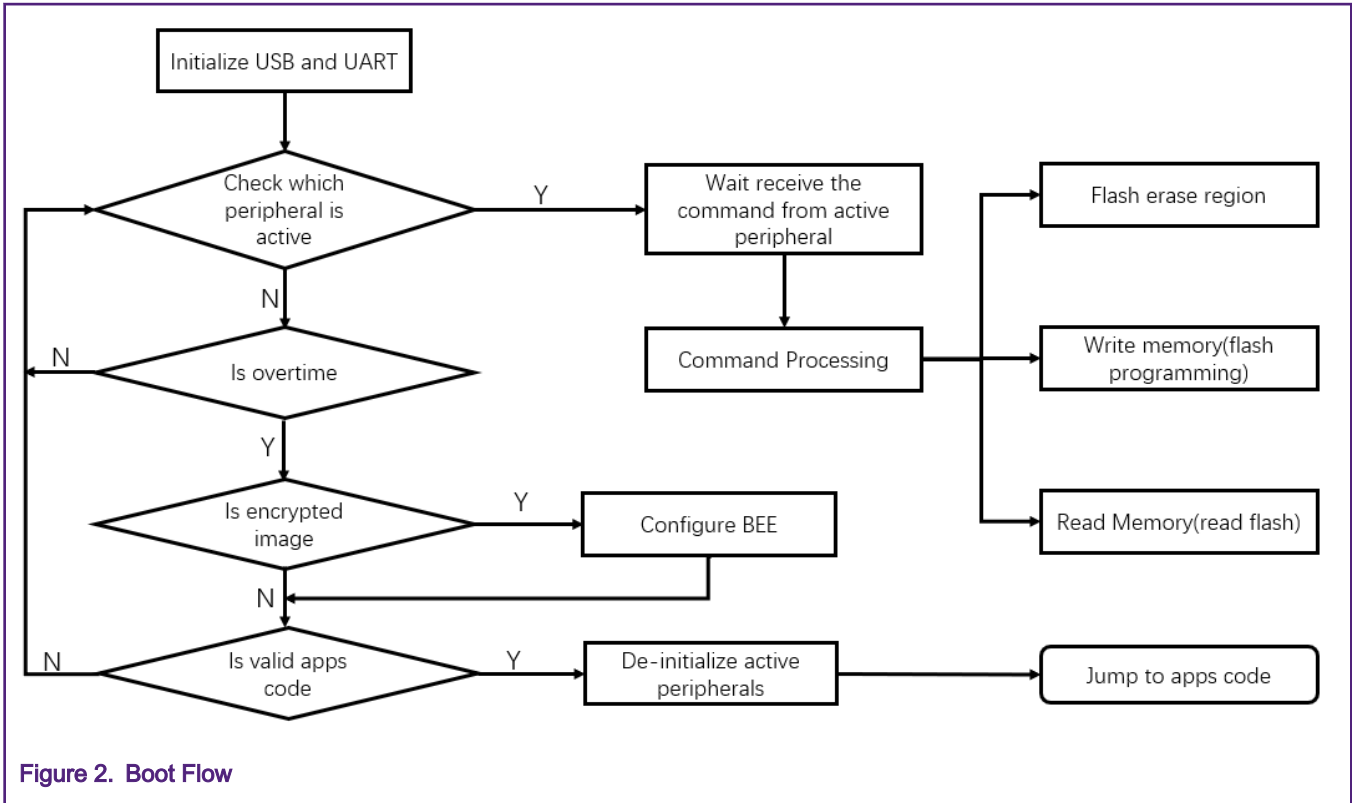
Here is an example to introduce how to implement XIP second boot loader in i.MX RT1050. Refer to [IMXRT1050 EVKB Board Hardware User's Guide](#) to rework board boot by QSPI flash.

For software architecture, it contains three sessions, flash operation, communication, and BEE configuration. Boot flow is shown in [Figure 2](#).



