NXP Semiconductors
Application Notes

Document Number: AN12183
Rev. 0, 05/2018

How to Enable Debugging for FLEXSPI NOR Flash

Contents

1. Introduction 1
2. MIMXRT1050 EVK board settings 1
   2.1. EVK Settings 2
   2.2. EVKB Settings 3
3. XIP boot flow 3
4. OpenSDA firmware Update 7
5. Examples 7
   5.1. How to add or remove boot header for XIP targets. 7
   5.2. Program the image to On-Board Hyper Flash 10
   5.3. Program the image to On-Board QSPI NOR Flash11
   5.4. Program the image to a new QSPI NOR Flash 12
6. Revision history 15

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 external storage device. In order to program image to flash and boot from flash and debug, the new Dap-link Firmware and SDK are provided. This application note will show how to program, debug and configure a new FLEXSPI NOR flash. For information about Flashloader, MfgTool, please refer to the application note How to Enable Boot from HyperFlash and SD Card AN12107 and How to Enable Boot from QSPI Flash AN12108.

The software used for example in this application note are based on the MIMXRT1050 SDK (Release Version: 2.3.1). The development environment is IAR Embedded Workbench 8.22.1. The hardware development environment is IMXRT1050-EVKB Board.

2. MIMXRT1050 EVK board settings

There are two On-Board Flashes on the EVK board: Hyper Flash and QSPI NOR Flash. The Hyper Flash is the default Flash. In order to enable the On-Board QSPI NOR Flash, EVK Board needs to change.

2.1. EVK Settings

Step1:

- The On-Board Hyper Flash should be removed, otherwise it will impact the QSPI NOR Flash read and write timing.

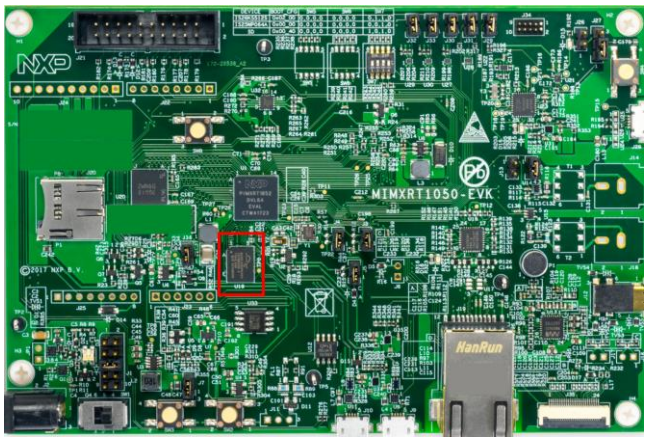

Figure 1. Remove the Hyper Flash

Step2:

- Weld 0 Ω resistor to the pad from R153 to R158.

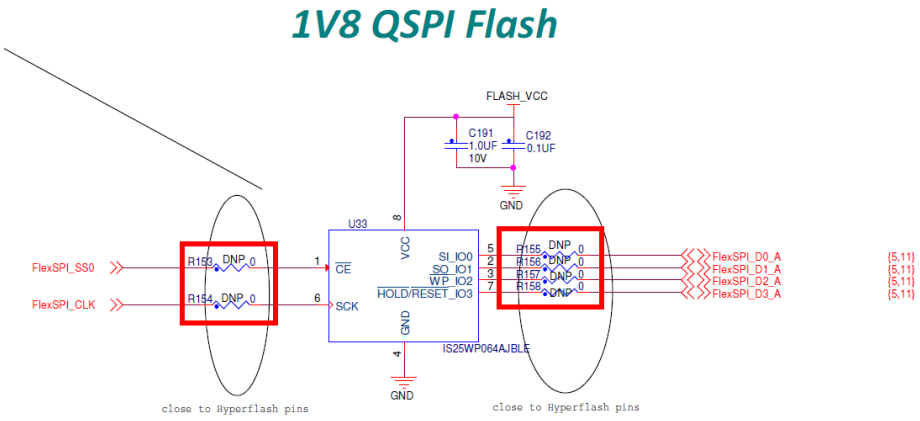

Figure 2. Weld 0 Ω resistor to the pad from R153 to R158

Step3:

- The firmware of OpenSDA needs to be replaced. The default firmware On-Board is used to Hyper Flash, so that the firmware should be replaced to QSPI NOR Flash. Both Hyper Flash and QSPI NOR Flash's firmware can be downloaded from NXP Website.

2.2. EVKB Settings

For EVKB board, the On-Board Hyper Flash doesn't need to remove.

Removed resistors: R356, R361 - R366.

Weld 0 Ω resistors: R153 - R158.

Follow the Step3 of chapter 2.1 to update the OpenSDA firmware.

Now the On-Board QSPI NOR Flash is ready to use.

3. XIP boot flow

The boot process begins at the Power-On Reset (POR) where the hardware reset logic forces the Arm core to begin the execution starting from the on-chip boot ROM. The boot ROM uses the state of the BOOT_MODE register and eFUSEs to determine the boot device. For development purposes, the eFUSEs used to determine the boot device may be overridden using the GPIO pin inputs. The boot ROM code also allows to download the programs to be run on the device. The example is a provisioning program that can make further use of the serial connection to provide a boot device with a new image. Typically, the internal boot is selected for normal boot, which is configured by external BOOT_CFG GPIOs. The *Table* 1shows the typical Boot Mode and Boot Device settings.

