Freescale Semiconductor
Application Note

Document Number: AN4186
Rev. 0, 09/2010

Using the QuadSPI Module on MPC56xxS

by: Daniel McKenna
Microcontroller Solutions Group
East Kilbride
Scotland

Contents

1 Introduction

This application note will discuss the QuadSPI module on the MPC560xS device and demonstrate how to use the module to configure, program, and read external memories with QuadSPI support.

This application note is accompanied by a zip file containing all the code discussed in it, as well as drivers to implement QuadSPI support for various debuggers.

2 QuadSPI overview

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.

At the time of publication of this application note all available QuadSPI memory devices use 24-bit addressing, giving a maximum capacity of 16 MB per chip.

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 (0x03)” command is sent, followed by the 24-bit address to be read. The data is then sent to the microcontroller.

There are several suppliers of QuadSPI-compatible memory, such as Winbond, Spansion, Macronix, and Numonyx. This application note will focus on the Spansion and Winbond devices as they were readily available at the time of publication. 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. At the time of design of the QuadSPI module on MPC560xS, only Winbond had a final specification released. Because of this, all Winbond commands are supported while some of the unique Spansion instructions are not. Any differences will be highlighted in the appropriate section within this document.

The command set supported by the QuadSPI module is shown in table Table 1. Instructions that are not required or supported by Spansion devices are in shaded table rows.

Table 1. Supported Commands

Command	Instruction code
Write Enable	0x06
Write Disable	0x04
Read Status Reg	0x05
Read Config Reg	0x35
Write Reg	0x01
Page Program	0x02
Quad Page Program	0x32
32K Block Erase	0x52
64K Block Erase	0xD8
4K Sector Erase	0x20
Chip Erase	0xC7/0x60
Erase Suspend	0x75
Erase Resume	0x7A
Power down	0xB9
High Performance Mode	0xA3
Read Data	0x03
Fast Read	0x0B
Fast Read Dual Output	0x3B
Fast Read Dual I/O	0xBB
Fast Read Quad Output	0x6B
Fast Read Quad I/O	0xEB
Octal Word Read Quad I/O	0xE3
Continuous Mode Bit Reset	0xFF
Release High Perf/ Power Down	0xAB
Read Manufacturer/Device ID	0x90
Read Unique ID	0x4B
Read JEDEC ID	0x9F

Both the Winbond and Spansion flash contain a Status register and a Configuration register. The former contains flags denoting error, flash busy, etc., whilst the latter is used to enable/disable Quad Mode and to set up memory erase protection if required. These registers are read and written using the commands in table Table 1.

Notice the numerous different read instructions — these select the number of data pins to use for the operation and also control the format in which the data is received. These will be fully discussed in section Programming.

3 QuadSPI module

In order to ensure backwards compatibility, the QuadSPI module on the MPC560xS device has support for both QuadSPI and SPI communications. This application note will focus on the QuadSPI features. For further information on using the module for an SPI interface refer to Freescale document AN2867,"Using the DSPI module on the MPC5500 Family," available at freescale.com.

When QuadSPI mode is enabled by setting the QUAD bit in the Module Configuration Register (MCR), the base address of the external flash is mapped to 0x8000_0000 of internal memory, and fills the range 0x8000_0000—0x87FF_FFF7. Note that the external memory is mirrored throughout the entire region — thus for a 32 Mbit memory, address 0 can be accessed at 0x8000_0000, 0x8040_0000, 0x8080_0000, etc.

The memory map of the QuadSPI module contains registers which pertain to both DSPI and QuadSPI mode. The registers that are only for DSPI will have no effect in QuadSPI mode. Only the Module Control Register (MCR) is applicable to both DSPI and QuadSPI.