Contents

- 1 Introduction..... 1
- 2 Overview..... 1
- 3 Second Boot loader implement..... 1
- 4 Generate Encrypt Image..... 5
- 5 Run Boot Loader..... 9
- 6 Conclusion..... 10
- 7 Revision history..... 11



After power-on, it first initializes the USB and LPUART1, and then checks which peripheral is active. Meanwhile, detects if it is greater than default TIMEOUT value(default is 10S for debug project, and 5S for release project). User can also change this setting in file “bootloader_config.h”, as shown in [Figure 3](#).

```

    #if DEBUG
    #define BL_DEFAULT_PERIPHERAL_DETECT_TIMEOUT 10000
    #else
    #define BL_DEFAULT_PERIPHERAL_DETECT_TIMEOUT 5000
    #endif // DEBUG

    // The bootloader will check this address for the application vector tabl
    #if !defined(BL_APP_VECTOR_TABLE_ADDRESS)
    #define BL_APP_VECTOR_TABLE_ADDRESS (0x10000+0x60000000)
    #endif
    #define BL_PARAMETERS_ADDRESS (0xF000+0x60000000)
    #endif // __BOOTLOADER_CONFIG_H__

    #define FLASH_VALID_START BL_APP_VECTOR_TABLE_ADDRESS
    #define FLASH_VALID_END BL_FLEXSPI_AMBA_BASE+8*1024*1024
  
```

Figure 3. BootLoader default parameters

3.1 Flash operation

i.MX RT support FlexSPI to interface QSPI, Octal and hyper flash. Here we take QSPI as an example to show how to implement flash erasing and programming on XIP. As ROM initializes FlexSPI and QSPI flash configuration on boot phase, it is not required

to initialize FlexSPI once more. By default it do not support flash erasing and programming operations, It only initialize LUT to support fast read command by ROM, so the user has to update LUT to support more commands as per customer requirements. To get stable operation, follow the below guidelines:

- Disable interrupt and prefetch buffer before doing any FlexSPI operation. To abort on-going prefetching, set SWRESET bit in register MCR0 to enable FlexSPI software reset. For example, update LUT.
- Allocate all the code of operating FlexSPI to internal or external RAM, say updating LUT, flash erase and program API function.
- After flash erasing and programming complete, set SWRESET bit to enable FlexSPI software reset.

All the flash API function can be found in file “flexspi_nor_flash_ops.c”, and for i.MX RT1060, also can call the ROM API to implement flash operation.

ROM provides the entry address(0x0020001c) for all API tree, it need to initialize API tree. One example as below:

```
g_bootloaderTree = (bootloader_api_entry_t*)(uint32_t*)0x0020001c;
```

An snippet for API entry is as below:

```
typedef struct
{
    const uint32_t version;                //!< Bootloader version number
    const char *copyright;                //!< Bootloader Copyright
    void (*runBootloader)(void *arg);     //!< Function to start the bootloader executing
    const uint32_t *reserved0;            //!< Reserved
    const flexspi_nor_driver_interface_t *flexSpiNorDriver;  //!< FlexSPI NOR Flash API
    const uint32_t *reserved1;            //!< Reserved
    const clock_driver_interface_t *clockDriver;
    const rtwdog_driver_interface_t *rtwdogDriver;
    const wdog_driver_interface_t *wdogDriver;
    const uint32_t *reserved2;
} bootloader_api_entry_t;
```

Figure 4. ROM API Entry

User can call appropriate API by this API entry. For details refer to attached example of “rt1060_rom_api.zip”.

3.2 Communication

This boot loader currently supports UART and USB for boot, default support LPUART1, and band rate is 115200 bps, available to use host tool “blhost.exe” to get image upgrade, to simplify the operation, remove some commands and supported commands list as below:

- get_property
- write memory
- flash erase region
- read memory

For detail protocols introduction, refer to doc MCUX Flashloader Reference Manual and [Kinetic blhost User's Guide.pdf](#).

