SPI-MEM-CTRL

Design Specification

Document Revision History

Revision	Date	Comments
1.00	31-Jan-2010	○ First release.
2.01	11-Nov-2010	○ New clock domain **`spi_2sclk`** instead of spi_4sclk. ○ Addition of Boot Function feature. ○ Addition of multiple Serial Flash devices feature.
3.00	25-Apr-2011	○ Addition of support for Serial Flash devices greater than 128Mbits feature. ○ Addition of full quad support feature (READ4IO, PP4IO instructions). ○ Addition of support for non JEDEC compliant devices. ○ Addition of the JEDEC information register.
3.01	19-May-2011	○ Updated Section 6. ○ Minor corrections.
4.00	25-Jun-2012	○ Correction in Table 4-3. ○ Interrupt Support. ○ Custom instruction feature. ○ Status Register update. ○ Two asynchronous reset lines. ○ Updated Default Memory register.
4.01	11-Jul-2012	○ Updated Table 6-1.
4.02	12-Dec-2012	○ Control Register: SPI-Mode configurable during synthesis ○ RX_delay field – added timing diagram
4.03	21-Jan-2013	○ Added Figures for READ and PAGE PROGRAM instructions ○ Minor Bugfix section 4.5.12
4.04	15-Apr-2013	○ Added programmable number of dummy cycles for read instructions ○ Added Read Dummy Cycles Register ○ Section 4.6.9
5.00	28-May-2013	○ Added Full Read Access Request ○ Added Execute on the Fly optional interface ○ Updated Extended Addressing mode register ○ Minor corrections.
5.01	16-Dec-2013	○ Updated Host Interface Timing Diagram

5.02	22-Apr-2014	○ RX-delay field increased to support 8 delay steps and moved to Read Dummy Cycles register.
5.03	11-Jul-2014	○ Added RX-delay value for automatic programming via configuration ROM.
5.04	8-Sep-2014	○ Updated Figure 4-9.
5.05	9-Mar-2016	○ Improvement of Custom Instruction to support arbitrary transfer length ○ Added support for page program instructions with 512-byte page size
5.06	8-Sep-2017	○ Update clk_delay constraints
5.07	9-Feb-2017	○ Added Extended Configuration Register (11h)
5.08	3-Sep-2018	○ Added SPI NAND Flash support.

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION AND FEATURES

1.1 Introduction

The SPI-MEM-CTRL core provides the necessary functionality to a host in order to communicate with a Serial Flash device using the Serial Peripheral Interface (SPI). The core supports three types of memory accesses: read, write and erase, as well as custom instructions.

The SPI-MEM-CTRL core is designed to provide to a host a simple interface for controlling SPI Serial Flash Memories. The core is highly flexible and can be configured before synthesis or programmed during runtime to support a large number of SPI Serial Flash memories, even less standard ones. Support for newer serial flash devices with densities larger than 128Mbits is also included.

A host can interface to the Serial Flash in a number of ways. Transferring data from the Flash memory to the host is done with minimum effort with a Block Read Interface, which uses a DMA mechanism to transfer a block of data to the host's memory space. Alternatively, the host can introduce a read request using the core's programmable registers. Then the core serves this request and sends the necessary instructions to the Serial Flash device. An additional RAM like interface which permits on-the-fly code execution is also available.

The design uses two clock domains. The first clock domain is used for the host interface of the core, while the second clock domain is used for generating the SPI clock and must be at least 2 times faster than the target SPI clock.

The core features configurable size read and write FIFOs and an interface to an external configuration memory.

Being carefully designed and rigorously verified, the SPI-MEM-CTRL is a reliable and easy-to-integrate core. Ease of integration is served by a complete

verification environment, and additional aids for system on chip simulation, such as an example simulation test-vector files.

This document describes the specification of the SPI-MEM-CTRL core. It includes the overall architectural description, detailed functional specifications, programming specification and interface definitions.

1.2 Features

Device Independent

- Supports most SPI NOR and compatible SPI NAND Flash memory devices
- Automatic identification of a variety of memories
- Configurable memory features to allow support of less standard Serial Flash devices
- Ability to communicate with up to 8 Serial Flash devices
- Supports memories with densities greater than 128Mbits

Efficient Bandwidth Utilization

- Single SPI
- Dual data output SPI
- Dual data input / output SPI
- Quad data input / output SPI
- Configurable size Read / Write FIFOs

Flexible Access Model

- Registered mapped I/O
    - Host issues access request, reads and writes data via register accesses

NOR SPI Flash features

- Read access size from 1 byte up to the entire memory size
- Read accesses starting from any address offset
- Write access sizes from 4 bytes up to the entire memory size
- Write accesses starting from any address offset that is multiple of 4
- Erasure of:
    - any sector (4KB)
    - any block (64KB)
    - whole chip

NAND SPI Flash features

- Read access size from 1 byte up to the entire memory size
- Read accesses starting from any address offset that is 32-bit word aligned
- Write access sizes from 4 bytes up to the entire memory size
- Write accesses starting from any address offset that is 32-bit word aligned
- Separate read and write requests for the ECC data area in each page
- Erasure of:
    - any block (128KB)

Ease of Integration

- Capable to auto-detect the connected Serial Flash devices
- Deep Power-down Mode support to minimize power consumption
- Block Read Interface simplifies read transfers implementing a DMA mechanism
    - Block Read Interface can be enabled to automatically initiate a read transfer after powering-up
- Execute on-the-fly (XOTF)
    - Random access data input
    - Permits direct code execution from microcontrollers without using cache
    - Minimum overhead on sequential address reads

- Ability to send custom instructions and read back memory response of arbitrary length using single SPI.
- Interrupts
- Programmable Number of Read Dummy Cycles for READ2IO and READ4IO instructions.

2 TERMINOLOGY AND SYMBOL CONVENTIONS

2.1 Abbreviations & Acronyms

In the following table the abbreviations and acronyms used throughout this document are presented and explained.

Table 2-1: Abbreviations & Acronyms

Term	Description
DPM	Deep-power Down Mode
FIFO	First-In First-Out memory
FPGA	Field Programmable Gate Array
SPI	Serial Peripheral Interface

2.2 Terms Defined

In the following table the key terms used throughout this document are defined.

Table 2-2: Terms Defined

Term	Description
BRI	Block Read Interface
LLSPIC	Low Level SPI Controller
Port	A top entity input or output
R/W	Read / Write access
SP	Synthesis Parameter
WIP	Write In Progress
XOTF	eXecute On-The-Fly

3 REFERENCES

3.1 Related Documents

SPI-MEM-CTRL Release Notes, Rev. r0, Dated 5 Oct, 2018

SPI-MEM-CTRL Integration Manual, Rev. 5.05, Dated 5 Oct, 2018

3.2 Web Sites

SPI in Wikipedia – http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

ALMA Technologies – https://www.alma-technologies.com

4 ARCHITECTURAL SPECIFICATION

4.1 Symbol

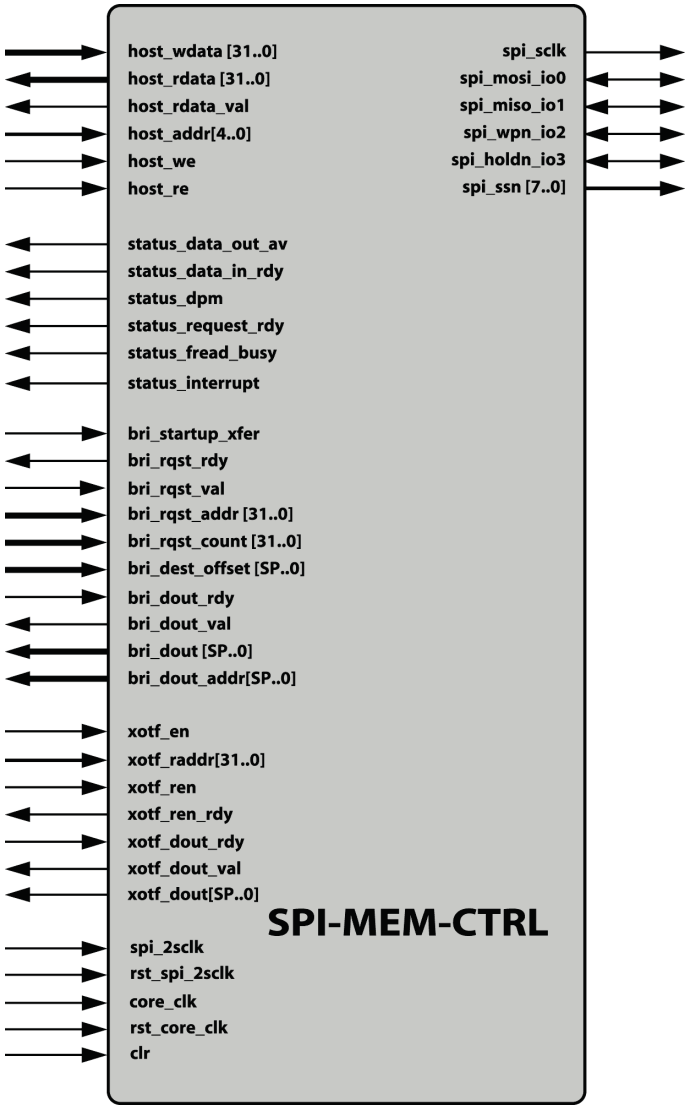

Figure 4-1: Core's Symbol

4.2 Pin Description

Table 4-1: Pin Description

Pin Name	Type	Polarity/ Bus size	Comments
Global Inputs			
`core_clk`	I	Rising	Core Clock
`spi_2sclk`	I	Rising	Clock 2 times faster than desired spi_sclk
`clr`	I	SP[1]	Synchronous clear
`rst_core_clk`	I	SP[1]	Asynchronous reset for core_clk domain
`rst_spi_2sclk`	I	SP[1]	Asynchronous reset for spi_2sclk domain
Host Interface			
`host_wdata`	I	32	Input Data Bus
`host_rdata`	O	32	Output Data Bus
`host_rdata_val`	O	High	Output Data Bus Valid
`host_addr`	I	5	Address Bus
`host_we`	I	High	Write Enable
`host_re`	I	High	Read Enable
Status Interface			
`status_data_out_av`	O	High	Read data available
`status_data_in_rdy`	O	High	Ready to accept write data
`status_dpm`	O	High	On Deep Power-down Mode
`status_request_rdy`	O	High	No access request in progress
`status_interrupt`	O	High	Interrupt
`status_fread_busy`	O	High	Full Read (FREAD) is in progress
SPI Interface			
`spi_sclk`	O	-	Serial Clock
`spi_ssn0 to spi_ssn7`	O	Low	Slave Select 0 … 7
`spi_mosi_io0`	I/O	1	Serial Data Output during Single Mode or Serial Data IO_0 during Dual - Quad Mode
`spi_miso_io1`	I/O	1	Serial Data Input during Single Mode or Serial Data IO_1 during Dual - Quad Mode
`spi_wpn_io2`	I/O	Low	Write Protect during Single Mode or Serial Data IO_2 during Dual - Quad Mode
`spi_holdn_io3`	I/O	Low	Pauses the device during Single Mode or Serial Data IO_3 during Dual - Quad Mode
Block Read Interface (BRI)			
`bri_startup_xfer`	I	High	Perform a Block Read transfer on startup
`bri_rqst_rdy`	O	High	Acknowledge block read request
`bri_rqst_val`	I	High	Block Read Valid
`bri_rqst_addr`	I	32	Starting address of the block to transfer from the serial flash address space
`bri_rqst_count`	I	32	Number of bytes to Read
`bri_dest_offset`	I	SP[1]	Address offset for computing the `bri_dout_addr`
`bri_dout_rdy`	I	High	Data Out Ready
`bri_dout_val`	O	High	Data Out Valid
`bri_dout`	O	SP[1]	Data Out
`bri_dout_addr`	O	SP[1]	Destination Address for `bri_dout` in the host

Pin Name	Type	Polarity/ Bus size	Comments
			memory space
Execute On the Fly Interaface (XOTF)			
xotf_en	I	High	Enable/Disable XOTF interface
xotf_ren_rdy	O	High	Acknowledge read enable request
xotf_ren	I	High	Read Enable
xotf_raddr	I	32	Read Address
xotf_dout_rdy	I	High	Data Out Ready
xotf_dout_val	O	High	Data Out Valid
xotf_dout	O	SP[1]	Data Out

Table Notes:
[1] Configurable during synthesis

4.3 Operation

The operation of the SPI-MEM-CTRL core is illustrated in Figure 4-2. This flowchart shows the basic actions that can be performed on the core to initiate data transfers or control the Serial Flash by programming the core's internal registers via the Host interface.

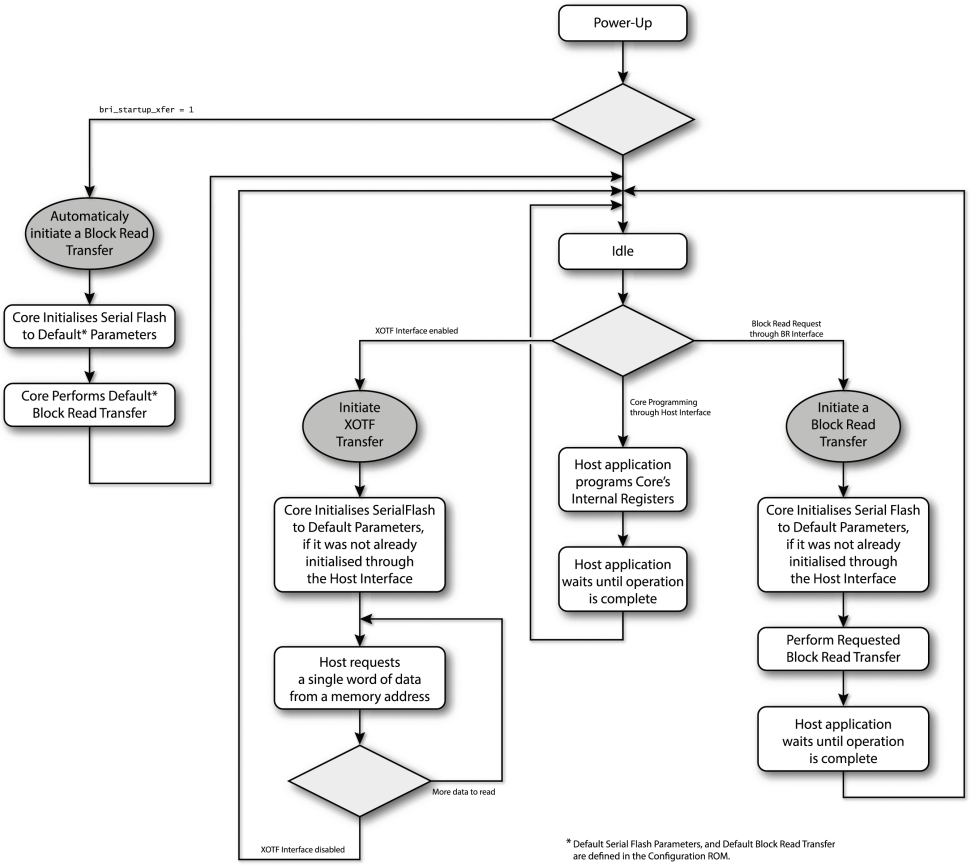

Figure 4-2: Flowchart illustrating the core's operation

After powering-up, or after an asynchronous reset or a synchronous clear, the core checks the value of the **`bri_startup_xfer`** input port. If this input is hardwired to logic 1, the core will automatically configure the Serial Flash memory using predefined parameters stored in a configuration memory. After the Serial Flash is initialized, the core will automatically initiate a Data Read transfer

and will start reading data from the Serial Flash. The parameters for this Block Read transfer are also stored in the configuration memory. This mechanism is useful when a block of data from the Serial Flash memory needs to be transferred during system boot to the host system's memory space.

After the automatic – "on power-up" – Block Read is complete, or if this mechanism is disabled, the core enters an idle mode, and it is ready to accept commands via programming of its internal registers. Besides using the Host Interface a Block Read transfer request can be issued prior to sending any command or programming to the core, using the core's independent Block Read Interface. Alternatively if the XOTF interface is enabled the core can accept single data word read requests.

In the case that a request is sent to the Block Read Interface, without prior initialization of the Serial Flash memory, this will be done automatically by the core using the default initialization parameters stored in the configuration memory. So in this case it is important that the correct parameters for the Serial Flash device used are stored in the configuration ROM.

After the first write operation to the Default Memory Register of the SPI-MEM-CTRL core, the core checks if the attached memory is one of those listed in Table 6-3. The SPI-MEM-CTRL core can also support any other memory that has similar behavior to those listed in Table 6-3. This is achieved by setting the Default Memory Register. If the memory cannot be auto-detected, the values of the Default Memory Register will be used to initialize the core. After this initialization phase, accesses to the Serial Flash device are performed at the maximum possible bandwidth. If the Serial Flash device supports Quad SPI the host must set it into quad mode respectively using Custom Instruction Register prior to any READ or WRITE access request.

The host can issue access requests via the **`host_wdata`** port. An access request consists of the access type (READ, WRITE, ERASE and FREAD), the address offset and the access length. Once the access request is issued, the core begins communication with the memory.

Note that READ requests can be issued through: the Host Interface, the Block Read Interface or the Execute on-the-fly (XOTF) Interface. The Block Read Interface is similar to a DMA mechanism, and returns a block of data without the host's interaction. The XOTF Interface is a mechanism for requesting random read accesses suitable for direct code execution by a microcontroller without the need for cache memories.

4.3.1 Host Interface - READ Access Request

If the core receives a READ access request, it begins reading data from a specific memory address of the Serial Flash, the Request Offset. These data are sequentially transferred to the Read-FIFO until all the data requested (Request Length) are being read. The **`status_data_out_av`** pin is used to signal the host when the number of words in the Read-FIFO is greater than the **`Read-FIFO Threshold`**. This threshold is programmable. If the Read-FIFO becomes full the read operation from the memory pauses until the host gets some data out of the Read-FIFO. Then the read operation resumes.

4.3.2 Host Interface – WRITE Access Request

When the core receives a WRITE access request, it begins writing to a specific memory address (Request Offset) and writes the data that the host has written into the Write-FIFO. The **`status_data_in_rdy`** pin is used to signal the host when the number of words in the Write-FIFO is lower than the **`Write-FIFO Threshold`**. This threshold is programmable. If the Write-FIFO becomes empty then the core pauses the write operation until there is new data in the Write-FIFO. Then the write operation resumes and bytes are written from the last point before the pause.

4.3.3 Host Interface – ERASE Access Request

If the host requests an ERASE access, the core erases the memory part that the host requested. The host must first make sure that the corresponding parts are not block protected. At that case the host must send custom instructions (using

the Custom Instruction Registers) in order to disable the Block Protection manually.

4.3.4 Host Interface – Full READ (FREAD) Access Request

If the core receives a FREAD access request, it begins reading data from a specific memory address of the Serial Flash, the Request Offset. These data are sequentially transferred to the Read-FIFO until the host signals a **`Break_Fread`** command into Access Request Register 2. The **`status_data_out_av`** pin is used to signal the host when the number of words in the Read-FIFO is greater than the **`Read-FIFO Threshold`**. This threshold is programmable. If the Read-FIFO becomes full the read operation from the memory pauses until the host gets some data out of the Read-FIFO. Then the read operation resumes. The **`status_fread_busy`** pin is asserted when a FREAD access request is issued and is deasserted after the **`Break_Fread`**.