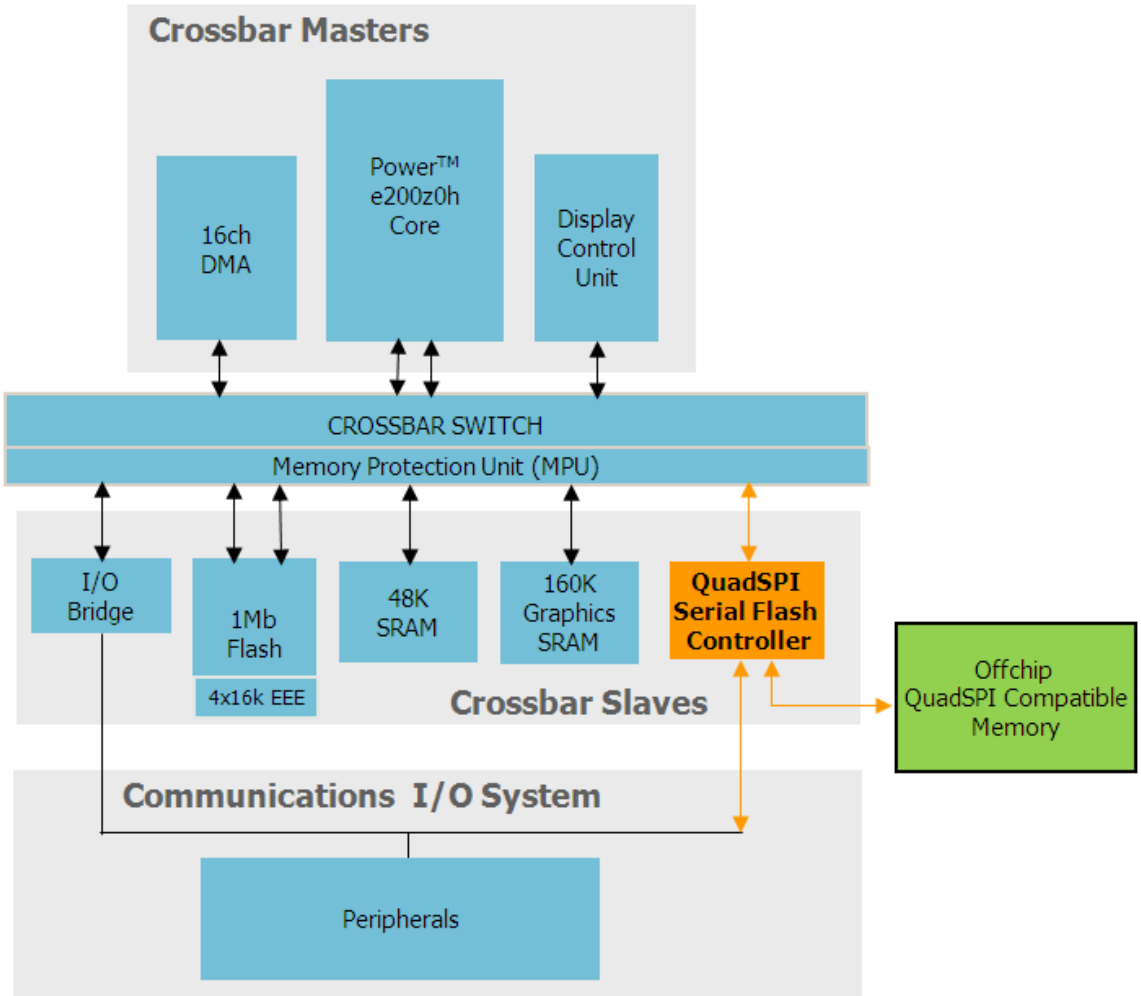

Figure 1. Overview of crossbar configuration on MPC560xS

The QuadSPI module can be used to read, write, and erase an external flash memory. Note from figure 1 that the QuadSPI can be accessed via two different routes: either across the I/O bridge like a typical peripheral, or directly over the crossbar like the other memory ranges.

Writes and erases are carried out by accessing the module across the I/O bridge like a standard peripheral and sending commands to the external memory. Read commands can additionally be automatically carried out over the crossbar. This means that any bus master accessing an address which is memory-mapped to the external flash will automatically receive the data at that address, without any software intervention. This is a powerful feature which allows the QuadSPI to store graphical information that the DCU can then pull directly. As such, after configuration, for all read operations the external QuadSPI memory is treated the same as other internal memories. The specific waveforms required by read/write/erase commands are generated automatically by the module; therefore, to change between a read and a fast read is as simple as changing the command.

Figure 1 shows the crossbar configuration on the MPC560xS. All masters (DCU, Z0 core, and DMA) are able to access the QuadSPI directly over the crossbar.

4 Hardware considerations

The QuadSPI device should be connected to the MPC560xS as shown in Figure 2.

The layout of the flash device from the microcontroller plays a major factor in the maximum speed that can be achieved. Tracks to the flash device should be as short as possible and each signal should be matched as much as possible. The data sheet for the memory being used should be consulted to ensure that the layout matches its specification.

