Freescale Semiconductor, Inc.
Application Notes

Document Number: AN5244
Rev. 0, 01/2016

How to use QuadSPI on KL8x Series

1. Overview

Earlier, to expand the flash space with external flash device and get XiP functionality you had to use NOR flash with parallel interface, which has high cost and consumes more I/O interface. The serial NOR flash (say SPI NOR flash) just used to store data. The KL8x device now contains one Quad Serial Peripheral Interface (QSPI) module, which is a single dual-QuadSPI module with up to eight data lines for XiP functionality.

This document describes the steps to use QuadSPI module on KL8x series and use the QuadSPI NOR flash.

2. Quad Serial Peripheral Interface

The Quad Serial Peripheral Interface (QuadSPI) block acts as an interface to one single or two external serial flash devices, each with up to eight bi-directional data lines. It supports singles, dual, quad, or octal data lines in single (SDR) or double (DDR) data rate configurations. SDR mode supports up to 96 MHz and DDR mode supports up to 72 MHz.

It provides the two ways to access QuadSPI peripheral bus and AHB bus. The QuadSPI block also provides a flexible AHB buffer to accelerate data read. The basic QuadSPI block diagram is as below.

Contents

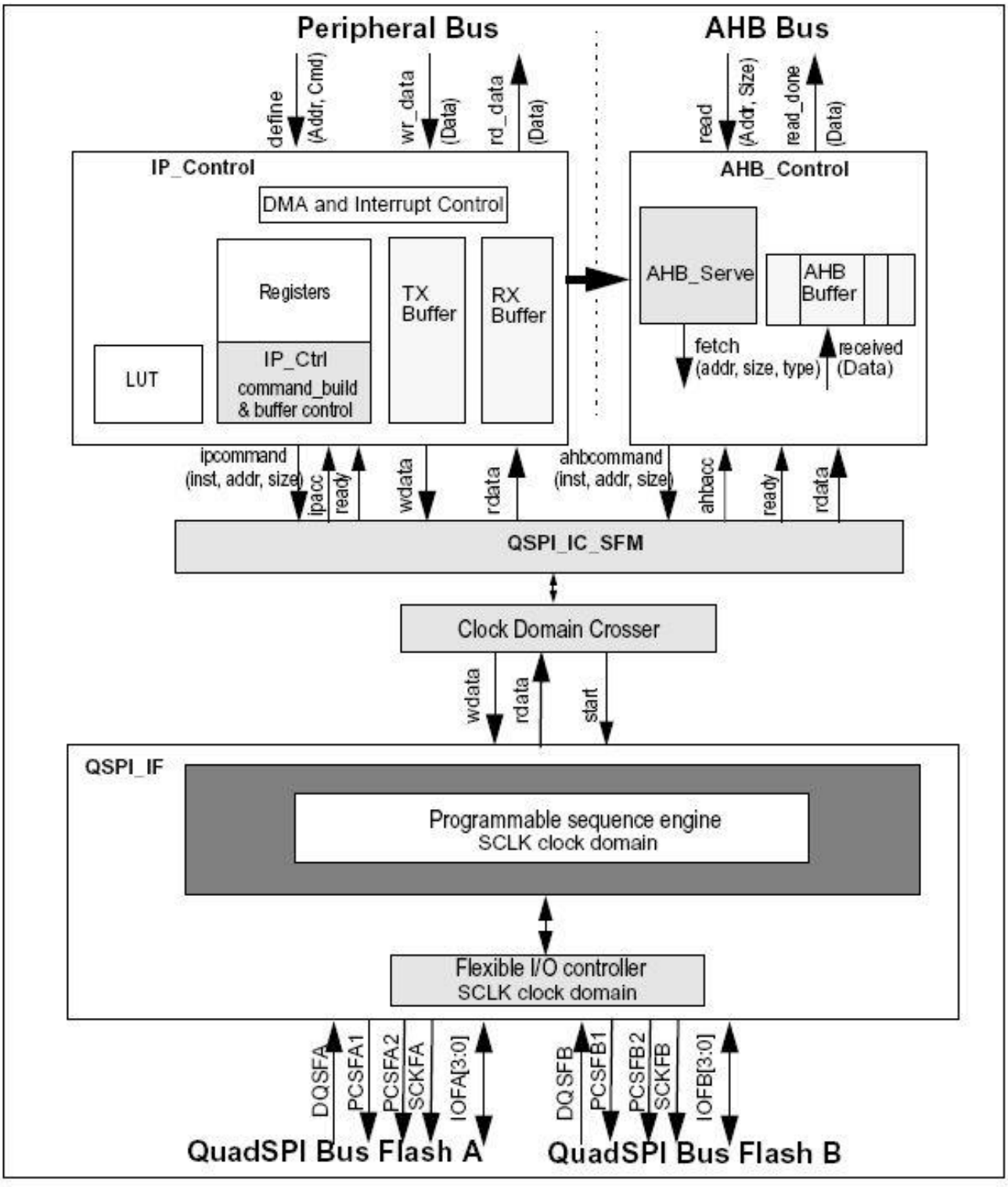


Figure 1. QuadSPI block diagram

3. Hardware design

The port (PORTE) is dedicated to the QuadSPI interface with fast pad, and it has a dedicated power pin (VDDIO_E) for PORTE.

NOTE

This design provides the option to power different voltage to PORTE and ensures that VDDIO_E is not less than VDD; otherwise, it may result in extra current leakage.

The hardware connection is as below:

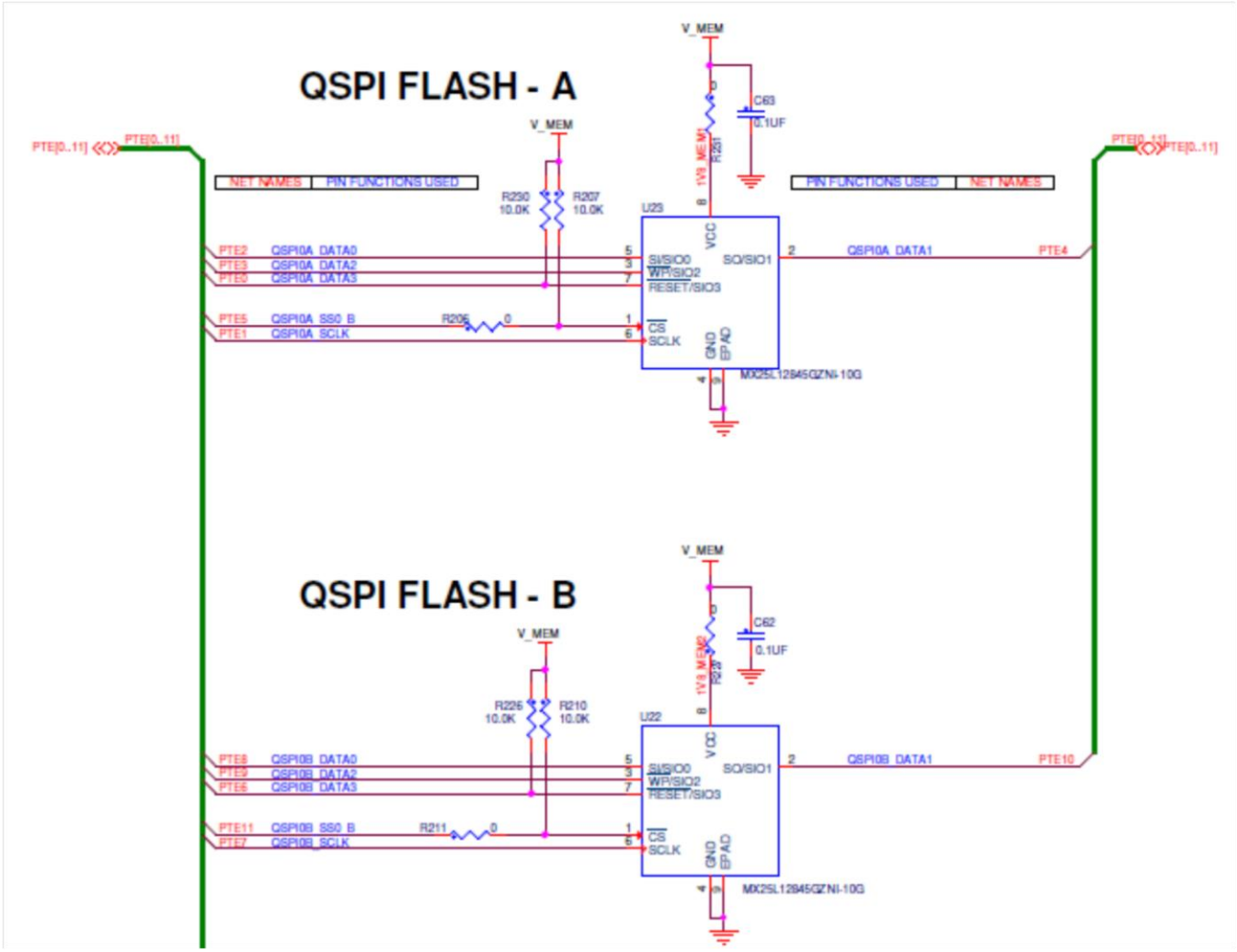


Figure 2. QSPI connection

In Figure 2 two QSPI devices are connected in parallel mode to yield eight data line performance.