Table 1. Typical Boot Mode and Boot Device settings

SW7-1	SW7-2	SW7-3	SW7-4	Boot Device
OFF	ON	ON	OFF	Hyper Flash
OFF	OFF	ON	OFF	QSPI NOR Flash
ON	OFF	ON	OFF	SD Card

Figure 4 shows FlexSPI NOR Flash Boot flow. The ROM expects the 512-byte FlexSPI NOR configuration parameters to be present at offset 0 in Serial NOR flash. The ROM reads these configuration parameters using the read command specified by BOOT_CFG2[2:0] with Serial clock operating at 30 MHz. The Flash Configuration Parameters including read command sequence, FlexSPI frequency, quad mode enablement sequence (optional), etc (More details in RM 8.6.3). Rom code will configure FlexSPI with these parameters.

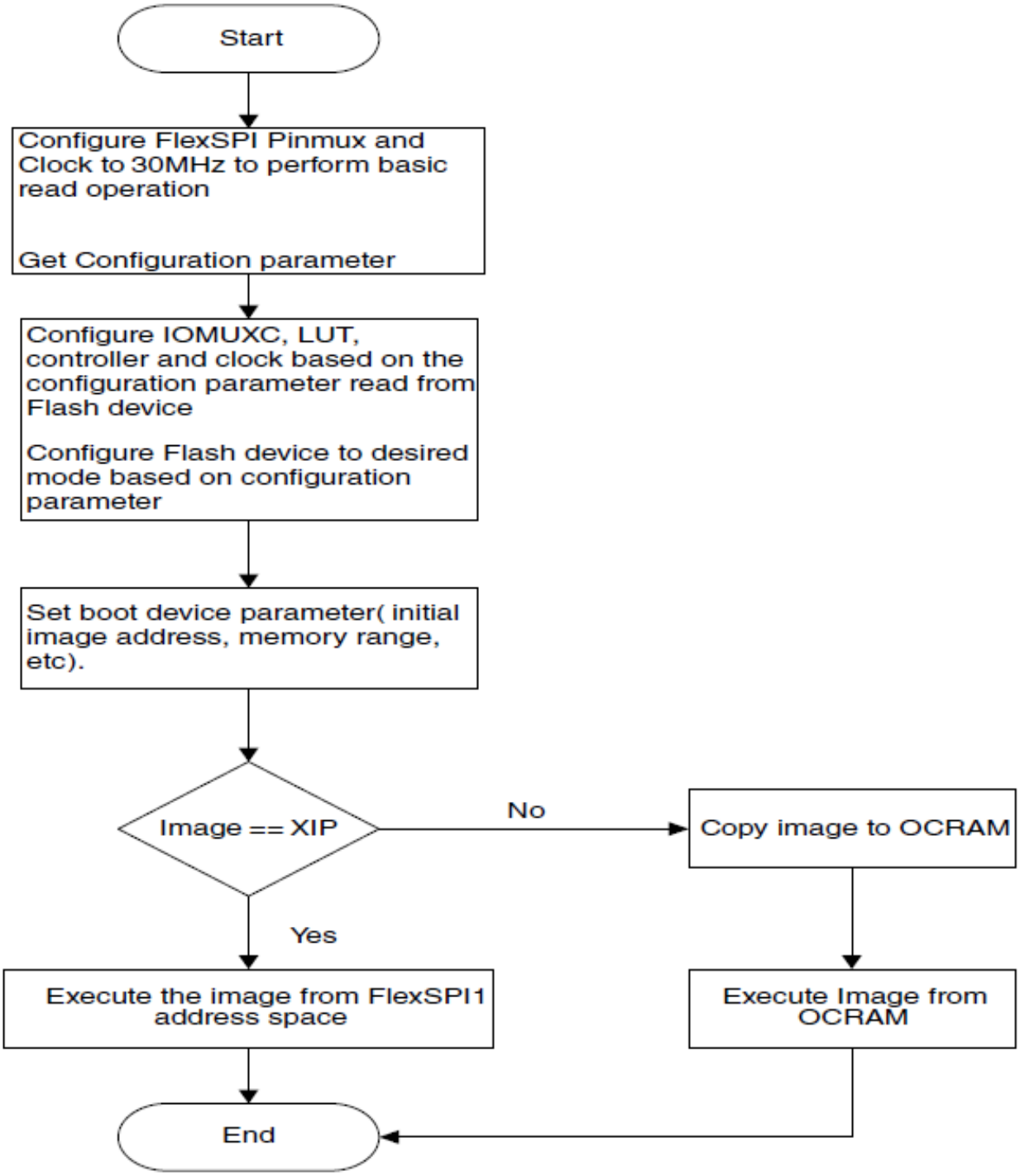

Figure 3. FlexSPI NOR Boot Flow

Then Rom code will get some key information about App Image, IVT (Image Vector Table), Boot Data and DCD (Device Configuration Data). IVT, Boot Data, DCD and user's code make up an App image.

A boot image which can program to FlexSPI NOR Flash directly should consist of:

• **Flash Configuration Parameters** —Read command sequence, FlexSPI frequency, quad mode enablement sequence (optional), etc(More details in RM 8.6.3). Search for “hyperflash_config” on SDK, the setting can be found on SDK.

• **Image Vector Table (IVT)** — a list of pointers located at a fixed address that the ROM examines to determine where the other components of the program image are located. Search for “image_vector_table” on SDK, the setting can be found on SDK. More details in RM 8.7.1.