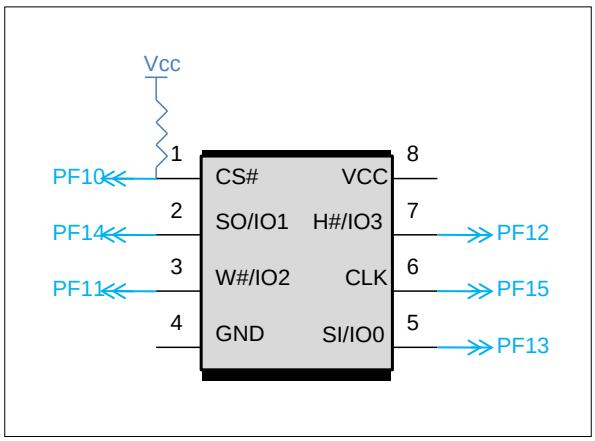

Figure 2. Connecting an external QuadSPI memory to MPC560xS

Both Winbond and Spansion parts also recommend a hardware pullup to VCC on the CS pin. This is to ensure that, during power-up, the voltage on the CS pin tracks that on VCC. The correct value of resistor to be used will be given in the part's data sheet.

Both Spansion and Winbond memories are initially set at the factory to operate in SPI mode. In order to enable QuadSPI operation, a nonvolatile bit in the flash's configuration register must be set by sending the correct command. This has to take place only once in the life of the memory.

Code to perform this operation is included in Sample code for external flash control.

5 Setting up QuadSPI

This section shows the basic configuration needed in order to interface with the external flash. These steps must be completed before a read/write/erase operation can be carried out.

Step 1: Set up the ports

On MPC560xS devices the QuadSPI is assigned to PortF[10] to PortF[15]. As such, the Port Configuration Registers (PCR) must be set up to designate these pins for use by the QuadSPI module. The number of ports required is dependent on how many data I/O lines are to be used: one, two, or four.

```
/* configure ports */
SIU.PCR[80].R = 0x0A00;   //QuadSPI pin config CS
SIU.PCR[81].R = 0x0B00;   //QuadSPI pin config IO2
SIU.PCR[82].R = 0x0B00;   //QuadSPI pin config IO3
SIU.PCR[83].R = 0x0700;   //QuadSPI pin config IO0
SIU.PCR[84].R = 0x0700;   //QuadSPI pin config IO1
SIU.PCR[85].R = 0x0600;   //QuadSPI pin config SCK
```

All I/O ports being used should have both their input and output buffers enabled. Depending on hardware setup, it may also be necessary to vary the slew rate of the output signals. An oscilloscope can be used to find which settings present the most suitable signals.

Step 2: Set up the clock

The speed at which the QuadSPI interface operates is restricted by a number of factors, including board layout and pad speeds. With a good layout it should be possible to communicate at speeds of up to approximately 48 MHz on the MPC560xS (figure correct at time of writing – feasibility of operating at faster speeds is currently under investigation. Please see the device data sheet for the most up-to-date details.

On the MPC560xS the QuadSPI module is clocked through Auxiliary Clock 3 which is configured in the Clock Generation Module (CGM). The clocking options are:

- System clock
- System clock ÷ 2
- Secondary FMPLL
- Secondary FMPLL ÷ 2

In the following instance the QuadSPI clock has been set to the system clock divided by two.

```
CGM.AC3SC.B.SELCTL = 1; //set QuadSPI clock as sysclk/2
```

Step 3: Change module to Quad mode

In order to maintain backwards compatibility, the QuadSPI module will default to DSPI mode and will have identical functionality to other DSPI modules. In order to enter Serial Flash Mode (SFM), the Module Configuration Register (MCR) must be altered.

```
QUADSPI_0.MCR.R = 0x00000088; //Enter SFM, Vendor ID=Winbond
```

Note that on the MPC560xS the vendor ID bits should be set to Winbond (0b0001) even if Spansion or another brand of flash is used. Almost all instructions are compatible. At the time of design, only Winbond had a finalized spec available, which is the reason that this is the only vendor ID listed. In future devices Spansion parts will be given an ID of 0b0010.

The Serial Flash should now be accessible. This can be tested by opening a memory window in the debug environment and accessing address 0x80000000. The debugger should be able to perform reads on the flash across the crossbar. In a new device all memory will be set to 0xFF.

6 Using the QuadSPI

This section will describe how to perform read, write, and erase operations on the QuadSPI module. It will also cover module behavior upon entering low-power mode.

6.1 Reading

As previously stated, the QuadSPI module has two different methods for reading from the external flash: IP commands and AHB commands. Each of these methods has its own internal buffers.

