AN12773

How to Enable Master Boot from Serial NOR Flash

Rev. 0 — March 12 2020 Application Note

Contents

1 Introduction

The i.MX RT Series is industry's first crossover processor provided by NXP. This document describes how to program a bootable image into the Serial NOR Flash device and enable the i.MX RT to boot from this primary Flash device. Booting from primary boot media is so-called master boot.

ROM bootloader supports automated booting from a Serial NOR (Quad or Octal SPI Flash, HyperFlash) device and eXecute-In-Place (XIP) from this serial NOR Flash. This is the main feature of ROM bootloader.

The release includes the PC-hosted `blhost` command-line application. This application is used for downloading application to Flash device in development phase. This release also includes `elftosb` command-line application and it is used to generate bootable image for i.MXRT 600 ROM.

The software used for example in this document is based on the i.MXRT 685 SDK 2.7.0. The development environment is IAR Embedded Workbench 8.40.2. The hardware development environment is X-M IMXRT 685 -EVK (Rev.E).

2 i.MXRT600 boot overview

2.1 Boot features

The internal ROM memory is used to store the boot code. After a reset, the Arm® processor starts its code execution from this memory. The boot loader code is executed every time when the part is powered-on or is reset.

Since the i.MX RT600 has no internal Flash for code and data storage, images must be stored elsewhere for loading upon reset or the CPU can execute from an external memory (XIP). Images can be loaded into on-chip SRAM from external Flash or downloaded via the serial ports (UART, SPI, I^2C, USB). The code is then validated, and boot ROM will jump to on-chip SRAM.

Depending on the values of the OTP bits and ISP pins and the image header type definition, the bootloader decides whether to download codes into the on-chip SRAM or run from external memory. The bootloader checks the OTP bit settings first and then the ISP pins. If bit `[3:0]` in OTP word `BOOT_CFG [0]` is not programmed (4b'0000), the boot source is determined by the states of the ISP boot pins (`PIO1_15`, `PIO1_16`, and `PIO1_17`).

2.2 Boot settings

If `PRIMARY_BOOT_SRC` bits in OTP are not set, the i.MX RT600 will read the status of the ISP pins to determine boot source.

Table 1. Boot mode and ISP Downloader modes based on ISP pins

Boot mode	ISP2 pin `PIO1_17`	ISP1 pin `PIO1_16`	ISP0 pin `PIO1_15`	Description
—	Low	Low	Low	Reserved

Table continues on the next page...

Table 1. Boot mode and ISP Downloader modes based on ISP pins (continued)

Boot mode	ISP2 pin PIO1_17	ISP1 pin PIO1_16	ISP0 pin PIO1_15	Description
SDIO0 (SD Card)	Low	Low	High	Boot from an SD card device connected to SDIO 0 interface. The i.MXRT600 will look for a valid image in the SD card device. If there is no valid image found, the i.MXRT600 will enter the ISP boot mode based on OTP DEFAULT_ISP_MODE bits (6:4, BOOT_CFG [0])).
FlexSPI Boot from Port B	Low	High	Low	Boot from Quad or Octal SPI Flash devices connected to the FlexSPI interface 0 Port B. The i.MXRT600 will look for a valid image in external Quad/Octal SPI Flash device. If there is no valid image found, the i.MXRT600 will enter ISP boot mode.
FlexSPI Boot from Port A	Low	High	High	Boot from Quad/Octal SPI Flash devices connected to the FlexSPI interface 0 Port A. The i.MXRT600 will look for a valid image in external Quad/Octal SPI Flash device. If there is no valid image found, the i.MXRT600 will enter ISP boot mode.
SDIO 0 (eMMC)	High	Low	Low	Boot from an SD card device connected to SDIO 0 interface. The i.MXRT600 will look for a valid image in the SD card device. If there is no valid image found, the i.MXRT600 will enter the ISP boot mode based on OTP DEFAULT_ISP_MODE bits (6:4, BOOT_CFG [0]))
USB DFU (master boot)	High	Low	High	USB DFU class is used to download a boot image over the USB High-speed port into on-chip SRAM.
Serial ISP (UART, SPI, I^2C, USB-HID)	High	High	Low	The Serial Interface (UART, SPI, and I^2C,USB-HID) is used to program OTP, external Flash, SD or eMMC device.
Serial Master Boot(UART, SPI, I^2C, USB-HID)	High	High	High	Serial Master boot (SPI Slave, I^2C Slave, or UART, USB-HID) is used to download a boot image over the serial interface (SPI Slave, I^2C slave or UART,USB-HID).

2.3 Boot image offset

The bootloader looks for the boot image from a specified offset on a boot media. See the details in Table 2.

Table 2. Image offset on different boot media

Boot media	Image offset
FlexSPI Boot (Serial NOR Flash device)	0x1000
SD Boot (SD card)	0x1000
eMMC boot (eMMC memory)	0x1000
Recovery Boot (SPI NOR Flash device)	0x1000

2.4 Boot image header

Once the boot mode is determined, and the boot image is available on the selected external memory device (SD, eMMC or Serial NOR Flash), the ROM bootloader starts to copy the first 64 bytes of image header from the external memory device into on-chip SRAM. The beginning of the image follows the format mentioned in .

Table 3. Image header format

Offset	Field	Description
0x00 - 0x1F	Reserved	—
0x20	imageLength	The image length
0x24	imageType	Image Type 0x0000 - Plain Image 0x0001 - Plain Signed Image 0x0002 - Plain CRC Image 0x0003 - Encrypted Signed Image 0x0004 - Plain Signed XIP Image 0x0005 - Plain CRC XIP Image 0x8001 - Plain Signed Image with KeyStore included 0x8003 - Encrypted Signed Image with KeyStore included
0x28	authBlockOffset/crcChecksum	Authenticate Block Offset or CRC32 checksum
0x2C - 0x33	Reserved	—
0x34	imageLoadAddress	Image Load Address
0x38 - 0x3F	Reserved	—

The bootloader begins scanning for user images by examining the image type located at offset 0x24 (imageType). If a valid image type is detected, the validation of an image header starts. Qualification of the image header continues by reading the image load address at offset 0x34 (imageLoadAddress) in the image header and using it as a pointer to a valid image header structure. If the imageType and imageLoadAddress are both non-zero, the address pointed by the imageLoadAddress must contain the image header under examination.

After the completion of the validation of the image header, the qualification continues by examining the image type field. If a bootable (not XIP) image resides in the external flash, the entire image will be loaded into the on-chip SRAM first, then the imageLength field in the image header will be used as the length to perform a CRC check if the CRC check feature is enabled.

2.5 Serial ISP boot

i.MXRT600 includes In-System Programming (ISP) functions. The bootloader provides flash programming utility that operates over a serial connection on the MCUs. It enables quick and easy programming of MCUs through the entire product lifecycle, including application development, final product manufacturing, and beyond. Host-side command line and GUI tools are available to communicate with the bootloader. Users can utilize host tools to read and program application code and do manufacturing via the bootloader.

Here are brief ISP features:

- Supporting UART, SPI, I^2C and USB peripheral interfaces.
- Automatic detection of the active peripheral.
- Programming OTP.
- Programming serial NOR Flash.
- Programming SD card.
- Programming eMMC device.

Each kind of boot device has unique config option block. It indicates the flash device properties. The config option block should be passed to host tool. Table 4 describes the config option block for FlexSPI master boot device.

Table 4. FlexSPI boot config option block

Offset	Field	Description	
0x00	Option 0	**[31:28] Tag**	**Must be 0xC**
		[27:24] Option Size	Size in bytes = (Option Size + 1) × 4 It is 0 if only option 0 is required.
		[23:20] Device Type	Device Detection Type 0 - Read SFDP for SDR commands 1 - Read SFDP for DDR Read commands 2 - HyperFLASH 1V8 3 - HyperFLASH 3V 4 - Macronix Octal DDR 5 - Macronix Octal SDR 6 - Micron Octal DDR 7 - Micron Octal SDR 8 - Adesto EcoXiP DDR 9 - Adesto EcoXiP SDR
		[19:16] Query CMD Pad	Data pads during Query command (read SFDP or read MID) 0 - 1 2 - 4 3 - 8
		[15:12] CMD Pad	Data pads during Flash access command

Table continues on the next page...