To get good performance in the DDR mode, it need to keep the equal length of SCK and DATA0 – DATA3, and if external DQS signals is used, also require to keep DQS in equal length traces with SCK and DATA0 – DATA3.

This QuadSPI module also supports internally generated DQS signals. In this mode, the internal reference clock is fed as a strobe to QuadSPI for read data sampling. The data strobe must be generated in a way such that the data is correctly sampled by the QuadSPI module.

To use the DDR mode with internal DQS signals, you must ensure not to use DQS signal pin for other function, and add a matching capacitor on this pin. The matching capacitor should be close to the input capacitance of the QSPI device (Cin).

4. Software design

There are two ways to enable QuadSPI work. One way is to use ROM boot loader, configure FOPT, and enable boot from QSPI, and then send out the configuration information of external QSPI device by communication interface (UART, I2C, SPI, and USB etc.). For more information see "*Freescale Kinetis Bootloader K80 Tools*".

The other way is to initialize the QuadSPI module in the user code and implement the related function, such as QSPI flash erase and program.

4.1. QSPI initialization

The QuadSPI module provides flexible configuration options to meet different device (for example: QSPI NOR flash) connection. However, it needs to configure correctly and enabled to work well. Before initializing QuadSPI, some things need to be considered, such as, the clock source and the QuadSPI mode (parallel or single, DDR or SDR and so on).

4.1.1. Clock source options

You can set additional QuadSPI clock source options using QuadSPI_SOCCR[2:0].

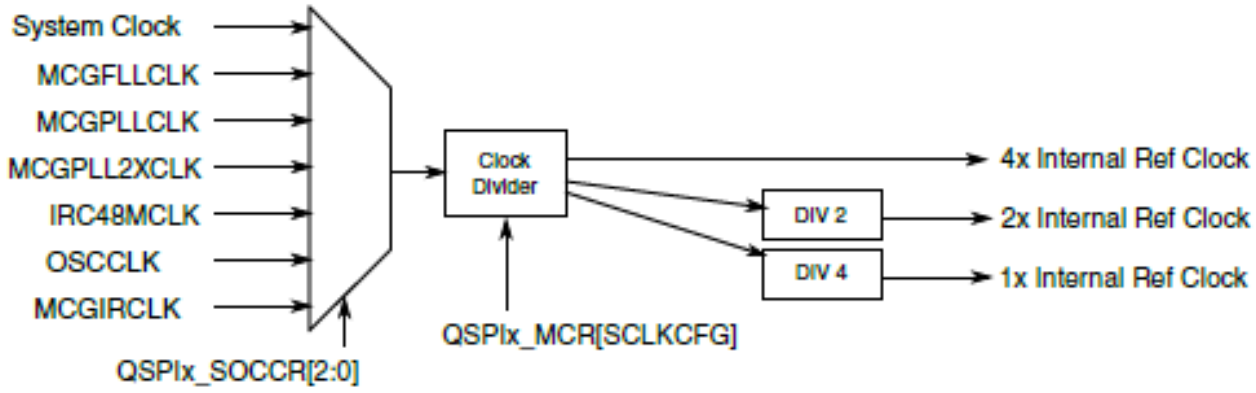


Figure 3. QSPI clock generation

The DDR mode of the QSPI requires a 4x, 2x, and 1x internal reference clock. In DDR mode, the clock divider output is used as the 4x internal reference clock. MCGPLL2XCLK is especially for the DDR mode to achieve the 4x speed (72Mhz in DDR mode).

The QSPI module also enables the clock divider to generate QSPI clock, which can be set by register QuadSPI_MCR[27:24].

The register SIM_CLKDIV1[OUTDIV5] is used to set the divider of clock for QSPI IP bus, which helps in accelerating the QSPI register operation.

NOTE

You need to ensure that (OUTDIV1 + 1) : (OUTDIV5 + 1) = 1 : 1 or 1 : 2 as other configuration may result in unexpected failure, which means to must have QSPI IP bus clock equal to system clock or ½ system clock.