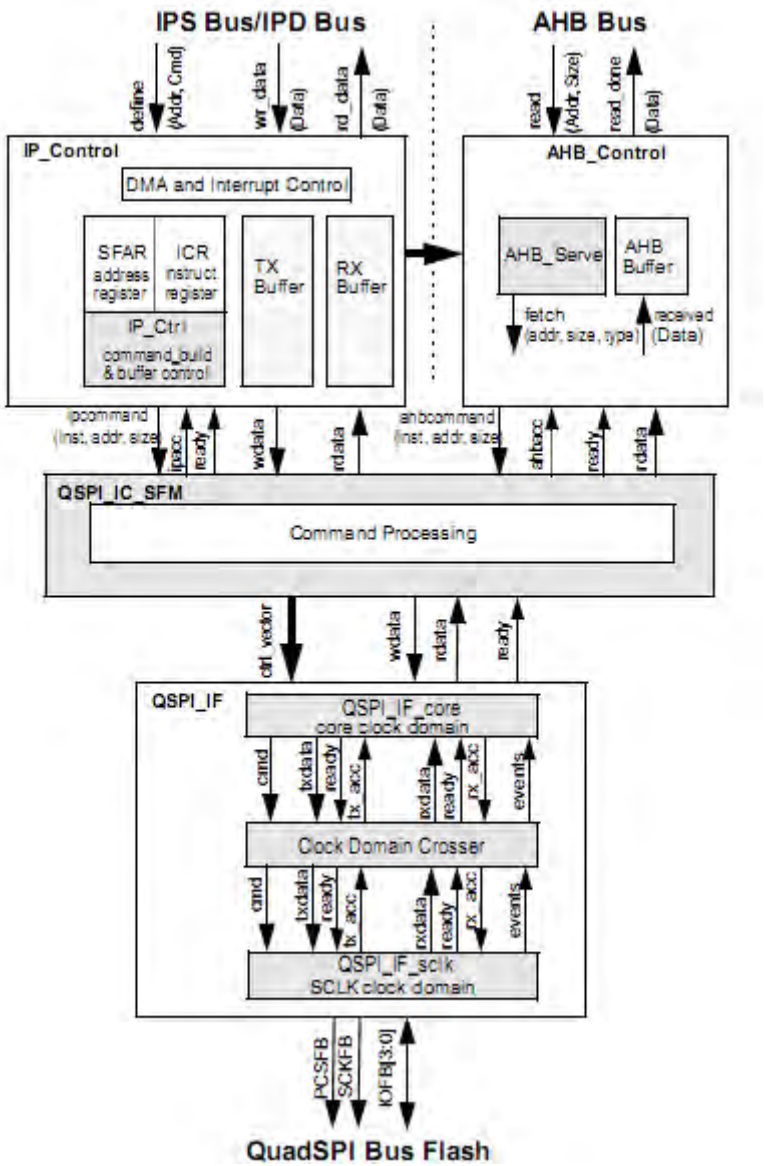

Figure 3. Block diagram of QuadSPI module (DSPI omitted)

IP commands (so named because they are written to the QuadSPI module over the I/O system via the IPS interface) are created by writing individual parameters (instruction code, address, size) to the correct registers. These are then passed to Serial Flash Mode (SFM) Command Processing where the full command is built up. This command is then passed across the Clock Domain Crosser and is output on the correct pins. The received data is stored in the Rx Buffer FIFOs (RBDR[0..14]).

Everything above the Clock Domain Crosser is clocked by the system clock, whereas everything below it is clocked by the QuadSPI clock (see Setting up QuadSPI for further details).

AHB bus accesses have their instruction code and size configured in advance. When the crossbar attempts to access an address which corresponds to the external flash, the code, size, and address are sent to the SFM Mode Command Processing as before. The received data is stored in the AHB Rx Buffer. This is not memory-mapped — the data is fed directly to the master which made the access.

6.1.1 IP commands

Once the module has been initialized (see Setting up QuadSPI), a read operation can be carried out. Several read commands are available: these are listed in Table 2 with a brief explanation for each. A fuller description of each command and its associated timings will be available in the relevant manufacturer's datasheet.

Table 2. Read commands

Instruction	Code	Description
Read Data	0x03	Single line read — frequency limited to 40–50 MHz.
Fast Read	0x0B	Single line read — up to maximum frequency.
Fast Read Dual Output	0x3B	Dual line read.
Fast Read Dual I/O	0xBB	Dual line read, dual-address TX. Continuous mode available.
Fast Read Quad Output	0x6B	Quad line read.
Fast Read Quad I/O	0xEB	Quad line read, quad-address TX. Continuous mode available.
Octal Word Read Quad I/O	0xE3	Winbond only. Quad line read, quad-address TX with least significant four bits of address set to zero. Continuous mode available.