Table 4. FlexSPI boot config option block (continued)

Offset	Field	Description	
			0 - 1 2 - 4 3 - 8
		[11:8] Quad Enable Type	Quad Mode Enable Setting 0 - Not configure 1 - Set bit 6 in Status Register 1 2 - Set bit 1 in Status Register 2 3 - Set bit 7 in Status Register 2 4 - Set bit 1 in Status Register 2 vis 0x31 command **NOTE** This field will be effective only if device is compliant with JESD216 only (9 longword SDFP table).
		[7:4] Misc	Miscellaneous Mode 0 - Not enabled 1 - Enable 0-4-4 mode for High Random Read performance 3 - Data Order Swapped mode (for MXIC Octal Flash only) 5 - Select the FlexSPI data sample source as internal loop back 6 - Config the FlexSPI NOR flash running at stand SPI mode
		[3:0] Max Freq	Max Flash Operation speed 0 - Don't change FlexSPI clock setting Others – See
`0x04`	Option 1	[31:28] Flash connection	Flash connection option: 0 - Single Flash connected to port A 1 - Parallel mode 2 - Single Flash connected to Port B
		[27:24] Drive Strength	The Drive Strength of FlexSPI Pads
		[23:20] DQS pinmux group	The DQS pin mux Group Selection
		[19:16] Pinmux group	The pin mux group selection
		[15:8] Status override	Override status register value during device mode configuration
		[7:0] Dummy Cycles	Dummy cycles for read command 0 - Use detected dummy cycle Others - dummy cycles provided in flash data sheet

3 FlexSPI master boot mode

In the i.MXRT600, the Serial NOR Flash device is supported as the primary boot media.

3.1 Device pin assignment

The ROM bootloader supports read, write, and erase external Serial NOR Flash devices via the FlexSPI module.

See the pin list of the FlexSPI peripheral in Table 5.

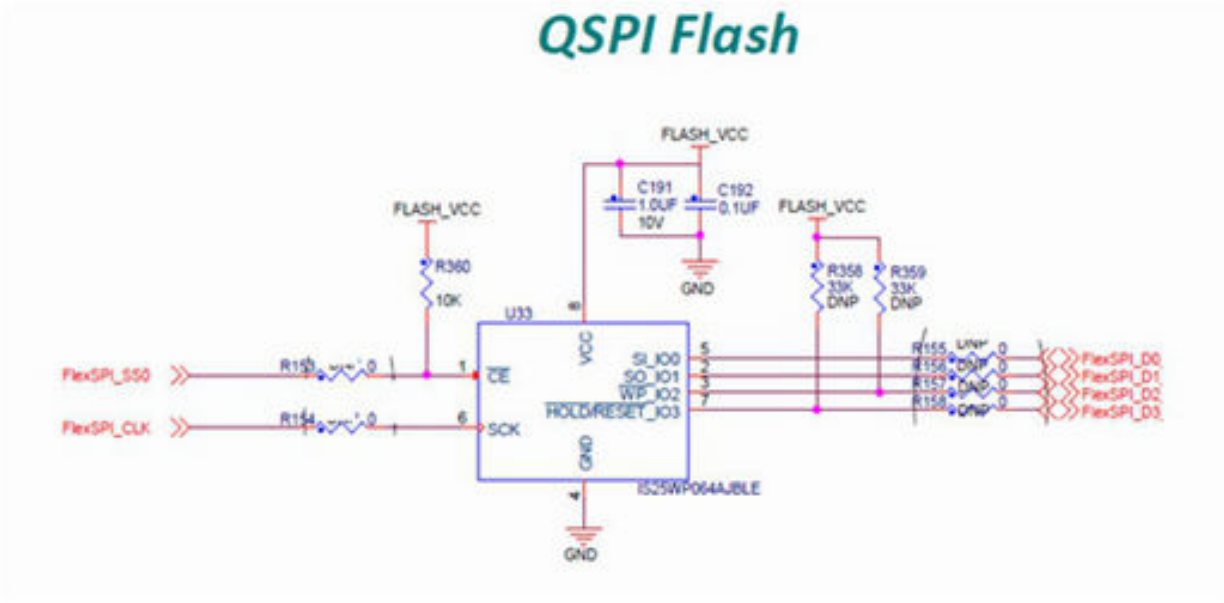
Table 5. FlexSPI pin assignments for NOR flash connections

Boot interface	Pin(s)	Function
FlexSPI0 PORT A	P1(18)	FLEXSPI_PORT_A_CLK0
	P1(19)	FLEXSPI_PORT_A _SSEL0
	P1(20)	FLEXSPI_PORT_A _D0
	P1(21)	FLEXSPI_PORT_A _D1
	P1(22)	FLEXSPI_PORT_A _D2
	P1(23)	FLEXSPI_PORT_A _D3
	P1(24)	FLEXSPI_PORT_A _D4
	P1(25)	FLEXSPI_PORT_A _D5
	P1(26)	FLEXSPI_PORT_A _D6
	P1(27)	FLEXSPI_PORT_A _D7
	P1(28)	FLEXSPI_PORT_A _DQS
	P1(29)	FLEXSPI_PORT_A _SSEL1
FlexSPI0 PORT B	P2(19)	FLEXSPI_PORT_B _SSEL0
	P2(21)	FLEXSPI_PORT_B _SSEL1
	P1(29)	FLEXSPI_PORT_B _CLK0
	P1(11)	FLEXSPI_PORT_B _D0
	P1(12)	FLEXSPI_PORT_B _D1
	P1(13)	FLEXSPI_PORT_B _D2
	P1(14)	FLEXSPI_PORT_B _D3
	P2(17)	FLEXSPI_PORT_B _D4
	P2(18)	FLEXSPI_PORT_B _D5
	P2(22)	FLEXSPI_PORT_B _D6
	P2(23)	FLEXSPI_PORT_B _D7

3.2 Device connection

There are four kinds of Flash connection for FlexSPI master boot.

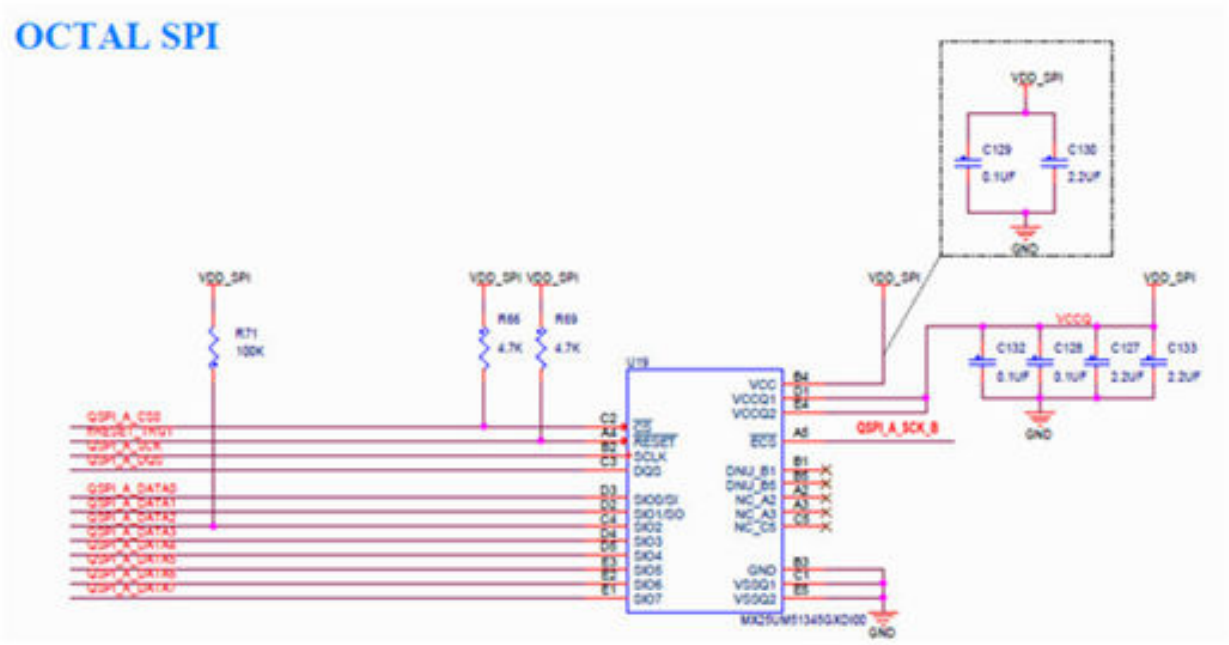
- The first two ways are to connect QSPI Flash to FlexSPI Port A or Port B, as shown in Figure 1.