4.3.5 Host Interface – Deep Power-down Mode

The core offers the host the ability to put the memory in Deep Power-down mode in order to minimize the power consumption. Once this is done, the core waits until the host requests the exit from DPM. Until then the core ignores any access request over the Host Interface. The period of time needed for the memory to enter and exit DPM is configurable.

4.3.6 Host Interface – Custom Instruction

The core provides to the host the ability to send a custom instruction to the Serial Flash device and read its response. This instruction is sent using Single SPI interface. Custom Instruction can be used for example to set the Serial Flash device in quad mode, unblock memory partitions before erase requests, exit 32 bit addressing mode or to send any other command in single mode.

4.3.7 Block Read Interface (BRI)

The core includes the Block Read Interface which simplifies significantly the data read transfers.

A Block Read transfer procedure can be initiated by sending a request using the **bri_rqst_val** input port of the core. The core receives the information needed to perform the transfer via the **bri_rqst_addr**, **bri_rqst_count** and **bri_dest_offset**. The start address of the data block in the Serial Flash device that will be transferred is specified on the **bri_rqst_addr** port. The base address in the destination address space is specified on the **bri_dest_offset** port. The number of bytes that will be transferred is specified on the **bri_rqst_count** port. The value of **bri_rqst_count** must be a multiple of 4.

As long as the **bri_rqst_rdy** port is low, the input ports of the host interface are ignored and the output ports of the host interface along with the ports of the status interface are driven to 0. When the **bri_rqst_rdy** port is high the host interface ports and the status interface ports have normal behavior. Then the host can have direct access to the SPI-MEM-CTRL core.

After the Block Read transfer is completed the core asserts the **bri_rqst_rdy** port. At this point the core is ready to receive a new Block Read request, or an access request command or a custom instruction through the Host Interface.

4.3.8 Execute On-The-Fly (XOTF) Interface

The core includes the Execute On-The-Fly Interface which simplifies significantly the random data read transfers.

The XOTF Interface is activated when the **xotf_en** is 1 and it is deactivated when the **xotf_en** is 0.

First of all, the XOTF mechanism checks if the SPI-MEM-CTRL core is initialized. If it is not initialized the XOTF runs the initialization procedure. The host requests

the data stored into the **xotf_raddr** address of the serial flash device. After the request is accepted the core begins to sequentially transfer data from the requested address and higher to an internal FIFO. If subsequent read requests refer to data from the addresses already present into the internal FIFO the XOTF will continue the read operation in a burst mode. In case of an address miss the XOTF mechanism breaks the current read operation, flushes the internal FIFO and initiates a new read operation to the requested address. The request data are placed to the **xotf_dout** output.

4.3.9 Connecting to Serial Flash Memory

The SPI-MEM-CTRL core supports SPI NOR or SPI NAND serial Flash devices. Depending on the type of the connected memory (NOR or NAND), the core must be set in the respective operation mode. This is done by programming the Control Register accordingly. Form a user's point of view operation in both modes is similar. The main difference is that NAND memories develop bad blocks, so the system software must implement an error detection / correction mechanism. The SPI-MEM-CTRL core provides extra request types for NAND operation that provide access to the ECC data area of each page of the NAND memory array.

The user can configure the core to communicate with one of up to 8 Serial Flash devices by writing into the Default Memory Register the Serial Flash device number to communicate with. Every time the host wants to change the selected device there must be a write into the Default Memory Register. These 8 devices can be any of the Serial Flash devices supported by the SPI-MEM-CTRL core. Each time there is a write into the Default Memory Register the core identifies the Serial Flash device attached.

The core automatically identifies the Serial Flash device by reading the Device identification, JEDEC information (Manufacture ID, Memory Type, Memory Density). There are some Serial Flash devices that don't support device identification. The core provides manual identification in order to support these Serial Flash devices.

If a device supports reading/writing instructions using quad mode, the host must first put the Serial Flash device in Quad mode respectively. This is not standardized and depends on the Serial Flash device manufacturer. The host should use the Custom Instruction Registers to send the respective commands as specified by the Serial Flash device datasheet.

Once the Serial Flash device is put in Quad mode the SPI-MEM-CTRL core can execute Read and Write Request using Quad Instructions.

If the host needs to write or erase a part of the Serial Flash device then the host must first make sure that the corresponding parts are not block protected. At that case the host must send custom instructions (using the Custom Instruction Registers) in order to disable the Block Protection.

4.4 Interfaces Specification

4.4.1 Global inputs, Clocks, and Reset

The SPI-MEM-CTRL core receives two clock signals via the **core_clk** and **spi_2sclk** ports. All registers in the design are positive edge triggered D-type flip-flops and the design does not contain any latches. In the cases of bidirectional ports, the tri-state buffers are separated from the rest of the core and they are instantiated in a top-level wrapper.

Two asynchronous reset lines the **rst_core_clk** and **rst_spi_2sclk** are used throughout the design one for each clock domains. The **rst_core_clk** and **rst_spi_2sclk** can be asserted at any time.

A synchronous reset line, the **clr**, is also provisioned. This signal is sampled with the **core_clk** and there are internal metastability registers to synchronize it with the **spi_2sclk**. It must stay high for at least three clock cycles of the slowest between the two clocks.

The behavior of the **rst_core_clk**, **rst_spi_2sclk** and **clr** ports can be configured before synthesis. The active level of the **rst_core_clk** and

rst_spi_2sclk signals is configurable, as well as whether the core will use the asynchronous reset or the synchronous clear or both.

4.4.2 The Host Interface

Data read from the memory or status registers are read from the core by placing the address of the respective register on the **host_addr** port masked with the **host_re** signal. The core returns the data requested on the **host_rdata** port, masked with the **host_rdata_val** signal.

The data to be written to the memory or control registers are fed to the core through the **host_wdata** port and written to the internal register selected using the **host_addr** port. Valid data must be masked with the **host_we** signal. Figure 4-3 provides the timing diagram showing the operation of the Data interface.

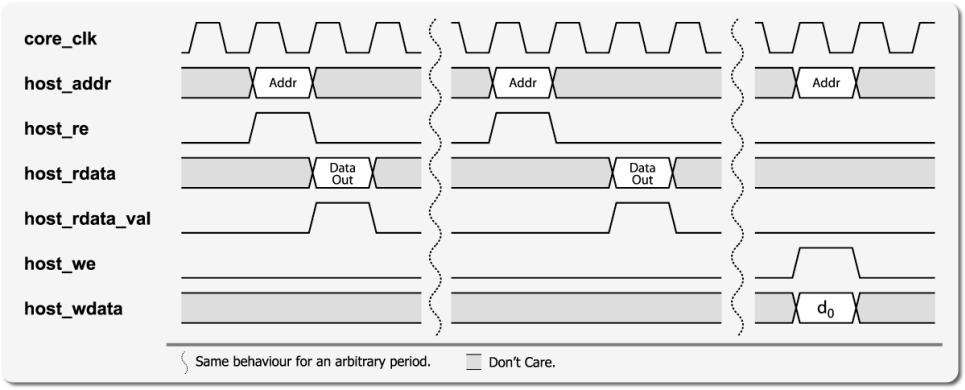

Figure 4-3: Data Input / Output Interface

4.4.3 The Status Interface

The **status_data_out_av** status port is set when the number of words in the Read-FIFO is greater than **Read-FIFO Threshold**. This flag indicates that there is enough data in the Read-FIFO to be read. The **status_data_out_av** status flag is reset when the number of words in Read-FIFO is equal or lower than **Read-FIFO Threshold**. When a read request is finished the

data_out_av status port stays high for as long as there are words in the Read FIFO.

The **status_data_in_rdy** status port is set when the number of words in the Write-FIFO is lower than **Write-FIFO Threshold**. This flag indicates that there is enough empty space in the Write-FIFO for the host to write data. The **status_data_in_rdy** status flag resets when the number of words in Write-FIFO is equal or greater than the **Write-FIFO Threshold**. See section 4.5.1 about Control Register.

The **status_DPM** status port is set immediately after the core starts serving the DPM request. It stays high while the memory enters DPM (**duration_enter_DPM** × 256 × $Period_{core\_clk}$). When the host requests the core to exit the memory from DPM the **status_DPM** status flag is still high. It stays high until the memory exits DPM (**duration_exit_DPM** × 256 × $Period_{core\_clk}$). The **status_DPM** status flag resets when the memory has leaved DPM and is in standby mode. See section 4.5.4 about duration DPM Register.

The **status_request_rdy** status port is reset when an access request is in progress or when the **status_DPM** status flag is set. It is set when an access request is served and the system is not in DPM.

The **status_interrupt** status port is set when one of the status signals enabled in the Interrupt Mask Register is triggered. It resets when all interrupt signals inside the Interrupt Request Register are cleared.

The Status flags are synchronous to the **core_clk**.

4.4.4 The Block Read Interface (BRI)

The Block Read Interface signals behave as shown on Figure 4-4. When the core is ready to accept requests on the Block Read Interface, the **bri_rqst_rdy** signal is active. The core will recognize Block Read requests by the **bri_rqst_val** signal, which masks valid parameters for the Block Read

request, on the **bri_rqst_addr**, **bri_rqst_count** and **bri_dest_offset** inputs.

After a request is received, the **bri_rqst_rdy** signal will be de-asserted and the core will start the Block Reading process. Data read from the Serial Flash is produced to the **bri_dout** port, masked with the **bri_dout_val** signal. The **bri_dout_addr** port holds the respective destination address for the current data on **bri_dout**, and can be used to write the data to the host system's memory space, or can be ignored.

Depending on the relation between **core_clk** and **spi_2sclk**, data can be sent to the host in a faster rate than they are read from the Serial Flash memory. In this case, the **bri_dout_val** signal will not stay constantly to logic 1, as the core will pause the data output stream until the internal FIFO is filled with enough data from the Serial Flash memory.

The host can also pause the output data stream, by de-asserting the **bri_dout_rdy** input until it is ready to receive more data.

The Block Read operation cannot be aborted; the data block requested must be entirely read before the Block Read operation finishes.

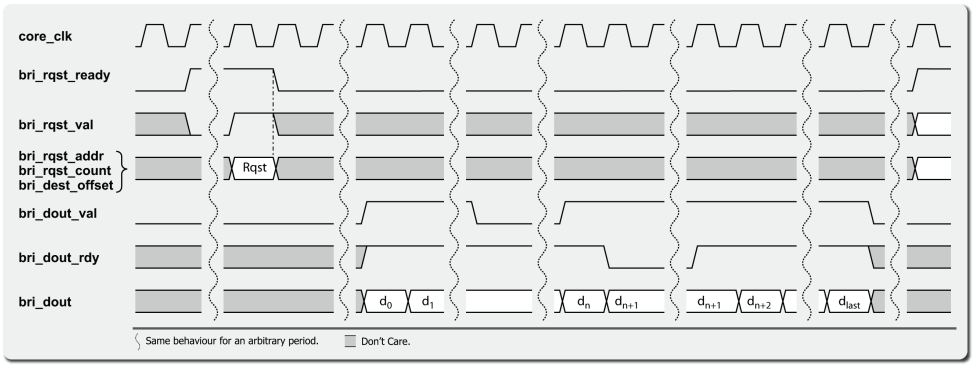

Figure 4-4: Block Read Interface

4.4.5 Block Read transfer on Power-up

If the core after a reset / clear detects the **bri_startup_xfer** port to be active, then the automatic Block Read transfer on startup is initiated. The **bri_rqst_rdy** port will stay low until the startup Block Read transfer operation ends.

The parameters for the startup transfer (**bri_rqst_addr**, **bri_rqst_count**, **bri_dest_offset**) are defined in the configuration memory. When the **bri_rqst_rdy** port reasserts the Serial Flash device is in DPM and the host has complete access to the core via the host_interface.

During the startup Block Read operation the core is automatically programmed for the Serial Flash parameters. These default parameters are defined in the configuration memory.

In case the host needs the core to use Quad instructions for Read and Write, it is necessary to make sure that before startup the Serial Flash device is already in Quad mode. The SPI-MEM-CTRL core will not put the Serial Flash device in Quad mode automatically as this is a non-volatile operation in many Serial Flash devices. The Serial Flash device can be manually put in Quad mode using custom command feature of the core.

4.4.6 The Execute On-The-Fly (XOTF) Interface

The Execute On-The-Fly (XOTF) interface signals behave as shown on Figure 4-5.

The host requests the data stored into the **xotf_raddr** address of the serial flash device by asserting the **xotf_ren** signal. The core asserts the **xotf_ren_rdy** output pin to indicate that it accepted the current request masked by the **xotf_ren** input signal.

Available output data on the **xotf_dout** output pins are masked with the **xotf_dout_val** output pin. The host uses the **xotf_dout_rdy** input pin to signal that the masked data was accepted.

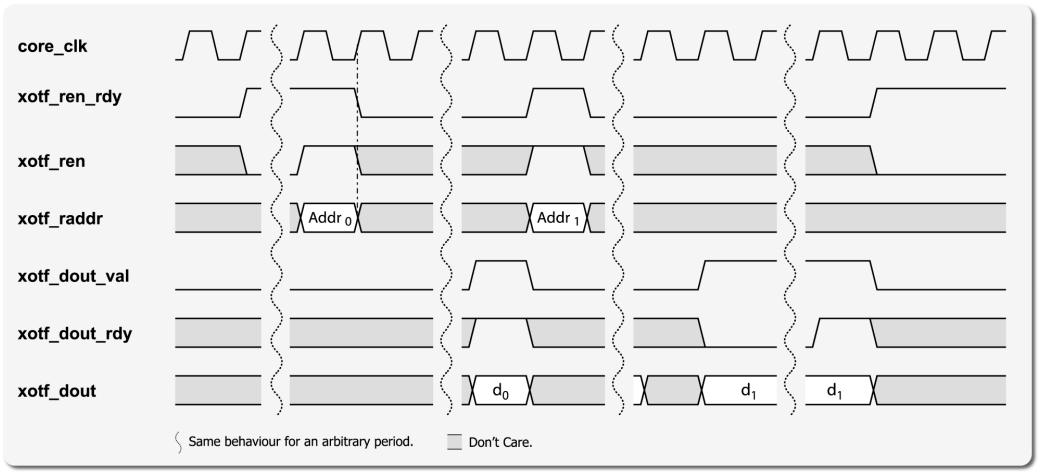

Figure 4-5: The XOTF Interface

4.4.7 Supported SPI Bus Waveforms

There are many different SPI bus protocol variations. The master on a SPI bus has to generate a serial clock. Besides setting the clock frequency it must also configure the clock polarity and phase with respect to the data. The two options which define clock polarity and phase are CPOL and CPHA respectively.

At CPOL = 0 the clock base is 0. For CPHA = 0 data are read on the clock's rising edge and data are changed on a falling edge (high->low clock transition). For CPHA = 1, data are read on the clock's falling edge and data are changed on a rising edge.

At CPOL=1 the base value of the clock is 1. For CPHA = 0, data are read on clock's falling edge and data are changed on a rising edge. For CPHA=1, data are read on clock's rising edge and data are changed on a falling edge.

The memories that use the SPI bus, support two modes (CPOL = 0, CPHA = 0 and CPOL = 1, CPHA = 1). These two modes have similar behavior which means that serial data are sampled on the rising edge of **`spi_sclk`** and serial data are shifted out on falling edge. The difference between these two modes is the initial value of the **`spi_sclk`**. SPI-MEM-CTRL core supports both modes and the host

can choose between them by configuring the Control Register (see section 4.5.1). Figure 4-6 presents the supported SPI bus waveforms.

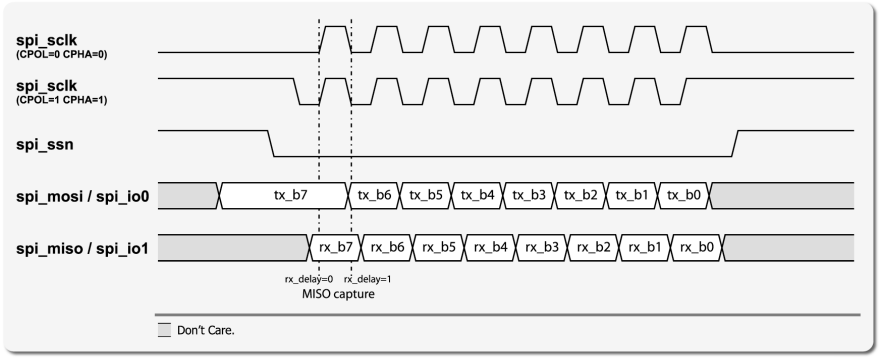

Figure 4-6: SPI Waveforms

4.5 Core Programming

The core is programmed via addressable registers. Table 4-2 shows the addressing of the registers of the core and a short description for quick reference.

Table 4-2: Register Address Map

Address	Name	Access Type	Description
00h	Control reg.	R/W	Core's parameters
01h	Status reg.	R	Information about core's status
02h	Access Request reg. 0	R/W	Access Request, word 0
03h	Access Request reg. 1	R/W	Access Request, word 1
04h	Access Request reg. 2	R/W	Access Request, word 2
05h	Duration DPM reg.	R/W	Set the duration required to enter/exit DPM
06h	Read/Write Data reg.	R/W	Read data when read, write data when written
07h	FIFOs Status reg.	R	The number of items in Read and Write FIFOs
08h	Default Memory reg.	W	Default Attributes of the Serial Flash device
09h	Extended Addressing Mode reg.	R/W	Set the instruction required to enter 32-bit SPI addressing mode
0Ah	Memory Specification reg.	R	3 Byte specification: 1 byte manufacture ID, 1–byte memory type, 1-byte density
0Bh	Interrupt Mask reg.	R/W	Interrupts enable disable mask
0Ch	Interrupt Request reg.	R/W	Reason for triggering interrupt signal
0Dh	Custom Instruction Setup reg.	R/W	Custom Instruction, Setup Register.
0Eh	Custom Instruction Data reg. 0	R/W	Custom Instruction, Data I/O Register 0
0Fh	Custom Instruction Data reg. 1	R/W	Custom Instruction, Data I/O Register 1
10h	Read Dummy Cycles reg.	R/W	Number of dummy cycles for Read Instructions
11h	Extended Configuration reg.	R/W	Extended configuration options for spi instructions

4.5.1 Control Register (00h R/W)

In this register the **Clk_delay**, **Read-FIFO Threshold**, **Write-FIFO Threshold**, **Enter_DPM**, **SPI mode**, **Soft_Reset** and **Clk_divisor** values are programmed. Table 4-3 describes the Control Register's fields.