In continuous mode the instruction is sent only once. Further reads require only an address to be sent, reducing overhead. This will be discussed further in Setup for maximum read performance.

Before executing a new instruction, it is necessary to ensure that any previous instruction is complete. This can be done by checking the BUSY flag in the SFMSR register.

```
while(QUADSPI_0.SFMSR.B.BUSY);
```

The memory-mapped 32-bit address to be accessed is first entered in the Serial Flash Address Register (SFAR). Thus to access 0x100 on the external flash, the register is set as follows:

```
QUADSPI_0.SFAR = 0x80000100; //Start Address
```

The instruction code, along with any instruction code options, is then entered into the Instruction Code Register (ICR). This triggers the transmission of the instruction.

Table 3. Instruction Code Register (ICR)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	R	ICO															
	W																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	R	ICO								IC							
	W																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The ICO bits in the ICR register are divided in a way that is dependent on the instruction being sent. The lower eight bits (register bits 16–23) are used for the mode bits which are applicable for some instructions — they are generally used to select whether the instruction should make use of continuous mode or not. The upper sixteen bits (register bits 0–15) are used as the size field, allowing for up to 64 KB of adjacent bytes to be read from a single instruction. Read addresses do not have to be aligned — partial reads (that is, reads of less than one word) are also supported.

Reads triggered by IP are stored in the fifteen QuadSPI RX buffers (RBDR[0..14]). Because these buffers are memory-mapped it is possible to read the data directly from them. However, the recommended method is to read incoming data from the (AMBA) register, which will always point to the most recent data in the buffer. The AMBA register is accessible straight from the crossbar — there is no need to go through the I/O bridge in order to access it. Therefore the data can be retrieved more quickly. As previously noted, the AMBA register is not grouped with the other QuadSPI registers but resides in the final word of the memory-mapped QuadSPI region: 0x8FFF_FFFC. Thus this register is not included in the standard MPC560xS header file and has to be declared by the user as follows:

```
#define QUADSPI_ARDB (*(vuint32_t *)(0x87FFFFFC));
```

Thus, once a read command has been sent, the received data can be read in the following manner:

1. Poll the Receive Buffer Not Empty flag (RXNE) in the Status register (SFMSR) and wait until it has been set.
2. Read the data from the ARDB register.
3. Clear the RXNE status bit by writing to the RBDF flag. This updates the RX buffer, causing ARDB to point to the next unread received data.
4. Repeat steps 1 to 3 for data size.

6.1.2 AHB commands

The read command used for AHB accesses is set up in the AMBA Control Register (ACR) register. This register is shown below.

Table 4. AMBA Control Register (ACR)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	R									ARMB							
	W																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	R		ARSZ							ARIC							
	W																
	Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1

The mode (ARMB) and Instruction Code (ARIC) are the same as those used for IP commands. The size field (ARSZ) is slightly different: the value entered is the multiple of four bytes to be sent — therefore an ARSZ value of ten will set up 40-byte transfers. The ARSZ value must be between one and sixteen, giving transfer sizes in the range of 4–64 bytes. Notice that the default values of the register will set up a 4-byte read using the Read Data (0x03) command. This uses a single data line, and is the slowest transfer rate available.

When using Winbond flash, some instructions will require a High Performance Mode Instruction to be sent before the actual instruction is transmitted. More details on this requirement can be found in the Winbond data sheet. This instruction should be sent using the IP Command method before the ACR register is configured. Note that this instruction is ignored by Spansion devices.

Any access of the memory-mapped external flash by a bus master will be automatically carried out and the result obtained without any software intervention.

6.1.2.1 AHB buffer

All data retrieved from crossbar accesses (in other words, from AHB commands) to the QuadSPI are stored in an internal 16x32-bit buffer (64 bytes). This is completely separate from the Rx Buffer used to store the received data from IP commands.

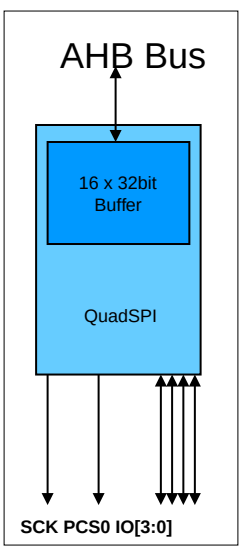

Figure 4. AHB Buffer on MPC560xS

All data in the AHB buffer is tagged with its address on the external memory. Thus any crossbar access is first checked with the contents of the buffer — if the data is present in the buffer then it can be accessed immediately with no wait states. This allows for single-cycle access. In this instance no QuadSPI instruction is executed.

