NXP Semiconductors
Application Notes

Document Number: AN12193
Rev. 0, 05/2018

Using S32K148 QuadSPI Module

by: NXP Semiconductors

1. Introduction

This application note describes the QuadSPI module on the S32K148 devices. It provides a description of how the module is implemented on these devices, specifically focusing on setting up LUT sequences, using commands to interface with an external memory and using the AHB interface. More details about the QuadSPI module can be found in the devices respective reference manual.

The application note is supported by two software examples, a bare metal example code and an SDK example. The bare metal example can be found in the attached zip file, while the SDK example is part of the SDK release.

Contents

2. QuadSPI protocol

Quad Serial Peripheral Interface (QuadSPI) is a communications protocol used for communications between a microcontroller and external flash memory. It is based on the popular Serial Peripheral Interface (SPI). Whereas an SPI makes use of up to four connections – Data In, Data Out, Clock, and Chip Select (used to signify that a transmit or receive is active) – QuadSPI uses Clock, up to six Chip Select channels, and up to four bi-directional data channels. This extra connectivity allows for data to be read from the flash in a prompt manner, making QuadSPI an excellent choice for using additional off-chip memory

Due to the smaller number of pins, requests for reads/writes/erases are carried out by sending commands across the bus. For example, to read data from flash memory the “Read Data (0xEB)” command is sent, followed by the 24-bit address to be read. The data is then sent to the microcontroller. The figure below shows a typical read instruction using four data lines.