Table 4-3: Control Register

Bit	Field	Description	Reset Value
7:0	**Clk_delay**	The **Clk_delay** value. This is used to specify the tSHSL, tWHSL and tSHWL times of the memory. **Clk_delay** × $Period_{core\_clk}$ = max(tSHSL, tWHSL, tSHWL)	80h[1]
15:8	**Read-FIFO Threshold**	The **status_data_out_av** port is set if the number of words in Read-FIFO is greater than this value	04h[1]
23:16	**Write-FIFO Threshold**	The **status_data_in_rdy** pin is set if the number of words in Write-FIFO is lower than this value	04h[1]
24	**Enter_DPM**	1: Enter DPM 0: Exit DPM	0
25	**SPI_Mode**	1: CPOL = 1, CPHA = 1 0: CPOL = 0, CPHA = 0	0[1]
26	**Soft_Reset**	1: Software reset	0
27	**Mem_type_nand**	Select operation mode depending on connected memory type: 0: enable SPI **NOR** Flash device mode 1: enable SPI **NAND** Flash device mode	0[1]
31:28	**Clk_divisor**	Clock divisor value. This is used to select the **spi_sclk** clock period. $Period_{spi\_sclk}$ = $Period_{spi\_2sclk}$ × 2 × (CDV+1)	0h[1]

Table Notes:
[1] Synthesis Parameter: Configurable during synthesis

4.5.1.1 Clk_delay field

The value of the **Clk_Delay** is used to specify to the core the number of **core_clk** clock cycles needed to meet the memory timing constraints of tSHSL, tWHSL and tSHSW.

For every memory there is a minimum amount of time that the Slave Select pin must stay high. This timing constraint is set by the tSHSL value. The most Serial Flash devices have a write protect pin. For this pin there are timing constraints

about setup and hold time. These constraints are set by the tWHSL and tSHWL. Figure 4-7 illustrates these timing constraints.

So the host must write into **Clk_delay** field of the control register the proper value that meets the following equation:

Clk_delay × $Period_{core\_clk}$ ≥ max(tSHSL, tWHSL, tSHWL)

For example, if the **core_clk** is 100MHz, and the tSHSL, tWHSL, tSHWL are 100ns, 20ns and 100ns respectively Any **Clk_delay** value equal or greater than 0Ahex meets the equation **Clk_delay** × 10ns ≥ max(100ns, 20ns, 100ns)

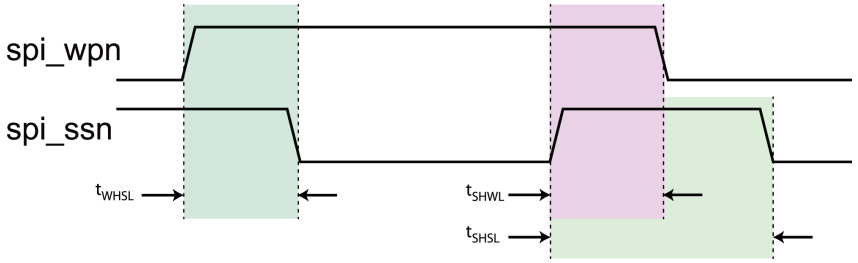

Figure 4-7: tWHSL, tSHWL, tSHSL timing constraints

Important Note: The signals **spi_wpn** and **spi_ssn** are synchronous to the **spi_2sclk** clock domain, so the actual delay will be rounded-up and the transition of **spi_wpn** or **spi_ssn** will happen at the next rising edge of **spi_2sclk** after the programmed delay.

4.5.1.2 Read-FIFO Threshold field

The **Read-FIFO Threshold** value defines the number of 32bits words stored in the Read-FIFO, beyond which the **status_data_out_av** status flag is set. That is the case when the core asks the host to take out of the Read-FIFO some data. The reset value is configurable during synthesis as the half of the FIFO size. The **Read-FIFO Threshold** must be [4, 2RFIFO_ADDR_BITS-1].

4.5.1.3 Write-FIFO Threshold field

The **`Write-FIFO Threshold`** value defines the number of 32bits words in the Write-FIFO, below which the **`status_data_in_rdy`** status flag is set. That is the case when the core asks the host to write new data to the Write-FIFO. The reset value is configurable during synthesis as the half of the FIFO size. The **`Write-FIFO Threshold`** must be [2, 2RFIFO_ADDR_BITS-2].

4.5.1.4 Enter_DPM field

The **`Enter_DPM`** control bit is used to force the memory to enter/exit Deep Power-down Mode (DPM). To enter DPM the host sets the **`Enter_DPM`** control bit. If the **`request_rdy`** status bit is low (there is an access request in progress) the enter DPM command is pending. When the **`request_rdy`** status bit is set, the core initiates the process of 'enter in DPM' command: first the core resets the **`request_rdy`** status bit, then it sends the enter DPM instruction to the memory and finally waits for **`duration_enter_DPM`** × 256 × $\text{Period}_{\texttt{core\_clk}}$. When this time expires the core rechecks the **`Enter_DPM`** control bit. If this bit is low the core starts processing the exit DPM command: first it sends the memory a Release from power-down instruction, then it waits for **`duration_exit_DPM`** × 256 × $\text{Period}_{\texttt{core\_clk}}$. When this time expires the **`request_rdy`** status bit is reset, so the core is available to get a new access request. The **`status_DPM`** status bit is set at the beginning of the processing of the enter DPM command and it is reset at the end of the processing of the exit DPM command.

Note: For more details about status bits, Access Request Registers, **`duration_enter_DPM`** or **`duration_exit_DPM`** please see sections 0, 4.5.3 and 4.5.4 respectively.

4.5.1.5 SPI_Mode field

The **`SPI_Mode`** defines the polarity and the phase of the **`spi_sclk`**. If **`SPI_Mode`** is high then CPOL = 1 and CPHA = 1, else CPOL = 0 and CPHA = 0.

When there is a change on **SPI_Mode** the host must wait at least two **spi_2sclk** clock cycles before entering any new access request.

4.5.1.6 Soft_Reset field

The **Soft_Reset** is synchronous software reset. The host has to write a one to this bit and then manually deassert it by writing a zero. The software reset function is implemented only if the core is configured with a synchronous clear enabled.

4.5.1.7 Mem_type_nand field

The **Mem_type_nand** field is used to set the operation mode of the SPI-MEM-CTRL core. When programmed to '0' the NOR mode is enabled. When programmed to '1' the NAND mode is enabled. The initial value of this field after reset or after power-up, is controlled by a synthesis parameter in the configuration package file. The SPI-MEM-CTRL core cannot identify the type (NOR or NAND) of the connected SPI Flash device, so this field is required to select the type of the connected memory. This field can also be set using the Configuration Memory **BRI Control Register**, for BRI startup transfers.

4.5.1.8 Clk_divisor field

The value of the **Clk_divisor** is used to divide the **spi_2sclk**'s clock period, to create the SPI serial clock (**spi_sclk**). The period of the generated **spi_sclk** is given by the following formula:

$$Period_{spi\_sclk} = Period_{spi\_2sclk} \times 2 \times (CDV+1)$$

For example, if the **spi_2sclk** is 100MHz, and the **Clk_divisor** value programmed is 5, the generated **spi_sclk** will be 100/[2 × (5+1)]= 8.33MHz, giving a bitrate of 8.33Mbps

4.5.2 Status Register (01h R)

Reading from this address, returns the contents of the Status Register. Refer to Table 4-4 for the bit-mapping of the status flags.

Table 4-4: Status Register

Bit	Field	Description	Reset Value
0	**data_out_av**	1: The host can read new data from the memory	0
1	**data_in_rdy**	1: The host can write new data to the memory	1
2	**status_DPM**	1: In Deep Power-down Mode	0
3	**request_rdy**	1: Enter new access request	0
4	**fread_busy**	1: FREAD in progress	0
7:5	**Reserved**	Reserved for future use	000
23:8	**Reserved**	Reserved for future use	0000h
31:24	**device_sr**	Current value of Serial Flash device's Status Register	00h

The **data_out_av** status bit is set when the number of words in Read-FIFO is greater than the **Read-FIFO Threshold**. This flag signals the host when there is enough data in the Read-FIFO to be read. The **data_out_av** status bit is reset when the number of words in Read-FIFO is equal or lower than **Read-FIFO Threshold**. When a read request is finished the **data_out_av** stays high as long as there are words in the Read FIFO.

The **data_in_rdy** status bit is set when the number of empty words in Write-FIFO is greater than the **Write-FIFO Threshold**. This flag signals the host when there is enough empty space in the Write-FIFO for the host to write data. The **data_in_rdy** status bit is reset when the number of empty words in Write-FIFO is equal or lower **Write-FIFO Threshold**.

The value of the **status_DPM** status bit identifies whether or not the memory is in DPM. It is set immediately after the core starts serving the DPM request. It

stays high while the memory enters DPM (**duration_enter_DPM** × 256 × $Period_{core\_clk}$). When the host requests the core to force the memory out of DPM the **status_DPM** status bit is still high. It stays high while the memory exits DPM (**duration_exit_DPM** × 256 × $Period_{core\_clk}$). The **status_DPM** status bit resets when the memory has exit DPM and is in standby mode. See section 4.5.4 about duration DPM Register.

The **request_rdy** status bit defines the case when the core is available to serve one new access request. The **request_rdy** status bit is de-asserted when there is another access request still in progress or the memory enters/is/exits DPM. The **request_rdy** status bit is asserted when the system is not in DPM (**status_DPM** = 0) and there is no access request in progress. As long as the **request_rdy** is low the core ignores any write into the Access Request Registers addresses 2 and 3 (see section 4.5.3).

The **fread_busy** status bit indicates that a Full READ access request is in progress. It is asserted when a Full READ access request is accepted and deasserted when a fread_break signal is sent to the core (Access Request Register 2, bit 4) and the operation is complete (see section 4.5.3).

The **device_sr** status bits include the current value of the Status Register of the Serial Flash device. When a device has two bytes status register the **device_sr** includes the value of the low byte. The SPI-MEM-CTRL core sends to the Serial Flash device the appropriate instruction to read its Status Register. Every time this takes place the **device_sr** status bits are updated.

4.5.3 Access Request Registers 0, 1, 2 (02h, 03h, 04h R/W)

These three registers include the memory access request parameters that define the access request. Refer to Table 4-5, Table 4-6 and Table 4-7 for the bit mapping.

When the **request_rdy** status bit is high (i.e. no access request is in progress, the system is not in DPM, no BRI procedure is in progress and no custom instruction is in progress), the host can write into the Access Request Registers and define the access request parameters. The host must first write the first word (addr 2), then the second word (addr 3) and finaly the third (addr 4). A new access request is issued every time the third word (addr 4) is written. Then the **request_rdy** status bit is reset. At that case any further write in the Access Request Registers will be ignored until the access request is served.

Table 4-5: Access Request Register 0

Bit	Field	Description	Reset Value
31:0	**Request_Offset**	The first address of the access request.	00000000h

The **Request_Offset** contains the starting address for a READ or WRITE access request. If the access request is a WRITE or if the SPI NAND Flash mode is enabled, the **Request_Offset** must contain any address that the two least significant bits are both 0 i.e. the **Request_Offset** must be multiple of 4. If those two bits are not 0 then the write into memory must not exceed page boundaries. If the access request is an ERASE the **Request_Offset** must contain any address within the sector/block/chip that the host wants to be erased.

Table 4-6: Access Request Register 1

Bit	Field	Description	Reset Value
31:0	**Request_Length**	The length of the access request. For SPI NOR Flash devices, during Erase, bits (31:2) are ignored and bits (1:0) = 00: Sector Erase (4KB) = 01: Block Erase (64KB) = 10: Chip Erase For SPI NAND Devices, the value of this register must be zero, and Block Erase is always performed.	00000000h

The **Request_Length** contains the number of consecutive bytes that the host wants to be read or written in the READ or WRITE access request respectively. The host can read and write any number of bytes that is multiple of 4. If, for example, the **Request_Length** is X with the two least significant bits of X not 0, then the core will read or write floor(X/4) bytes. Most Serial NOR Flash devices support three types of memory erase: section erase (4KB), block (64KB) or chip/bulk erase (whole chip). The two least significant bits of the **Request_Length** define these three types of erasure. The rest 30 most significant bits must all be 0. If the 30 most significant are not all 0 the erase access request will be ignored. For serial NAND Flash devices only Block Erase is supported and the **Request_Length** must be zero.

Table 4-7: Access Request Register 2

Bit	Field	Description	Reset Value
3:0	**Request_Type**	0000 : READ Request. 0001 : WRITE Request. 0010 : ERASE Request. 0011 : FREAD Request. 0100 : ECCREAD Request for NAND page ECC data. 0101 : ECCWRITE Request for NAND page ECC data. 0110 - 1111 : Reserved	0h
4	**Break_Fread**	1 : Break FREAD Request	0
5	**PP_Page_Size**	0: 256-bytes per page size 1: 512-bytes per page size	0
7:6	Reserved	Reserved for future use.	00
31:8	Reserved	Reserved for future use.	000000h

The **Request_Type** bits define the type of the access request (READ, WRITE, ERASE, FREAD, ECCREAD, ECCWRITE).

It must be noted that during a read access address will not wrap around page or sector boundaries but to the end of the address space. In the same way, during a write access address will not wrap around page, sector or chip boundaries. Unpredicted results will occur if the write address exceeds the address space.

When operating in SPI NAND mode, ECC data at the end of the pages are not accessible with the READ and WRITE requests. So, the addresses for READ and WRITE requests must be in the following format:

Table 4-8: NAND mode addressing

Block Address																Page Address						Byte in page (0 - 2047)											
A_{31}	A_{30}	A_{29}	A_{28}	A_{27}	A_{26}	A_{25}	A_{24}	A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0

ECCREAD and ECCWRITE Requests are issued in order to read or write the extra data at the end of **one** selected page, to be used for error detection purposes. These data are needed to implement bad block detection on SPI NAND Flash devices. A zero **Request_Offset** points to the first extra byte in one page (column address = 2048 + **Request_Offset**). The **Request_Length** must not exceed the size of extra data area for one page.

FREAD Request is issued in order to continuously read the Serial Flash Device starting from address **Request_Offset**. The FREAD will be terminated when the host sets the **Break_Fread** bit to 1.

The **PP_Page_Size** field is used to set the page size for the PP, PP2O, PP4O and PP4IO write commands. For SPI NOR Flash devices, to set the page size to 512 this field must be programmed to 1. This field is ignored for SPI NAND Flash devices where page size in most cases is fixed to 2048 + Extra.

Note:
Although the host cannot write into Access Request Registers when the **request_rdy** output pin is 1, it is permitted to write the **Break_Fread** bit of the Access Request Register 2 when the FREAD Access Request is in progress. This bit is self cleared.

4.5.4 Duration DPM Register for SPI NOR devices (05h R/W)

This register contains the values of the two parameters **duration_enter_DPM** and **duration_exit_DPM**. The memory needs **duration_enter_DPM** × 256 × $Period_{core\_clk}$ and **duration_exit_DPM** × 256 × $Period_{core\_clk}$ to enter or exit Deep Power-down Mode respectively. During this time the core ignores any access request. Refer to Table 4-9 for the bit-mapping.

Table 4-9: Duration DPM Register

Bit	Field	Description	Reset Value
15:0	**duration_enter_DPM**	Number Of **core_clk** Clock Cycles needed by the memory to enter DPM	FFFFh
31:16	**duration_exit_DPM**	Number Of **core_clk** Clock Cycles needed by the memory to exit DPM	FFFFh

4.5.5 Read/Write Data Register (06h R/W)

A read from address 06h returns to **host_rdata** a word read from the memory. Each word contains 4 bytes read from the memory. Data from the lowest address goes on the least significant byte of the register. If the access request is a read

for less than 4 bytes the core will read 4 consecutive bytes. A write to address 6 adds the word on **host_wdata** in the Write-FIFO. Data on the least significant byte of the register will be written on the lowest address.

4.5.6 FIFOs Status Register (07h R)

This register contains the number of words currently stored in the Read and Write FIFOs. Refer to Table 4-10 for bit-mapping.

Table 4-10: Fifos Status Register

Bit	Field	Description	Reset Value
15:0	**read_fifo_status**	Number of 32 bit words in Read FIFO	0000h
31:16	**write_fifo_status**	Number of 32 bit words in Write FIFO	0000h

4.5.7 Default Memory Register (08h W)

This register contains the default values of the attributes that the attached Serial Flash device supports. After power-up when the core is in idle state, the core waits for a write access in the Default Memory Register. Then it communicates with the Serial Flash device and tries to identify it. If this memory is not one of the memories listed in Table 6-3, then the core gets the values that the host has written to the Default Memory register. Refer to Table 4-11 for bit-mapping.

Note that it is not mandatory to program the Default Memory Register before requesting a Block Read transfer via the Block Read Interface. In this case, the core will try to identify if the memory is one of the ones listed in Table 6-3. If the memory is not identified, the core will be programmed with the default values for the Default Memory Register given in the configuration memory. This would be also the case for a startup Block Read transfer.

Writing to the Default Memory Register also selects which of the 8 slave select lines will be active. The host can select a different Serial Flash device by re-writing the Default Memory Register. The first write into the Default Memory Register must take place in the beginning. Any other write must take place only

when the **status_request_rdy** port is high. Otherwise the write operation in the Default Memory Register will be ignored. Every time there is a valid write into the Default Memory Register, the SPI-MEM-CTRL core communicates with the Serial Flash and tries to identify it.

Important: After any write into the Default Memory register the host must wait at least 4 core_clk clock cycles before writing any other register.

The SPI-MEM-CTRL core provides the host the ability to deactivate the automatic identification of the Serial Flash device by setting the appropriate bit.

Table 4-11: Default Memory Register

Bit	Field	Description	Reset Value
2:0	**def_mem_read_opcode**	000: FAST_READ (opcode 0Bh) 001: READ2O (opcode 3Bh) 010: READ2IO (opcode BBh) 011: READ4O (opcode 6Bh) 100: READ4IO (opcode EBh) 101 - 111: Reserved	000
5:3	**def_mem_write_opcode**	**SPI NOR** Flash mode: 000: PP (opcode 02h) 001: PP2O (opcode A2h) 010: PP4O (opcode 32h) 011: PP4IO (opcode 38h) 100 - 111: Reserved **SPI NAND** Flash mode: 000: PLRDx1 (opcode 84h) 001: PLRDx4 (opcode 34h)	000
6	**def_mem_attr_32**	0: 24-bits address 1: 32-bits address (SPI NOR Flash only)	0
7	**def_mem_attr_dpm**	0: Memory does not support DPM 1: Memory supports DPM (SPI NOR Flash only)	0
10:8	**def_mem_chip_select**	Active Slave Select Line, e.g. 000: Activates Device on Slave Select line 0 101: Activates Device on Slave Select line 5	000
11	**def_mem_auto_id**	0: Manual Identification 1: Automatic Identification	1
31:12	Reserved	Reserved for future use	00000h

4.5.7.1 def_mem_read_opcode field

The **def_mem_read_opcode** contains the default value for the instruction that will be used for READ accesses. If the host chooses the READ4O or READ4IO instruction for READ accesses (Quad mode accesses), it is necessary to ensure that the Serial Flash device is in Quad mode. The Serial Flash device can be set to Quad mode by sending custom instructions (using the Custom Instruction Registers).