• **Boot data** — a table that indicates the program image location, program image size in bytes, and the plugin flag. Search for “boot_data” on SDK, the setting can be found on SDK.

• **Device Configuration Data (DCD)** — IC configuration data (ex: SDRAM register config). More details for DCD Format can be found in RM 8.7.2. Because DCD data is stored in binary, it is hard to understand and modified. There is a DCD Tool that can convert the configuration text file to a binary file. Search for “dcd_data[]” on SDK, the setting can be found on SDK.

• **User code and data**.

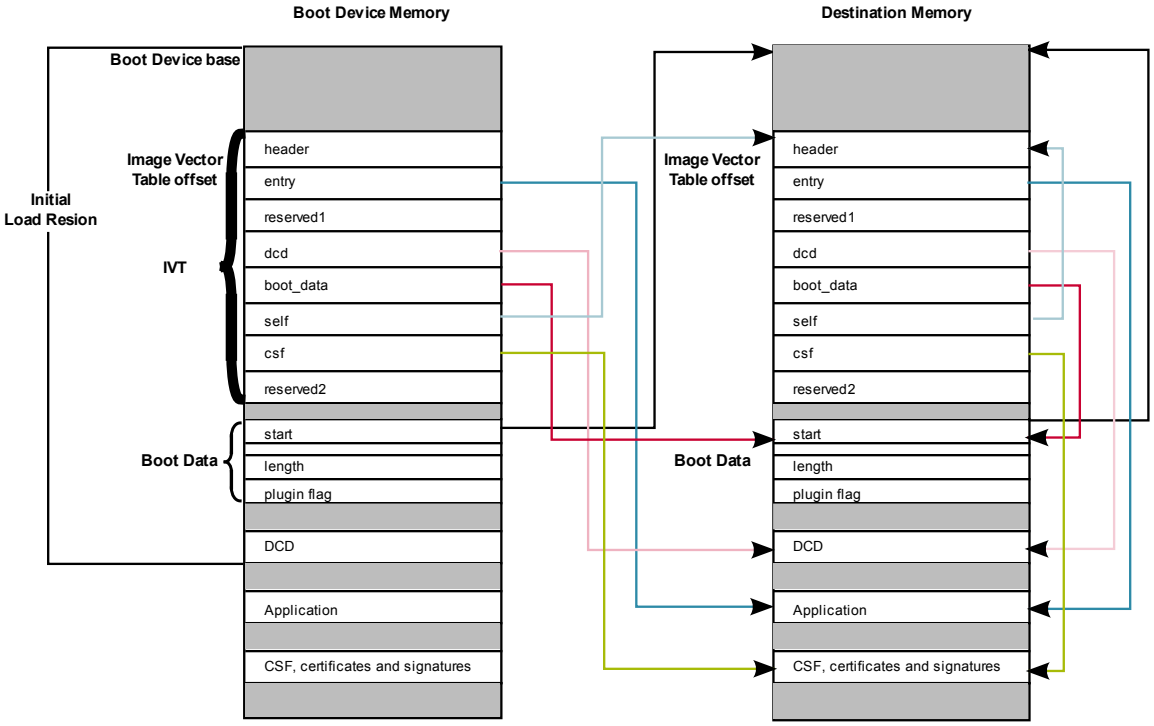

Figure 4. Bootable image layout

Open the link file *MIMXRT1052xxxxx_flexspi_nor.icf*, the address layout of Flash Configuration Parameters, IVT, Boot Data and DCD Data can be found.

```
define exported symbol m_boot_hdr_conf_start = 0x60000000;
define symbol m_boot_hdr_ivt_start           = 0x60001000;
define symbol m_boot_hdr_boot_data_start     = 0x60001020;
define symbol m_boot_hdr_dcd_data_start      = 0x60001030;
```

Figure 5. Bootable image address layout

Open a generated image, such as hello_world.bin. The Flash Configuration Parameters are at the front. The tag of Flash Configuration Parameters is *0x42464346*, ascii is *FCFB* as *Figure* 6. More details can be found on RM 8.6.3.1.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	46	43	46	42	00	04	01	56	00	00	00	00	03	03	03	03	FCFB...V........
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000040	59	00	00	00	00	08	07	00	00	00	00	00	00	00	00	00	Y...............
00000050	00	00	00	04	00	00	00	00	00	00	00	00	00	00	00	00	....
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000070	00	00	00	00	00	00	00	00	10	00	10	00	00	00	00	00	....
00000080	a0	87	18	8b	10	8f	06	b3	04	a7	00	00	00	00	00	00	.??..??.....□□h□
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001c0	00	02	00	00	00	00	04	00	00	01	00	00	00	00	00	00	....
000001d0	00	00	04	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

Figure 6. Flash Configuration Parameters address layout

The tag of IVT is 0xD1, the tag can be found on 0x1000. The boot start address offset is 0x1020, the data is 0x6000000 which matches with Flash start address. And the DCD start address offset is 0x1030, the data is 0xD2 which matches with the Tag of DCD.

```
00001000 d1 00 20 41 00 20 00 60 00 00 00 00 30 10 00 60 ? A. .`....0..`□
00001010 20 10 00 60 00 10 00 60 00 00 00 00 00 00 00 00  ..`...`........
00001020 00 00 00 60 00 00 00 04 00 00 00 00 ff ff ff ff ...`........
00001030 d2 04 30 41 cc 03 ac 04 40 0f c0 68 ff ff ff ff ?0A??@.類
```

Figure 7. IVT, Boot Data and DCD Data start address layout

4. OpenSDA firmware Update

Almost all demos on SDK 2.3.1 support XIP demo. That means when using the default XIP target demos, the raw image will be added the Flash Configuration Parameters, IVT, Boot Data and DCD. So that no longer need OpenSDA firmware to add these information to the raw image. Either using the On-Board Hyper Flash or QSPI NOR Flash, the firmware needs to update to use the XIP demos.