BT Device	Func Name	Func	PAD	WLCSP114	VFBGA176	FOWLP249
FlexSPI0 Port A	flexspi0_A_SCLK	1	PIO1_18	✓	✓	✓
	flexspi0_A_SS0_N	1	PIO1_19	✓	✓	✓
	flexspi0_A_DATA[0]	1	PIO1_20	✓	✓	✓
	flexspi0_A_DATA[1]	1	PIO1_21	✓	✓	✓
	flexspi0_A_DATA[2]	1	PIO1_22	✓	✓	✓
	flexspi0_A_DATA[3]	1	PIO1_23	✓	✓	✓
FlexSPI0 Port B	flexspi0_B_SCLK	6	PIO1_29		✓	✓
	flexspi0_B_SS0_N	6	PIO2_19	✓	✓	✓
	flexspi0_B_DATA[0]	6	PIO1_11	✓	✓	✓
	flexspi0_B_DATA[1]	6	PIO1_12	✓	✓	✓
	flexspi0_B_DATA[2]	6	PIO1_13	✓	✓	✓
	flexspi0_B_DATA[3]	6	PIO1_14	✓	✓	✓

Figure 1. QSPI flash connection

- The third way is to connect Octal/Hyper flash to FlexSPI Port A, as shown in Figure 2.



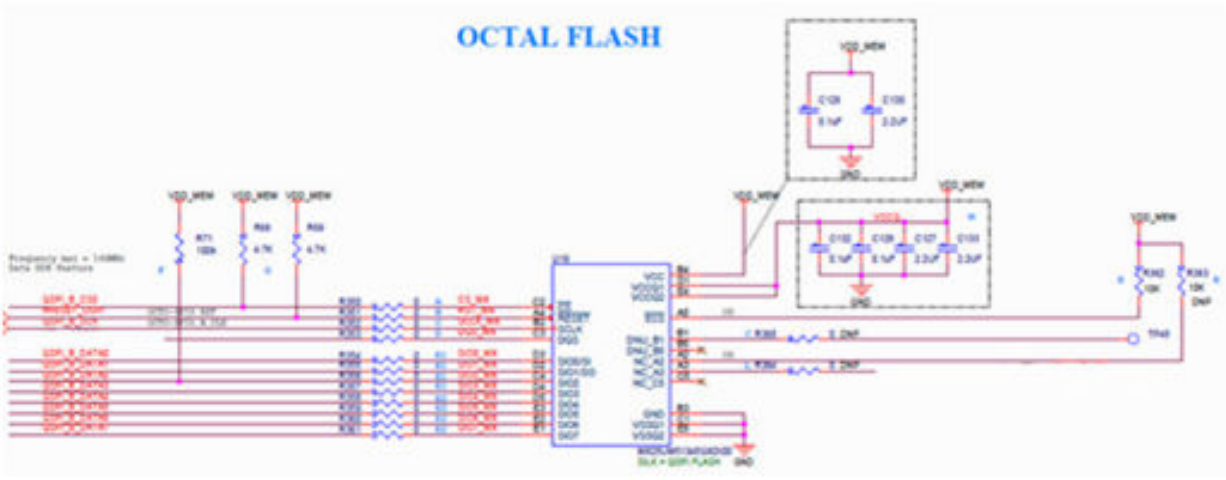
BT Device	Func Name	Func	PAD	WLCSP114	VFBGA176	FOWLP249
FlexSPI0 Port A	flexspi0_A_SCLK	1	PIO1_18	✓	✓	✓
	flexspi0_A_SS0_N	1	PIO1_19	✓	✓	✓
	flexspi0_A_DATA[0]	1	PIO1_20	✓	✓	✓
	flexspi0_A_DATA[1]	1	PIO1_21	✓	✓	✓
	flexspi0_A_DATA[2]	1	PIO1_22	✓	✓	✓
	flexspi0_A_DATA[3]	1	PIO1_23	✓	✓	✓
	flexspi0_A_DATA[4]	1	PIO1_24		✓	✓
	flexspi0_A_DATA[5]	1	PIO1_25		✓	✓
	flexspi0_A_DATA[6]	1	PIO1_26		✓	✓
	flexspi0_A_DATA[7]	1	PIO1_27		✓	✓
	flexspi0_A_DQS	1	PIO1_28		✓	✓
	flexspi0_A_SCLK_N	5	PIO1_29		✓	✓

Figure 2. Octal/Hyper Flash connected to Port A

- The last way is to connect Octal/Hyper flash to FlexSPI Port B, as shown in Figure 3.

NOTE

There is no DQS pin in FlexSPI Port B, so the flash cannot run at high speed in this way.



BT Device	Func Name	Func	PAD	WLCSP114	VFBGA176	FOWLP249
FlexSPI0 Port B	flexspi0_B_SCLK	6	PIO1_29		✓	✓
	flexspi0_B_SS0_N	6	PIO2_19	✓	✓	✓
	flexspi0_B_DATA[0]	6	PIO1_11	✓	✓	✓
	flexspi0_B_DATA[1]	6	PIO1_12	✓	✓	✓
	flexspi0_B_DATA[2]	6	PIO1_13	✓	✓	✓
	flexspi0_B_DATA[3]	6	PIO1_14	✓	✓	✓
	flexspi0_B_DATA[4]	6	PIO2_17	✓	✓	✓
	flexspi0_B_DATA[5]	6	PIO2_18	✓	✓	✓
	flexspi0_B_DATA[6]	6	PIO2_22		✓	✓
	flexspi0_B_DATA[7]	6	PIO2_23		✓	✓

Figure 3. Octal/Hyper Flash connected to Port B

3.3 Image link region

FlexSPI boot image can be either XIP image or Non-XIP image.

- An XIP image can only be linked at address `0x08001000`, and the first 4 KB of FlexSPI map region is used to store flash config block.
- An Non-XIP image should be linked into internal 4.5 MB SRAM. As the first 112 KB SRAM has been occupied by ROM until after boot, and the region `0x1C000` - `0x7FFFF` is shared memory between DSP and Cortex-M33, it is better to link Non-XIP image from `0x80000`. For applications which do not use the DSP, Non-XIP image can be linked starting from `0x1C000`.