4.5.7.2 def_mem_write_opcode filed

The **def_mem_write_opcode** contains the default value for the instruction that will be used for WRITE accesses. If the host chooses the PP4O or PP4IO instruction for WRITE accesses (Quad mode accesses), it is necessary to ensure that the Serial Flash device is in Quad mode. The Serial Flash device can be set to Quad mode by sending custom instructions (using the Custom Instruction Registers).

4.5.7.3 def_mem_attr_32 field (for SPI NOR Flash devices)

The SPI-MEM-CTRL core supports Serial NOR Flash devices with densities greater than 128Mbits. So Read, Write and Erase instructions with 32 address bits can be executed. The default instructions use 24 address bits. If the attached Serial Flash device has a density greater than 128Mbits the **def_mem_attr_32** must be set. On other case, density lower or equal to 128Mbits, **def_mem_attr_32** must be unset.

4.5.7.4 def_mem_attr_dpm field (for SPI NOR Flash devices)

The **def_mem_attr_dpm** defines whether or not the attached serial NOR flash supports instruction for entering/exiting Deep Power-down Mode.

4.5.7.5 def_mem_chip_select field

The **def_mem_chip_select** defines which of the 8 Slave Select lines will be used. Each time that the host needs to access a different Serial Flash device must

write into Default Memory Register. Every slave select line that is not active has always the value 1.

4.5.7.6 def_mem_auto_id field

The SPI-MEM-CTRL core can automatically identify what Serial Flash device is attached to it by reading the JEDEC id of the device and matching it against the list of device entries provided in the configuration ROM (See 4.6.10). If the **def_mem_auto_id** is 1 the SPI-MEM-CTRL core communicates with the Serial Flash device and tries to identify it by reading its JEDEC id. If the attached device is not found in the list the controller will use the values that the host has programmed in the first four fields of the Default Memory Register. Table 6-3 lists the Serial Flash devices which are the default device entries present inside the configuration ROM (See 4.6.10).

The host can override the automatic identification and manually select the device attributes by resetting the **def_mem_auto_id**. If the **def_mem_auto_id** is 0 the controller uses the values that the host has programmed in the first four fields of the Default Memory Register.

4.5.8 Extended Addressing Mode Register for SPI NOR devices (09h R/W)

Most Serial NOR Flash devices support 24 address bits which limits their density to 128Mbits. Some manufacturers propose Serial Flash devices with 32 address bits with densities greater than 128Mbits. The SPI-MEM-CTRL core by default supports 24 address bits. In order to support densities greater than 128Mbits an instruction that sets the Serial Flash device to 32 address bits mode must be used.

If the attached Serial Flash device is detected to have mem_attr_32 equal to 1 (i.e. 32 address bits , for more details please see section 4.5.7) the SPI-MEM-CTRL core checks the Extended Addressing Mode Register and sends the appropriate instruction to the Serial Flash device in order to enable 32 address bits mode.

Table 4-12: Extended Addressing Mode Register

Bit	Field	Description	Reset Value
7:0	**ext_am_opcode**	Set the opcode that enters the 32-bit addressing mode	B7h[1]
15:8	**ext_am_byte0**	Byte 0 that follows the opcode	00h
23:16	**ext_am_byte1**	Byte 1 that follows Byte 0	00h
25:24	**ext_am_mode**	00: don't send any instruction 01: send opcode 10: send opcode, byte0 11: send opcode, byte0, byte1	00
26	**ext_am_check_wip**	1= Wait for WIP=0 before issuing instruction	0
27	**ext_am_wren**	Send WREN before instruction	0
31:28	Reserved	Reserved for future use	0h

Table Notes:
[1] Configurable during synthesis

4.5.8.1 ext_am_opcode field

The **ext_am_opcode** contains the opcode of the instruction that must be sent to the Serial Flash device in order to enable the 32 address bits.

4.5.8.2 ext_am_byte0 field

The **ext_am_byte0** contains the byte0 of the instruction that must be sent to the Serial Flash device in order to enable the 32 address bits.

4.5.8.3 ext_am_byte1 field

The **ext_am_byte1** contains the byte1 of the instruction that must be sent to the Serial Flash device in order to enable the 32 address bits.

4.5.8.4 ext_am_mode field

The **ext_am_mode** defines the number of bytes that determine the instruction that must be sent to the Serial Flash device in order to enable the 32 address bits. If the **ext_am_mode** has the value of 00 then no instruction is sent to the Serial Flash device. If the **ext_am_mode** has the value of 01 the instruction that will be sent to the Serial Flash device consists of 8 bits: the **ext_am_opcode**. If

the **ext_am_mode** has the value of 10 the instruction that will be sent to the Serial Flash device consists of 16 bits: the **ext_am_opcode** followed by the **ext_am_byte0**. If the **ext_am_mode** has the value of 11 the instruction that will be sent to the Serial Flash device consists of 24 bits: the **ext_am_opcode** followed by the **ext_am_byte0** and the **ext_am_byte1**.

4.5.8.5 ext_am_check_wip field

If the **ext_am_check_wip** bit is set to 1, the SPI-MEM-CTRL will wait until the WIP bit of the Serial Flash device's status register is 0. Then the instruction that sets the device to 32-bit addressing mode is issued.

4.5.8.6 ext_am_wren field

If the **ext_am_wren** bit is set to 1, the SPI-MEM-CTRL will send a WREN (Write Enable opcode 06h) instruction before sending the instruction that sets the device to 32-bit addressing mode.

4.5.9 Memory Specification register (0Ah R)

If the SPI-MEM-CTRL core is programmed to automatically detect what is the attached Serial Flash device a read identification id instruction (RDID) is sent. This instruction returns 3 bytes which consist of the JEDEC specifications of the Serial Flash device. This information is stored in the Memory Specification Register. Refer to Table 4-13 for bit mapping.

Table 4-13: Memory Specification Register

Bit	Field	Description	Reset Value
7:0	**memspecs_dev_cap**	Memory capacity	00h
15:8	**memspecs_dev_type**	Memory type	00h
23:16	**memspecs_man_id**	Manufacture ID	00h
31:24	Reserved	Reserved for future use	00h

4.5.10 The Interrupt Mask register (0Bh R/W)

The Interrupt Mask Register contains the enable values that trigger the **status_interrupt** signal. Refer to Table 4-14 for bit mapping. When the host writes a value into the Interrupt Mask Register the SPI-MEM-CTRL core set the **status_interrupt** signal whenever one or more of the enabled signals are set. When **status_interrupt** signal is set and the host reads the Interrupt Mask Register the **status_interrupt** signal resets.

Table 4-14: Interrupt Mask Register

Bit	Field	Description	Reset Value
0	**imr_data_out_av**	1: Enable Interrupt on **status_data_out_av** = 1	0
1	**imr_data_in_rdy**	1: Enable Interrupt on **status_data_in_rdy** = 1	0
2	**imr_request_rdy**	1: Enable Interrupt on **status_request_rdy** = 1	0
3	Reserved	Reserved for future use	0
31:4	Reserved	Reserved for future use	0000000h

4.5.11 The Interrupt Request Register (0Ch R/W)

A read to the Interrupt Request Register will return the status of the core's interrupt requests. It is used to determine the reason an interrupt was generated. After an interrupt has been serviced, the respective interrupt request is cleared by writing the corresponding clear bit.

Table 4-15: Interrupt Request Register

Bit	Field	Description	Reset Value
0	`irr_data_out_av`	Read 1: Pending Interrupt Request on **`status_data_out_av`** Write 1: Clear Interrupt Request on **`status_data_out_av`**	0
1	`irr_data_in_rdy`	Read 1: Pending Interrupt Request on **`status_data_in_rdy`** Write 1: Clear Interrupt Request on **`status_data_in_rdy`**	0
2	`irr_request_rdy`	Read 1: Pending Interrupt Request on **`status_request_rdy`** Write 1: Clear Interrupt Request on **`status_request_rdy`**	0
3	Reserved	Reserved for future use	0
31:4	Reserved	Reserved for future use	0000000h

Custom Instruction Registers (0Dh, 0Eh, 0Fh R/W)

The SPI-MEM-CTRL core can issue a programmable custom instruction to the Serial Flash device. These three registers can be used to define the custom instruction to be sent to the Serial Flash device, and read its response. Please refer to Table 4-16,

Table 4-17 and Table 4-18 for bit-mappings.

When the **request_rdy** status bit is high (i.e. no access request is in progress, the system is not in DPM, no BRI procedure is in progress and no custom instruction is in progress), the host can write the Custom Instruction Registers and define the custom instruction parameters. The host must first write the data registers (addr 14, addr 15) and last the setup register (addr 13). A new custom instruction is set every time the setup register (addr 13) is written. Then the **request_rdy** status bit is reset. At that case any further write in the Custom Instruction Registers will be ignored until the custom instruction is complete.

When a custom instruction is complete the **request_rdy** status bit is set and the host can read the Custom Instruction Data Registers (addr 14, addr 15) to get the response of the Serial Flash device.

The Custom instruction mechanism can be used for example to set the Serial Flash device in the desired operating mode (e.g. to Quad mode, 24-bit addressing mode) or to use other device features such as bank protection.

Table 4-16: Custom Instruction Setup Register (addr13)

Bit	Field	Description	Reset Value
7:0	ci_opcode	Opcode of the Custom Instruction	00h
11:8	ci_length	The length of the Custom Instruction. 0000 : do not use 0001 : Send opcode 0010 : Send opcode, byte0 0011 : Send opcode, byte0, byte1 0100 : Send opcode, byte0 -> byte2 0101 : Send opcode, byte0 -> byte3 0110 : Send opcode, byte0 -> byte4 0111 : Send opcode, byte0 -> byte5 1000 : Send opcode, byte0 -> byte6 1001 : Send opcode, byte0 -> byte7 1010 - 1111 : do not use	0h
12	ci_spi_wpn	The value of the **spi_wpn_io2** pin during execution of Custom Instruction	0
13	ci_spi_holdn	The value of the **spi_holdn_io3** pin during execution of Custom Instruction	1
14	ci_check_wip	1= Wait for Write-in-Progress or for Operation-in-Progress before issuing Custom Instruction	0
15	ci_wren	Send WREN before Custom Instruction	0
16	ci_lf_en	Enable Long Frame mode	0
17	ci_lf_stop	Finalize Long Frame	0
19:18	Reserved	Reserved for future use	00
31:16	Reserved	Reserved for future use	0000h

Table 4-17: Custom Instruction Data Register 0 (addr 14)

Bit	Field	Description	Reset Value
7:0	ci_byte0	Byte 0 of the Custom Instruction	00h
15:8	ci_byte1	Byte 1 of the Custom Instruction	00h
23:16	ci_byte2	Byte 2 of the Custom Instruction	00h
31:24	ci_byte3	Byte 3 of the Custom Instruction	00h

Table 4-18: Custom Instruction Data Register 1 (addr 15)

Bit	Field	Description	Reset Value
7:0	ci_byte4	Byte 4 of the Custom Instruction	00h
15:8	ci_byte5	Byte 5 of the Custom Instruction	00h
23:16	ci_byte6	Byte 6 of the Custom Instruction	00h
31:24	ci_byte7	Byte 7 of the Custom Instruction	00h

4.5.11.1 ci_opcode field

The **ci_opcode** field is programmed with the opcode that will be transmitted with the Custom Instruction. When Long Frame mode is enabled the opcode will only be sent in the first Custom Instruction call that initiates an arbitrary length of Custom Instruction bytes. See **ci_lf_en** and **ci_lf_stop** for more details on how to send custom instruction of arbitrary byte length.

4.5.11.2 ci_length field

The **ci_length** field is programmed with the number of bytes that will be transferred on each Custom Instruction call. When Long Frame mode is enabled the **ci_opcode** will only be sent in the first Custom Instruction call that initiates an arbitrary length of Custom Instruction bytes. See **ci_lf_en** and **ci_lf_stop** fields for detailed description on how to send a custom instruction of arbitrary byte length.

4.5.11.3 ci_spi_wpn field

The **ci_spi_wpn** field forces a value to the **spi_wpn_io2** pin during execution of Custom Instruction.

4.5.11.4 ci_spi_holdn field

The **ci_spi_holdn** field forces a value to the **spi_holdn_io3** pin during execution of Custom Instruction.

4.5.11.5 ci_check_wip field

When the **ci_check_wip** field is set to 1, the SPI-MEM-CTRL will wait until the WIP bit of the Serial Flash device's status register is 0 for SPI NOR devices, or until the Operation-in-Progress bit of the status register is 0 for SPI NAND devices. Then the Custom Instruction is issued.

4.5.11.6 ci_wren field

When the **ci_wren** field is set to 1, the SPI-MEM-CTRL will send a WREN (Write Enable opcode 06h) instruction before sending the Custom Instruction.

4.5.11.7 ci_lf_en, ci_lf_stop fields

The **ci_lf_en** and **ci_lf_stop** fields control the operation of the Long Frame mode. The Long Frame mode is a mechanism that permits arbitrary byte length custom instructions. While in Long Frame mode a large custom instruction sequence is split in multiple Custom Instruction calls. Each Custom Instruction call may transfer a number of bytes specified in the **ci_length** field.

To enable the Long Frame mode every Custom Instruction call must have **ci_lf_en** field set to 1. The **ci_opcode** will be transmitted in the first Custom Instruction call and it will be omitted in every subsequent call. Then **ci_length** number of data bytes specified in the **ci_byte<i>** fields are transferred. If the **ci_check_wip** or **ci_wren** fields are set the SPI-MEM-CTRL core will wait once for WIP or send one WREN instruction before the **ci_opcode** is sent. The values of the **ci_spi_wpn** and **ci_spi_holdn** fields are set for the first Custom

Instruction call and affect the state of **spi_wpn_io2** and **spi_holdn_io3** pins respectively for all the Custom Instruction calls until the Long Frame is finished.

To finalize the Long Frame, the **ci_lf_stop** field must be set to 1 on the last Custom Instruction call. Then the SPI-MEM-CTRL core will transfer the last data bytes and exit the Long Frame mode.

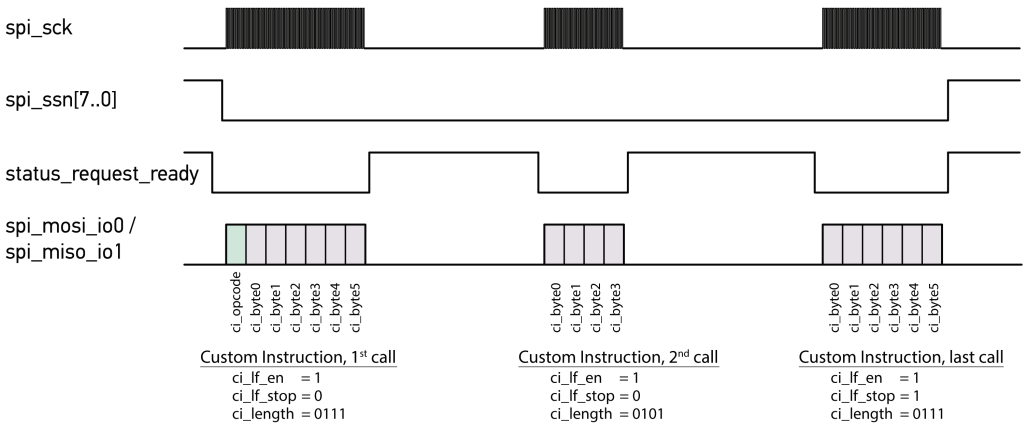

Figure 4-8: Long Frame mode

4.5.11.8 ci_byte<i> fields

The **ci_byte<i>** fields, where **<i>** is [0 … 7], define the data byte **<i>** to be transferred with the Custom Instruction.

The SPI-MEM-CTRL core first sends the **ci_opcode**, then the **ci_byte0**, **ci_byte1** etc using Single SPI (via **spi_mosi_io0** pin). The Serial Flash device's response (received through **spi_miso_io1** pin) replaces the respective **ci_byte<i>** fields of the Custom Instruction Data Registers 0 and 1. When the Custom Instruction is complete the **request_rdy** status bit is set and the host can read the Custom Instruction Data Registers 0 and 1 (addr 14, addr 15) to get the response of the Serial Flash device.

4.5.12 Read Dummy Cycles Register for Serial NOR devices (10h R/W)

Most Serial NOR Flash devices support the following Read Instructions: FAST_READ, READ2O, READ2IO, READ4O and READ4IO (see Table 6-1 for details). There are some Serial Flash devices that support these instructions with non-standard number of dummy cycles in order to be able to achieve higher serial clock frequencies. The SPI-MEM-CTRL core can support custom number of dummy cycles for Instructions READ2IO and READ4IO.

For READ2IO instruction (opcode BBh) the default number of dummy cycles is 4.

For READ4IO instruction (opcode EBh) the default number of dummy cycles is 6.

The host can write into the Read Dummy Cycles Register the desired number of dummy cycles that will be inserted during the respective Read Instructions. For each Read Access Request the SPI-MEM-CTRL core will use the number of dummy cycles specified in the Read Dummy Cycles Register while executing READ2IO or READ4IO instructions.

The value of this register is ignored when SPI NAND mode is enabled.

Refer to Table 4-19 for bit mapping.

Table 4-19: Read Dummy Cycles Register

Bit	Field	Description	Reset Value
3:0	rdc_read2io	Number of Dummy Cycles for READ2IO Instruction (opcode BB) 0000: Default (4 dummy cycles) 0001-0011: reserved 0100: 4 dummy cycles 0101: 5 dummy cycles 0110: 6 dummy cycles 0111: 7 dummy cycles 1000: 8 dummy cycles 1001: 9 dummy cycles 1010: 10 dummy cycles 1011-1111: reserved	0h[1]
7:4	rdc_read4io	Number of Dummy Cycles for READ4IO Instruction (opcode EB) 0000: Default (6 dummy cycles) 0001-0101: reserved 0110: 6 dummy cycles 0111: 7 dummy cycles 1000: 8 dummy cycles 1001: 9 dummy cycles 1010: 10 dummy cycles 1011-1111: reserved	0h[1]
10:8	rx_delay		001
11	Reserved	Reserved for future use	0
31:12	Reserved	Reserved for future use	00000h

Table Notes:
[1] Configurable during synthesis

4.5.12.1 rx_delay field

This field determines the timing for the sampling of the input serial data. The value of the **`rx_delay`** field specifies the number of delay steps for capturing the serial data inputs. Each delay step is equal to one **`spi_2sclk`** clock cycle. If set to 0, input serial data is captured with the rising edge of the generated SPI Clock. The input timing is illustrated on Figure 4-9. In this example, the best value to use would be **`rx_delay = 2`**. Supported values range from 0 to 7.