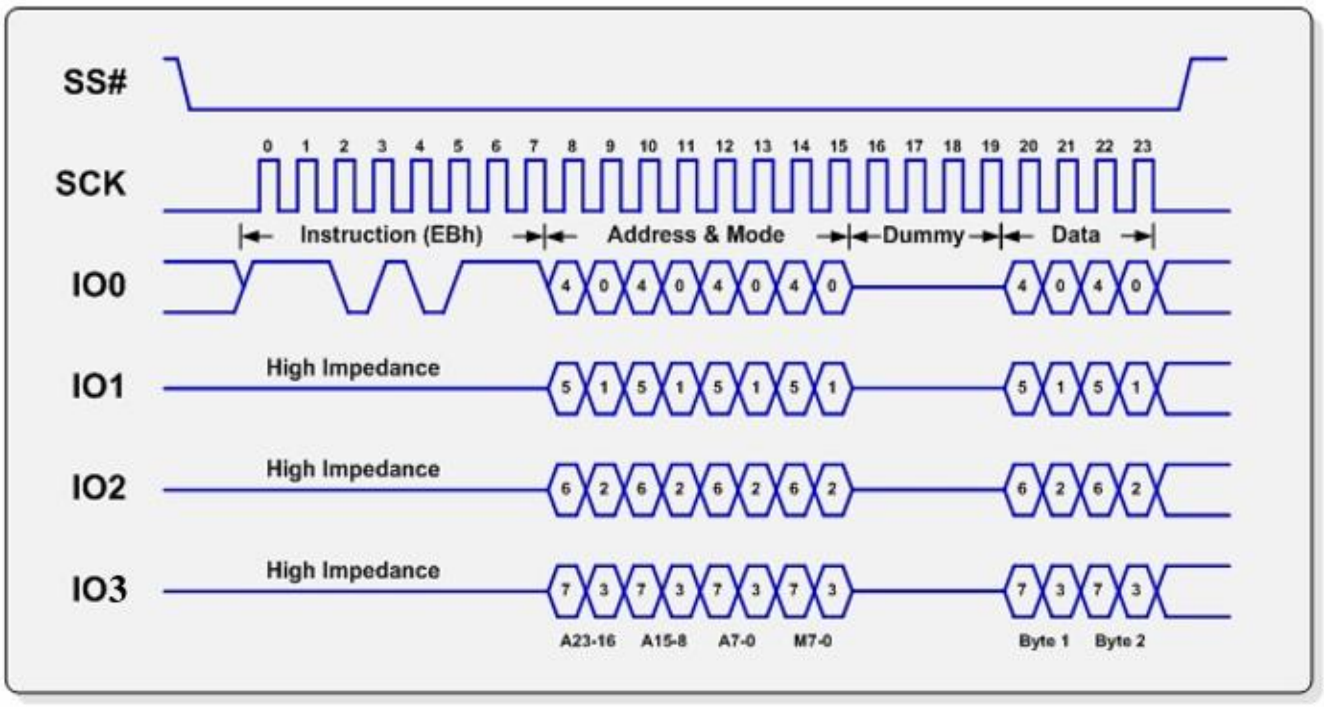


Figure 1. **Read command (QuadSPI frame)**

There are several suppliers of QuadSPI-compatible memory, such as Winbond, Spansion, Macronix, and Numonyx. The examples provided in this application note will focus on the Macronix devices as the external memory populated on the S32K148 EVB is a Macronix chip. Like SPI before it, QuadSPI does not adhere to a set standard, but as a rule different manufacturers’ devices interface via a similar command set.

3. S32K148 QuadSPI implementation

The following section will describe a couple of features of the QuadSPI module that only applies on the S32K148 due to the way It was implemented on these

3.1. Side A and side B

The QuadSPI module is divided into two “sides.” A and B, mainly due to the limited amount of high speed pads available in the device, each side has its advantages and disadvantages so it is up to the application to select between them.

The main advantage of side A is speed, it supports up to 80 MHz. However, it does not support DDR, neither Hyperbus functionality.

On the other hand, Side B operates slower, up to 20 MHz, but it does support DDR and Hyperbus protocol for HyperRAM devices.

It is important to clarify that even though there are two “sides” of the QuadSPI module, It does not mean that it can be implemented as if there were two separate instances of the QuadSPI module. One side can only be used at the same time.