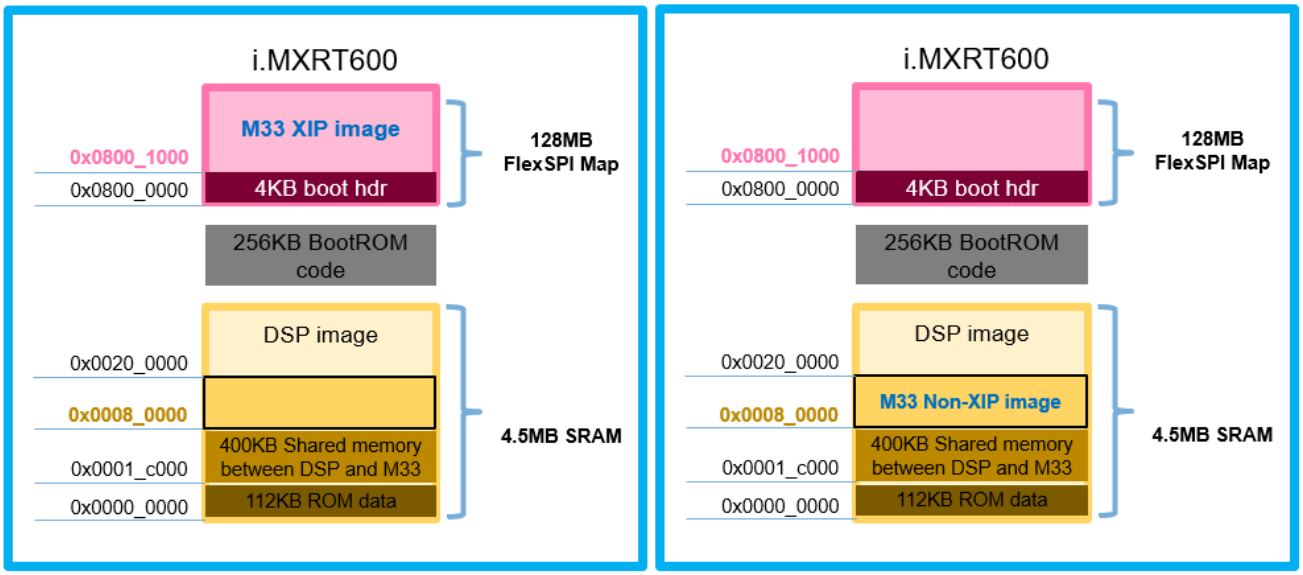


Figure 4. Image link region

3.4 FlexSPI boot flow

The ROM bootloader enters the FlexSPI master boot mode if this mode is determined by ISP pins or OTP configuration.

Figure 5 illustrates the brief FlexSPI boot flow.

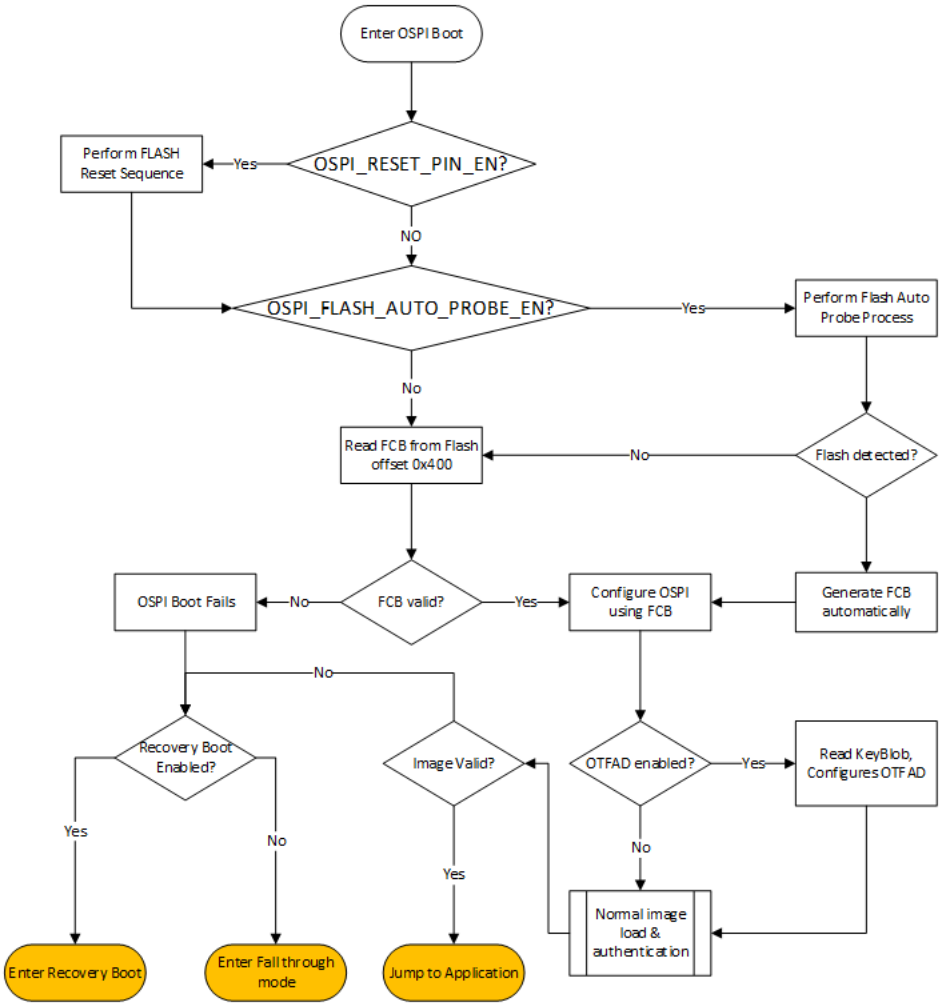


Figure 5. FlexSPI boot flow

When the FlexSPI boot process starts, the ROM bootloader will try to configure FlexSPI peripheral to access the Serial NOR flash device by employing Flash Configuration Block (FCB) at offset 0x400 on the Flash device or the Flash Auto-probe feature specified in OTP.

Only when the Serial NOR flash device has been well accessed, the ROM bootloader stays loading the boot image from the external flash device to the on-chip SRAM if it is a Non-XIP image, or keeping it where it is if it is a XIP image, and then do integrity check/image authentication with the image.

The bootloader jumps to the boot image if the integrity check/authentication passes, otherwise, it falls through to the recovery boot mode or ISP mode.

3.5 FlexSPI boot OTP settings

The ROM bootloader supports access to different QSPI/OSPI NOR Flash devices from various vendors via the OSPI interface using 1-bit, 2-bit (dual), 4-bit (quad) or 8-bit (octal) mode.

If `FLEXSPI_AUTO_PROBE_EN` in OTP is blown, the ROM bootloader will perform Flash auto-probe sequence using parameters blown in `FLEXSPI_PROBE_TYPE` field which defines the Flash Auto-probe type, `FLEXSPI_FLASH_TYPE` field which defines the Flash type, and `FLEXSPI_FREQUENCY` which defines the Flash access speed.

Table 6. FlexSPI boot OTP field

Field name	Enum name	Description	Offset	Width	Value
FLEX_SPI_AUTO_PROBE_EN	—	Quad/Octal-SPI flash auto probe feature enable.	0	1	—
FLEX_SPI_PROBE_TYPE	—	Quad/Octal-SPI flash probe type.	1	3	—
	QSPI_NOR	QuadSPI NOR	—	—	b'000
	MICRON_OCTAL	Micron Octal FLASH	—	—	b'001
	MACRONIX_OCTAL	Macronix Octal FLASH	—	—	b'010
	ADESTO_OCTAL	Adesto Octal FLASH	—	—	b'011
	—	Reserved	—	—	b'100
		Reserved			b'101
	—	Reserved	—	—	b'110
	—	Reserved	—	—	b'111
FLEX_SPI_FLASH_TYPE	—	Define typical Serial NOR Flash types	4	3	—
	QSPI_ADDR_3B	Device supports 3B read by default	—	—	b'000
	—	Reserved			b'001
	HYPER_1V8	HyperFLASH 1V8	—	—	b'010
	HYPER_3V3	HyperFLASH 3V3	—	—	b'011
	OSPI_DDR_MXIC	MXIC Octal DDR	—	—	b'100
	OSPI_DDR_MICRON	Micron Octal DDR	—	—	b'101
	—	Reserved	—	—	b'110
	—	Reserved	—	—	b'111
FLEX_SPI_FREQUENCY	—	Q/O-SPI flash interface frequency.	11	3	—
	QSPI_60MHZ	60 MHz	—	—	b'000
	QSPI_80MHZ	80 MHz	—	—	b'001
	QSPI_90MHZ	90 MHz	—	—	b'010
	QSPI_100MHZ	100 MHz	—	—	b'011
	—	Reserved	—	—	b'100
	—	Reserved	—	—	b'101
	—	Reserved	—	—	b'110
	—	Reserved	—	—	b'111

3.6 Flash config block

If FLEXSPI_AUTO_PROBE_EN in OTP is not blown, the ROM bootloader will look at offset 0x400 on the Flash device. If data at offset 0x400 equals to 0x42464346, the ROM bootloader will read the whole 512-byte FCB into on-chip SRAM and configure the FlexSPI controller using this FCB accordingly.

Table 7. FlexSPI boot Image layout

Offset	Width (Bytes)	Field	Description
0x0000_0400	512	Flash Config Block	The OSPI FLASH configuration block. This block is required if the FLEXSPI_AUTO_PROBE_EN is not blown on the OTP.
0x0000_1000	Image size	Bootable Image	The boot image, starts with valid image header.