4.1.2. Serial flash address assignment

Based on different flash devices, some flash have two dies, or connect to 4-data line flash in parallel. The serial flash address assignment may be modified by writing into Serial Flash A1 Top Address (QuadSPI_SFA1AD) and Serial Flash A2 Top Address (QuadSPI_SFA2AD) for device A and into Serial Flash B1Top Address (QuadSPI_SFB1AD) and Serial Flash B2Top Address (QuadSPI_SFB2AD) for device B.

Figure 4 shows how different access modes are related to the address specified for the next SFM Command.

NOTE

This address assignment is valid for both IP and AHB commands.

Parameter	Function	Access Mode
QSPI_AMBA_BASE ((31:10) - 22 bits)	QuadSPI AHB base address	
TOP_ADDR_MEMA1(TPADA1)	Top address for the external flash A1 (first device of the dual die flash A, or the first of the two independent flashes sharing the IOFA)	Any access to the address space between TOP_ADDR_MEMA1 and QSPI_AMBA_BASE will be routed to **Serial Flash A1**
TOP_ADDR_MEMA2(TPADA2)	Top address for the external flash A2 (second device of the dual die flash A, or the second of the two independent flashes sharing the IOFA).	Any access to the address space between TOP_ADDR_MEMA2 and TOP_ADDR_MEMA1 will be routed to **Serial Flash A2**
TOP_ADDR_MEMB1(TPADB1)	Top address for the external flash B1 (first device of the dual die flash B, or the first of the two independent flashes sharing the IOFB)	Any access to the address space between TOP_ADDR_MEMB1 and TOP_ADDR_MEMA2 will be routed to **Serial Flash B1**
TOP_ADDR_MEMB2(TPADB2)	Top address for the external flash B2 (second device of the dual die flash B or the second of the two independent flashes sharing the IOFB)	Any access to the address space between TOP_ADDR_MEMB2 and TOP_ADDR_MEMB1 will be routed to **Serial Flash A2**

Figure 4. Serial flash address assignment

Figure 5 show how to set and configure serial flash address. It connects two 128-Mbit serial flashes with parallel mode. The QSPI_AMBA_BASE address is 0x68000000

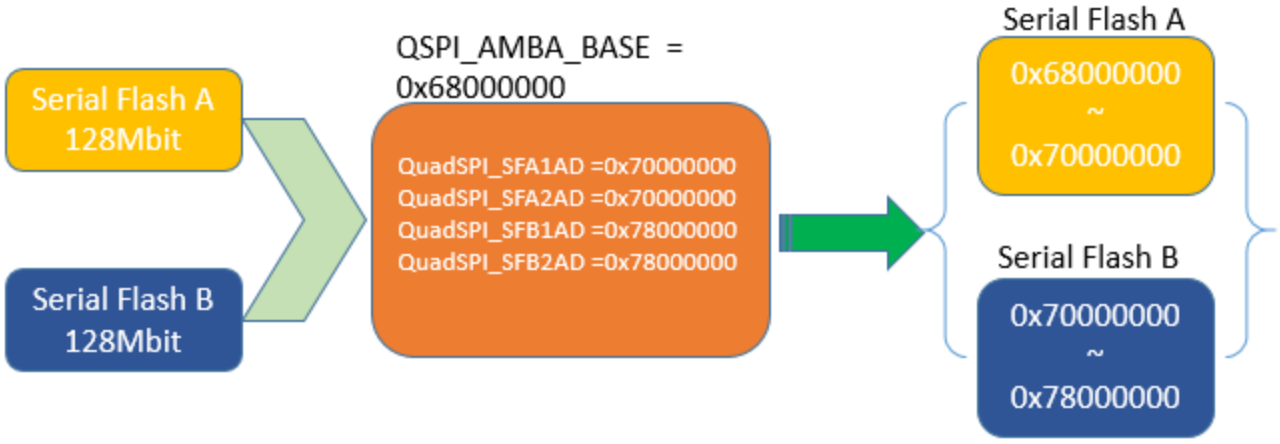


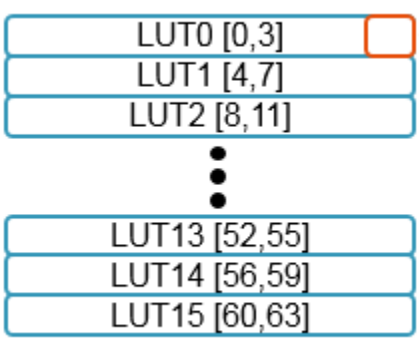
Figure 5. Example of address assignment

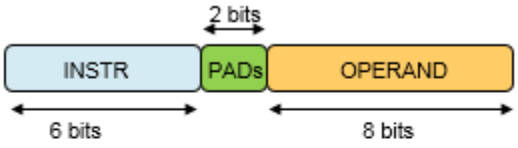
4.1.3. Look-up table configuration

This device consists of a total of 64 LUT register, and these 64 registers are divided into groups of four registers that make a valid sequence. Therefore, QSPI_LUT[0], QSPI_LUT[4], QSPI_LUT[8] till QSPI_LUT[60] are the starting registers of a valid sequence. Each of these sets of four registers can have a maximum of eight instructions, which helps accelerate to run the command saved in look-up table.

