cadence®

QSPI Flash Controller and PHY

Internal Name: QSPI
Part Number: IP6515E

NDA #
Design Specification
Revision: 1.01

Confidentiality Notice

Trademarks

Restricted Permission

Disclaimer

Restricted Rights

Table of Contents

List of Figures

List of Tables

Preface

This chapter provides a general introduction to this manual, and contains the following sections:

1. About This Document

This manual is intended for chip designers who will use the QSPI Flash Memory Controller in their application. Readers of this document should be familiar with QSPI Flash Devices specifications.

2. Organization of This Document

This manual contains the following chapters:

Preface	Describes the manual and lists the typographical conventions and symbols used. It also provides information on how to get technical assistance.
Chapter 1, *Module Data*	This chapter provides general information about controller architecture and features.
Chapter 2, *Programming the QSPI Flash Controller*	This chapter describes the controller application interface as viewed by software engineers. It provides detailed description of implemented operation modes and examples of usage.
Chapter 3, *Integrating the QSPI Flash Controller*	This chapter describes some guidelines and recommendations which should be taken into consideration during integration of the QSPI Flash Controller in SoC.
Chapter 4, *PHY Module Specification*	This chapter provides all data related to the optional PHY Module. It includes the exemplary programming sequences and implementation details.

3. Acronyms and Abbreviations

The following acronyms and abbreviations are used in this manual:

Table 1. Acronyms and Abbreviations

Term	Meaning
FSM	Finite State Machine
SRAM	Static Random Access Memory
POR	Power On Reset
DAC	Direct Access Controller

Term	Meaning
INDAC	Indirect Access Controller
STIG	Software Triggered Instruction Generator
XIP	EXecute In Place
WEL	Write Enable Latch
DLL	Delay Line
QSPI	Quad Serial Peripheral Interface
DMA	Direct Memory Access
MSB	Most Significant Byte
LSB	Least Significant Byte
FIFO	First In First Out
AMBA	Advanced Microcontroller Bus Architecture
AXI	Advanced eXtensible Interface
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
PHY	Physical Layer of the OSI Model
RTL	Register Transfer Level

4. Typographic Conventions

The following conventions are used throughout this guide:

Table 2. Typographic Conventions

Typographic Convention	Description
courier font	Indicates code.
italic	Indicates signals. For example: *mem_reset*
MSB and LSB	Bits within registers are always listed from MSB to LSB.
AA [n:0]	Indicates the bits within a register when more than one bit is involved in a description. For description purpose, the bits are presented as if they were contiguous within a single register. However, this is not always the case. Refer to Programming Interface for exact bit locations.
0x	Indicates the Hex values. For example, 0xFFFB is the value written to a 2-byte field in a register.
h	Alternative indication of the Hex values. For example, 'h5 is the value written to a 4-bit field in a register or a 4-bit signal. To make some expressions more readable, it is permitted to insert size of the logic vector before apostrophe. For example, 32'h0 means the 32 bit vector filled with zeroes.
b	Indicates binary values. For example, 'b0110 is the value written to a 4-bit field in a register or a 4-bit signal. To make some expressions more readable, it is permitted

Typographic Convention	Description
	to insert size of the logic vector before apostrophe. For example, 32'b0 means the 32 bit vector filled with zeroes.
n/a	Indicates that corresponding data is not applicable.
assert and de-assert	Assert: an active-high signal is pulled high or that an active-low signal is pulled low to ground. The word de-assert means that an active-high signal is pulled low to ground or that an active-low signal is pulled high.
Fields	Any register may contain multiple fields or a single field.
Reserved fields	When a field in a table is described as reserved, the controller shall ignore writes to this field. When the host tries to read these fields, it will get all zeros values.

5. Getting Help

If you have any problems with using this product or understanding the documentation, you can submit a Service Request (SR) to Cadence Support. When doing so, you must provide enough information about the problem so that it can be investigated efficiently. Describe the problem in detail, provide the version of the controller you are using, and state the exact circumstances in which the problem occurs.

5.1. Service Requests

Service Requests are your way of giving feedback, asking questions, getting solutions, and reporting problems. Unless told otherwise, Cadence support staff will respond to your service request. If Cadence Support cannot answer your question, Cadence Research and Development personnel will get involved. It is important to specify the severity level of the service request as accurately as possible.

There are three levels of severity:

- Critical — You cannot proceed without a solution to the issue.
- Important — You can proceed, but you need a solution to the issue.
- Minor — You prefer to have a solution, but you can wait for it.

Note

You can request support to increase the severity level of an issue. Therefore, do not use Critical unless immediate resolution of an issue is absolutely necessary and urgently required.

5.2. Using Cadence On-line Support

Cadence encourages you to submit requests using Cadence On-line Support. With Cadence On-line Support you can also track your open service requests.

To use Cadence On-line Support to submit a service request:

1. If you do not have a Cadence On-line Support account, go to http://support.cadence.com and under the heading **New User?**, click *Register Now*.

   You must provide a valid *Host-ID* for any product. The *Host-ID* is contained in the SERVER line of your product license file.

Note

If you already have a Cadence On-line Support account, then you only need to update your Cadence On-line Support preferences to include a valid *Host-ID* for a product.