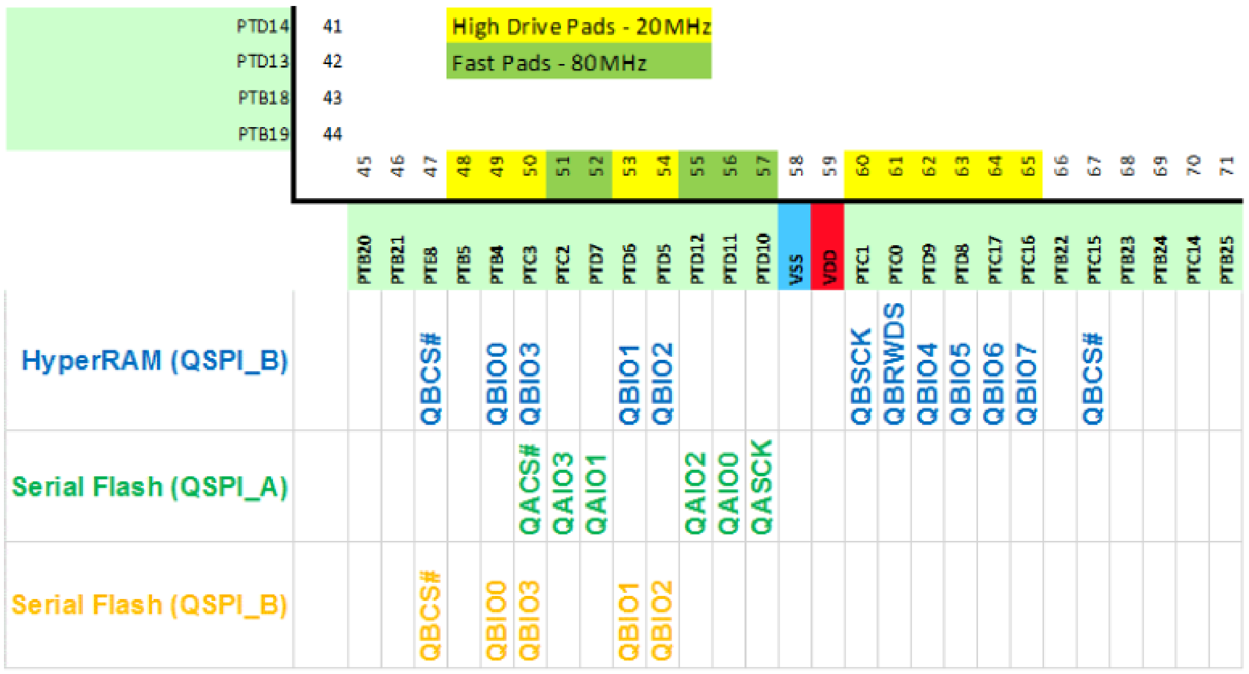

PTD14 41
PTD13 42
PTB18 43
PTB19 44
High Drive Pads - 20MHz
Fast Pads - 80MHz
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
PTB20 PTB21 PTE8 PTB5 PTB4 PTC3 PTC2 PTD7 PTD6 PTD5 PTD12 PTD11 PTD10 VSS VDD PTC1 PTC0 PTD9 PTD8 PTC17 PTC16 PTB22 PTC15 PTB23 PTB24 PTC14 PTB25
HyperRAM (QSPI_B)
QBCS# QBIO0 QBIO3 QBIO1 QBIO2 QBSCK QBRWDS QBIO4 QBIO5 QBIO6 QBIO7 QBCS#
Serial Flash (QSPI_A)
QACS# QAIO3 QAIO1 QAIO2 QAIO0 QASCK
Serial Flash (QSPI_B)
QBCS# QBIO0 QBIO3 QBIO1 QBIO2

Figure 2. **S32K148 Pinout: QuadSPI module side A and side B**

4. Look-up Table (LUT) functionality

The Look-up table also known as LUT is the mechanism used by the QuadSPI module to communicate with the external memory. It is used for either sending commands, reading, writing or waiting. 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.

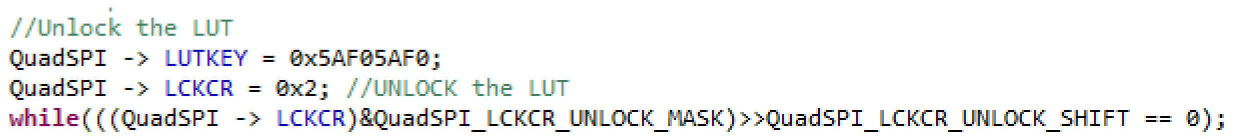
The following table lists some of the most common commands for LUT operations. For the complete list go to **Table 34-14 Instruction set** of reference manual:

Table 1. **Common LUT commands**

Command	Coding (6 bits)
CMD	0x01
ADDR	0x02
DUMMY	0x03
MODE	0x04

Command	Coding (6 bits)
READ	0x07
WRITE	0x08
STOP	0x00

As a safety mechanism, the LUT table is locked by default. Therefore, the first step to start using the LUT is to unlock it. To unlock it, the key must be written into the LUTKEY register. The key value is 0x5AF05AF0, then a value of 0x02 must be written into the Lock configuration register. The LUT must be unlocked at this point. The following code snip shows how this looks on the S32K148 device.



```
//Unlock the LUT
QuadSPI -> LUTKEY = 0x5AF05AF0;
QuadSPI -> LCKCR = 0x2; //UNLOCK the LUT
while(((QuadSPI -> LCKCR)&QuadSPI_LCKCR_UNLOCK_MASK)>>QuadSPI_LCKCR_UNLOCK_SHIFT == 0);
```

Figure 3. **Unlock LUT code**

Once the LUT has been unlocked the user can modify the LUT sequences having in consideration that QSPI_LUT[0], QSPI_LUT[4], QSPI_LUT[8] till QSPI_LUT[60] are the starting registers of a valid sequence. Some of the features of the look-up table are:

- Each instruction-operand unit is 16-bit wide. However, LUT registers are 32-bit wide, so two instructions can be placed within each LUT register.
- Depending on the complexity of the QSPI transaction, a sequence may consist of a single instruction-operand set or several of them.

Each LUT instruction-operand has the following structure:

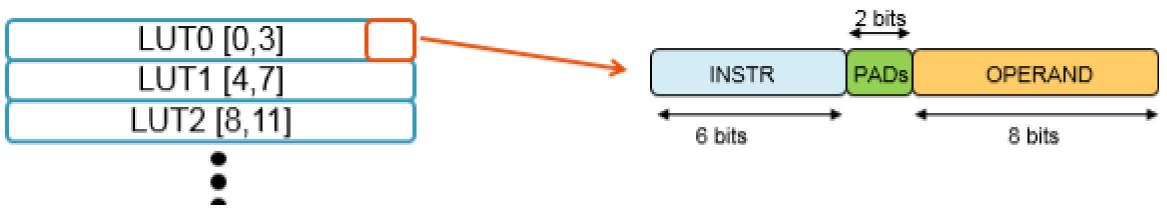


Figure 4. **LUT operand structure**

Where the INSTR field represents the LUT commands presented previously in Table 1, the PADs field represents the amount of data lines used by the command and the operand field varies depending on the INSTR used, this can be found in **Table 34-14 Instruction set** of reference manual.