If the number bigger than TR18132215, the firmware of OpenSDA will not add the configure information to the raw image. If not, please update the firmware from NXP web.

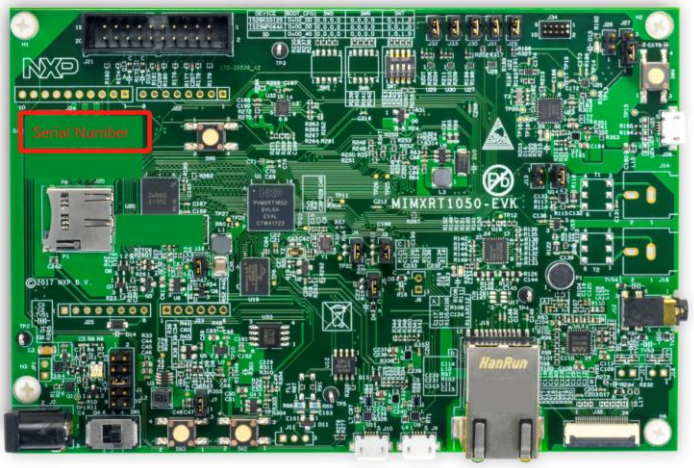

Figure 8. Serial Number

5. Examples

5.1. How to add or remove boot header for XIP targets

Now SDK for i.MX RT1050 provides `flexspi_nor_debug & flexspi_nor_release` targets for each example/demo which supports XIP (eXecute In Place). These two targets will add *XIP_BOOT_HEADER* to the image by default. Then ROM can boot and run this image directly on external flash.

NOTE

When using DapLink to debug flexspi_nor_debug & flexspi_nor_release targets, please set the breakpoint type to hardware breakpoint.

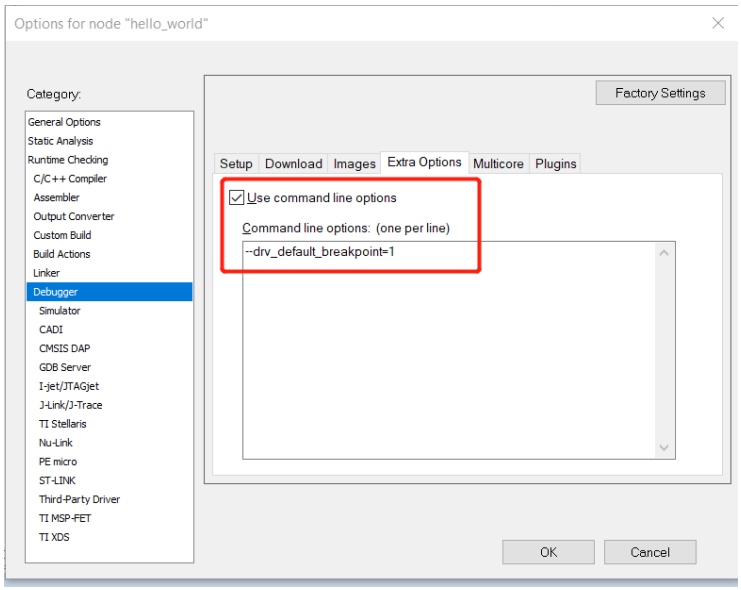

Figure 9. How to set the hardware breakpoint

5.1.1. Macros for the boot header

The *Table* 2 shows three macros that are added in flexspi_nor targets to support XIP:

Table 2. Macros for the boot header

XIP_EXTERNAL_FLASH	1: Exclude the code which will change the clock of flexspi. 0: make no changes.
XIP_BOOT_HEADER_ENABLE	1: Add flexspi configuration block, image vector table, boot data and device configuration data(optional) to the image by default. 0: Add nothing to the image by default.
XIP_BOOT_HEADER_DCD_ENABLE	1: Add device configuration data to the image. 0: Do **NOT** add device configuration data to the image.

The *Table* 3 shows the different effect on the built image with different combination of these macros:

Table 3. Different effect on the built image with difference macros

		XIP_BOOT_HEADER_DCD_ENABLE=1	XIP_BOOT_HEADER_DCD_ENABLE=0
XIP_EXTERNAL_FLASH=1	XIP_BOOT_HEADER_ENABLE=1	Can be programed to hyperflash by IDE and can run after POR reset if hyperflash is the boot source. SDRAM will be initialized.	Can be programed to hyperflash by IDE and can run after POR reset if hyperflash is the boot source. SDRAM will **NOT** be initialized.
	XIP_BOOT_HEADER_ENABLE=0	Can **NOT** run after POR reset if it is programed by IDE even if hyperflash is the boot source.	
XIP_EXTERNAL_FLASH=0		This image can **NOT** do XIP because when this macro is set to 1, it will exclude the code which will change the clock of flexspi.	