2. Log in to Cadence On-line Support, and on the page, under the heading **My Requests**, click *CREATE SERVICE REQUEST*.

   A form opens for submitting your service request. Select *Design IP* in the *Product* list box. Also fill the other fields that are mandatory and press **Continue** to complete the Service Request.

6. Additional information

For more information on Cadence and its products, please visit: http://www.cadence.com

Part I. User Guide

Table of Contents

1. Module Data

The following topics are discussed in this chapter:

1.1. General Description

This chapter covers general description of the design architecture. It presents the block diagram with all implemented IO signal interfaces.

1.1.1. Features

The main features of Cadence QSPI Flash Memory Controller (for the configuration described in this document):

- Memory mapped ‘direct’ mode of operation for performing FLASH data transfers and executing code from FLASH memory
- Software triggered ‘indirect’ mode of operation for performing low latency and non processor intensive FLASH data transfers
- Optional DMA Peripheral interface to communicate indirect mode status with external DMA
- Local SRAM of configurable size to reduce AHB overhead and buffer FLASH data during indirect transfers
- Set of software APB accessible FLASH control registers to perform any FLASH command, including data transfers up to 8-bytes at a time
- Additional addressable Memory Bank to accommodate more than 8-bytes at a time
- Supports any device clock frequency, including current market device frequencies of 133MHz SDR or 80 MHz DDR
- Support for XIP (Execute in Place), sometimes referred to as continuous mode
- Support for DDR Mode and DTR Protocol
- Support for single, dual or quad I/O instructions
- Programmable device sizes
- Programmable write protected regions to block system writes from taking effect
- Programmable delays between transactions
- Legacy mode allowing software direct access to low level transmit and receive FIFOs, bypassing the higher layer processes

- An independent reference clock to decouple AHB clock from SPI clock – allows slow system clocks
- Programmable baud rate generator to generate divided clocks
- Features included to improve high speed read data capture mechanism
- Option to use adapted clocks to further improve read data capturing
- Programmable interrupt generation
- Up to four external device selects
- Programmable AHB decoder, enables continuous addressing mode for each of connected devices and auto-detection of boundaries between devices
- Support BOOT mode
- Full integration with PHY Module dedicated to more flexible and power efficient transfers

1.1.2. Block Diagram

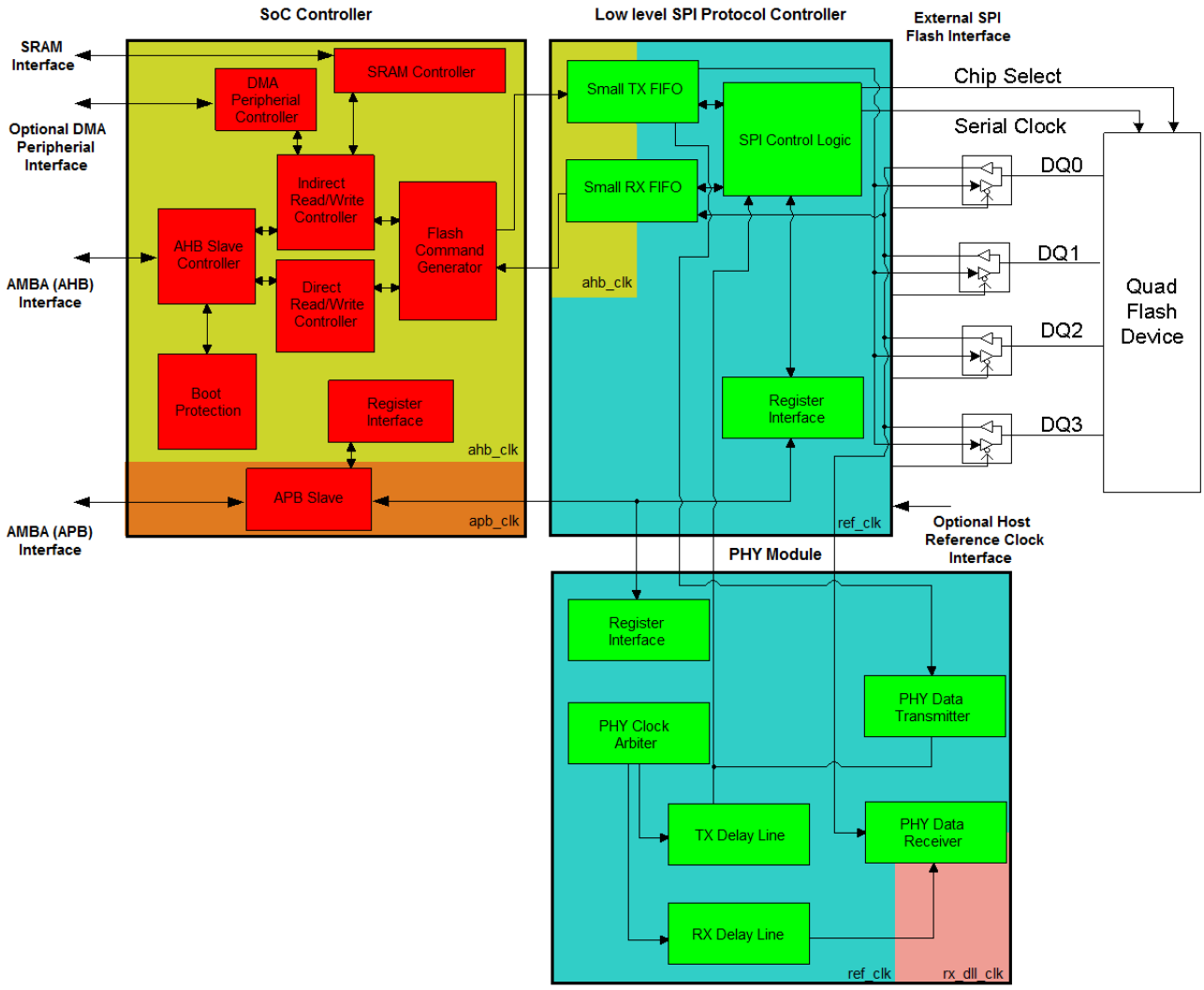
The QSPI Flash Controller (QSPI_FLASH_CTRL) can be used to provide access to Serial Flash devices. Standard Serial Peripheral Interface (SPI) is supported along with high performance Dual and Quad SPI variants.