For example, having the value 0x1C08. If viewed as binary the value is 0b0001110000001000. If we divide it into the different fields:

INSTR						PADs		OPERAND							
0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0

INSTR = 0x07 (Read)

PADs = 0x00 (1 PAD)

OPERAND = 0x08 (8 bytes)

When this sequence is launch the module will read 8 bytes of data through 1 data line.

Once all the LUT sequences had been filled the LUT table must be locked again. The sequence to locking down the LUT is very similar to unlocking it. The key must be written into the LUTKEY register. The key value is 0x5AF05AF0, then a value of 0x01 must be written into the Lock configuration register. The following code snip shows how this looks on the S32K148 device.

```
//Lock the LUT
QuadSPI -> LUTKEY = 0x5AF05AF0;
QuadSPI -> LCKCR = 0x1; //LOCK the LUT
while(((QuadSPI -> LCKCR)&QuadSPI_LCKCR_LOCK_MASK)>>QuadSPI_LCKCR_LOCK_SHIFT == 0);
```

Figure 5. **Lock LUT code**

5. Peripheral bus (commands) interface

The QSPI module offer two different paths to communicate with an external memory Peripheral Bus (left side of the figure below) or AHB bus (left side of the figure below). In this section the Pheripheral Bus interface will be explained in more detail.