If the data is not in the buffer and is not being fetched by the currently executing command, then the current command is aborted and the buffer is flushed. A new command is then executed with the correct address. It is possible to read from the buffer whilst it is being filled; for example, if a 64-byte read is executed it will be possible to read the first 32 bytes whilst the last 32 bytes are being fetched. This helps keep crossbar access times to a minimum.

6.1.2.2 Setup for maximum read performance

Maximum data throughput is achieved by using the Fast Quad Read I/O (0xEB). (On Winbond devices, the Octal Word Read Quad I/O (0xE3) instruction can give even faster throughput but it is restricted to 8-byte-aligned reads). The optimum access size for an application is generally sixty-four bytes. As previously mentioned, if only four bytes are required then a 64-byte read will still be carried out. However, once the required four bytes have been fetched, then any further crossbar accesses will abort the remainder of the instruction and transmit a new command. If several nonadjacent data reads are required, then a smaller read size could be more power-efficient, because a large read size would lead to the majority of the fetched data being disposed.

From a performance point of view, the peak can be achieved by setting the size to its maximum value. This is due to the fact that transmitting an instruction adds an overhead and takes up clock cycles which could otherwise be used to receive data. More details on this can be found in the command overhead section below.

The QuadSPI module clock should be configured for maximum speed (see Setting up QuadSPI), then the following line will configure the module for maximum performance.

```
QUADSPI_0.ACR.R = 0x00A040E3 // 0xE3 w continuous burst. 64byte transfers
```

Continuous Burst Mode Reads

Several QuadSPI read instructions support burst mode, which gives a decrease in the instruction overhead by requiring the instruction code to be transmitted only once. Future writes require only an address to be sent and the memory will return the required data. This leads to increased throughput.

Continuous burst instructions can be used for either IP or AHB accesses. In order to exit from burst mode (in other words, to change to use a write or an erase command), the Continuous Read Mode Reset instruction (0xFF) must be sent in order to notify both the external memory and the QuadSPI module. Note that although the 0xFF instruction is not supported by the Spansion memory, it must still be sent in order to notify the QuadSPI module — the Spansion memory will simply ignore the instruction. This should also be remembered when awakening the MPC560xS from a low-power state — for example, if the MPC560xS is put into a low power mode but the QuadSPI remains powered, it will retain its previous state and will therefore still be in burst mode upon wakeup.

Command Overhead at Maximum Performance

Due to the nature of memory accesses on the external QuadSPI memory and its command-driven interface, the maximum performance is achieved during burst transfers rather than small, random, non-sequential accesses.

To illustrate this point we will show two accesses and their associated overheads: first a small random access, and second a large sequential overhead.

RANDOM ACCESS:

Using the optimum 0xEB instruction with continuous burst and an access size of 32 bits:

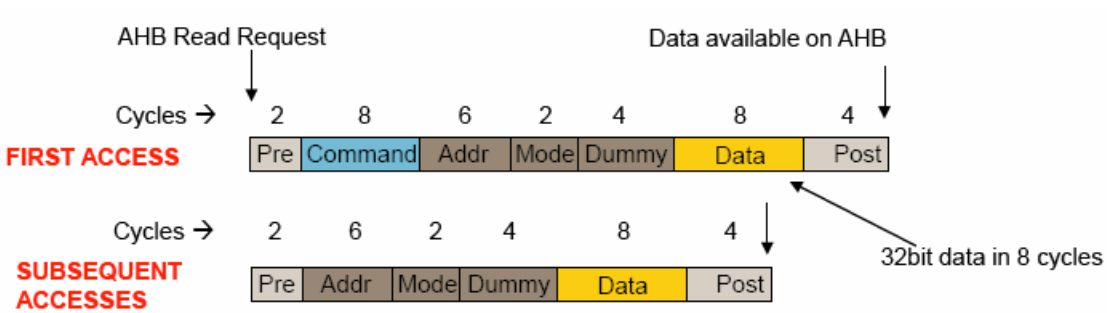

Figure 5. 32-bit access pattern: 34-26-26-26-….