The QSPI_FLASH_CTRL connects to an SoC environment through its AMBA AHB bus and APB bus interfaces. The AHB interface is used to transfer data, either in a memory mapped direct fashion (for example a processor wishing to execute code directly from external FLASH memory), or in an indirect fashion where the controller is set up via configuration registers to silently perform some requested operation, signalling its completion via interrupts or status registers. For indirect operations, data is transferred between system memory and external FLASH memory via an internal SRAM which is loaded for writes and unloaded for reads by an AHB master within the SoC environment at low latency AHB system speeds. Interrupts or status registers are used to identify the specific times at which this SRAM should be accessed using user programmable configuration registers. The size of the SRAM is configurable. An optional DMA peripheral bus is also available to optimize the data transfers between an external master and QSPI_FLASH_CTRL during indirect transfers.

There is an option to integrate PHY Module designed to ensure more granularity of SPI data transfers. More detailed description of this sub-block is covered in Chapter 4, *PHY Module Specification*.

Figure 1.1 shows a block diagram of the QSPI Flash Memory Controller and PHY.

Figure 1.1. Cadence QSPI Flash Memory Controller Architecture



1.1.3. Signal Interfaces

The QSPI_FLASH_CTRL includes the following signal interfaces:

- AMBA Advanced Peripheral Bus (APB) slave interface for accessing the programmable registers
- AMBA Advanced High Speed Bus (AHB) slave interface for data transfer
- An optional DMA peripheral interface

The AMBA interfaces fully conform to the ARM AMBA Revision 2.0 specification.

Table 1.1. AMBA APB Interface

Signal Name	I/O	Reset	Clock Domain	Description
n_preset	I	n/a	pclk	Active low AMBA reset (*nPRESET*). This signal must be asserted low asynchronously and de-asserted high synchronously with *pclk*. Resets all APB registers.
pclk	I	n/a	n/a	Peripheral bus clock (*PCLK*).
psel	I	n/a	pclk	Peripheral select (*PSEL*). Active high select signal to indicate that a valid access is being made to one of the programmable registers.
penable	I	n/a	pclk	Peripheral enable (*PENABLE*). This indicates the second clock cycle of an access and indicates that the write data may be strobed into a register on the next rising edge of *pclk*, or that the read data is expected to be valid at the next rising edge of *pclk*.
pwrite	I	n/a	pclk	Peripheral write strobe (*PWRITE*). This indicates that a write access is taking place (if *psel* is active).
paddr[7:0]	I	n/a	pclk	Address bus from selected master (*PADDR*). Indicates which register is being accessed. This address is word-aligned and therefore bits [1:0] should always be 'b00.
pwdata[31:0]	I	n/a	pclk	Write data. Data to be written into the addressed register.
prdata[31:0]	O	32'h0	pclk	Read data. Data read from the addressed register. For APB Rev 2.0, it is driven to logic 0 when not addressed.
pslverr	O	1'b0	pclk	Indicates an APB access to an invalid address. Should be sampled when *penable* is high. This signal is optional and not a standard AMBA signal.
pready	O	1'b1	pclk	Indicates the current APB access is to be delayed with wait states. The controller provides *pready* being tied high – no wait states needed.
interrupt	O	1'b0	pclk	Level sensitive active high interrupt.

Table 1.2. AMBA AHB Interface

Signal Name	I/O	Reset	Clock Domain	Description
n_hreset	I	n/a	hclk	Active low asynchronous reset. This signal must be asserted low asynchronously and de-asserted high synchronously with *hclk*.
hclk	I	n/a	n/a	AHB bus clock.
hsel	I	n/a	hclk	AHB slave select. This signal should be only asserted during valid AHB transfer between external master and the controller.