5.1.2. Where to change the macros in SDK?

Take *hello_world* as an example.

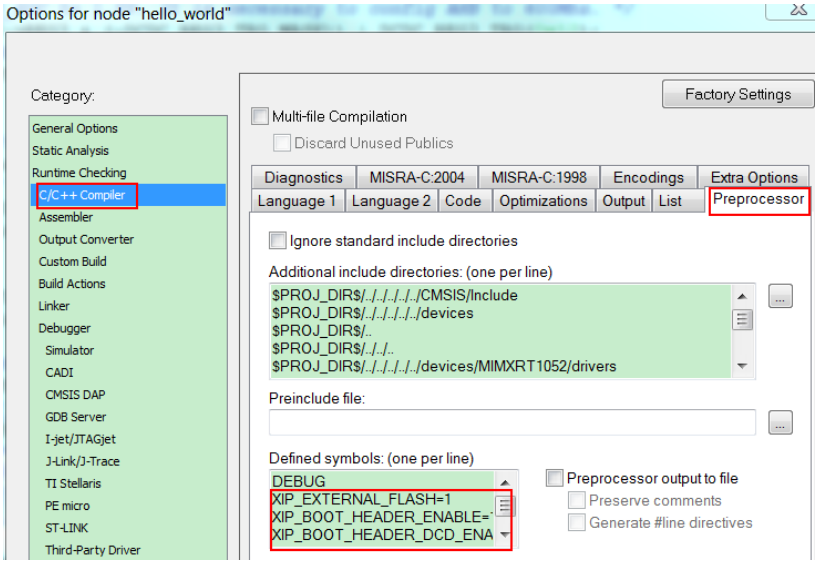

Figure 10. Where to change the SDK macros based on IAR

5.2. Program the image to On-Board Hyper Flash

Step1:

Configure the board to Hyper Flash Boot Mode by pull-up SW7-2 and SW7-3 and pull-down others. Then power on the EVK Board.

Step2:

Open the `hello_world` demo in the SDK and select the project configuration as `flexspi_nor_debug`. Then build the project and program the image to the Flash.

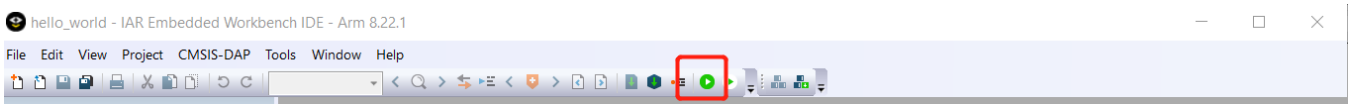

Figure 11. Build and program the project

Step3:

Open and configure the Terminal Window:

- Baud rate: 115200
- Data bits: 8
- Stop bit: 1
- Parity: None
- Flow control: None

Press SW3 to reset the EVK Board and “hello world” will be printed to the terminal.

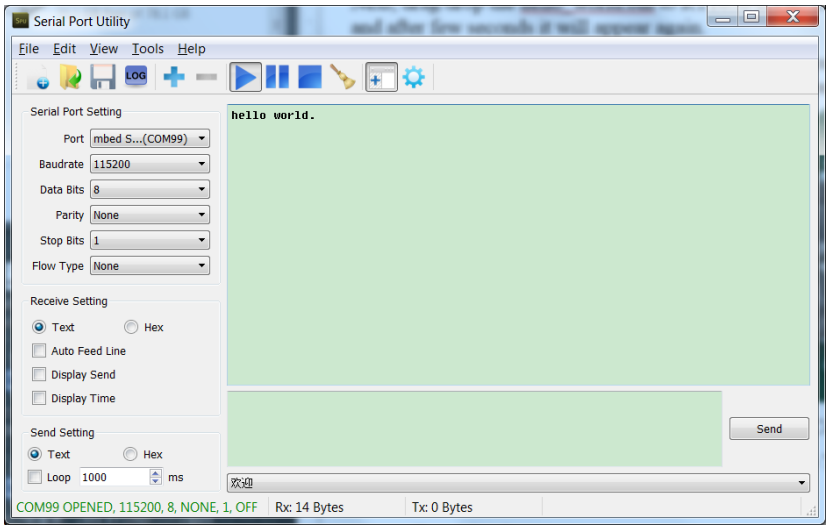

Figure 12. “Hello World.”

5.3. Program the image to On-Board QSPI NOR Flash

Step 1:

Configure the board to QSPI NOR Flash Boot Mode by pull-up SW7-3 and pull-down others. Change the firmware of OpenSDA to QSPI NOR Flash. Then power on the EVK Board.

Step2:

Open the `hello_world` demo in the SDK and select the project configuration as `flexspi_nor_debug`. Find "`evkbimxrt1050_hyper_config.c`" as *Figure* 13.