Figure 6 shows an example FCB for Octal Flash.

```
const flexspi_boot_config_t flexspi_config = {
    .tag                  = FLASH_CONFIG_BLOCK_TAG,
    .version              = FLASH_CONFIG_BLOCK_VERSION,
    .readSamplingOption   = 3,
    .csHoldTime           = 3,
    .csSetupTime          = 3,
    .columnAddressWidth   = 0,
    .deviceModeCfgEnable  = 1,
    .deviceModeType       = 0,
    .waitTimeCfgCommands  = 1,
    .deviceModeSeq        = 0x0601,
    .deviceModeArg        = 0,
    .configCmdEnable      = 1,
    .configModeType       = {0, 2, 0},
    .configCmdSeqs        = {0x0701, 0x0a01},
    .configCmdArgs        = {0x2, 0x1},
    .controllerMiscOption = (1u << kFlexSpiMiscOffset_SafeConfigFreqEnable) | (1u << kFlexSpiMiscOffset_DdrModeEnable),
    .deviceType           = 0x1,
    .sflashPadType        = kSerialFlash_8Pads,
    .serialClkFreq        = kFlexSpiSerialClk_96MHz,
    .flashA1Size          = 0,
    .flashA2Size          = 0,
    .flashB1Size          = BOARD_FLASH_SIZE,
    .flashB2Size          = 0,
    .csPadSettingOverride = 0,
    .sclkPadSettingOverride = 0,
    .dataPadSettingOverride = 0,
    .dqsPadSettingOverride  = 0,
    .timeoutInMs          = 0,
    .coarseTuning         = 0,
    .fineTuning           = 0,
    .samplePoint          = 0,
    .dataHoldTime         = 0,
    .busyOffset           = 0,
    .busyBitPolarity      = 0,
    .lut                  = {[0] = 0x071307ec,  [1] = 0x33140b20,  [2] = 0x2704,      [4] = 0x24040405,  [8] = 0x07fa0705,
            [9] = 0x33140b20,  [10] = 0x2704,     [12] = 0x0406,     [16] = 0x07f90706, [20] = 0x07de0721,
            [21] = 0x0b20,     [24] = 0x04000472, [25] = 0x04030400, [26] = 0x20010400, [28] = 0x04000472,
            [29] = 0x04020400, [30] = 0x20010400, [32] = 0x072307dc, [33] = 0x0b20,     [36] = 0x07ed0712,
            [37] = 0x23042B20, [40] = 0x04000472, [41] = 0x04000400, [42] = 0x20010400, [44] = 0x079f0760},
    .pageSize           = 0x100,
    .sectorSize         = 0x1000,
    .ipcmdSerialClkFreq = 1,
    .isUniformBlockSize = 0,
    .blockSize          = 0x10000,
    .isNonBlockingMode  = 0,
};
```

Figure 6. Flash config block

4 MIMXRT685 EVK board settings

There are two memories that can be connected to FlexSPI port in EVK board: Octal Flash and pSRAM. So for FlexSPI boot, there is nothing to do with board settings.

The octal Flash is **MX25UM51345** and it is connected to FlexSPI0 Port B.

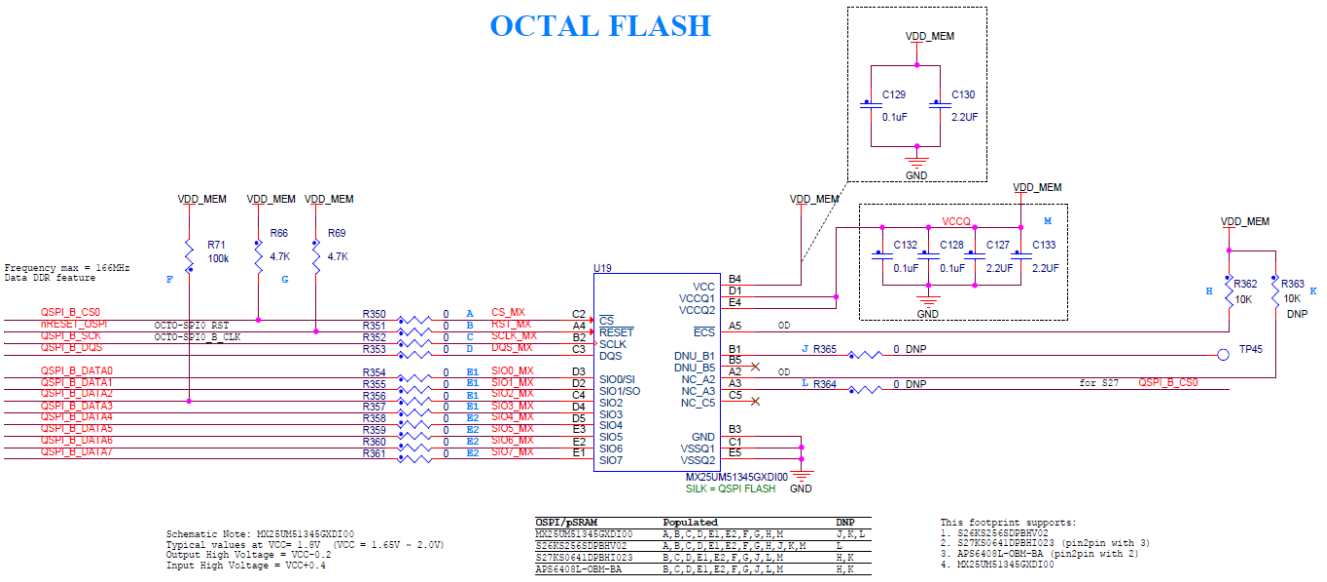


Figure 7. Octal Flash connected to PORT B

5 Program tools

5.1 Using Blhost to enable FlexSPI boot

This chapter shows the steps that use the `blhost` tool to program an XIP image to Octal Flash and Boot from the Octal Flash.

The `blhost` is a command-line host program used to interface with devices running ROM Bootloader. The version of blhost should be v2.3 or higher.

1. Open the `\SDK_2.7.0_EVK-MIMXRT685\boards\evkmimxrt685\driver_examples\gpio\led_output` example and select the project configuration as `flash_debug`. The settings are as shown in Figure 8.