Signal Name	I/O	Reset	Clock Domain	Description
hready_in	I	n/a	hclk	Combined form of *hready* from all slaves in AHB system.
htrans[1:0]	I	n/a	hclk	AHB transfer type as follows: • 'b00 - Idle • 'b01 - Busy • 'b10 - Non-sequential • 'b11 - Sequential
haddr[31:0]	I	n/a	hclk	AHB address.
hwrite	I	n/a	hclk	High for AHB write, low for AHB read.
hsize[2:0]	I	n/a	hclk	Transfer size. Only 8,16 and 32 bit transfers are supported. Supported widths are as follows: • 'b000 - 8 bit transfers • 'b001 - 16 bit transfers • 'b010 - 32 bit transfers
hburst[2:0]	I	n/a	hclk	Burst type as follows (wrapping bursts are not supported for writes): • 'b000 - Single transfer • 'b001 - Incrementing burst • 'b010 - 4-beat wrapping burst • 'b011 - 4-beat incrementing burst • 'b100 - 8-beat wrapping burst • 'b101 - 8-beat incrementing burst • 'b110 - 16-beat wrapping burst • 'b111 - 16-beat incrementing burst
hwdata[31:0]	I	n/a	hclk	Write data from external AHB Master (32 bit only)
hrdata[31:0]	O	32'h0	hclk	Read data to external AHB Master (32 bit only). Unused bits will be tied to logic 0.
hresp	O	1'b0	hclk	Slave response as follows: • 'b0 - OK • 'b1 - not OK (corresponding interrupt is triggered if unmasked)

Signal Name	I/O	Reset	Clock Domain	Description
				AMBA split and retry operations are not supported.
hready_out	O	1'b0	hclk	*hready* generated by the Flash Controller slave. When low, the flash controller is inserting wait states.

Table 1.3. Host Reference Clock Interface

Signal Name	I/O	Description
n_ref_rst	I	Active low ref_clk reset. This signal must be asserted low asynchronously, and de-asserted high synchronously with ref_clk. Resets all registers in the reference clock domain.
ref_clk	I	Reference clock

Table 1.4. SRAM Interface

Signal Name	I/O	Reset	Clock Domain	Description
sram_cs	O	1'b0	hclk	Active high chip select.
sram_we	O	1'b0	hclk	Active high write enable.
sram_re	O	1'b0	hclk	Active high read enable.
sram_add[N:0]	O	(N+1)'h0	hclk	SRAM address bus with address width being configurable.
sram_do[31:0]	O	32'h0	hclk	SRAM write data bus.
sram_di[31:0]	I	n/a	hclk	SRAM read data bus.

The SRAM ports (used for indirect modes of data transfer only) are passed to the top level.

Table 1.5. External SPI Flash interface

Signal Name	I/O	Reset	Clock Domain	Description
mi0	I	n/a	async	Multiple-SPI master mode input, bit 0 The master input may be up to 4-bits wide in master mode, with 1-bit for standard serial SPI operation, 2-bits for Dual-SPI, and 4-bits for Quad-SPI. Internally, the Quad-SPI IP must switch between mi[1] for 1-bit wide input and mi[0] for the LSB of dual-mode or quad-mode input.
mo0	O	1'b1	ref_clk	MOSI signal, Multiple_SPI master mode output, bit 0.
n_mo_en[0]	O	1'b1	ref_clk	Master mode output enable for bit 0. Active low. This pin retains its original function for serial SPI mode, but additionally will be driven dynamically for Dual- and Quad-SPI modes, based on the enabled mode and the current stage of the transfer.
mi1	I	n/a	async	MISO pin, Multiple-SPI master mode input, bit 1.

Signal Name	I/O	Reset	Clock Domain	Description
				The master input may be up to 4-bits wide in master mode, with 1-bit for standard serial SPI operation, 2-bits for Dual-SPI, and 4-bits for Quad-SPI. Internally, the Quad-SPI IP must switch between mi[1] for 1-bit wide input and mi[0] for the LSB of dual-mode or quad-mode input.
mo1	O	1'b1	ref_clk	MOSI signal, Multiple-SPI master mode output, bit 1.
n_mo_en[1]	O	1'b1	ref_clk	Master mode output enable for bit 1. Active low.
mi2	I	n/a	async	MISO pin, Multiple-SPI master mode input, bit 2. The master input may be up to 4-bits wide in master mode, with 1-bit for standard serial SPI operation, 2-bits for Dual-SPI, and 4-bits for Quad-SPI.
mo2_wpn	O	1'b1	ref_clk	MOSI signal, Multiple-SPI master mode output, bit 2, for quad-mode writes to external SPI slave and used for write protect output to flash devices, active low, when not in quad-mode.
n_mo_en[2]	O	1'b1	ref_clk	Master mode output enable for bit 2. Active low. Only utilized if quad-mode commands are performed.
mi3	I	n/a	async	MISO pin, Multiple-SPI master mode input, bit 3. The master input may be up to 4-bits wide in master mode, with 1-bit for standard serial SPI operation, 2-bits for Dual-SPI, and 4-bits for Quad-SPI.
mo3_hold	O	1'b1	ref_clk	MOSI signal, Multiple-SPI master mode output, bit 3, for quad-mode writes to external SPI slave and used for hold output to flash devices, active low, when not in quad-mode. NOTE: The use of the HOLD pin is not supported by the controller.
n_mo_en[3]	O	1'b1	ref_clk	Master mode output enable for bit 3. Active low. Only utilized if quad-mode commands are performed.
n_ss_out[3:0]	O	4'hf	ref_clk	Peripheral select outputs. The IP supplies up to 4 CS.
sclk_out	O	1'b0	ref_clk	Multiple-SPI master clock output.