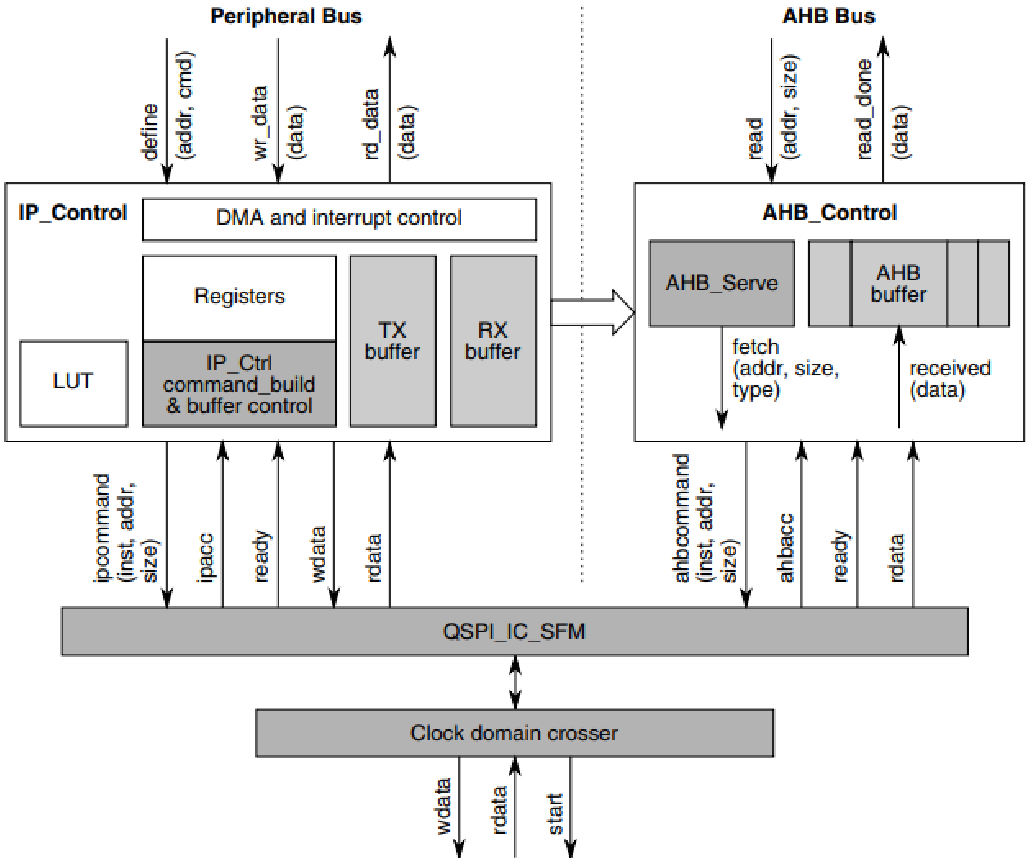


Figure 6. **QSPI block diagram**

If they user wants to write, erase or change the configuration of the external memory the only option is the Peripheral bus interface. It uses the LUT table sequences to communicate with the external memory. Once the LUT table has been filled out with the required LUT sequences the user can simple launch the desired sequence. For example, suppose that the LUT is filled out as the following figure:

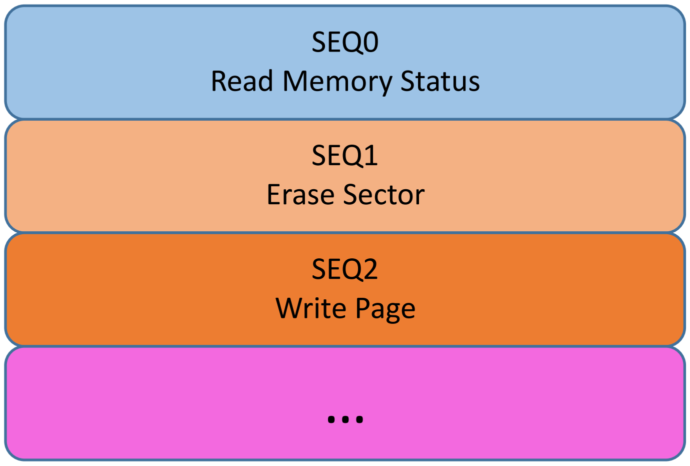
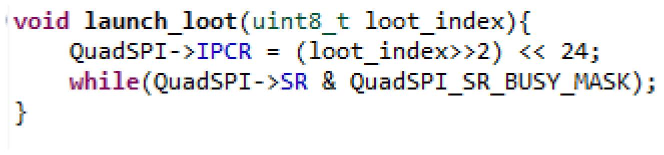


Figure 7. **Example LUT**

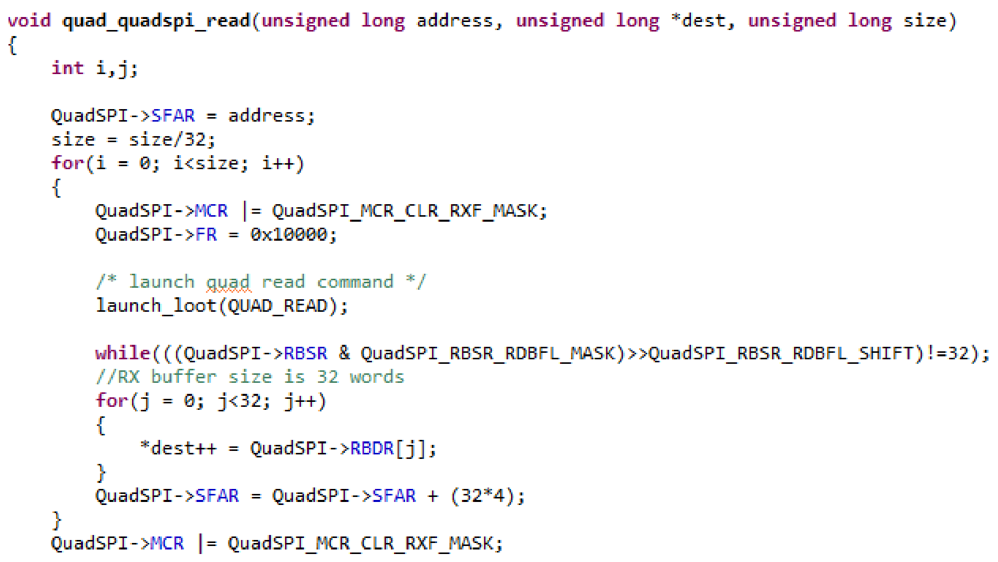
Sequence 0 contains the necessary commands to read the status of the memory, sequence 1 the necessary commands to erase a selected sector, and the sequence 2 the necessary commands to write the whole page of the memory. If for example, the user would like to erase one sector of the memory, it would be as simple of calling out the sequence 1 of the LUT as many times as the application needs. The following code snip exemplifies how simple is to call out a sequence of the LUT table.



```
void launch_loot(uint8_t loot_index){
    QuadSPI->IPCR = (loot_index>>2) << 24;
    while(QuadSPI->SR & QuadSPI_SR_BUSY_MASK);
}
```