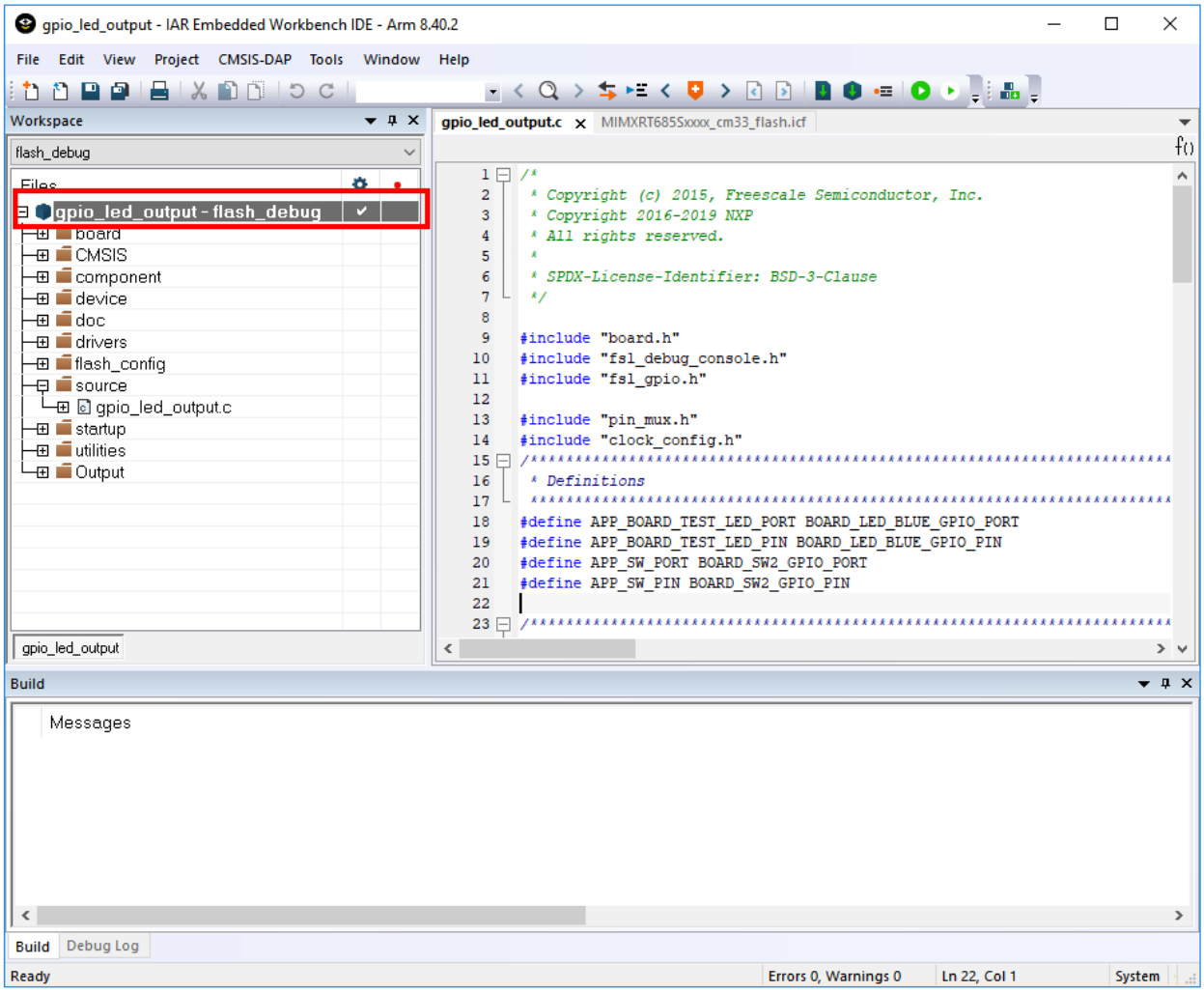


Figure 8. Selecting the project configuration as flash_debug

2. Make sure that `BOOT_HEADER_ENABLE` is set to **1** in **Preprocessor Option**. Only when this MACRO is enabled, the image binary will contain FCB data.

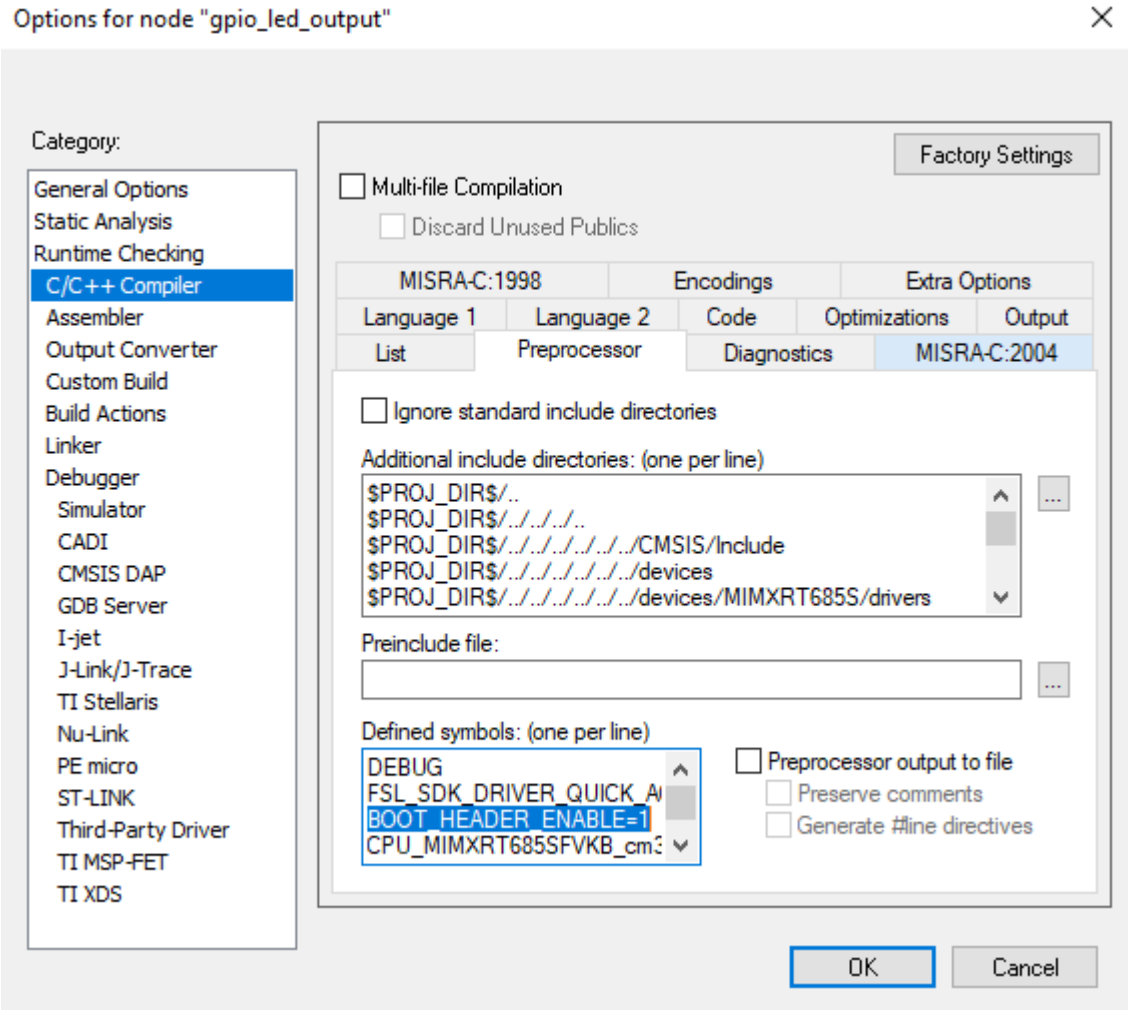


Figure 9. Setting BOOT_HEADER_ENABLE

3. Build the project and generate an image with the `.bin` format. You can find the `gpio_led_output.bin`, as shown in Figure 10. This binary contains FCB data and correct image header.