Table 1.6. Optional DMA Peripheral Interface

Signal Name	I/O	Reset	Clock Domain	Description
indwr_drtype[1:0]	O	2'h0	hclk	Driven by controller to indicate the request type the indirect write logic wants the DMA to perform (single or burst).
indwr_drlast	O	1'b0	hclk	The last Indirect write DMA request
indwr_drvalid	O	1'b0	hclk	Indirect write controller request to DMA to perform a transfer

Signal Name	I/O	Reset	Clock Domain	Description
indwr_drready	I	n/a	hclk	Indication that DMA has accepted the indirect write DMA request
indwr_datype[1:0]	I	n/a	hclk	Driven by DMA to indicate to the controller the type of transfer that is being acknowledged. The controller only uses this signal to identify an incoming DMA flush request.
indwr_davalid	I	n/a	hclk	Driven by DMA to indicate that the Indirect write DMA request is completed.
indwr_daready	O	1'b1	hclk	Driven by controller to indicate that it saw the *indwr_davalid*.
indrd_drtype[1:0]	O	2'h0	hclk	Driven by controller to indicate the request type the indirect read logic wants the DMA to perform (single or burst).
indrd_drlast	O	1'b0	hclk	The last Indirect read DMA request
indrd_drvalid	O	1'b0	hclk	Indirect read controller request to DMA to perform a transfer
indrd_drready	I	n/a	hclk	Indication that DMA has accepted the indirect read DMA request
indrd_datype[1:0]	I	n/a	hclk	Driven by DMA to indicate to the controller the type of transfer that is being acknowledged. The controller only uses this signal to identify an incoming DMA flush request.
indrd_davalid	I	n/a	hclk	Driven by DMA to indicate that the Indirect read DMA request is completed.
indwr_daready	O	1'b1	hclk	Driven by controller to indicate that it saw the *indrd_davalid*.

Table 1.7. Adapted Loopback Device Clock

Signal Name	I/O	Reset	Clock Domain	Description
sclk_dlyd_i	I	n/a	async	This is an optional clock, used by the read delay capture circuit to sample the read data as it returns from the memory device. It is a delayed version of the "sclk_out" output from the controller. The delay itself is implemented externally. Refer to Section 2.1.11.2, *"Mechanism using external DLL"* for further details.

Table 1.8. DFT Mode

Signal Name	I/O	Reset	Clock Domain	Description
scanmode	I	n/a	ref_clk	This signal specifies the operation mode as follows: • 0 - Normal Operation Mode

Signal Name	I/O	Reset	Clock Domain	Description
				• 1 - Scan Mode (DFT Mode)

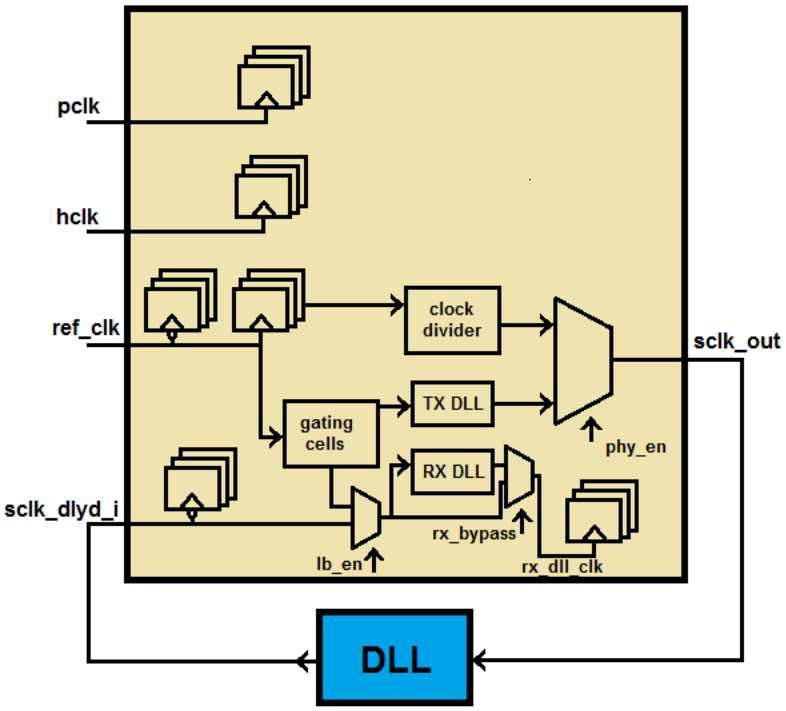
Table 1.9. PHY Module Interface

Description
The PHY Module Interface signals are listed and described in Chapter 4, *PHY Module Specification*.

1.1.4. Clock Diagram

AHB, APB and reference clocks are intended to being generated continuously. There is also sclk_out which is the generated clock passing further outside the chip into clock input of the Flash Device. QSPI Flash Memory Controller and PHY.