Figure 8. **Launch LUT sequence code**

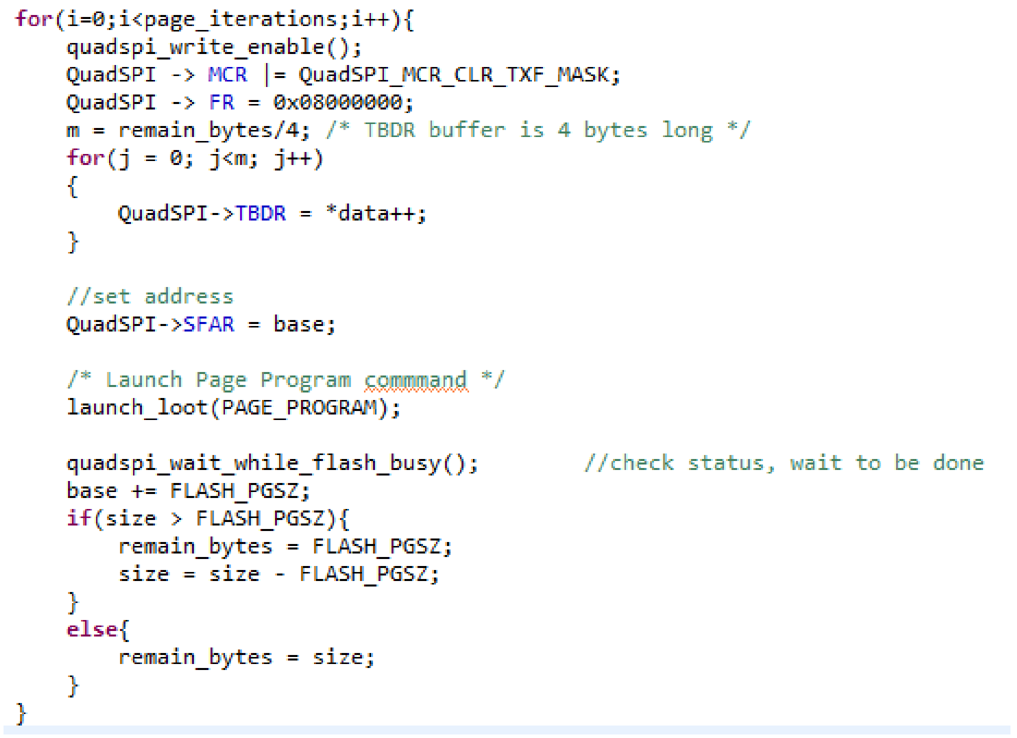
Even tough launching a LUT sequence will execute the necessary commands to communicate with the memory, some actions such as reading or writing require the use of other registers to correctly receive or send the data. For example, when reading the MCU stores the data in an internal data buffer accessed through the RBDR[0-31] registers, it also requires the user specify the address to be read in the SFAR register and clearing up the CLR_RXF flag. The following code snip shows a function that is able to read a configurable amount of bytes from an specified address.



```c
void quad_quadspi_read(unsigned long address, unsigned long *dest, unsigned long size)
{
    int i,j;

    QuadSPI->SFAR = address;
    size = size/32;
    for(i = 0; i<size; i++)
    {
        QuadSPI->MCR |= QuadSPI_MCR_CLR_RXF_MASK;
        QuadSPI->FR = 0x10000;

        /* launch quad read command */
        launch_loot(QUAD_READ);

        while(((QuadSPI->RBSR & QuadSPI_RBSR_RDBFL_MASK)>>QuadSPI_RBSR_RDBFL_SHIFT)!=32);
        //RX buffer size is 32 words
        for(j = 0; j<32; j++)
        {
            *dest++ = QuadSPI->RBDR[j];
        }
        QuadSPI->SFAR = QuadSPI->SFAR + (32*4);
    }
    QuadSPI->MCR |= QuadSPI_MCR_CLR_RXF_MASK;
```