Some of the features of the look-up table are:

- Each instruction-operand unit is 16-bit wide.
- Depending on the complexity of the QSPI transaction, a sequence may consist of a single instruction-operand set or several of them.
- Writing to the IPCR[SEQID] triggers the execution of the specified sequence.
- Reading from the AHB memory mapped QSPI area triggers the execution of the sequence specified on the BFGENCR[SEQID] field.





Instruction	Pad	Operand	Comment
CMD	0x0	0x03	Read Data byte command on one pad
ADDR	0x0	0x18	24 Addr bits to be sent on one pad
READ	0x0	0x04	Read 32Bytes
JMP_ON_CS	0x0	0x00	Jump to instruction 0 (CMD)

Figure 6. Look-up table

You must pre-populate LUT. Note that LUT0 and LUT1 have the default value 0x08180403h and 0x24001C08h respectively, so that it can support common flash read command.

Below is an example of LUT to access QSPI flash device. Here, Macronix (MU25L128) is taken as an example.

Table 1. Accessing QSPI flash drive

Instruction	PAD	OPERAND	COMMENT
CMD(6d'1)	2	0xEB	4xIO Read Command
ADDR(6d'2)	2	0x18	24 Bit address to be send on 4 pads
DUMMY(6d'3)	0	0x06	6 dummy cycles
READ(6d'7)	2	0x08	Read 8 bytes
STOP(0)	0	0	Stop execution; deassert CS

You can write LUT0 and LUT1 as below:

LUT0 = 0x0A1806EB;

LUT1 = 0x1E080C06;

After reset, the LUT may be reprogrammed according to the device connected on board. In order to protect its contents during a code run, the LUT may be locked, and a write to the LUT will not be successful until it has been unlocked again. The key for locking or unlocking the LUT is 0x5AF05AF0.

The process for locking and un-locking the LUT is as follows:

- Locking the LUT
  1. Write the key (0x5AF05AF0) in to the LUT Key Register (QuadSPI_LUTKEY).
  2. Write 0b01 to the LUT Lock Configuration Register (QuadSPI_LCKCR). Note that the IPS transaction should immediately follow the above IPS transaction (no other IPS transaction can be issued in between). A successful write into this register locks the LUT.
- Unlocking the LUT
  1. Write the key (0x5AF05AF0) into the LUT Key Register (QuadSPI_LUTKEY).
  2. Write 0b10 to the LUT Lock Configuration Register (QuadSPI_LCKCR). Note that the IPS transaction should immediately follow the above IPS transaction (no other IPS transaction can be issued in between). A successful write into this register unlocks the LUT.

And it available to implement the instruction by writing QuadSPI_IPCR register and SEQID field define index of LUT, and 4 LUT register is one group, so index of LUT0 is 0, and index of LUT4 is 1, the rest can be done in this way.

4.1.4. Take initialization with SDK

KL8x serial provide QuadSPI HAL and driver API functions base on SDK (version 1.3 or later), and it provide a configurable struct variable for QuadSPI initialization.

Add the following structure:

```
/*! @brief External flash configuration items*/
typedef struct QspiFlashConfig
{
    uint32_t flashA1Size; /*!< Flash A1 size */
    uint32_t flashA2Size; /*!< Flash A2 size */
    uint32_t flashB1Size; /*!< Flash B1 size */
    uint32_t flashB2Size; /*!< Flash B2 size */
    uint32_t lookuptable[FSL_FEATURE_QSPI_LUT_DEPTH]; /*!< Flash command in LUT */
    uint32_t dataHoldTime; /*!< Data line hold time. */
    uint32_t CSHoldTime; /*!< CS line hold time */
    uint32_t CSSetupTime; /*!< CS line setup time*/
    uint32_t cloumnspace; /*!< Column space size */
    uint32_t dataLearnValue;/*!< Data Learn value if enable data learn */
    qspi_endianness_t endian; /*!< Flash data endianess. */
    bool parallelmode; /*!< If enable parallel mode. */
    bool wordaddress; /*!< If enable word address.*/
    bool DQSEnable; /*!< If enable DQS mode. */
    qspi_dqs_config_t dqs_config; /*!< DQS configuration; If not supported, set to NULL */
    bool DDREnable; /*!< If enable DDR mode. */
    bool octalmode; /*!< If enable octal mode. */
} qspi_flash_config_t;
```