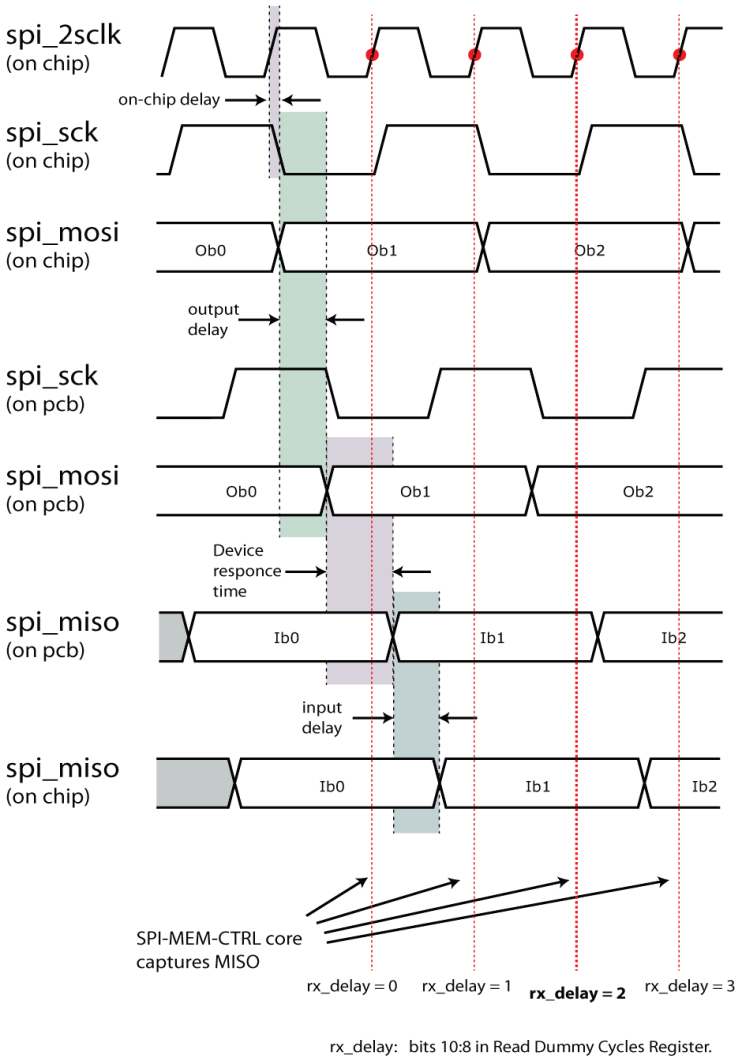

Figure 4-9: SPI Bus input timing diagram.

4.5.13 Extended Configuration Register (11h R/W)

Table 4-20: Extended Configuration Register

Bit	Field	Description	Reset Value
7:0	`read4io_mode`	The value of the mode bits that will be transmitted after the address on a READ4IO (EBh) command. Used only in SPI NOR Flash mode.	00
31:8	`reserved`	Reserved for future use	000000h

In the READ4IO instruction the first 8 bits (2 clock cycles) sent after the address bits are the mode bits. Most serial flash devices work with the default value of zero for the mode bits. However, there are a few serial flash devices that need these mode bits to have a different value. This value should be written to the **`read4io_mode`** field prior to a READ4IO instruction.

4.6 Configuration Memory

The SPI-MEM-CTRL core uses a Configuration Memory in which various configuration parameters are stored, such as the list of the supported Serial Flash devices, the default behavior of the core when the Serial Flash device is not detected, the startup Block Read transfer parameters, etc.

This memory can be implemented either as an on-chip ROM, or a RAM, or register file etc. A simple SRAM like interface is used to access this memory. An RTL model for a Read Only Memory is supplied with the core, but it is up to the user to implement in a different way to permit extended configurability of the core, even after it is implemented in hardware and changes are not possible.

Table 4-21 shows how the available configuration options are mapped to the address space of the configuration memory.

Table 4-21: Configuration Memory

32-bit Word Address	Name	Description
00h	BRI CONTROL REGISTER VALUE	Default value for the Control Register to be used with Block Read transfers
01h	BRI DURATION DPM REGISTER VALUE	Default value for the Duration DPM Register to be used with Block Read transfers
02h	BRI EXTENDED ADDRESSING MODE REGISTER VALUE	Default value for the Extended Addressing Mode Register to be used with Block Read transfers
03h	BRI DEFAULT MEMORY REGISTER VALUE	Default value for the Default Memory Register to be used with Block Read transfers
04h	BRI WAIT POWER UP CONSTANT	Default value for the Wait Power-up constant to be used with Block Read transfers
05h	STARTUP BRI_RQST_ADDR	Value for **bri_rqst_addr** to be used for a Startup Block Read transfer
06h	STARTUP BRI_RQST_COUNT	Value for **bri_rqst_count** to be used for a Startup Block Read transfer
07h	STARTUP BRI_DEST_OFFSET	Value for **bri_dest_offset** to be used for a Startup Block Read transfer
08h	NUMBER OF SPI DEVICES LISTED	The number of SPI devices in ROM
08h	NUMBER OF DUMMY BITS	Default value for Read Dummy Cycles Register to be used with the respective Read transfers
08h	RX_DELAY	Default value for rx_delay in Read Dummy Cycles Register
08h	NUMBER OF READ4IO MODE BITS	Default value for mode bits in READ4IO instruction
09h	DEVICE 0 ENTRY	1st Serial Flash memory device info
0Ah	DEVICE 1 ENTRY	2nd Serial Flash memory device info
…	…	…

4.6.1 Addr 00h – BRI Control Register value

This is a 32-bit word that holds the default value that will be used for the Control Register (see 4.5.1) for Block Read transfers, when the Control Register is not

explicitly programmed through the Host Interface, e.g. on a startup Block Read transfer.

4.6.2 Addr 01h – BRI Duration DPM Register Value

This is a 32-bit word that holds the default value that will be used for the Duration DPM Register (see 4.5.4) for Block Read transfers, when the Duration DPM Register is not explicitly programmed through the Host Interface.

4.6.3 Addr 02h – BRI Extended Addressing Mode Register Value

This is a 32-bit word that holds the default value that will be used for the Extended Addressing Mode Register (see 4.5.8) for Block Read transfers, when the Extended Addressing Mode Register is not explicitly programmed through the Host Interface.

4.6.4 Addr 03h – BRI Default Memory Register Value

This is a 32-bit word that holds the default value that will be used for the Default Memory Register (see 4.5.7) for Block Read transfers, when the Default Memory Register is not explicitly programmed through the Host Interface.

4.6.5 Addr 04h – BRI Wait power up constant

This is a 32-bit word that holds the Wait Power-up Constant value. This value determines how many **`core_clk`** cycles the core will wait after power up before Block Read transfers can take place.

4.6.6 Addr 05h – Startup bri_rqst_addr

This is a 32-bit word indicating the starting address in the Serial Flash memory that will be used for the Startup Block Read transfer.

4.6.7 Addr 06h – Startup bri_rqst_count

This is a 32-bit word indicating the number of bytes to transfer that will be used for the Startup Block Read transfer. This number must be a multiple of 4-bytes.

4.6.8 Addr 07h – Startup bri_dest_offset

This is a 32-bit word indicating the base address for the destination address space that will be used for the Startup Block Read transfer.

4.6.9 Addr 08h – The number of SPI devices in ROM, the Number of Dummy Cycles, rx_delay & the Mode bits in READ4IO

This is a 32-bit word containing the following field:

Bits [11:0] indicate how many Serial Flash Device Info entries are following from address 09h and after.

Bits [15:12] and [19:16] hold the default value for the **`rdc_read2io`** and **`rdc_read4io`** fields of the Read Dummy Cycles Register (see 4.5.12) that will be used during Startup Block Read transfers.

Bits [22:20] hold the default value for **`rx_delay`** field of the Read Dummy Cycles Register (see 4.5.12) that will be used during Startup Block Read transfers.

Bits [30:23] hold the value that will be programmed to the **`read4io_mode`** bits field of the Extended Configuration Register (see 4.5.13) on Startup Block Read or XOTF transfers.

See Table 4-22 for bit Mapping

Table 4-22: Number SPI Devices, Read Dummy Cycles Register Value

Bit	Field
11:0	Number of SPI Devices
15:12	Number of Dummy Cycles for **READ2IO** (for SPI NOR mode)
19:16	Number of Dummy Cycles for **READ4IO** (for SPI NOR mode)
22:20	**RX_Delay**
30:23	Mode bits for **READ4IO** during startup BRI or XOTF (for SPI NOR mode)
31	Reserved

4.6.10 Addr ≥ 09h – Serial Flash memory device info

Each 32-bit word from Address 09h and above, represents an entry for an SPI Serial Flash device. These entries are scanned by the core during the auto detection of the connected memory device.

The Entry for one memory device is shown in Table 4-23.

Table 4-23: Serial Flash Device Entry

	bit 31..24	bit 23..16	bit 15…8	bit 7	bit 6	bit 5…3	bit 2…0
NOR Flash	Manufacturer ID	Type ID	Density ID	**attr_dpm**	**attr_32**	**write_opcode**	**read_opcode**
NAND Flash	Manufacturer ID	Device ID	Unused	Unused	Unused	**write_opcode**	**read_opcode**

The Manufacturer ID, Type ID and Density ID, are the values returned by the device after a RDID command.

For more information on bits 7…0 refer to the description of the respective bits in the Default Memory Register, in 4.5.7.

5 HARDWARE SPECIFICATION

5.1 Block Diagram

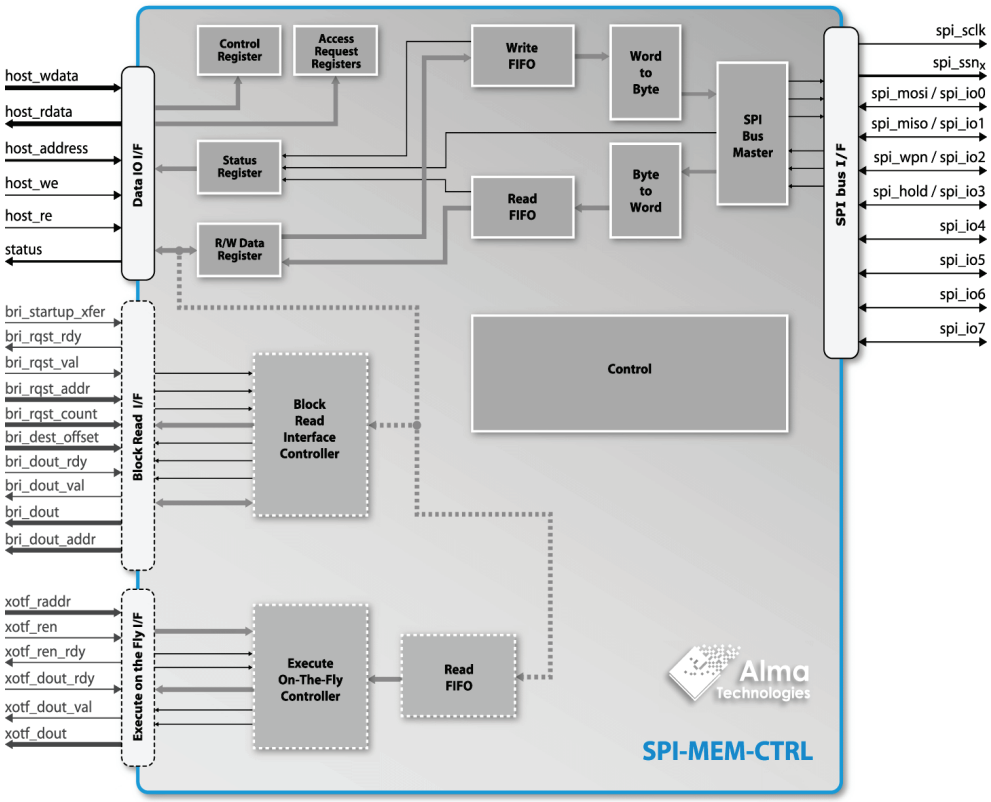

Figure 5-1: SPI-MEM-CTRL Block diagram

5.2 Block Description

The Control Register and the Access Request Registers are programmed to setup the core. When the access request is defined the main controller decodes the access request and coordinates the data transfer.

In READ access request the LLSPIC sends the memory the necessary instructions to read it and prepares itself to get the data. The data flow out of the memory in

byte streams. The LLSPIC gets those bytes and transfers them to the byte 2 word component. Then the 32 bit word is moved into the Read-FIFO. The first word of the Read-FIFO is pre-fetched. The host can get it by reading the Read/Write data Register.

In WRITE access request the host writes the Read/Write Data Register. The control unit moves that data into the Write-FIFO. The first word in the Write-FIFO is moved into the word 2 byte component. The LLSPIC sends the necessary instructions to the memory followed by the data.

The LLSPIC works in the **`spi_2sclk`** clock domain while the rest of the core uses the **`core_clk`**. The synchronization of the two clock domains is achieved with the use of two double-clocked asynchronous FIFOs. The one double-clocked FIFO is for input data and the other for output data.

The SPI-MEM-CTRL core receives two clock signals via the **`core_clk`** and **`spi_2sclk`** ports.

6 TECHNOLOGY SPECIFICATION

6.1 Supported Instruction Set for SPI NOR-Flash mode

The SPI-MEM-CTRL supports the instructions listed on Table 6-1 on SPI NOR-Flash mode.

Table 6-1: Instruction Set

Instruction	Opcode	Description
WREN	06	Write enable
RDSR	05	Read Status Register
FAST_READ	0B	Read bytes at higher speed
READ2O	3B	Dual Read Output
READ2IO	BB	Dual Read Input / Output
READ4O	6B	Quad Read Output
READ4IO	EB	Quad Read Input / Output
PP	02	Page Program
PP2O	A2	Dual Page Program Output
PP4O	32	Quad Page Program Output
PP4IO	38	Quad Page Program Input / Output
SE	20	Sector Erase
BE	D8	Block Erase
CE	C7	Chip Erase
RDID	9F	Read Device ID / Jedec ID
DP	B9	Deep Power-down
RES	AB	Release from Deep Power-down
EN4B	See reg 09h	Enable 32 bit address mode