Boot pins is as below:

Table 1. Boot Pins

Peripheral	Instance	IO Func	ALT	PAD
LPUART	1	lpuart1.TX	2	GPIO_AD_B0_12

Table continues on the next page...

Table 1. Boot Pins (continued)

		lpuart1.RX	2	GPIO_AD_B0_13
USB	1	USB_OTG1_DP	-	USB_OTG1_DP
		USB_OTG1_DN	-	USB_OTG1_DN

User may change UART instance and pin mux as below.

- Navigate the file “peripherals_MIMXRT1052.C” to change instance number.

```

///! @brief Peripheral array for MIMXRT1051.
const peripheral_descriptor_t g_peripherals[] = {
#if BL_CONFIG_LPUART_1
    // LPUART1
    { .typeMask = kPeripheralType_UART,
      .instance = 1,
      .pinmuxConfig = uart_pinmux_config,
      .controlInterface = &g_lpuartControlInterface,
      .byteInterface = &g_lpuartByteInterface,
      .packetInterface = &g_framingPacketInterface },
#endif // BL_CONFIG_LPUART_1

```

For example, available to change LPUART instance to 2 or others.

Here “.pinmuxconfig” specify the pin mux configuration function for LPUART. To change pin mux settings, navigate this function to modify it.

3.3 Security management

The second bootloader support to upgrade the encrypted or raw image. To encrypt image by default it uses SW_GP2 as key, and decrypted on-the-fly by configure BEE region1, and reserve BEE region0 for second boot loader, available to select OTPMK/SNVS as boot loader key, or use the same key(SW_GP2) for second boot loader. The image layout is shown in [Figure 5](#) .