Add the following example code.

```
qspi_flash_config_t single_config =
{
    .parallelmode  = 1,
    .DDREnable = 0,
    .dataHoldTime = 0,
    .flashA1Size = FLASH_SIZE, /* 16MB */
    .flashB1Size = FLASH_SIZE, /* 16MB */
    .lookuptable =
     {
         // Seq0 : Read  , default value of LUT0 and LUT1
       [0] = 0x08180403,
       [1] = 0x24001C08,
        // Seq1: Write Enable
        [4] = 0x406,
        // Seq2: Erase All
        [8] = 0x460,
        // Seq3: Read Status
        [12] = 0x1c010405,
        // Seq4: Page Program
        [16] = 0x08180402,
        [17] = 0x2004,
        // Seq5: Write Register
        [20] = 0x20040401, // 2 byte write
         // Seq7: Erase Sector
        [28] = 0x08180420,
        // Seq8: Dummy
        [32] = 0x4FF,
        // Seq9: Dual read
        [36] = 0x091804BB,  // dual read, 24bit address
        [37] = 0x1D8011A5,  // mode bits and read 128 bytes
        [38] = 0x2401,      // jump to address instruction
    },
     .endian = kQspi64LittleEndian,
};
```

To complete the QuadSPI initialization using SDK, add the following snippet code.

```
  QSPI_DRV_Init(0, &state);      // Initialize QSPI internal state and open clock for QSPI.
```

```
  qspi_config_t config;
  config.AHBbufferSize[3] = FLASH_PAGE_SIZE;
  QSPI_DRV_GetDefaultQspiConfig(&config);   // Get QSPI default settings
  QSPI_DRV_ConfigQspi(0, &config);          // configure the qspi
  QSPI_DRV_ConfigFlash(0, &single_config);  //According to serial flash feature to configure flash settings
```

After complete QSPI initialization, it is available to leverage LUT to execute the according flash operation, such as program, erase and others.

Add the example code:

```
/* Erase sector */
void erase_sector(uint32_t addr)
{
    //Clear tx buffer
    while(QSPI_DRV_GetQspiStatus(0, kQspiBusy));
    QSPI_DRV_ClearFifo(0, kQspiTxFifo);
    //Set the erase start address
    QSPI_DRV_SetIPCommandAddr(0,addr);
    //Enable flash write
    cmd_write_enable();
    QSPI_DRV_ExecuteFlashCommand(0,28);
}
```

4.2. Operation with parallel mode

QuadSPI can access two flashes in parallel. This can improve the throughput performance by two times. But note that only read operations and x1 mode write are allowed in parallel mode.

It is available to enable AHB flexible-buffers work in parallel mode by set QSPI_BFGENCR[PAR_EN] bit to '1' , and enable IP command work in parallel mode via set QSPI_IPCR[PAR_EN] bit to ‘1’, reads from any even address provides bits [7:4] of both serial flash devices and reads from any odd address provides bits [3:0] of both flash devices.

Parallel Flash Mode is valid for commands related to data read and data write in single io mode from the serial flash, that means it can write data with parallel mode, but limit to 1 pad mode, and read data is fully support with 1,2,4 pad operation.

In parallel mode, when it send out command to flash device, it will send out command on both QSPI A and B simultaneously, but if read status also by in parallel mode, it will get the wrong result because it is combination of bus data of QSPI A and QSPI B, hence it is limit to use parallel mode for command read, and it need to temporarily change it to single mode and read status register individually, below is a example for read status register of Macronix (MU25L128).