Figure 9. **QuadSPI read code**

Similar considerations apply for writing data. User must first full a buffer of data, which is then send to the external memory depending on the amount of data specified in the command. The buffer is filled out through the TBDR[0-31] registers. Just as the read sequence, user must specify the address to be written to in the SFAR register. The following snip of code shows a write routine.

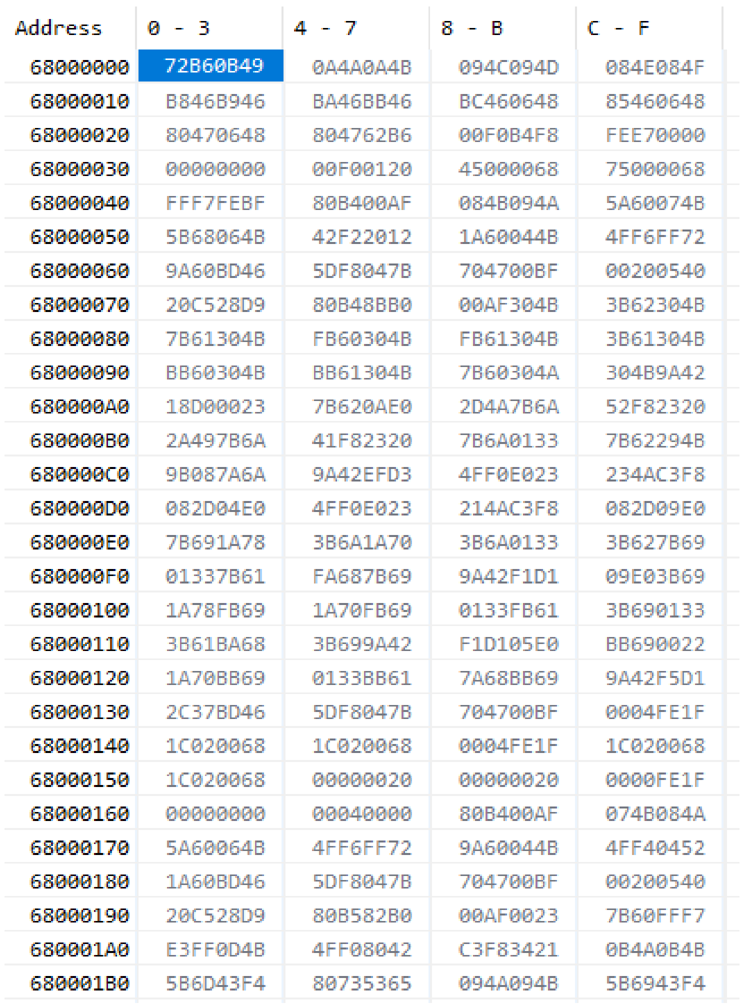


```c
for(i=0;i<page_iterations;i++){
    quadspi_write_enable();
    QuadSPI -> MCR |= QuadSPI_MCR_CLR_TXF_MASK;
    QuadSPI -> FR = 0x08000000;
    m = remain_bytes/4; /* TBDR buffer is 4 bytes long */
    for(j = 0; j<m; j++)
    {
        QuadSPI->TBDR = *data++;
    }

    //set address
    QuadSPI->SFAR = base;

    /* Launch Page Program commmand */
    launch_loot(PAGE_PROGRAM);

    quadspi_wait_while_flash_busy();        //check status, wait to be done
    base += FLASH_PGSZ;
    if(size > FLASH_PGSZ){
        remain_bytes = FLASH_PGSZ;
        size = size - FLASH_PGSZ;
    }
    else{
        remain_bytes = size;
    }
}
```