Note that the initial access takes thirty-four cycles from data request until data available. Further accesses do not require the command to be sent, therefore eight cycles are saved. Therefore all subsequent accesses require twenty-six cycles. The Dummy cycles are the fetching period required by the QuadSPI memory, Pre and Post are the cycles required to initialize the instruction (Pre) and make it available in the buffer (Post).

SEQUENTIAL ACCESS:

In this instance we use the same instruction as above but set the read size to 64 bytes:

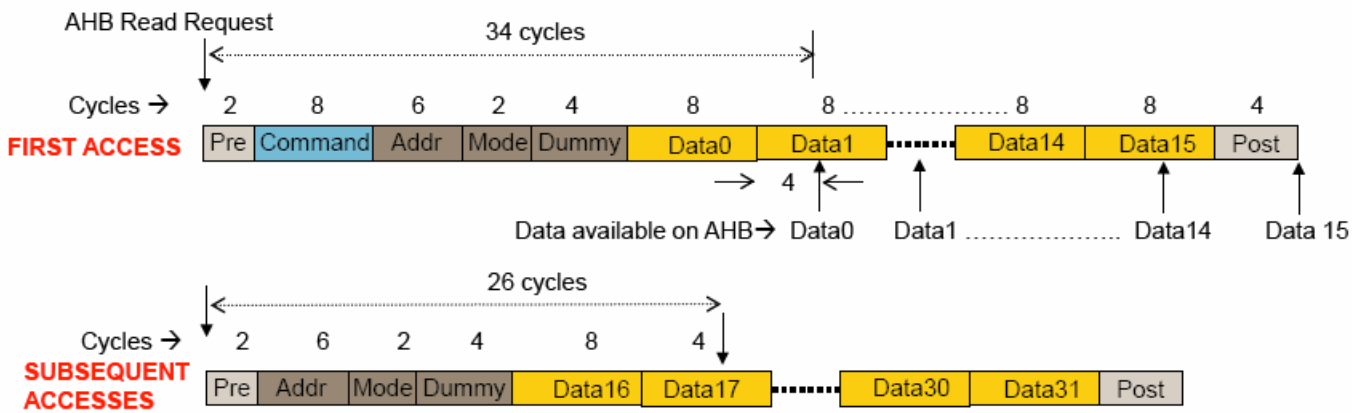

Figure 6. First 32-bit access pattern: 34-8-8-8-…. subsequent 32-bit access pattern: 26-8-8-8-….

NOTE

Any accesses to data already stored in the buffer can be accessed in a single cycle. No command is sent in this instance.

These performance characteristics make the QuadSPI useful for storing graphics on MPC560xS, which can then be fetched by the DCU.

6.1.3 DMA accesses

As DMA is a bus master, it is possible to set up the DMA to transfer data from an address that has been mapped to the external QuadSPI to another location. This will cause the QuadSPI module to initiate a transfer using the AHB command. However, this may not be the optimum method for carrying out DMA reads, as in this instance the DMA will have sole control over the QuadSPI's AHB interface whilst the data fetch is being executed. If your application has other masters carrying out large accesses to the QuadSPI, then access delays may become a problem. A better method is to trigger the DMA only when the received data is available. This is achievable by using IP Commands.

The QuadSPI module contains a trigger to the DMA which, when enabled (by setting the SFMRSER.RBDDE bit), will cause a DMA transfer to commence whenever there is unread data in the RX Buffer. The DMA channel should be set up to read four bytes from the ARDB register and store the four bytes in an arbitrary location. The CPU can then be used to execute a read using an IP command. The DMA will only trigger once the first 32-bit word is received.

An example of using the DMA in conjunction with the QuadSPI module can be found in SW pack.

6.2 Erasing

As is typical with large flash memories, erase instructions affect a larger block size than write operations. The size of an erasable sector differs between parts, and the data sheet should be consulted to find sector and block sizes. Also note that the erase size may not be uniform over the full memory map.

The erase instructions differ slightly between Winbond and Spansion parts, as shown in the table below:

Table 5. Erase Instruction Compatibility

Command	Instruction code	Win-bond	Span-sion
4K Sector Erase	0x20	X	X
32K Block Erase	0x52	X	
64K Block Erase	0xD8	X	X
Chip Erase	0xC7/0x60	X	X

Note that Spansion contains an 8K sector erase command (0x40) which is not supported by the QuadSPI module. As an alternative, two 4K erase commands should be used.