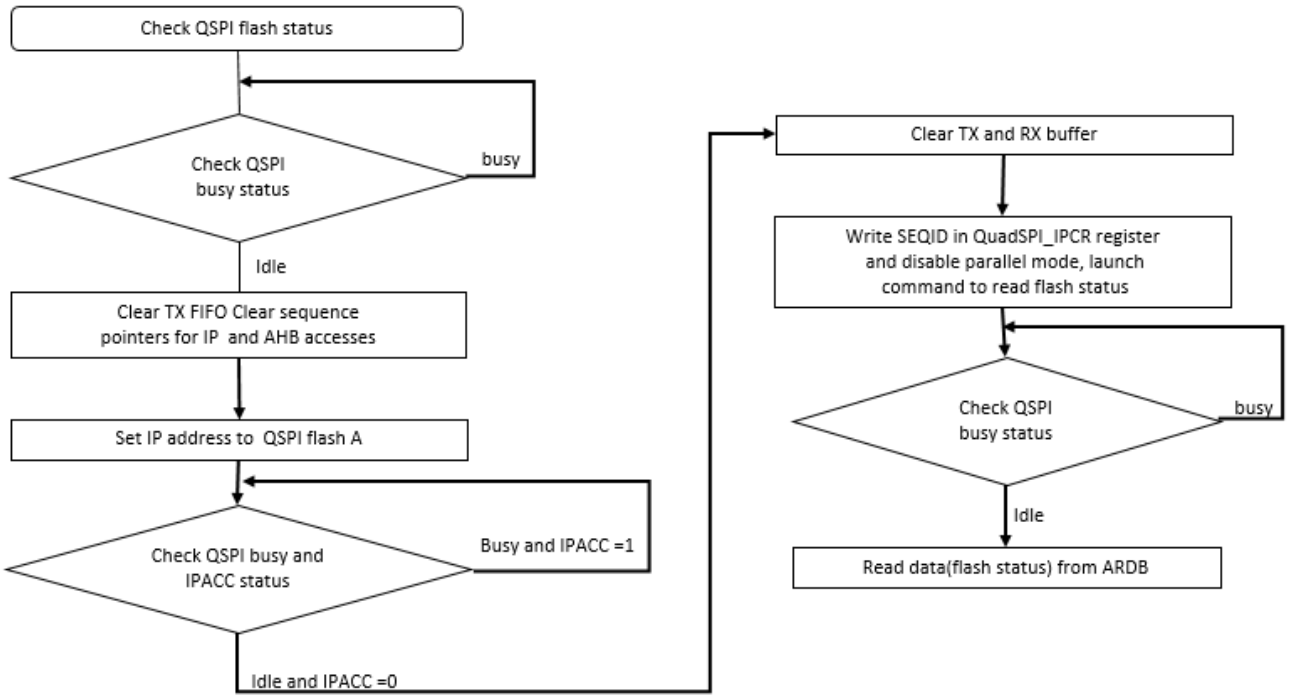


Figure 7. Workflow of reading the QSPI Flash A status

Add the following to read QSPI Flash B status register. You just need to change IP address to QSPI flash B, for example:

```
QSPI_DRV_SetIPCommandAddr(0,FLASH_B1_BASEADDRESS); //write QSPI flash B address to QuadSPI_SFAR
```

To read the QSPI status, add the following snippet code.

```
while(QSPI_DRV_GetQspiStatus(0, kQspiBusy));
QSPI_DRV_ClearFifo(0, kQspiTxFifo);
QSPI_HAL_ClearSeqId(QuadSPI0, kQspiBufferSeq);
QSPI_HAL_ClearSeqId(QuadSPI0, kQspiIPSeq);
//Set the address
QSPI_DRV_SetIPCommandAddr(0,FLASH_A1_BASEADDRESS);
while(QSPI_DRV_GetQspiStatus(0, kQspiBusy));
while(QSPI_DRV_GetQspiStatus(0, kQspiIPAccess));
QSPI_DRV_ClearFifo(0, kQspiTxFifo);
       QSPI_DRV_ClearFifo(0, kQspiRxFifo);
       QuadSPI0_IPCR = QuadSPI_IPCR_SEQID(3)|0x02;
       while(QSPI_DRV_GetQspiStatus(0, kQspiBusy));
       val = *(volatile uint32_t *)(FSL_FEATURE_QSPI_ARDB_ADDRESS);
```

Do not forgot to restore the QSPI mode to parallel mode by write QuadSPI_IPCR or QuadSPI_BFGENCR.

```
QuadSPI0_IPCR = IpcrValue;
while(QSPI_DRV_GetQspiStatus(0, kQspiBusy));
while(QSPI_DRV_GetQspiStatus(0, kQspiIPAccess));
```

4.3. Double Data Rate mode operation

The increasing requirement of improved throughput has introduced the double data rate (DDR) mode. In DDR mode, the data is transferred on both the rising and falling edges of the serial flash clock. The DDR serial flashes sample as well as drive the data on both rising and falling edges of serial flash clock.