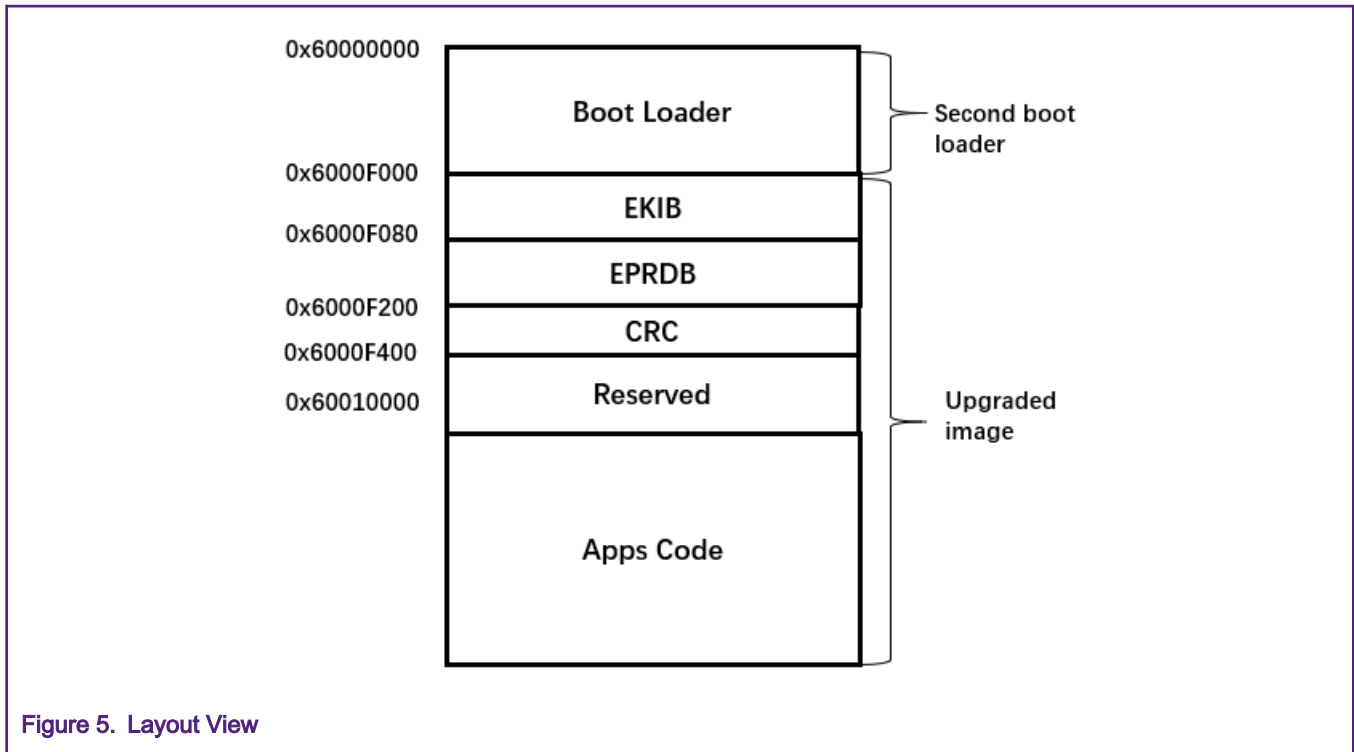


Figure 5. Layout View

It reserves the address space from 0x60000000 to 0x6000f000 for second boot loader, and address space from 0x6000f000 to 0x6000f400 is for configuration parameters of encrypted image. This is similar with ROM EKIB and EPRDB, EKIB encrypted by SW_GP2 and EPRDB encrypted by KIB with AES128 CBC mode.

After detect timeout, second boot loader tries to check if image is valid or not, if image is encrypted image, try to decrypt EKIB and EPRDB, and then configure BEE region1 with specified setting by PRDB, and reserve BEE region 0 for boot loader encryption.

NOTE

Customer must burn the SW_GP2 and set fuse "BEE_KEY1_SEL" to "0b11", this is to enable BEE region 1 key from SW_GP2.

4 Generate Encrypt Image

As second boot loader supports to program the encrypted the image and boot up, so it needs one tool to convert the plain image to encrypted image. A simple diagram, and image layout is below:

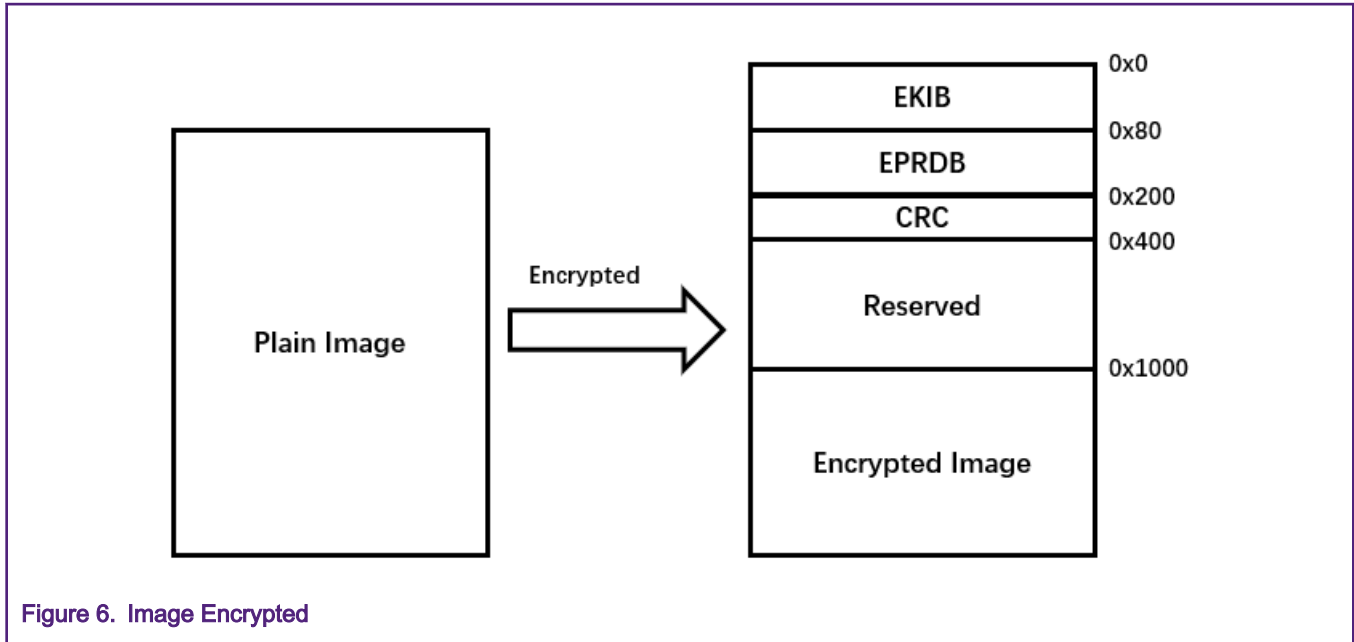


Figure 6. Image Encrypted

The description of encrypted image layout is as below:

Table 2. Encryption image parameters layout

Offset	Description
0x0000 - 0x0003	Tag, must equal to 0xEA5CA5A5
0x4 - 0x23	Encrypted Key Info Block(EKIB)
0x24 – 0x7f	Reserved
0x80 - 0x17F	Encrypted Protection Region Descriptor Block(EPRDB)
0x180 - 0xFFFF	Reserved
0x200 – 0x400 CRC	Save image CRC value for check
0x400 - 0xFFFF	Reserved

Key Info Block(KIB) consist of 128-bit AES Key and 128-bit Initial Vector.

Table 3. Key Info Block

Offset	Field	Description
0x00 - 0x0f	AES-128 KEY	128-bit AES KEY used for EPRDB decryption
0x10 - 0x1f	Initial Vector	128-bit Initial Vector used for EPRDB decryption

Protection Region Descriptor Block (PRDB) is a 128 byte crypto block using AES-ECB mode and the AES Key provisioned in eFUSE. The plaintext of PRDB is shown in Table4.

Table 4. Protection Region Descriptor Block

Offset	Field	Description
0x000 - 0x003	tagl	Must equal to 0x5F474154 (ASCII: "TAG_")
0x004 - 0x007	tagh	Must equal to 0x52444845 (ASCII: "EHDR")
0x008 - 0x00b	Version	[31:24] 'V' [23:16] Major version [15:08] Minor version [07:00] Bug fix Must equal to 0x56010000 for i.MX RT Family
0x00c - 0x00f	Reversed	

Table 4. Protection Region Descriptor Block (continued)

0x010-0x04f	encrypt_region_info	Information for encrypted region			
		Offset	Field	Description	
		0x00 - 0x03	Start	Absolute start address of encrypted region, must be 4 KB aligned	
		0x04 - 0x07	End	Absolute end address of encrypted region, must be 4 KB aligned	
		0x08 - 0x0b	Mode	AES encryption mode 0 - AES-ECB (128) 1 - AES-CTR (128), it is the recommend mode for Encrypted XIP	
		0x0c - 0x0f	lock_option	Region Lock options 0 - All regions are unlocked 3 - BEE regions 1 are locked.	
		0x10 - 0x1f	Counter	AES-CTR Counter, Valid only if mode is 1 Only the upper 96bits are configurable, the lower 32bit must be 0. Note: During encryption and decryption, the nonce/is	
				127:32	31:28
Counter [127:32]	0			FAC Region Start [31:4]	
0x20-0x3f	Reserved	Reserved for future use.			
0x050-0x0Ff	Reserved	Reserved for future use			

CRC block is to check the code integrity of programming to flash, second boot loader will make CRC check before jumping to user code, if failed, will stay in boot load image.