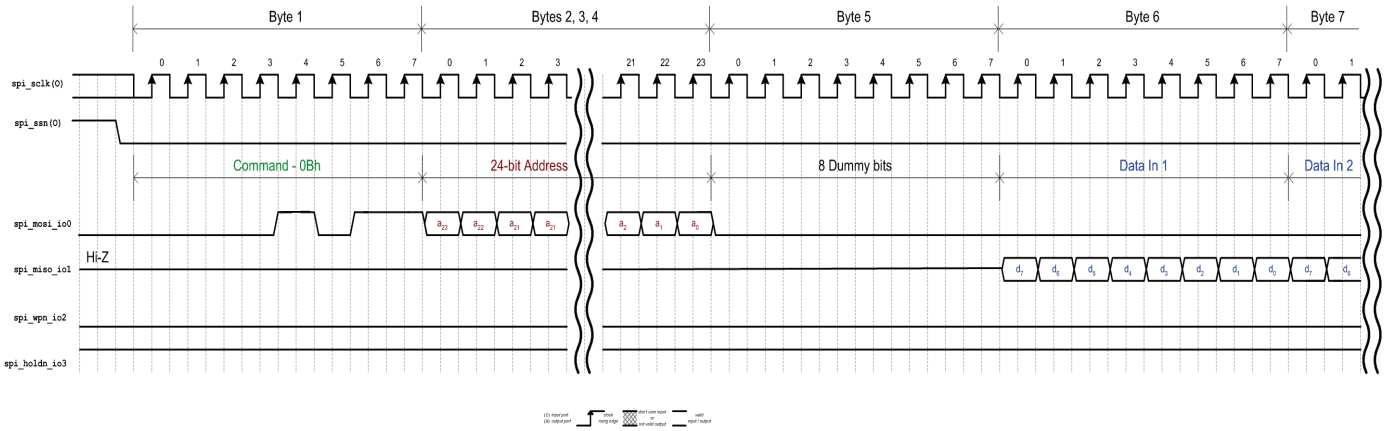

Figure 6-1: FAST_READ (opcode 0B) 24 address bit mode

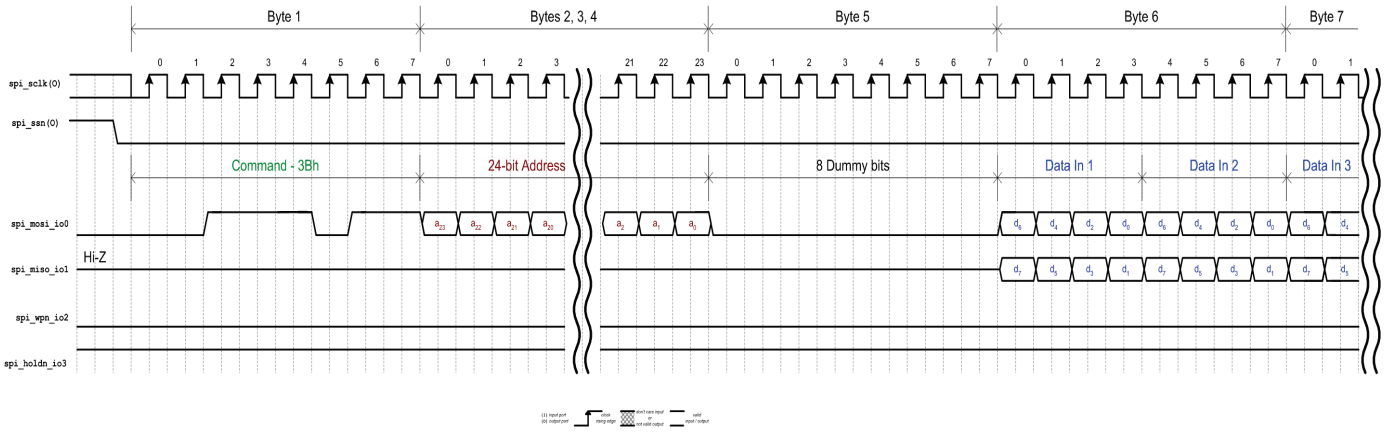

Figure 6-2:READ2O (opcode 3B) 24 address bit mode

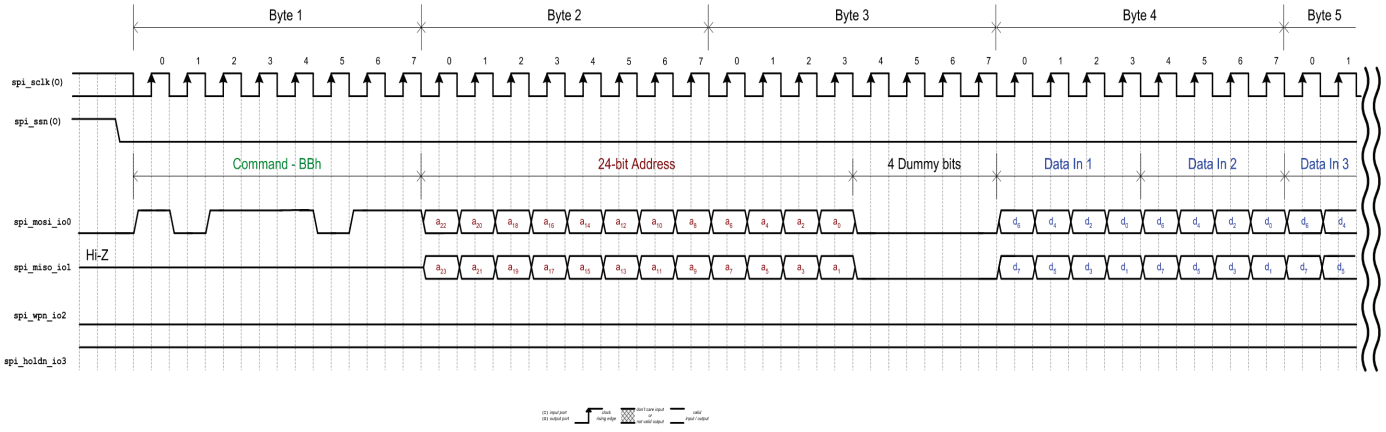

Figure 6-3: READ2IO (opcode BB) 24 address bit mode

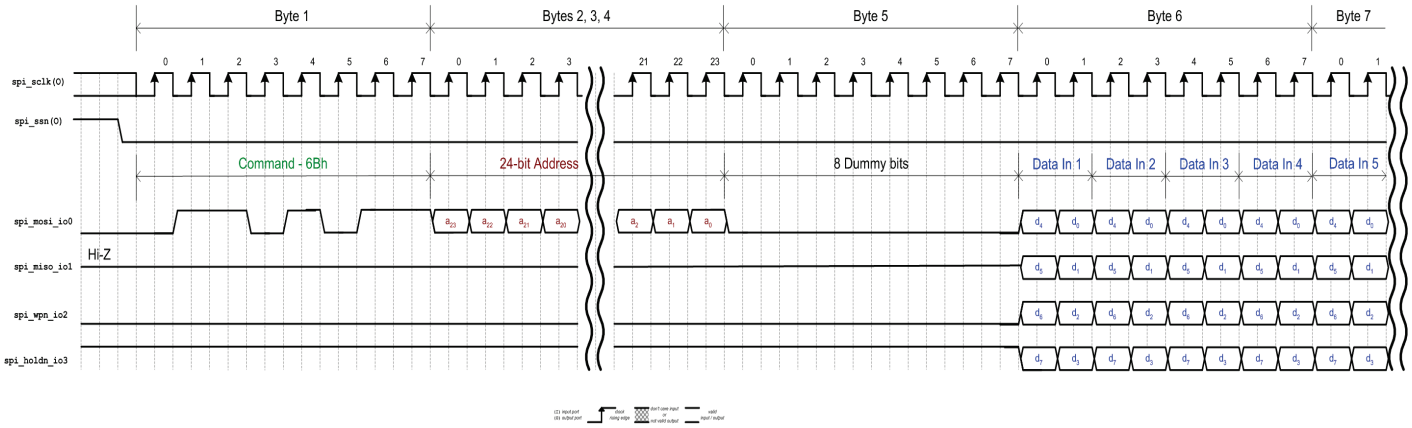

Figure 6-4: READ4O (opcode 6B) 24 address bit mode

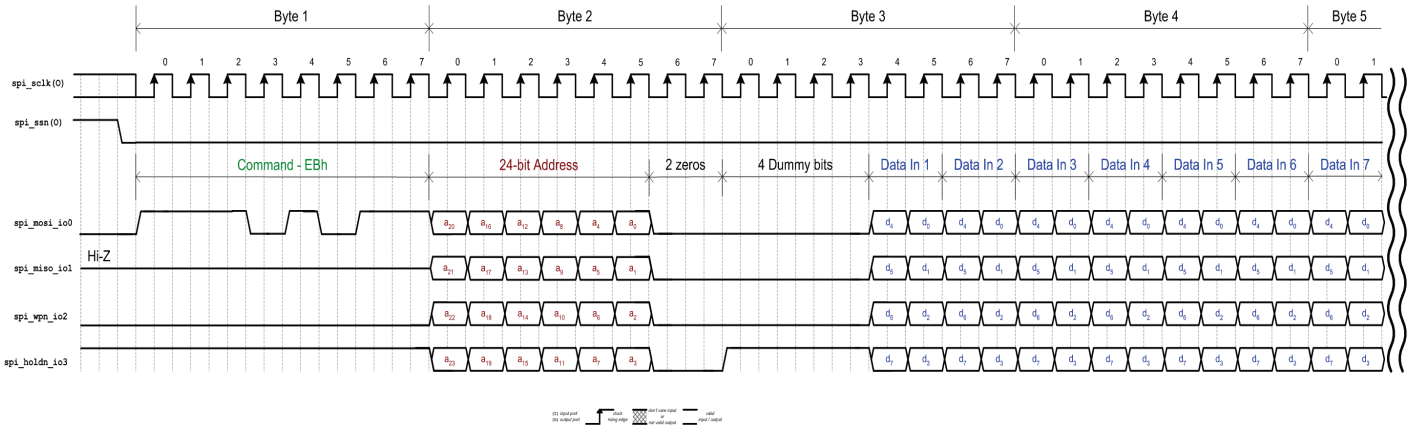

Figure 6-5: READ4IO (opcode EB) 24 address bit mode

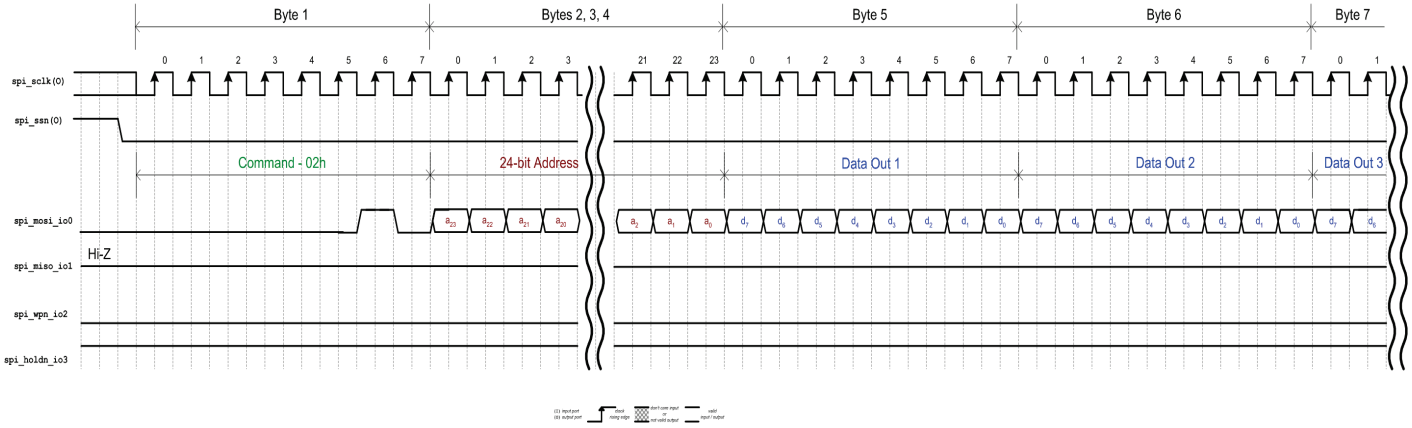

Figure 6-6: PP (opcode 02) 24 address bit mode

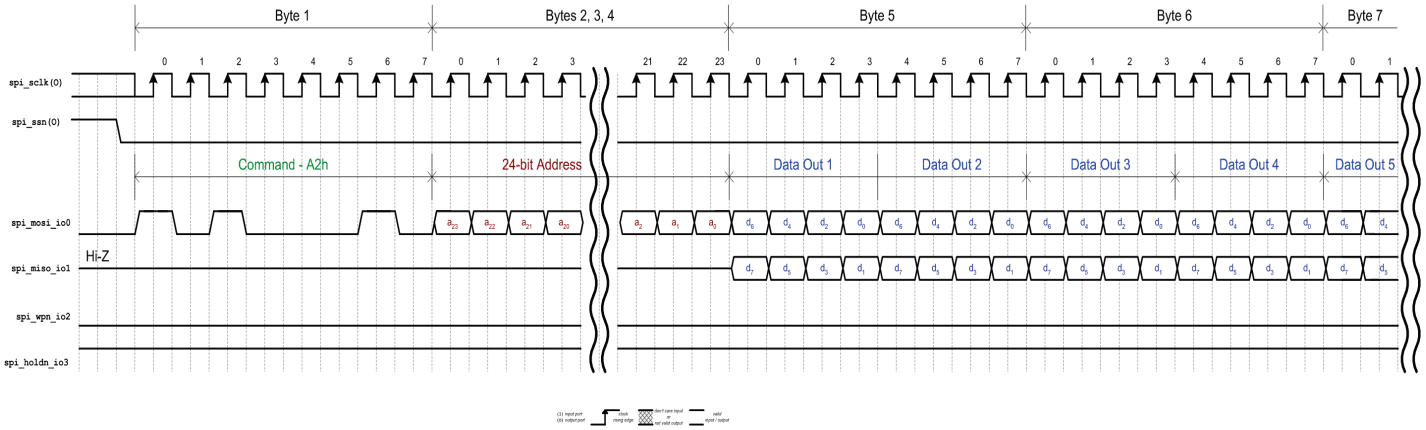

Figure 6-7: PP2O (opcode A2) 24 address bit mode

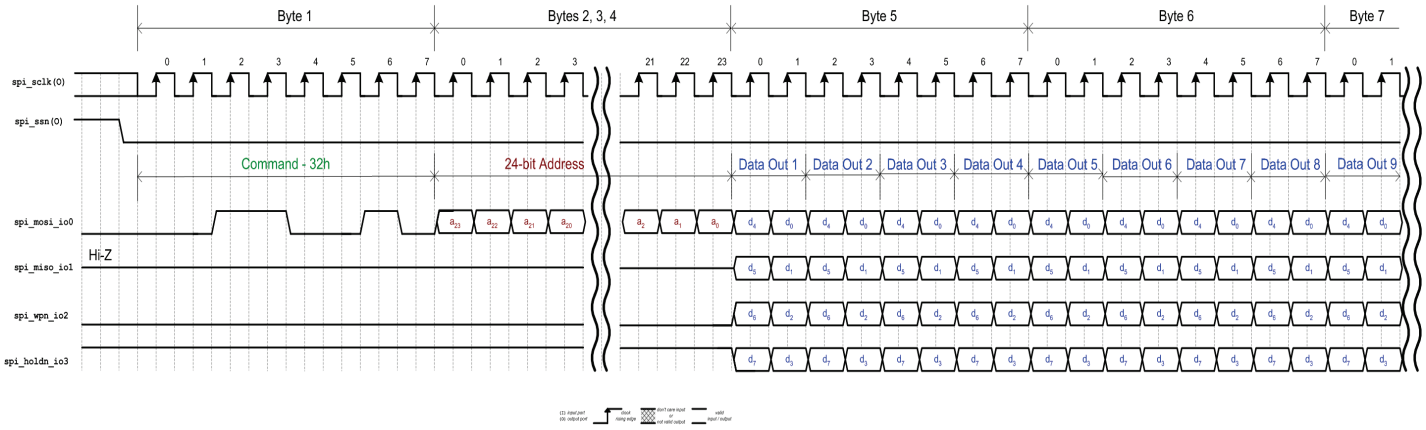

Figure 6-8: PP4O (opcode 32) 24 address bit mode

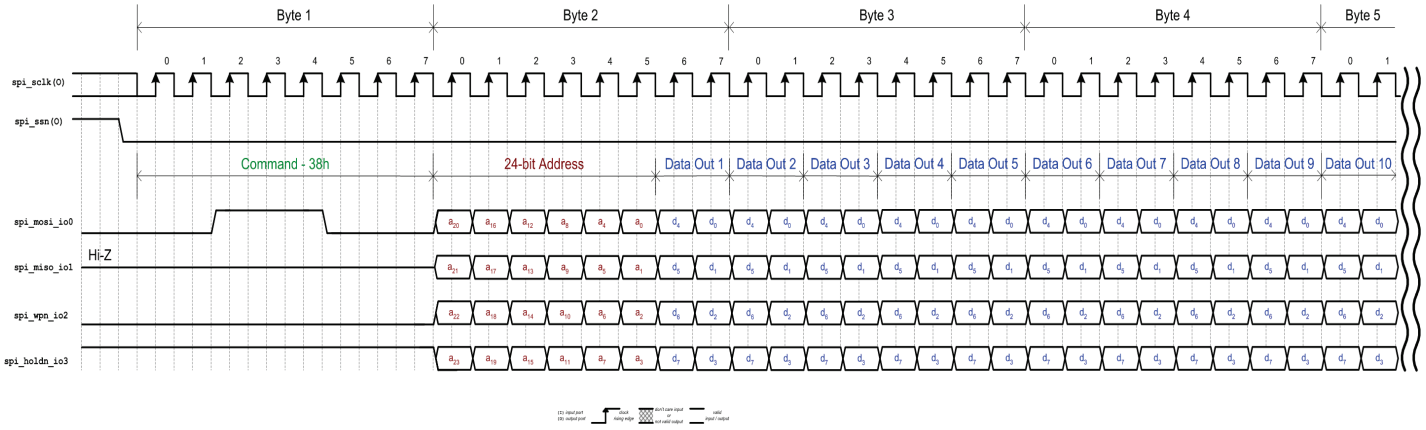

Figure 6-9: PP4IO (opcode 38) 24 address bit mode

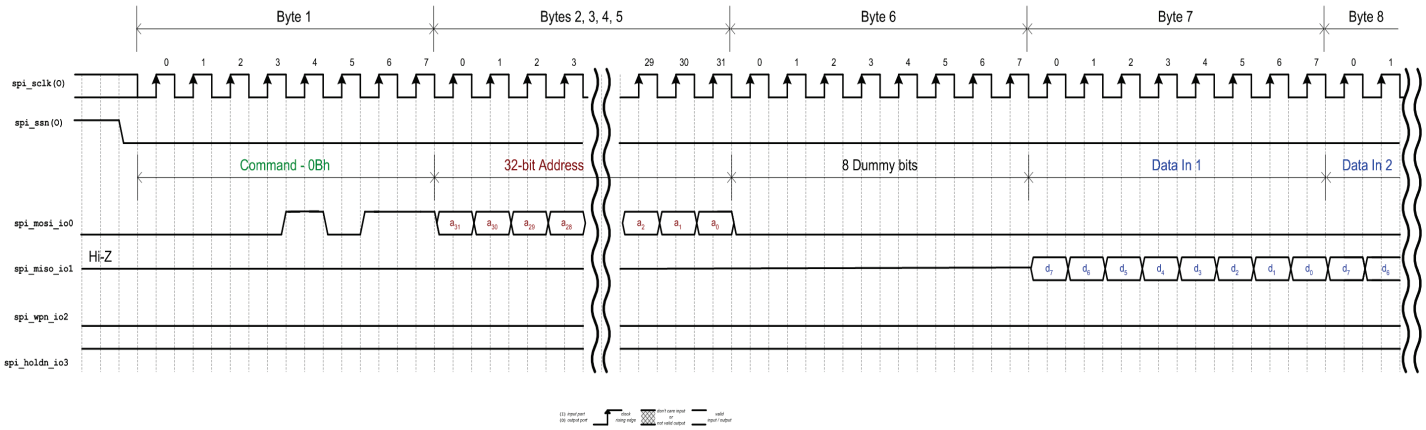

Figure 6-10: FAST_READ (opcode 0B) 32 address bit mode

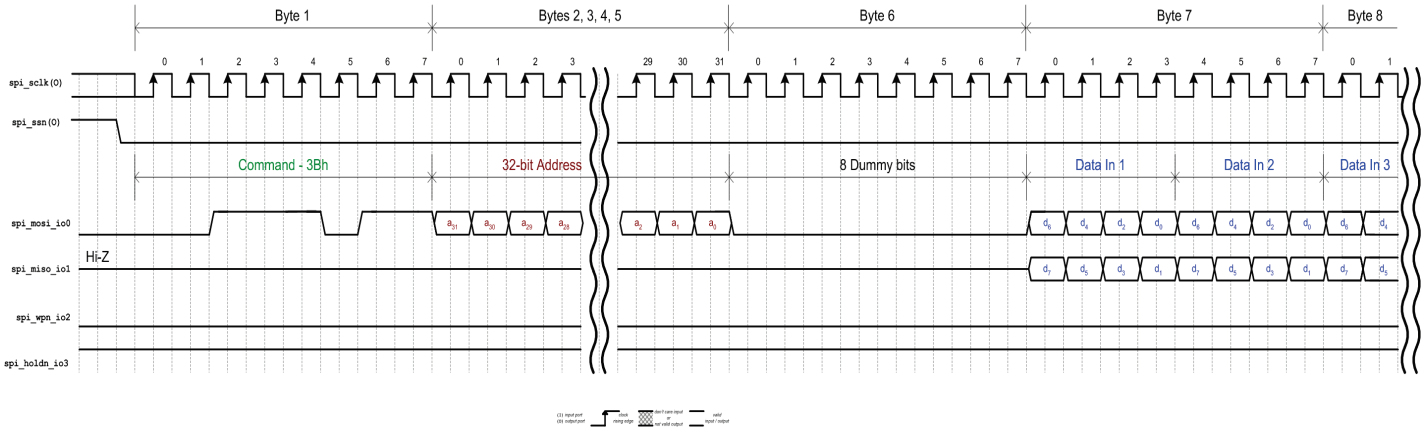

Figure 6-11: READ2O (opcode 3B) 32 address bit mode

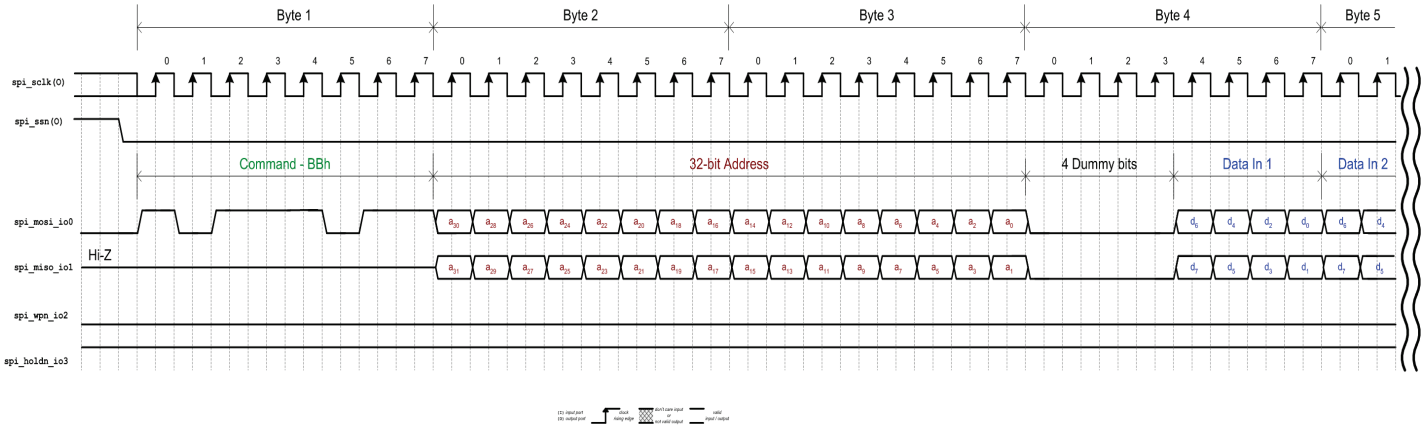

Figure 6-12: READ2IO (opcode BB) 32 address bit mode

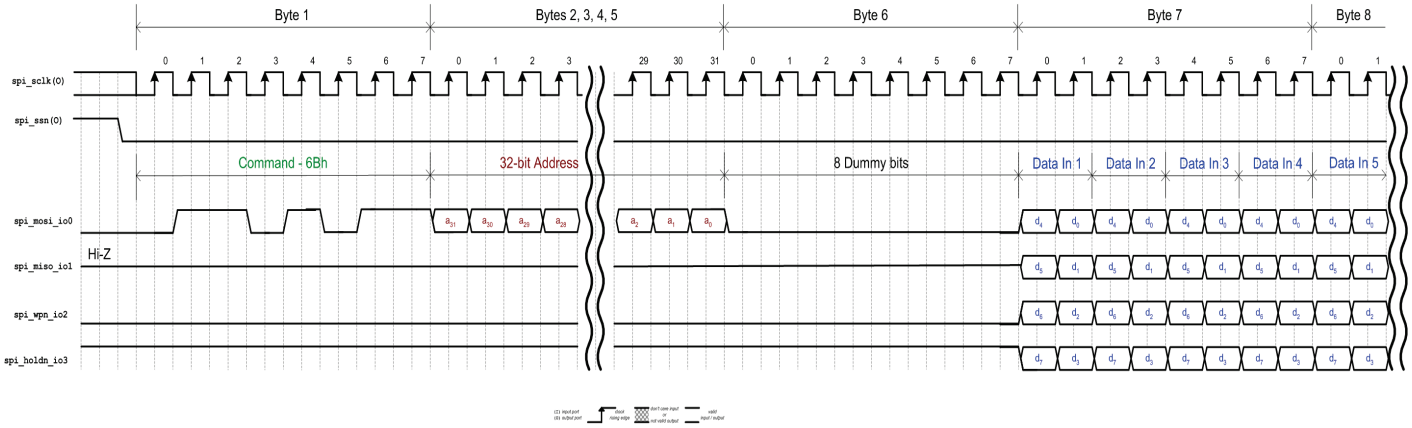

Figure 6-13: READ4O (opcode 6B) 32 address bit mode

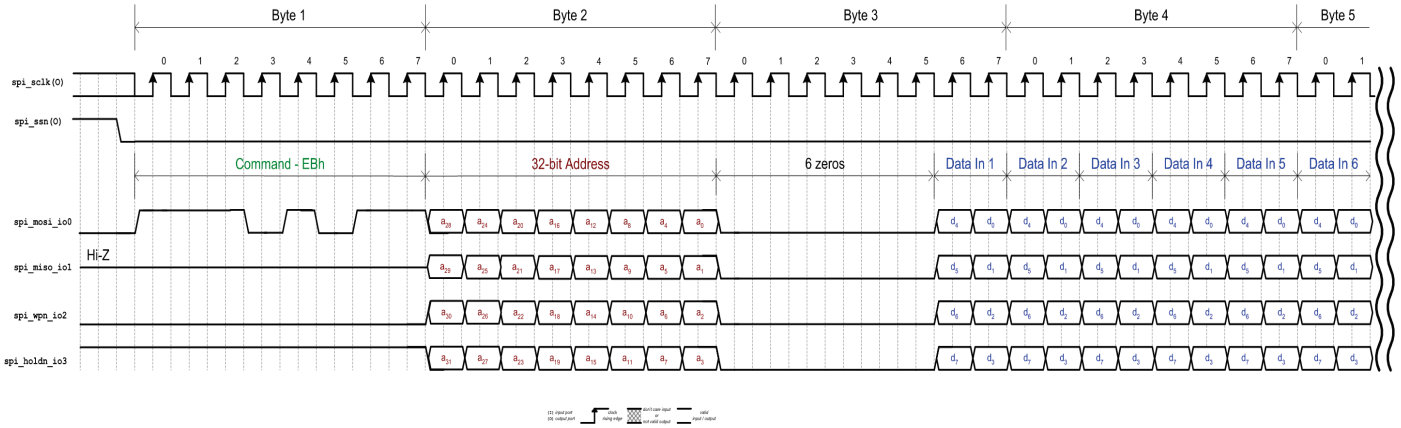