Figure 1.2. Cadence QSPI Flash Memory Controller Clock Architecture

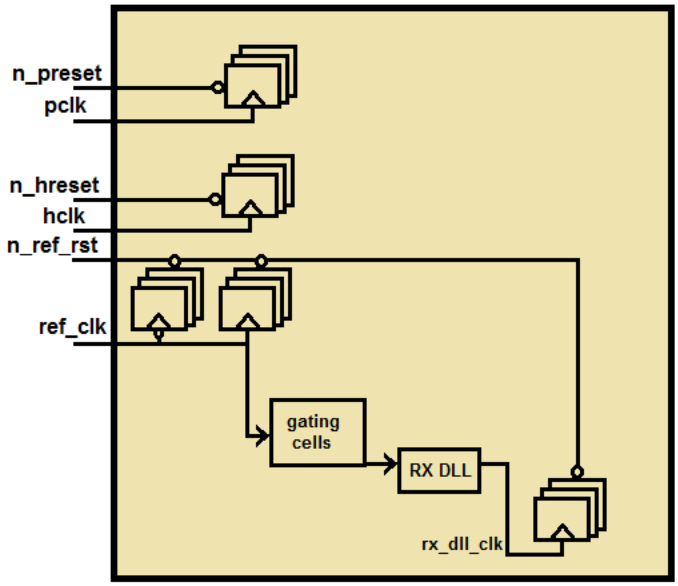


1.1.5. Reset Diagram

There are three asynchronous resets present in the design, one for each of the main clock domains (*ref_clk*, *hclk* and *pclk*). Internally generated *rx_dll_clk* is reset using *n_ref_rst* signal. These resets are used directly to reset the synchronous parts of the design. The reset signals may be set low asynchronously with respect to the clock, but must be set high synchronously. Since *sclk_dlyd_i* clock is used for sampling data only – there is no reset for this domain.

Figure 1.3 shows a reset diagram of the QSPI Flash Memory Controller and PHY.

Figure 1.3. Cadence QSPI Flash Memory Controller Reset Architecture



1.2. Timing Requirements

Table 1.10. APB Timings

Parameter	Description	Min	Max	Unit
Tclk	Clock period	10	undefined	ns
Tclkh	Clock high	n/a	n/a	
Tclkl	Clock low	n/a	n/a	
Tihnres	*n_preset* hold after *pclk*	n/a	n/a	
Tisnres	*n_preset* setup before *pclk*	n/a	n/a	
Tihpen	*penable* hold after *pclk*	0.5	-	ns
Tispen	*penable* setup before *pclk*	4.0	-	ns
Tihpel	*psel* hold after *pclk*	0.5	-	ns
Tispel	*psel* setup before *pclk*	4.0	-	ns
Tihpa	*paddr* hold after *pclk*	0.5	-	ns
Tispa	*paddr* setup before *pclk*	4.0	-	ns
Tihpw	*pwrite* hold after *pclk*	0.5	-	ns
Tispw	*pwrite* setup before *pclk*	4.0	-	ns
Tihpwd	*pwdata* hold after *pclk*	0.5	-	ns
Tispwd	*pwdata* setup before *pclk*	4.0	-	ns
Tohprd	*prdata* hold after *pclk*	0.5	-	ns
Tovprd	*paddr* valid after *pclk*	4.0	-	ns