An erase is carried out in this way:

1. The start address of the sector/block to be erased should be written to the SFAR register. If the address is not aligned with a sector or block, then the erase will still take place, and all data within the sector/block that contains the address will be erased.
2. All erase and write instructions require a Write Enable instruction (0x06) to be sent prior to the operation. The code should then pause until the module busy flag (SFMSR.BUSY) is cleared.
3. The instruction code for the required erase command should then be written to the ICR register.
4. The module should now wait for the module busy flag to clear (SFMSR.BUSY) and for the busy flag on the external flash to clear, denoting that the erase is complete.

The code to carry out a 4K sector erase at address 0x000000 of the external flash is shown here:

```
QUADSPI_0.ICR.R = 0x00000006; /* enable the flash for writing */
while(QUADSPI_0.SFMSR.B.BUSY ==1); /* while module busy */
QUADSPI_0.SFAR.R = 0x80000000;
QUADSPI_0.ICR.R = 0x00000020; /* 4k Erase */
while(QUADSPI_0.SFMSR.B.BUSY ==1); /* while module busy */
wait_while_flash_busy(); /* wait while flash busy – function code in Appendix */
```

6.3 Programming

External flash is programmed in pages of 256 bytes. Two commands are available, Page Program (0x02) which uses a single line to transmit data, and Quad Page Program (0x32) which, as the name suggests, uses all four lines. The write speed of the external flash will be much slower than its read speed — the exact figure will be in the data sheet but can typically take 1.5 ms. Because of this constraint there is no great benefit from using Quad Writes at higher bus speeds, because the bottleneck is not the transmission time but the flash programming time.

In order to program a page of flash the following routine should be used.

1. The start address of the page to be programmed must be written to the SFAR register. The address does not have to be aligned with the beginning of a page; however, if the end of the page is reached and there is still more data to be sent, then this will wrap around and continue writing at the first address of the page.
2. Send a Write Enable Command, and wait until the module busy flag is cleared.
3. Write the first sixteen bytes (or all the data if size <16 bytes) to be sent to the Transmit Buffer (TBDR) register.
4. Send the write command and size (in bytes) using the ICR register. The maximum size is 256 bytes (one full page).
5. The transmit buffer will now begin emptying as the data is sent. The (SFMSR.TXFULL) should be polled until it is cleared, at which point further data should then be put into the buffer. This should be repeated until all data has been transmitted.
6. Wait until both the module and the external flash are no longer busy.

Code to carry out the above steps is shown below, where len = size of data to be written in bytes, tx_data[i] = array of data to be written to external flash.

```
QUADSPI_0.SFAR.R = 0x80000000;

for(i=0; ((i<15)&(i<(len/4))); i++)
{
      QUADSPI_0.TBDR.R = tx_data[i];
}
QUADSPI_0.ICR.R = 0x00000006; /* enable the flash for writing */
while(SFMSR&QSPI_SFMSR_BUSY==1); /* while module busy */
QUADSPI_0.ICR.R = 0x00000002|((len)<<16); /* trigger single line write of size len */
for(i=15;(i<(len/4));i++)
{
      while(QUADSPI_0.SFMSR.B.TXFULL);
      QUADSPI_0.TBDR.R = tx_data[i];
}
while(QUADSPI_0.SFMSR.B.BUSY ==1); /* while module busy */
wait_while_flash_busy(); /* wait while flash busy – function code in Appendix */
```

6.4 Low-power mode entry

When the QuadSPI module receives a request to enter a low-power mode from the Mode Entry module on MPC560xS, the QuadSPI finishes any active transfer before acknowledging the low power request.

Appendix A Sample code for external flash control

This routine will check whether external flash device is currently busy (in other words, if a read/write or erase operation is currently executing).

```
void wait_while_flash_busy(void)
{
volatile unsigned int read_data;

read_data = 0x01000000;
while((read_data & 0x01000000) == 0x01000000) /* loop while BUSY bit is set */
{
ICR = 0x00010005; /* Query status register */
      while(SFMSR&QSPI_SFMSR_BUSY == 1);
while(SFMSR&QSPI_SFMSR_RXNE == 0);
read_data = ARDB;
SFMFR = 0x10000; /* read complete */
}
}
```

This routine will change external flash device to QuadSPI enabled mode. The Config_QSPI function must be run first to set up the clock and pin multiplexing.

```
/* change to QuadSPI mode */
qspi_flash_write_enable();
QUADSPI_0.TBDR = 0x00020000; //data to be writen
QUADSPI_0.ICR.R = 0x00020001; //write config register
while(QUADSPI_0.SFMSR.B.BUSY);
wait_while_flash_busy();
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4186
Rev. 0, 09/2010