Figure 6-14: READ4IO (opcode EB) 32 address bit mode

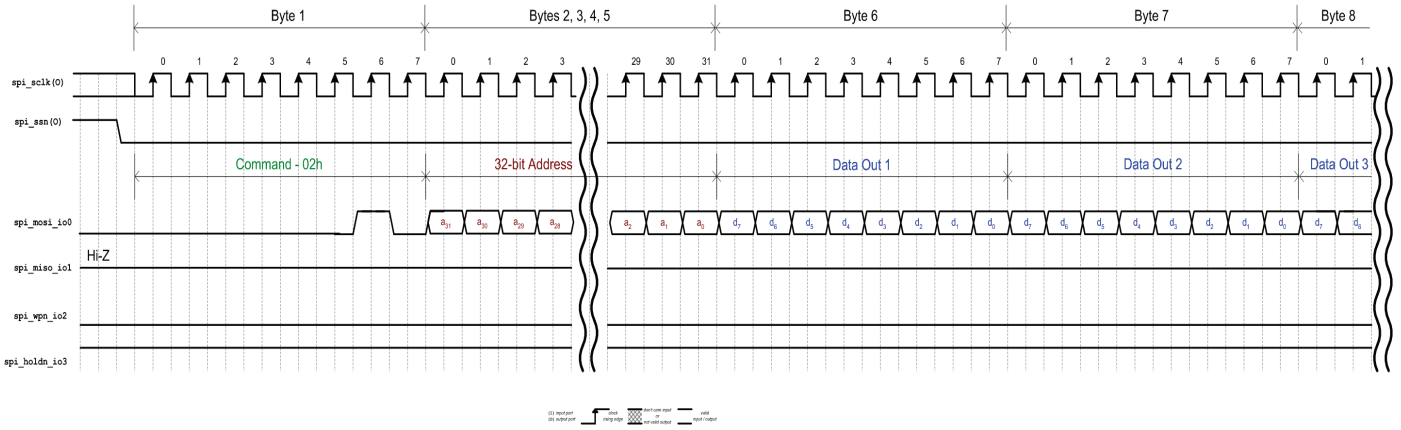

Figure 6-15: PP (opcode 02) 32 address bit mode

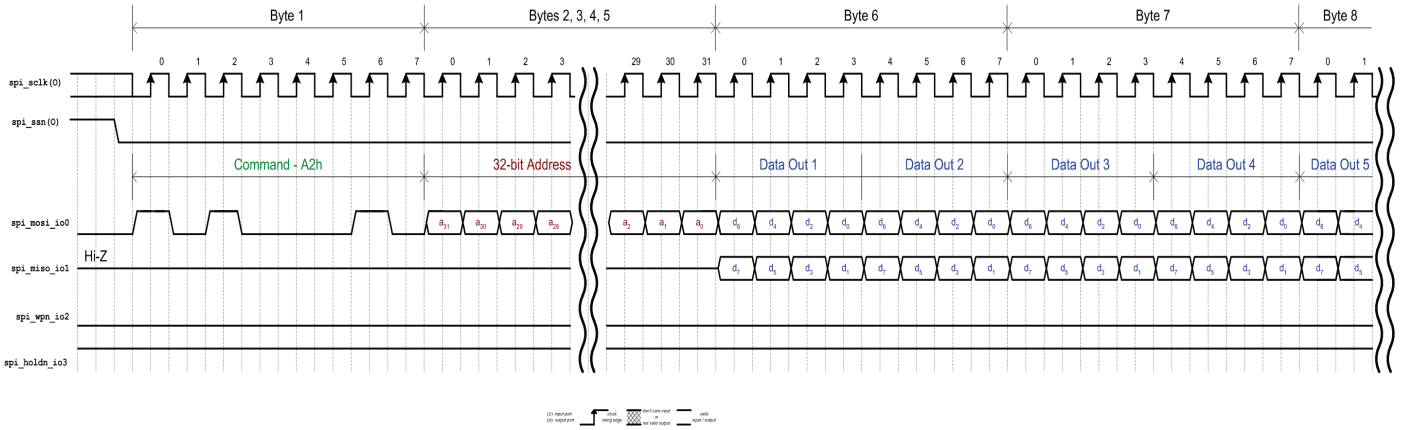

Figure 6-16: PP2O (opcode A2) 32 address bit mode

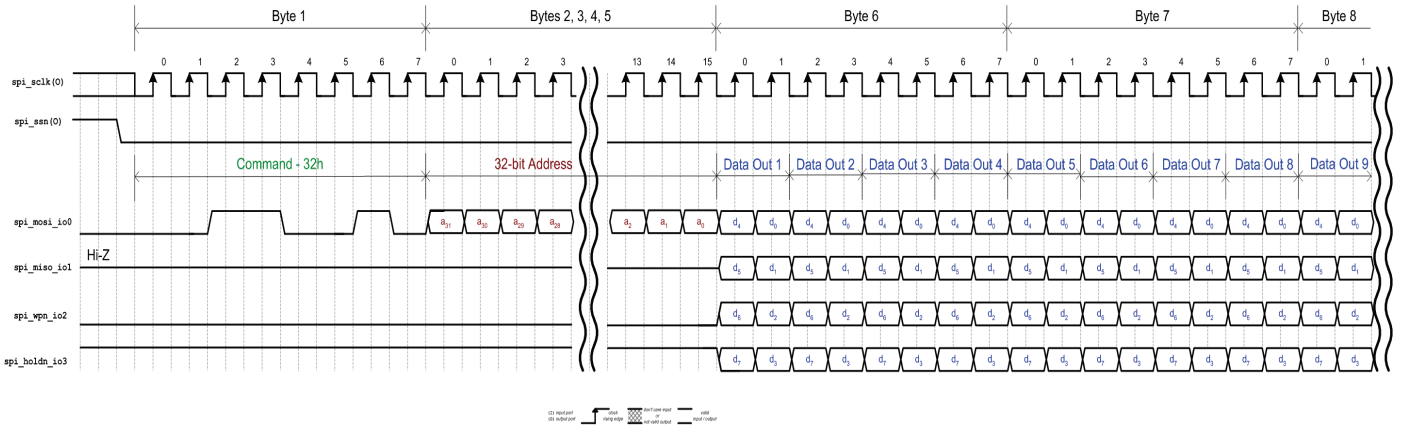

Figure 6-17: PP4O (opcode 32) 32 address bit mode

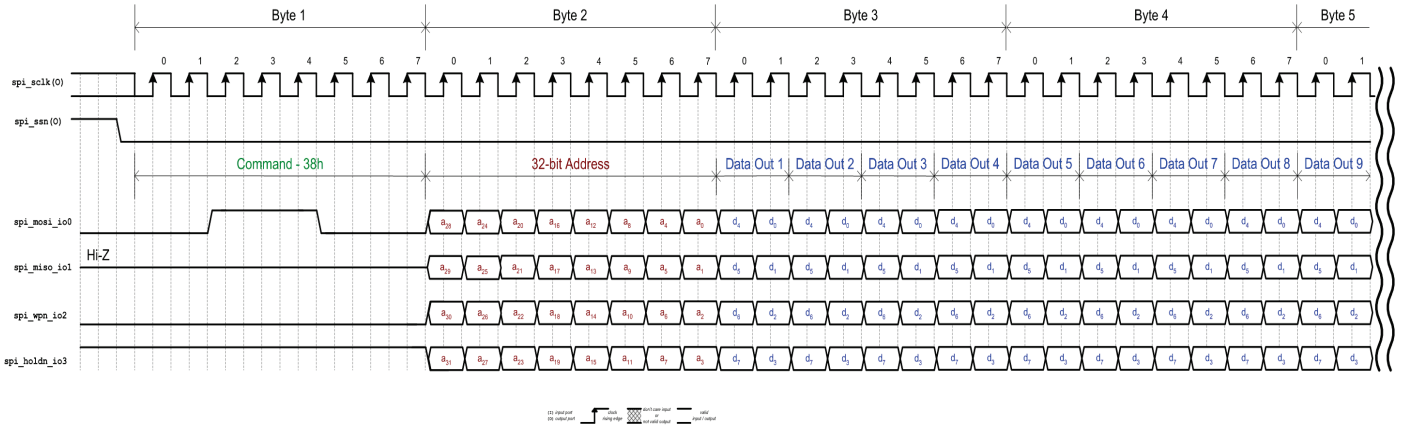

Figure 6-18: PP4IO (opcode 38) 32 address bit mode

6.2 Supported Instruction Set for SPI NAND-Flash mode

The SPI-MEM-CTRL supports the instructions listed on Table 6-1 on SPI NAND-Flash mode.

Table 6-2: Instruction Set

Instruction	Opcode	Description
WREN	06	Write enable
GETFEAT	0F	Read Status Register
READ_ID	9F	Read Identification Data
PAGE_READ	13	Prefetch page in Cache Register
FAST_READ	0B	Read bytes at higher speed
READ2O	3B	Dual Read Output
READ2IO	BB	Dual Read Input / Output
READ4O	6B	Quad Read Output
READ4IO	EB	Quad Read Input / Output
PLRDx1	84	Program Load Random Data x1
PLRDx4	34	Program Load Random Data x4
PEXEC	10	Program Execute
BE	D8	Block Erase

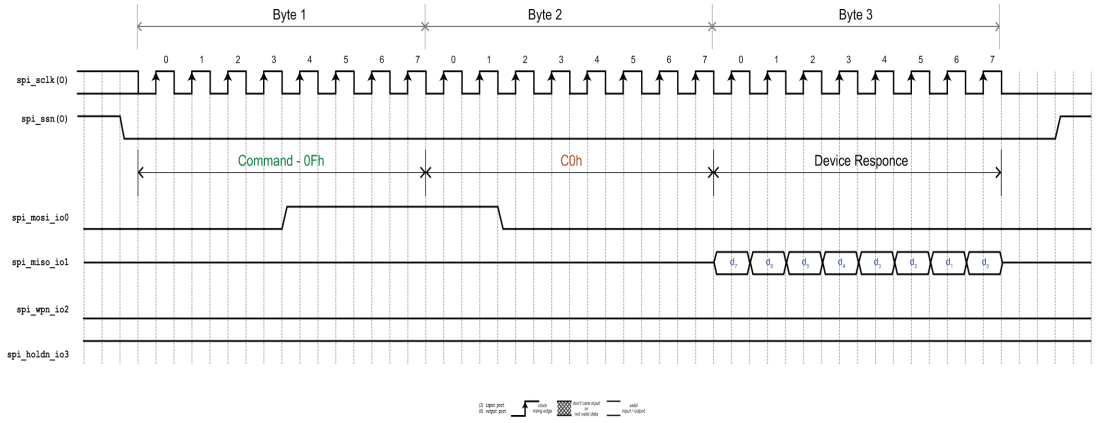

Figure 6-19: Get Status (Get Feature - opcode 0F)

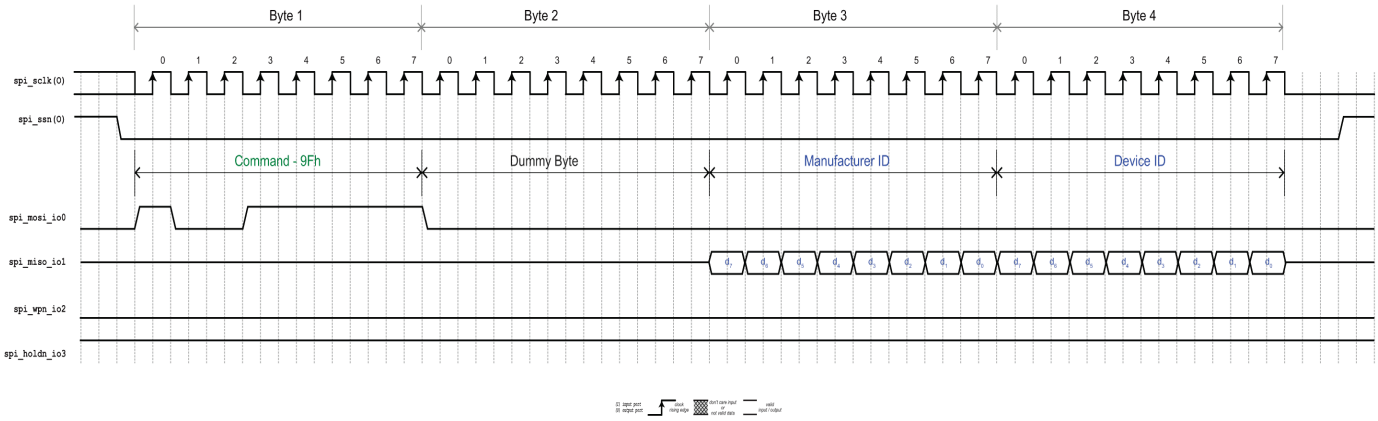

Figure 6-20:READ_ID (opcode 9F)

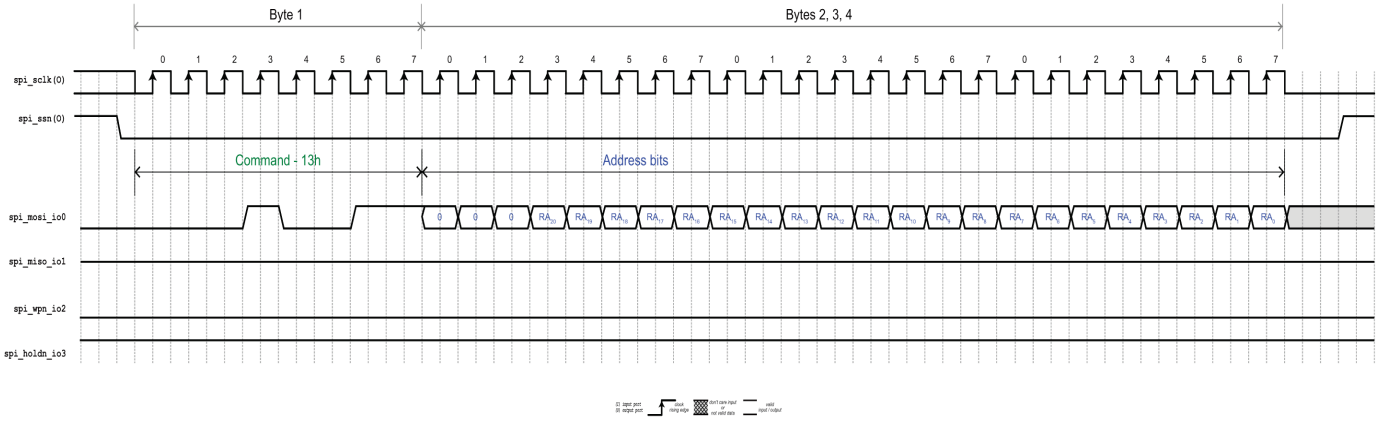

Figure 6-21: PAGE_READ (opcode 13)