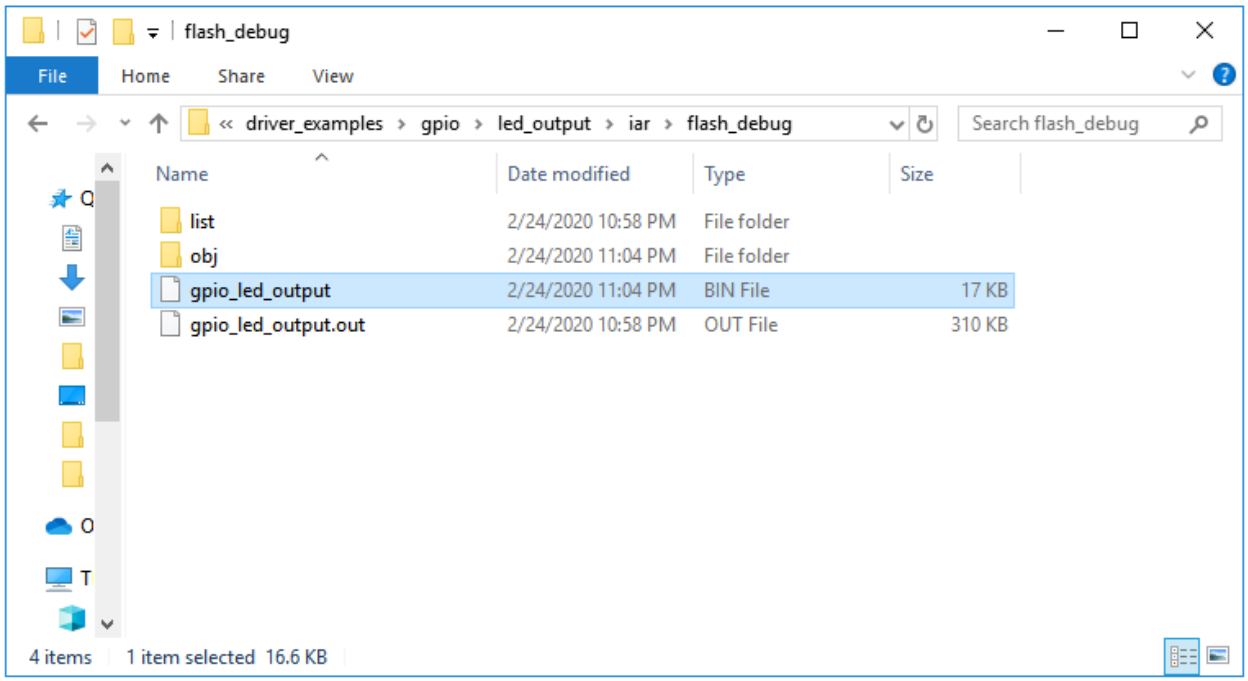


Figure 10. gpio_led_output.bin location

4. Copy the new `gpio_led_output.bin` to the `blhost` folder.

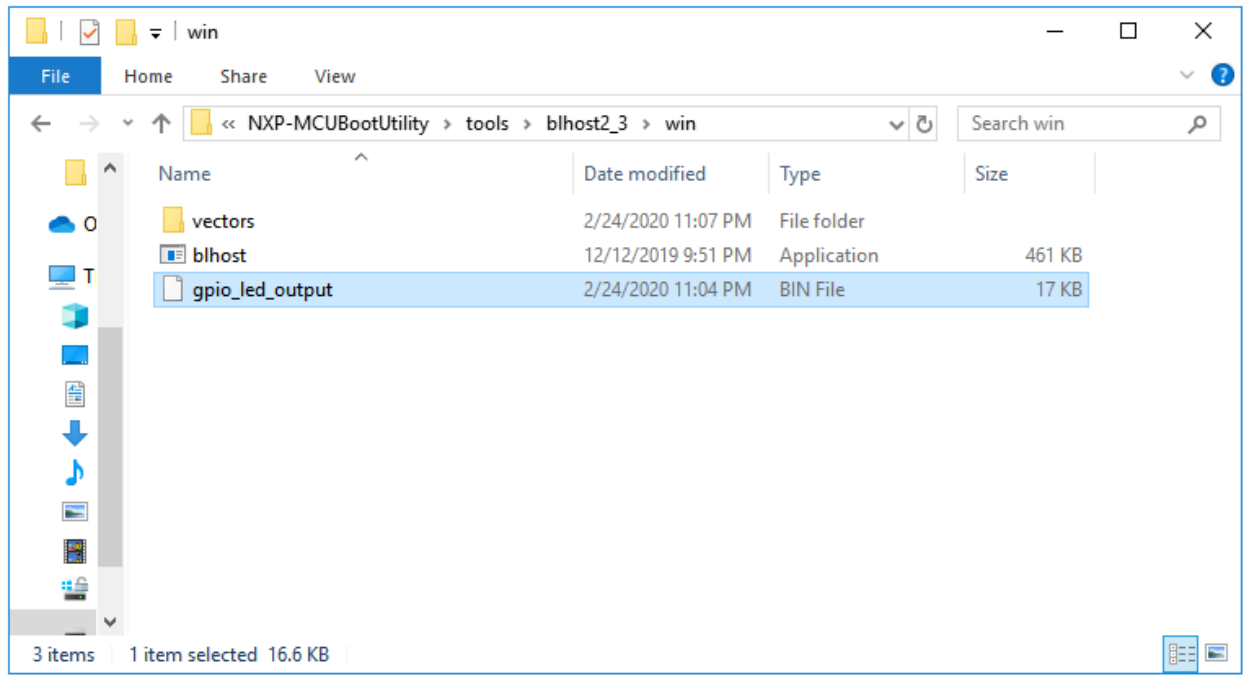


Figure 11. Copying gpio_led_output.bin

5. Switch the RT685-EVK board to Serial ISP mode by setting SW5 to **1-ON, 2-OFF, 3-OFF**, as shown in Figure 12.

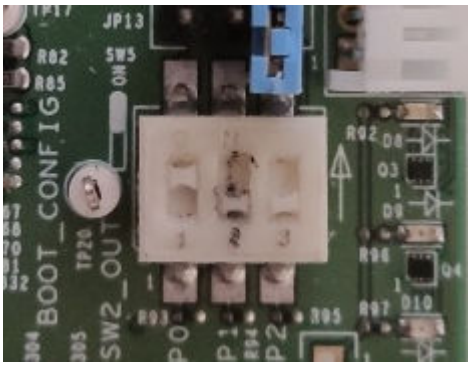


Figure 12. Setting SW5 to 1-ON, 2-OFF, 3-OFF

Connect a USB cable to J7 USB port and issue the `blhost` commands as shown in Figure 13.

```
Windows PowerShell
PS D:\NXP-MCUBootUtility\tools\blhost2_3\win> .\blhost.exe -u 0x1fc9,0x0020 -- fill-memory 0x1c000 4 0xc1503051
Inject command 'fill-memory'
Successful generic response to command 'fill-memory'
Response status = 0 (0x0) Success.
PS D:\NXP-MCUBootUtility\tools\blhost2_3\win> .\blhost.exe -u 0x1fc9,0x0020 -- fill-memory 0x1c004 4 0x20000014
Inject command 'fill-memory'
Successful generic response to command 'fill-memory'
Response status = 0 (0x0) Success.
PS D:\NXP-MCUBootUtility\tools\blhost2_3\win> .\blhost.exe -u 0x1fc9,0x0020 -- configure-memory 0x9 0x1c000
Inject command 'configure-memory'
Successful generic response to command 'configure-memory'
Response status = 0 (0x0) Success.
PS D:\NXP-MCUBootUtility\tools\blhost2_3\win> .\blhost.exe -u 0x1fc9,0x0020 -- flash-erase-region 0x08000400 0x6000
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.
PS D:\NXP-MCUBootUtility\tools\blhost2_3\win> .\blhost.exe -u 0x1fc9,0x0020 -- write-memory 0x08000400 .\gpio_led_output.bin
Inject command 'write-memory'
Preparing to send 17048 (0x4298) bytes to the target.
Successful generic response to command 'write-memory'
(1/1)100% Completed!
Successful generic response to command 'write-memory'
Response status = 0 (0x0) Success.
Wrote 17048 of 17048 bytes.
PS D:\NXP-MCUBootUtility\tools\blhost2_3\win>
```

Figure 13. `blhost` command sequences

The argument value **0xc1503051** and **0x20000014** in fill-memory command is FlexSPI boot config option block. For details, see .

6. Switch the RT685-EVK board to FlexSPI Port B boot mode by setting SW5 to **1-ON, 2-OFF, 3-ON**, as shown in Figure 14.

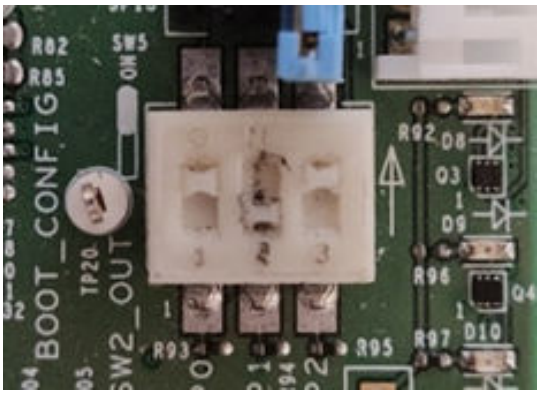


Figure 14. Setting SW5 to 1-ON, 2-OFF, 3-ON

7. Reset the board. The **gpio led** demo will run properly.

5.2 Using NXP-MCUBootUtility to enable FlexSPI boot

This chapter shows the steps that use the NXP-MCUBootUtility tool to program an image to Octal Flash and Boot from the Octal Flash.

The NXP-MCUBootUtility is a GUI tool used to interface with devices running ROM Bootloader. It is a real one-stop tool. The version of NXP-MCUBootUtility should be v2.3 or higher.