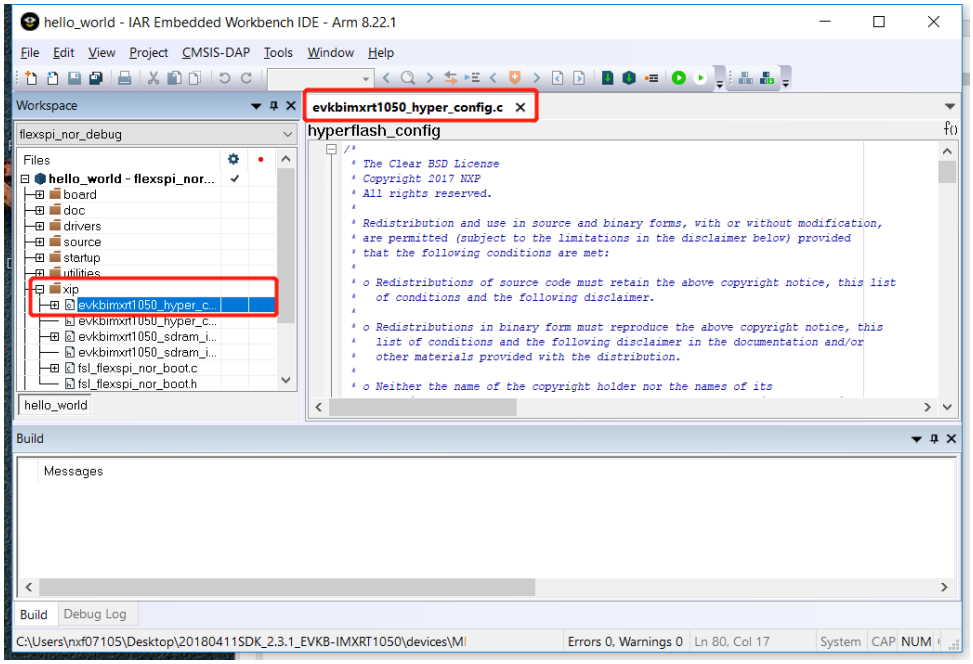

Figure 13. evkbimxrt1050_hyper_config.c

Step3:

Comment `const flexspi_nor_config_t hyperflash_config` and replace it as `const flexspi_nor_config_t qspiflash_config` (can replace evkbimxrt1050_hyper_config.c file in attachment. New file has been configured for QSPI NOR Flash).

```
const flexspi_nor_config_t qspiflash_config = {
    .memConfig =
        {
          .tag = FLEXSPI_CFG_BLK_TAG,
          .version = FLEXSPI_CFG_BLK_VERSION,
          .readSampleClkSrc = kFlexSPIReadSampleClk_LoopbackFromDqsPad,
          .csHoldTime = 3u,
          .csSetupTime = 3u,
          .columnAddressWidth = 0u,
          .configCmdEnable = 0u,
          .controllerMiscOption = 0u,
          .deviceType = kFlexSpiDeviceType_SerialNOR,
          .sflashPadType = kSerialFlash_4Pads,
          .serialClkFreq = kFlexSpiSerialClk_133MHz,
          .lutCustomSeqEnable = 0u,
          .sflashA1Size = 0x00800000u, /* 8MB/64Mbit */
          .lookupTable =
              {
                // Fast read sequence
                [0] = FLEXSPI_LUT_SEQ(CMD_SDR, FLEXSPI_1PAD, 0xEB, RADDR_SDR, FLEXSPI_4PAD, 0x18),
                [1] = FLEXSPI_LUT_SEQ(DUMMY_SDR, FLEXSPI_4PAD, 0x06, READ_SDR, FLEXSPI_4PAD, 0x02),
                [2] = FLEXSPI_LUT_SEQ(STOP, 0, 0, STOP, 0, 0),
                [3] = FLEXSPI_LUT_SEQ(STOP, 0, 0, STOP, 0, 0),
              },
        },
};
```

Figure 14. flexspi_nor_config_t qspiflash_config

Then build the project and program the image to the Flash. After these steps, “Hello World” can be printed on terminal.

5.4. Program the image to a new QSPI NOR Flash

5.4.1. How to program the image to GD25LQ64C

This section will outline how to use a new QSPI NOR Flash. Take GD25LQ64C for example.

Step1:

Replace the *const flexspi_nor_config_t hyperflash_config* as c*onst flexspi_nor_config_t qspiflash_config*.

Step2:

Open the IAR project(FlashIMXRT1050_EVK_FlexSPI_Example) in the attachment. Build the project and find FlashIMXRT1050_EVK_FlexSPI.out. Then copy it to IAR install path.

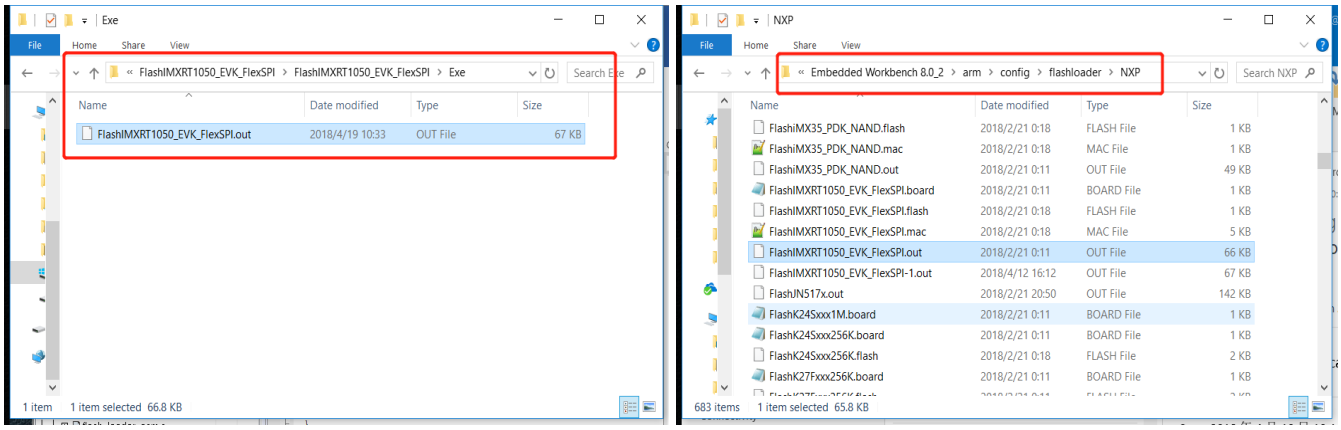

Figure 15. Update IAR flashloader

Step3:

Build the project and download. Then “Hello World” can be printed on terminal.