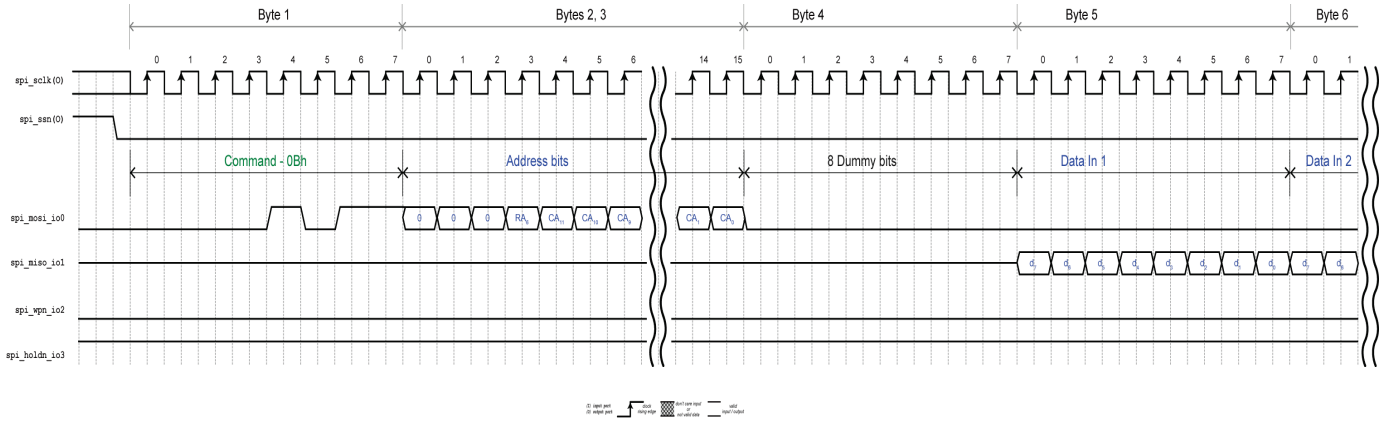

Figure 6-22: FAST_READ (opcode 0B)

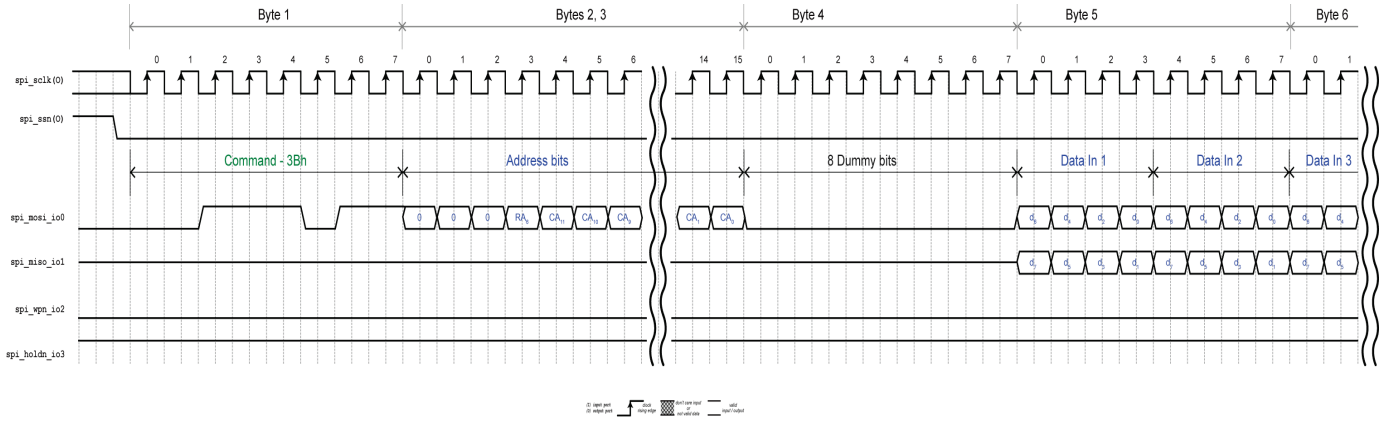

Figure 6-23: READ2O (opcode 3B)

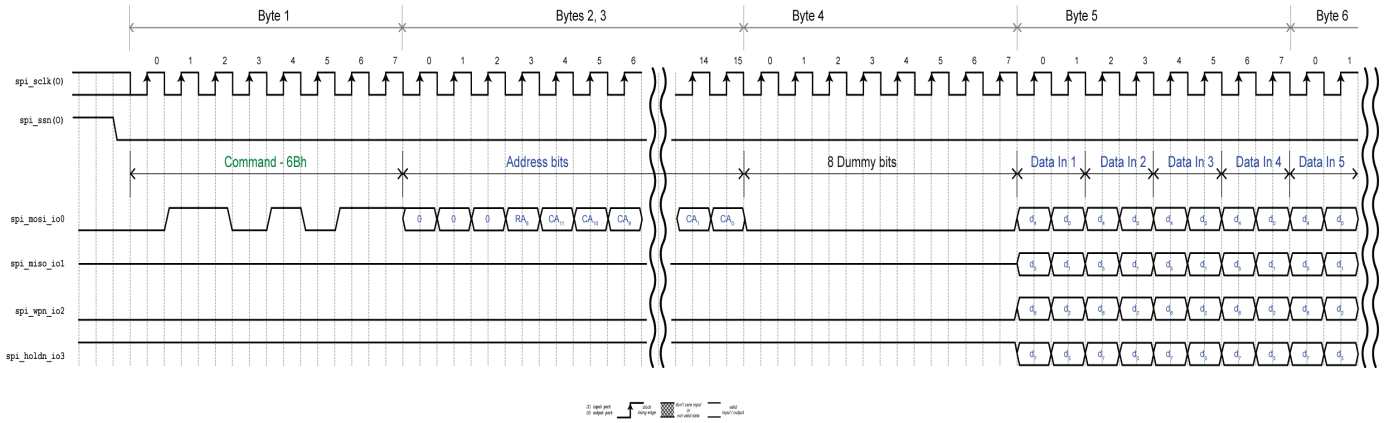

Figure 6-24: READ4O (opcode 6B)

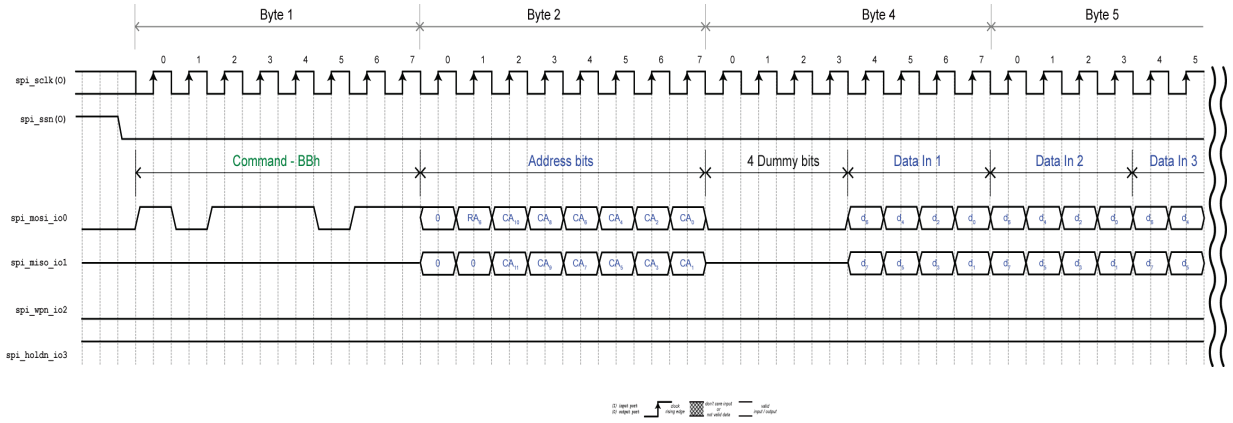

Figure 6-25: READ2IO (opcode BB)

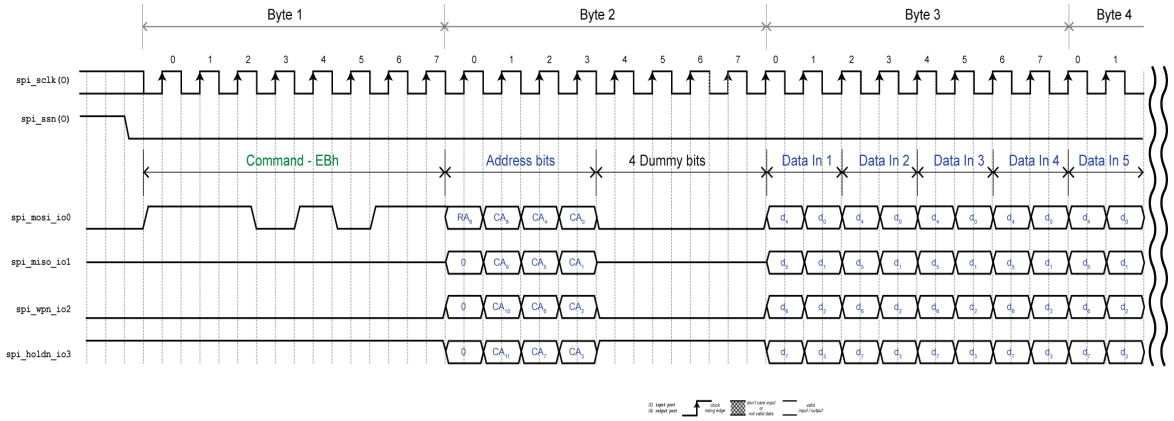

Figure 6-26: READ4IO (opcode EB)

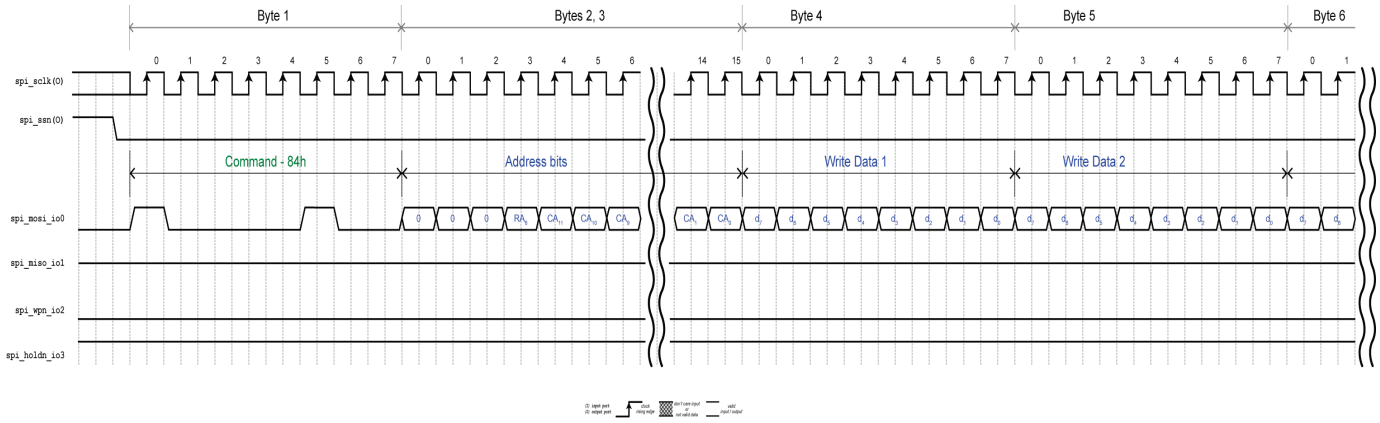

Figure 6-27: PROGRAM LOAD RANDOM DATA x1 (opcode 84)

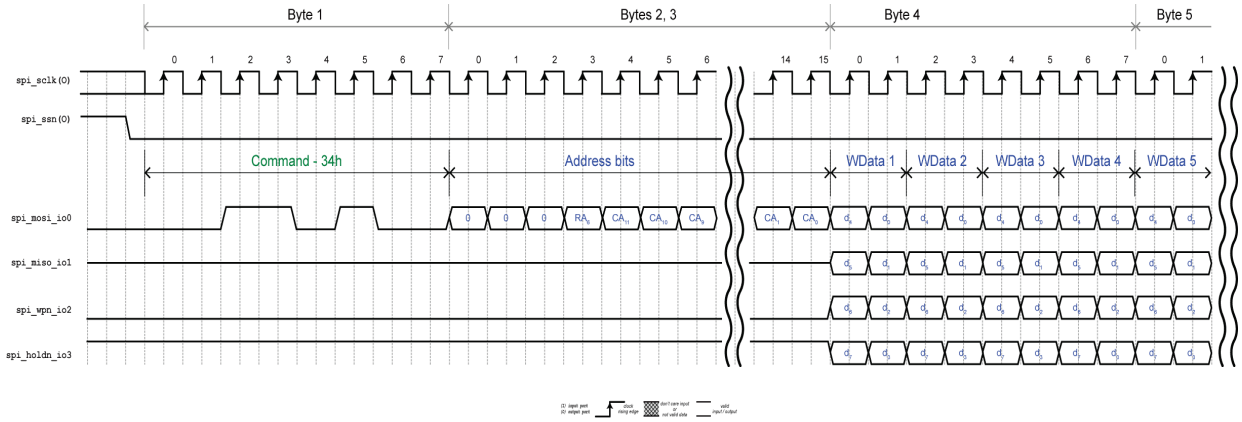

Figure 6-28: PROGRAM LOAD RANDOM DATA x4 (opcode 34)

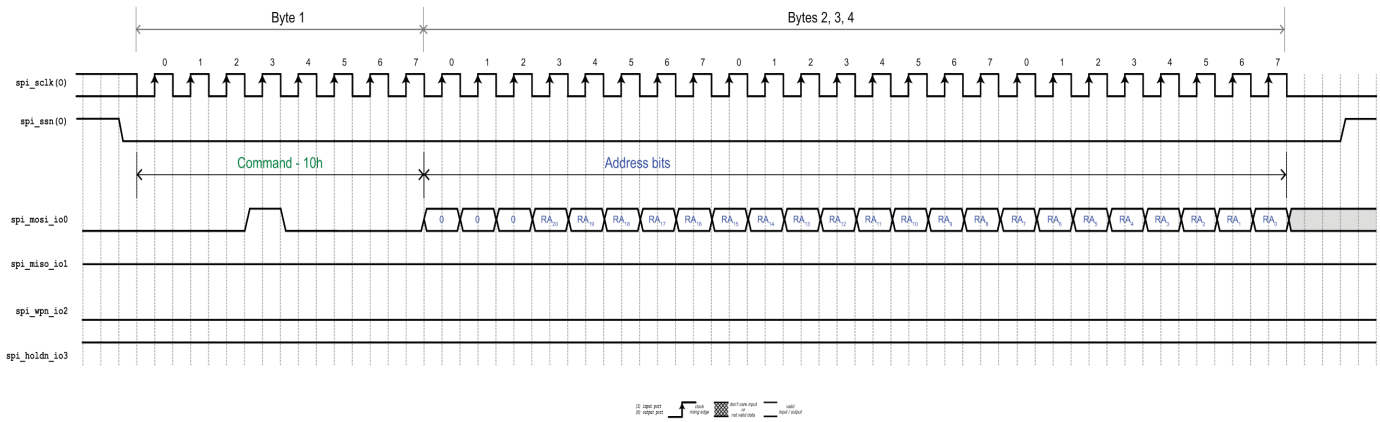

Figure 6-29: PROGRAM EXECUTE (opcode 10)

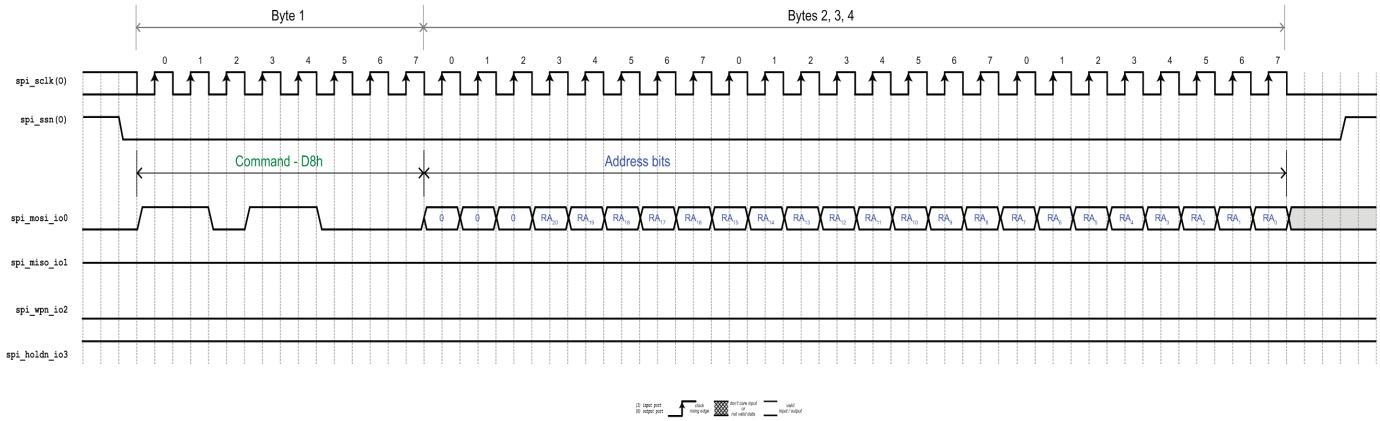

Figure 6-30: BLOCK ERASE (opcode D8)

6.3 Supported Memories

The SPI-MEM-CTRL supports a variety of memories. Some of these chips are listed on Table 6-3 and Table 6-4. Please contact Alma Technologies to ask if your chip is not included in this table.

Table 6-3: Supported SPI NOR Memory Chips

Supplier	Model	attr_DPM	attr_32	Faster supported write_opcode	Faster supported read_opcode
AMIC	A25L032	1	0	001	010
ATMEL	AT25FS040	0	0	000	000
EON	EN25B64	1	0	000	000
EON	EN25D16	1	0	000	001
ESMT	F25L16A	0	0	000	001
ESMT	F25L32Q	1	0	010	100
GigaDevice	GD25F40/80	1	0	000	000
GigaDevice	GD25D40/80	1	0	000	001
MACRONIX	MX25L512	1	0	000	001
MACRONIX	MX25L12805D	1	0	000	000
MACRONIX	MX25L3235D	1	0	011	100
MACRONIX	MX25L3237D	1	0	011	100
MACRONIX	MX25L25635E	1	1	011	100
MACRONIX	MX25L25735E	1	1	011	100
MICRON	N25Q256A13E	0	1	010	011
NUMONIX	M25PE16	1	0	000	000
NUMONIX	M25PX64	1	0	001	001
PMC	Pm25LV080B/016B	0	0	000	000
SPANSION	S25FL064A	1	0	000	000
SPANSION	S25FL512S	0	1	000	011
SST	SST25VF064C	0	0	001	010
WINBOND	W25X16, W25X16A, W25X32, W25X64	1	0	000	001
WINBOND	W25Q128BV	1	0	010	100
WINBOND	W25Q64BV	1	0	010	100

Table 6-4: Supported SPI NAND Memory Chips

Supplier	Model	Faster supported write_opcode	Faster supported read_opcode
Micron	MT29F1G01ABBFD	001	100
Winbond	W25M02GW	001	100
Winbond	W25N01GV	001	100
Winbond	W25M02GV	001	100

Notes:

The material in this document is for information only. ALMA Technologies assumes no responsibility for errors or omissions and reserves the right to change, without notice, product specifications, operating characteristics, etc. ALMA Technologies assumes no liability for damage resulting from the use of information contained in this document.