1. Rebuild `\SDK_2.7.0_EVK-MIMXRT685\boards\evkmimxrt685\driver_examples\gpio\led_output` project and generate an image with `.srec` format.
2. Switch the RT685-EVK board to the Serial ISP mode, connect a USB cable to J7 USB port, and open the NXP-MCUBootUtility. Set MCU device to **i.MXRT6xx** and Boot Device to **FLEXSPI NOR**. Click **Boot Device Configuration** to double check, and then click **Connect to ROM**.

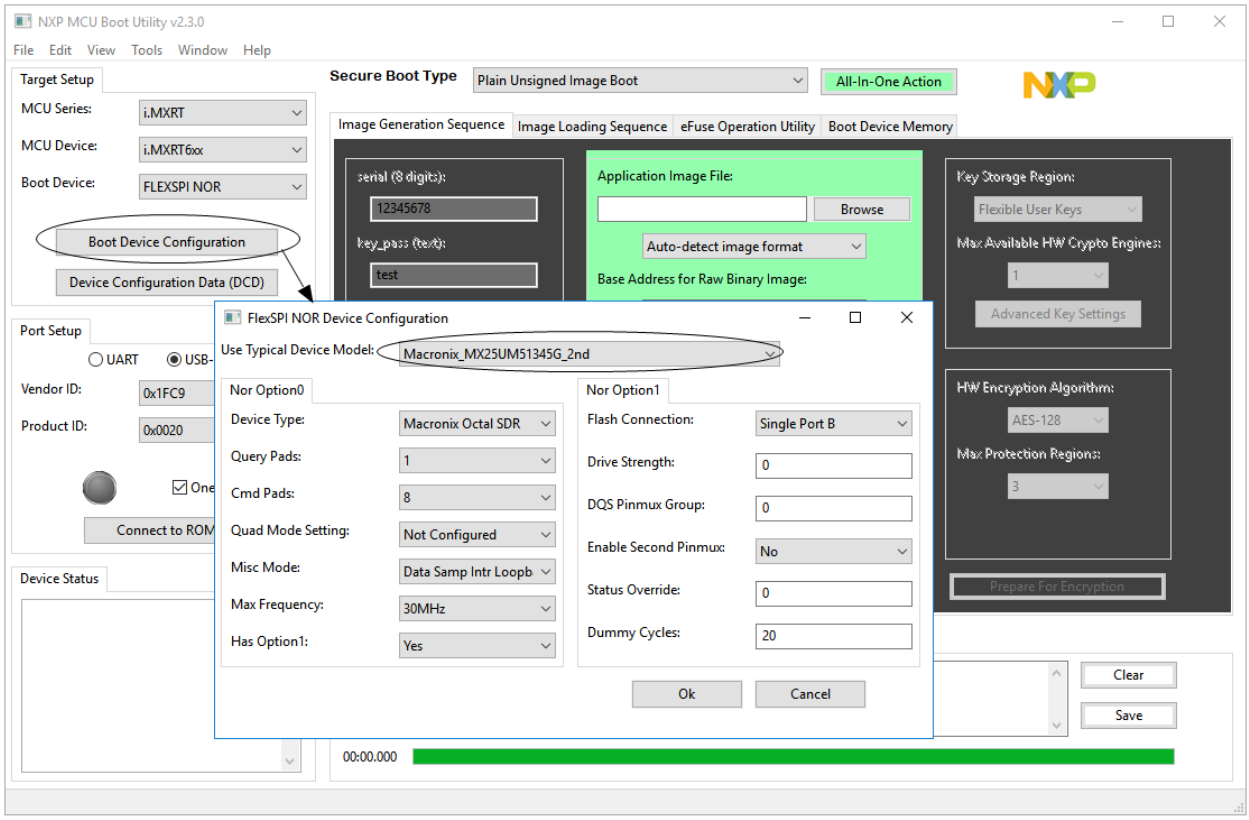


Figure 15. MCUBootUtility settings

3. If the tool can connect with RT600 BootROM successfully, the device information will be showed in **Device Status**. Browse the `gpio_led_output.srec` file and click **All-In-One Action**, as shown in Figure 16.

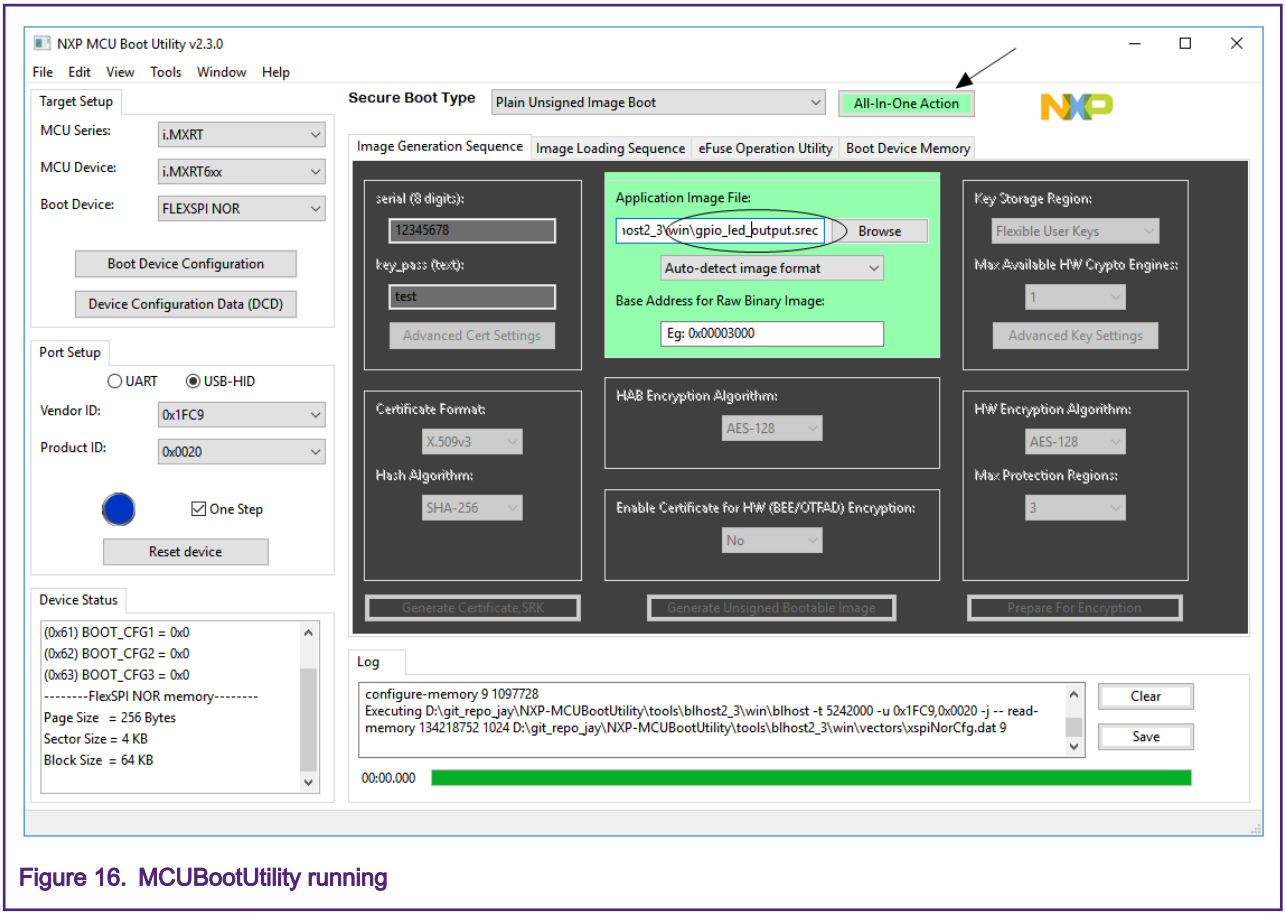


Figure 16. MCUBootUtility running

4. Switch the RT685-EVK board to FlexSPI Port B boot mode by setting SW5 to **1-ON, 2-OFF, 3-ON** to reset the board.
5. Reset the board. The **gpio led** demo will run properly.

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. 2020. **All rights reserved.**

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

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

Date of release: March 12 2020

Document identifier: AN12773

arm