5.4.1.1. What is the difference between the two Flash configure parameters?

The main difference is the LUT (Look Up Table). The LUT (Look Up Table) is an internal memory to preserve a number of preprogrammed sequences. Each sequence consists of up to 8 instructions which are executed sequentially. When a flash access is triggered by an IP command or an AHB command, FlexSPI controller will fetch the sequence from LUT according to sequence index/number and execute it to generate a valid flash transaction on SPI interface.

Second is Read Sample Clock Source, Hyper Flash uses External Input from DQS Pad but QSPI NOR Flash uses Loopback from DQS Pad.

Third is Serial Flash Type, Hyper Flash is Octal and the QSPI NOR Flash is Quad.

A comparison tool can help to find other differences.

5.4.1.2. What is the difference between the two flashloaders?

The main difference is that the QE bit position between of GD and ISSI are different. The *Figure* 16 shows the main difference between two flash loaders. The left one is the original function and the other one is the modified function.

```
/*Read A1 ID*/
ReadID(0x00000000,&id);

if(0x9D == (id & 0x000000FF))


{
  a1_size = 1<<((id >> 16) & 0x00FF);
  ReadStatusReg(0x00000000, &a1_StatusReg);

  if(!((a1_StatusReg &= 0xFC)& (1<<6)))
  {
    WriteEnable(0x00000000);
    WriteStatusReg(0x00000000, a1_StatusReg | (1<<6));
  }
}
```

```
/*Read A1 ID*/
ReadID(0x00000000,&id);

a_mf_id = id & 0x000000FF;

if(0xC8 == a_mf_id)
{
  a1_size = 1<<((id >> 16) & 0x00FF);
  ReadStatusReg(0x00000000, &a1_StatusReg1);
  ReadStatusReg2(0x00000000, &a1_StatusReg2);
  a1_StatusReg = ((a1_StatusReg2 | (1<<1)) << 8) | (a1_StatusReg1 & 0xFC);
  if(!(a1_StatusReg2 & (1<<1)))
  {
    WriteEnable(0x00000000);
    WriteStatusReg16(0x00000000, a1_StatusReg);
  }
}
else if(0x9D == a_mf_id)
{
  a1_size = 1<<((id >> 16) & 0x00FF);
  ReadStatusReg(0x00000000, &a1_StatusReg1);
  if(!((a1_StatusReg1 &= 0xFC)& (1<<6)))
  {
    WriteEnable(0x00000000);
    WriteStatusReg8(0x00000000, a1_StatusReg1 | (1<<6));
  }
}
```

Figure 16. Difference between the two flashloaders

Other difference can be found by comparison tool.

NOTE

The default flashloader can be found in your IAR install path: IAR Systems\Embedded Workbench 8.0_2\arm\src\flashloader\NXP\FlashIMXRT1050_EVK_FlexSPI

The modified flashloader can be found in the attachment file.

5.4.2. How to program the image to GD25Q64C

This section will outline how to use a new QSPI NOR Flash. Take GD25Q64C for example. Besides the value of power supply, there are some difference between GD25LQ64C and GD25Q64C.

NOTE

The power supply of GD25Q64C is 3.3 V, but the default power supply is 1.8V. Remember change the power supply voltage.

Table2. Commands (Standard/Dual/Quad SPI)

Command Name	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	n-Bytes
Write Enable	06H						
Write Disable	04H						
Volatile SR Write Enable	50H						
Read Status Register	05H	(S7-S0)					(continuous)
Read Status Register-1	05H	(S15-S8)					(continuous)
Write Status Register	01H	S7-S0	S15-S8				
Read Data	03H	A23-A16	A15-A8	A7-A0	(D7-D0)	(Next byte)	(continuous)
Fast Read	0BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)	(continuous)
Dual Output Fast Read	3BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)[1]	(continuous)
Dual I/O Fast Read	BBH	A23-A8[2]	A7-A0 M7-M0[2]	(D7-D0)[1]			(continuous)
Quad Output Fast Read	6BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)[3]	(continuous)
Quad I/O Fast Read	EBH	A23-A0 M7-M0[4]	dummy[5]	(D7-D0)[3]			(continuous)
Quad I/O Word Fast Read[7]	E7H	A23-A0 M7-M0[4]	dummy[6]	(D7-D0)[3]			(continuous)
Page Program	02H	A23-A16	A15-A8	A7-A0	D7-D0	Next byte	
Quad Page Program	32H	A23-A16	A15-A8	A7-A0	D7-D0		
Sector Erase	20H	A23-A16	A15-A8	A7-A0			
Block Erase(32K)	52H	A23-A16	A15-A8	A7-A0			
Block Erase(64K)	D8H	A23-A16	A15-A8	A7-A0			
Chip Erase	C7/60H						
Enable QPI	38H						
Enable Reset	66H						
Reset	99H						
Set Burst with Wrap	77H	W6-W4					
Program/Erase Suspend	75H						
Program/Erase Resume	7AH						
Release From Deep	ABH	dummy	dummy	dummy	(ID7-ID0)		(continuous)

13

will happen and WEL will not be reset.