Figure 10. **QuadSPI page program code**

6. AHB interface

The AHB block diagram is shown in the right side of Figure 6. Differently from the Peripheral Bus Access, the AHB interface only allows read operations. However, the main advantage of the AHB access is that it allows to see the external memory as if it was mapped to an internal memory address of the device, meaning that the user does not need to perform any LUT sequence launch. In the S32K148, the QuadSPI AHB region is 128 MB long, it is mapped to starting address 0x68000000. For example, if the user tries to access address 0x000000 of the memory using peripheral bus access, then a certain LUT sequence would need to be launched and the user would need to read the Rx buffer of QuadSPI to get the data. On the other hand, when using AHB the user could simply access memory address 0x68000000 (start address for QuadSPI) and get the data. The following figure shows the memory accessed through the debugger after the QuadSPI was programmed with some data, as it can be noticed, the data is read as internal memory.



```
Address    0 - 3      4 - 7      8 - B      C - F
68000000   72B60B49   0A4A0A4B   094C094D   084E084F
68000010   B846B946   BA46BB46   BC460648   85460648
68000020   80470648   804762B6   00F0B4F8   FEE70000
68000030   00000000   00F00120   45000068   75000068
68000040   FFF7FEBF   80B400AF   084B094A   5A60074B
68000050   5B68064B   42F22012   1A60044B   4FF6FF72
68000060   9A60BD46   5DF8047B   704700BF   00200540
68000070   20C528D9   80B48BB0   00AF304B   3B62304B
68000080   7B61304B   FB60304B   FB61304B   3B61304B
68000090   BB60304B   BB61304B   7B60304A   304B9A42
680000A0   18D00023   7B620AE0   2D4A7B6A   52F82320
680000B0   2A497B6A   41F82320   7B6A0133   7B62294B
680000C0   9B087A6A   9A42EFD3   4FF0E023   234AC3F8
680000D0   082D04E0   4FF0E023   214AC3F8   082D09E0
680000E0   7B691A78   3B6A1A70   3B6A0133   3B627B69
680000F0   01337B61   FA687B69   9A42F1D1   09E03B69
68000100   1A78FB69   1A70FB69   0133FB61   3B690133
68000110   3B61BA68   3B699A42   F1D105E0   BB690022
68000120   1A70BB69   0133BB61   7A68BB69   9A42F5D1
68000130   2C37BD46   5DF8047B   704700BF   0004FE1F
68000140   1C020068   1C020068   0004FE1F   1C020068
68000150   1C020068   00000020   00000020   0000FE1F
68000160   00000000   00040000   80B400AF   074B084A
68000170   5A60064B   4FF6FF72   9A60044B   4FF40452
68000180   1A60BD46   5DF8047B   704700BF   00200540
68000190   20C528D9   80B582B0   00AF0023   7B60FFF7
680001A0   E3FF0D4B   4FF08042   C3F83421   0B4A0B4B
680001B0   5B6D43F4   80735365   094A094B   5B6943F4
```

Figure 11. **QuadSPI memory region**

AHB access uses LUT sequence 0 as its default read sequence. Therefore, user must make sure to program sequence 0 with a valid read command before trying to use AHB access, by default sequence 0 is programmed with typical values for a simple (one data line) read.

It is important to notice that data read from external memory would be accessed significantly slower than from internal memory, due to the following factors:

- Data from external memory is retrieved at the QuadSPI clock frequency, while the internal data is accessed at core frequency.
- Internal memory data is within cache range, QuadSPI region is not.
- QuadSPI AHB buffer can be configured up to 4 KB, accessing data outside those 4 KB will require the QuadSPI to retrieve data from external memory, increasing the delay.

Another benefit of using AHB access is that it allows the execution of code from external memory, considering that it would be significantly slower as stated above.

7. Software example

This application note is accompanied by software. The software project can be open in the S32DS and runs over the S32K148 EVB using the MX25L6433F external memory available on the board. The example uses routines for both types of accesses, AHB and peripheral bus, it programs the external memory with a pre-compiled application and verifies that it was correctly written by reading it, both actions using peripheral bus access. Once it was verified, the program executes the application using AHB access. The application is a simple red LED toggling.

8. Reference

- AN5412, Quad Serial Peripheral Interface (QuadSPI) Module Updates
- AN4186, Using the QuadSPI Module on MPC56XXS
- AN5244, How to use QuadSPI on KL8x Series

Document Number: AN12193
Rev. 0
05/2018