Figure 1.4. APB Timings

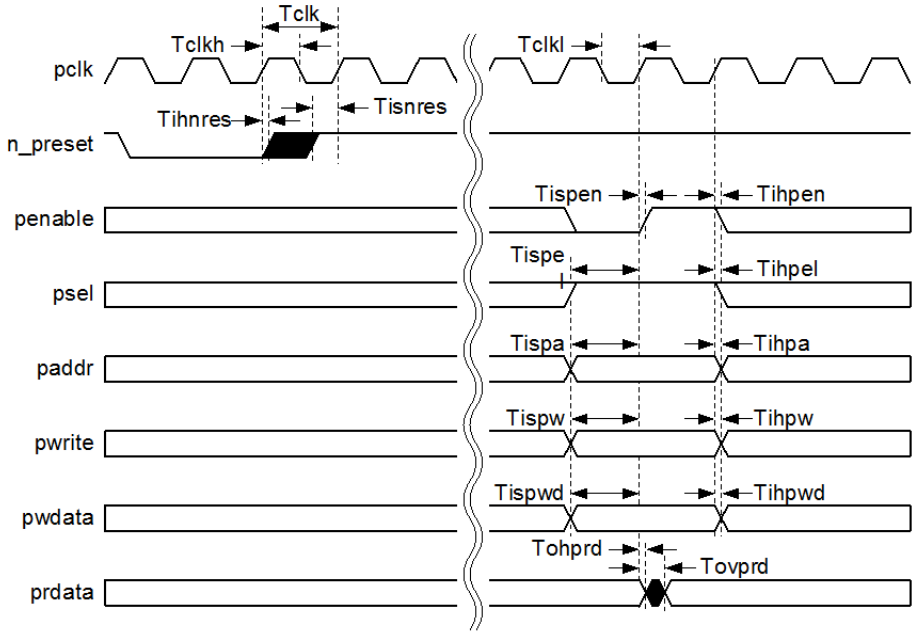


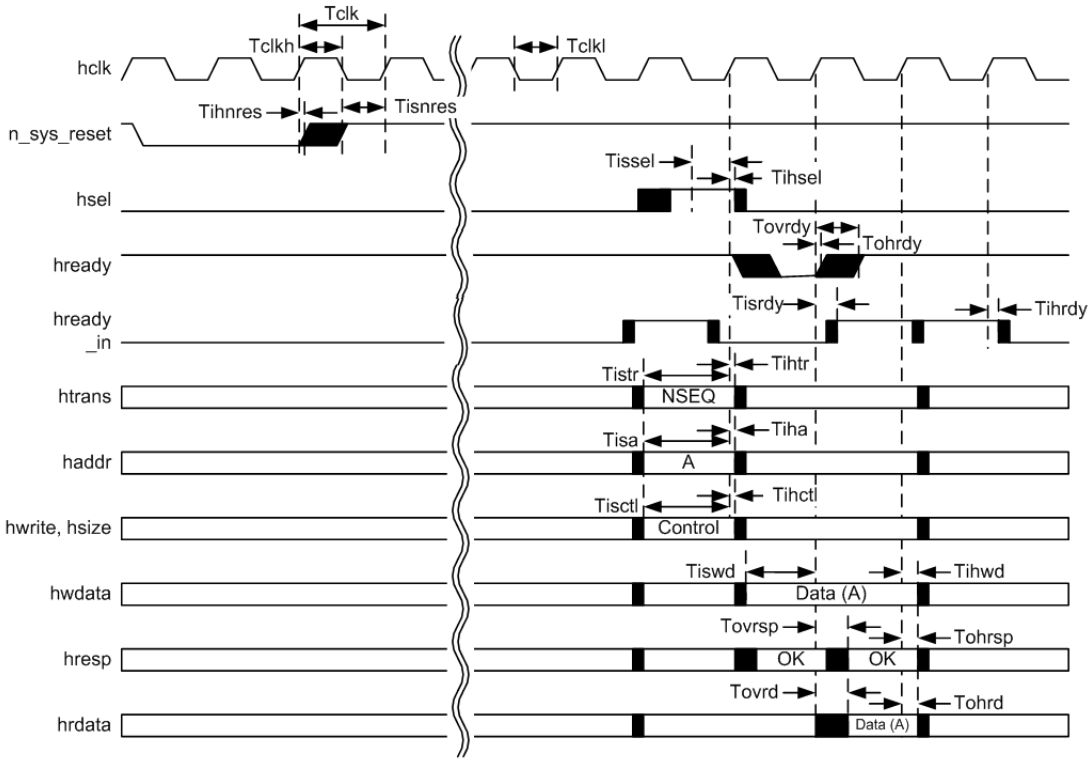
Table 1.11. AHB Timings

Parameter	Description	Min	Typ	Max	Unit
Tclk	Clock period, T0	1.6	10	undefined	ns
Tclkh	Clock high	40%	40%	-	T0
Tclkl	Clock low	40%	40%	-	T0
Tihnres	*n_sys_reset* hold after *hclk*	n/a	n/a		T0
Tihsres	*n_sys_reset* setup before *hclk*	n/a	n/a		T0
Tissel	*hsel* setup before *hclk*	60%	60%	-	T0
Tihsel	*hsel* hold after *hclk*	0.1	0.1	-	ns
Tohrdy	*hready_out* hold after *hclk*	0.1	0.1	-	ns
Tovrdy	*hready_out* valid after *hclk*	-	-	50%	T0
Tihrdy	*hready_in* hold after *hclk*	0.1	0.1	-	ns
Tisrdy	*hready_in* setup before *hclk*	60%	60%	-	T0
Tihtr	*htrans* hold after *hclk*	0.1	0.1	-	ns
Tistr	*htrans* setup before *hclk*	60%	60%	-	T0
Tiha	*haddr* hold after *hclk*	0.1	0.1	-	ns
Tisa	*haddr* setup before *hclk*	60%	60%	-	T0
Tihctl	*hwrite,hsize* hold after *hclk*	0.1	0.1	-	ns
Tisctl	*hwrite,hsize* setup before *hclk*	60%	60%	-	T0
Tihwd	*hwdata* hold after *hclk*	0.1	0.1	-	ns
Tiswd	*hwdata,hsize* setup before *hclk*	60%	60%	-	T0

Parameter	Description	Min	Typ	Max	Unit
Tohrsp	*hresp* hold after *hclk*	0.1	0.1	-	ns
Tovrsp	*hresp* valid after *hclk*	-	-	50%	T0
Tohrd	*hrdata* hold after *hclk*	0.1	0.1	-	ns
Tovrd	*hrdata* valid after *hclk*	-	-	50%	T0

Maximum achieved AHB frequency is equal to 600 MHz so this requirement was inserted into the table. There are some limitations which could hamper correct work on such high frequency depending on individual requirements of the system. Reference value of this clock is 100 MHz.

Figure 1.5. AHB Timings



NOTE:

All timings are specified relative to the target clock period, *pclk*/*hclk*. These timings are coded into the provided synthesis script, and have been achieved with a typical technology, but are for guidance only.