Table 5. CRC Block

Offset	Field	Description
0x00 - 0x04	Tag	Must equal to 0xccea5a5a

Table continues on the next page...

Table 5. CRC Block (continued)

0x04 - 0x07	CRC value	Final CRC results
0x08 – 0x0b	Byte count	Number of bytes for CRC processed

Attached zip file with the document provides one tool "image_generate.exe". To encrypt image, detail arguments introduction is described as below:

Table 6. Arguments View

ifile	Input file path	
ofile	Output file path	
base_addr	base address of input file	
region_key	hex string	
region_arg	[start, length, permission],..., [start, length, permission]	start, length, and permission are integer variables
region_lock	0/1/2	0/1/2
use_zero_key	0/1	0 - KIB and Nonce/Counter[1/2/3] are generated randomly 1 - KIB and Nonce/Counter[1,2,3] = 0
kib_key	hex string	
kib_iv	hex string	
region_iv	hex string	

Below is example to generate the encrypted image by "image_generate.exe" tool:

```
image_generate.exe ifile=iled_blinky.bin ofile=iled_blinky_encrypt.bin base_addr=0x60010000
region_key=00112233445566778899aabbccddeeff region_arg=1,[0x60010000,0xF000,0]
```

5 Run Boot Loader

To program the encrypted image to flash, second boot loader must first fuse SW_GP2 to i.MXRT10xx(part RT1020/RT1050/RT1060), and fuse "BEE_KEY1_SEL" to "0b11" to enable SW_GP2 as BEE region1 key selection.

Get the attached package, and then build second boot loader to program the code to flash by IDE or others tools. Attached code support to program to flash by IAR directly.

To update image follow the below steps:

1. Prepare one apps image for upgrade, it must modify the linker file to get interrupt vector address to be 0x60010000. Refer to attached linker file, below is snippet linker file.

```

define symbol m_interrupts_start      = 0x60010000;
define symbol m_interrupts_end      = 0x600103FF;

define symbol m_text_start          = 0x60010400;
define symbol m_text_end            = 0x63FFFFFF;

define symbol m_data_start          = 0x20000000;
define symbol m_data_end            = 0x2001FFFF;

```

Also, remove the XIP information by “XIP_BOOT_HEADER_ENABLE = 0”.

2. Program the second boot loader to flash by IDE or other tools.
3. Generate the encrypted image by attached tools “image_generate.exe” with below command:
image_generate.exe ifile=iled_blinky.bin ofile=iled_blinky_encrypt.bin base_addr=0x60010000
region_key=00112233445566778899aabbccddeeff region_arg=1,[0x60010000,0xF000,0]

This is optional, user can also directly program the plain image to flash.

4. Plug in USB cable, and try to run command in sequence.

USB command as below:

- blhost.exe -u -- get-property 1
- blhost.exe -u -- flash-erase-region 0x6000f000 0x10000
- blhost.exe -u -- write-memory 0x6000f000 iled_blinky_encrypt.bin

UART command as below:

- blhost.exe -p COM7,115200 -- get-property 1
- blhost.exe -p COM7,115200 -- flash-erase-region 0x6000f000 0x10000
- blhost.exe -p COM7,115200 -- write-memory 0x6000f000 iled_blinky_encrypt.bin

With above command, it can erase specified flash space and program the image file to flash.

NOTE

For encrypted image, start address of image is 0x6000f000, for raw image, start address is 0x60010000.

Also user can check flash contents by read-memory command. For detailed information on how to use blhost, refer to “blhost User’s Guide.pdf”.

5. After successfully write image to flash, reset MCU, and then it works, see the LED blinking, and print the message on serial terminal.

6 Conclusion

This document introduces one way to implement second boot loader based on XIP flash, which is helpful to implement second boot loader based customized applications.

7 Revision history

Table 7. Revision history

Revision number	Date	Substantive changes
0	06/2020	Initial release

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, UMEMS, 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. 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: June 2020
Document identifier: AN12604