Table2. Commands (Standard/Dual/Quad SPI)

Command Name	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	n-Bytes
Write Enable	06H						
Write Disable	04H						
Volatile SR Write Enable	50H						
Read Status Register-1	05H	(S7-S0)					(continuous)
Read Status Register-2	35H	(S15-S8)					(continuous)
Read Status Register-3	15H	(S23-S16)					(continuous)
Write Status Register-1	01H	S7-S0					
Write Status Register-2	31H	S15-S8					
Write Status Register-3	11H	S23-S16					
Read Data	03H	A23-A16	A15-A8	A7-A0	(D7-D0)	(Next byte)	(continuous)
Fast Read	0BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)	(continuous)
Dual Output Fast Read	3BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)[1]	(continuous)
Dual I/O Fast Read	BBH	A23-A8[2]	A7-A0 M7-M0[2]	(D7-D0)[1]	(Next byte)	(Next byte)	(continuous)
Quad Output Fast Read	6BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)[3]	(continuous)
Quad I/O Fast Read	EBH	A23-A0 M7-M0[4]	dummy[5]	(D7-D0)[3]	(Next byte)	(Next byte)	(continuous)
Quad I/O Word Fast Read[7]	E7H	A23-A0 M7-M0[4]	dummy[6]	(D7-D0)[3]	(Next byte)	(Next byte)	(continuous)
Page Program	02H	A23-A16	A15-A8	A7-A0	D7-D0	Next byte	continuous
Quad Page Program	32H	A23-A16	A15-A8	A7-A0	D7-D0	Next byte	continuous
Fast Page Program	F2H	A23-A16	A15-A8	A7-A0	D7-D0	Next byte	continuous
Sector Erase	20H	A23-A16	A15-A8	A7-A0			
Block Erase(32K)	52H	A23-A16	A15-A8	A7-A0			
Block Erase(64K)	D8H	A23-A16	A15-A8	A7-A0			
Chip Erase	C7/60H						
Enable Reset	66H						
Reset	99H						
Set Burst with Wrap	77H	dummy[8]					

13

Figure 17. Difference between GD25LQ64C(Left) and GD25Q64C(Right)

The difference is the value of Write Status Register and the command format, so that the value which related with these registers need to modify. Open the FlashIMXRT1050_EVK_FlexSPI_Example with IAR. Find the LUT table and modify the value like following Figure:

```
/*Write Status Register sequence*/
lut_table[WR_STATUS_REG_LUT_INDEX+0] = FLEXSPI_LUT_INST(LUT_CODE_CMD_SDR,   LUT_PADS_ONE, ISSI_CMD_WRSR);
lut_table[WR_STATUS_REG_LUT_INDEX+1] = FLEXSPI_LUT_INST(LUT_CODE_WRITE_SDR, LUT_PADS_ONE,              2);
lut_table[WR_STATUS_REG_LUT_INDEX+2] = FLEXSPI_LUT_INST(LUT_CODE_STOP,      0,                     0);
lut_table[WR_STATUS_REG_LUT_INDEX+3] = FLEXSPI_LUT_INST(LUT_CODE_STOP,      0,                     0);
lut_table[WR_STATUS_REG_LUT_INDEX+4] = FLEXSPI_LUT_INST(LUT_CODE_STOP,      0,                     0);
lut_table[WR_STATUS_REG_LUT_INDEX+5] = FLEXSPI_LUT_INST(LUT_CODE_STOP,      0,                     0);
lut_table[WR_STATUS_REG_LUT_INDEX+6] = FLEXSPI_LUT_INST(LUT_CODE_STOP,      0,                     0);
lut_table[WR_STATUS_REG_LUT_INDEX+7] = FLEXSPI_LUT_INST(LUT_CODE_STOP,      0,                     0);
```

Figure 18. Modify the value form ISSI_CMD_WRSR (0x01H) to 0x31H

Then, write register format needs to be changed to 8-bit as following *Figure 19*.

```
265  if(0xC8 == a_mf_id)
266  {
267    a1_size = 1<<((id >> 16) & 0x00FF);
268    ReadStatusReg(0x00000000, &a1_StatusReg1);
269    ReadStatusReg2(0x00000000, &a1_StatusReg2);
270    a1_StatusReg = ((a1_StatusReg2 | (1<<1)));//a1_StatusReg = ((a1_StatusReg2 | (1<<1)) << 8) | (a1_StatusReg1 & 0xFC);
271    if(!(a1_StatusReg2 & (1<<1)))
272    {
273      WriteEnable(0x00000000);
274      WriteStatusReg8(0x00000000, a1_StatusReg);//WriteStatusReg16(0x00000000, a1_StatusReg);
275    }
276  }
```

Figure 19. Modify the write register format

Finally, build this project and copy the .out file as chapter 5.4.1.

6. Revision history

Table 4. Revision history

Revision number	Date	Substantive changes
0	05/2018	Initial release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN12183
Rev. 0
05/2018