It is recommended to set QuadSPI_FLSHCR_TDH to non-zero in DDR mode, and also after complete QSPI initialization, it still available to change QSPI to DDR mode, a basic flow as below:

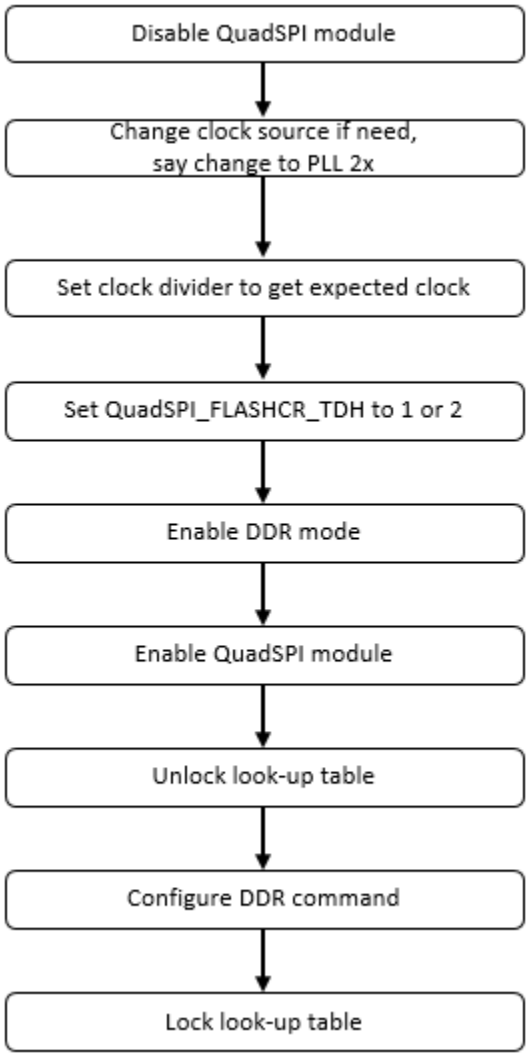


Figure 8. Flow of switching to DDR mode

To switch to the DDR mode, add the following snippet code.

```
QSPI_HAL_DisableModule(QuadSPI0);
QSPI_HAL_SetClockSrc(QuadSPI0,3);
QSPI_HAL_SetSCLK(QuadSPI0,288000000,288000000);
QSPI_HAL_SetDDRModeCmd(QuadSPI0,1);
qspi_flash_timing_t  qspi_flash_timming;
qspi_flash_timming.CSHoldTime = 3;
```

```
qspi_flash_timming.CSSetupTime = 3;
qspi_flash_timming.dataHoldTime = 1;
QSPI_HAL_SetFlashTiming(QuadSPI0,&qspi_flash_timming);
QSPI_HAL_EnableModule(QuadSPI0);
uiQSPI_ReadCmd[0] = 0x2a1806ED;            // configure read QSPI flash with DDR mode,
uiQSPI_ReadCmd[1] = 0x3a800E06;           // configure dummy is 6
uiQSPI_ReadCmd[2] = 0x2600;
QSPI_DRV_UpdateLUT(0,0,uiQSPI_ReadCmd);
```

5. Performance comparison

As the QuadSPI module supports XiP functionality, it is available to save the code to the external serial flash and then run it directly. The steps are as follows:

- Build code to the address range of external serial flash (see the memory map and assigned flash address).
- Program firmware by ROM boot loader or program it by apps code.
- After complete code programmed, configure LUT with expected read mode if need, and also can use the default LUT setting (LUT0, LUT1) to execute code saved in QSPI flash.
- Jump to address located in external serial flash.

Due to shortage of non-local cache supported, the performance of running code in the external serial flash on KL8x series is impacted. It can reach to half of performance of internal flash with cache enable base on the evaluation of running core mark test. If you remove the impact of cache and disable the flash cache, the performance is similar.

6. Conclusion

This document introduces the basic flow to use QuadSPI of KL8x series to access flash device. It also introduces the consideration of hardware and software design, which help user to easy to use QuadSPI module.

7. Reference

MKL82P121M72SF0RMRM *Reference Manual*

8. Glossary

QuadSPI/QSPI	*Quad Serial Peripheral interface*
SDR	*Single data rate*
DDR	*Double data rate*
LUT	*Look-up table*
MCG	*Multipurpose Clock Generator*

9. Revision history

Table 2 is an example of a revision history table.

Table 2. Sample revision history

Revision number	Date	Substantive changes
0	06/2015	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN5244
Rev. 0
01/2016