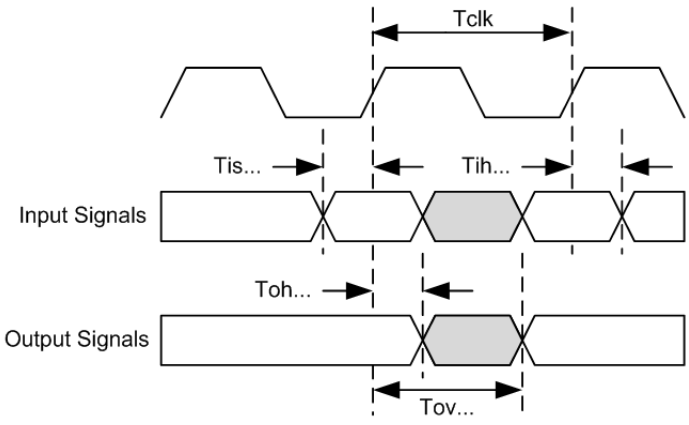
Table 1.12. Multiple-SPI Timings

Parameter	I/O	Description	Min	Max	Unit
Tclk		Clock period, T0	1.25	undefined	ns
Tihmi0	I	*mi0* hold after *ref_clk*	5%	-	Tclk
Tismi0	I	*mi0* setup before *ref_clk*	-	35%	Tclk
Tihmi1	I	*mi1* hold after *ref_clk*	5%	-	Tclk

Parameter	I/O	Description	Min	Max	Unit
Tismi1	I	*mi1* setup before *ref_clk*	-	35%	Tclk
Tihmi2	I	*mi2* hold after *ref_clk*	5%	-	Tclk
Tismi2	I	*mi2* setup before *ref_clk*	-	35%	Tclk
Tihmi3	I	*mi3* hold after *ref_clk*	5%	-	Tclk
Tismi3	I	*mi3* setup before *ref_clk*	-	35%	Tclk
Tohmo0	O	*mo0* hold after *ref_clk*	5%	-	Tclk
Tovmo0	O	*mo0* valid after *ref_clk*	-	50%	Tclk
Tohmo1	O	*mo1* hold after *ref_clk*	5%	-	Tclk
Tovmo1	O	*mo1* valid after *ref_clk*	-	50%	Tclk
Tohmo2	O	*mo2_wpn* hold after *ref_clk*	5%	-	Tclk
Tovmo2	O	*mo2_wpn* valid after *ref_clk*	-	50%	Tclk
Tohmo3	O	*mo3_hold* hold after *ref_clk*	5%	-	Tclk
Tovmo3	O	*mo3_hold* valid after *ref_clk*	-	50%	Tclk
Tohmoen	O	*n_mo_en* hold after *ref_clk*	5%	-	Tclk
Tovmoen	O	*n_mo_en* valid after *ref_clk*	-	50%	Tclk
Tohsso	O	*n_ss_out* hold after *ref_clk*	5%	-	Tclk
Tovsso	O	*n_ss_out* valid after *ref_clk*	-	50%	Tclk

Most outputs are registered on the ref_clk. There is no combinatorial logic on these outputs. Master Output Interface which is directly forwarded to external Flash Device has multiplexer on its output for the optional delay inserting feature which is used to adjust hold timing. Given that this interface toggles based on programmable divider synchronous for ref_clk – multi cycle path is ensured, therefore additional combinational delay is safe in this case. For the inputs, these are all data inputs, driven on one edge of sclk_in and sampled by the QSPI Flash controller on the ref_clk.

Figure 1.6. Multiple-SPI Timings



NOTE1:

Max timings are specified relative to the target clock period, Tclk. These are coded into the synthesis script provided. These timings have been achieved with a typical technology, but are for guidance only.

NOTE2:

Hold times specified are guidelines only. Min timings have no effect during synthesis.

NOTE3:

Minimum Tclk is limited by the requirements of the individual Flash Device. When PHY Mode is enabled. I.e. if maximum allowed SPI clock rate is 100 MHz, Tclk (min) takes 10 ns (no longer 2.5 ns).

1.3. Clock Domains

There are three main asynchronous clock sources for the QSPI Controller and one generated clock for clocking the external FLASH device. The AHB clock is the main system clock used to transfer data over the AHB bus between an external AHB master and the QSPI controller. The APB clock is used to access the register map of the QSPI controller and perform basic configuration and interrupt handling. The reference clock is used to serialize the data and drive the external SPI interface. Using the reference clock allows the core to decouple the frequency of the SPI FLASH device from the SoC system clocks, thereby providing a more flexible clocking solution.

1.4. Programming Interface

The following registers may be programmed for the QSPI Flash Controller module:

Table 1.13. Controller Register Map

Offset	Function	R/W
0x00	QSPI Configuration Register	R/W
0x04	Device Read Instruction Register	R/W
0x08	Device Write Instruction Register	R/W
0x0C	QSPI Device Delay Register	R/W
0x10	Read Data Capture Register	R/W
0x14	Device Size Register	R/W
0x18	SRAM Partition Register	R/W
0x1C	Indirect AHB Address Trigger Register	R/W
0x20	DMA Peripheral Register	R/W
0x24	Remap Address Register	R/W
0x28	Mode Bit Register	R/W
0x2C	SRAM Fill Level Register	RO
0x30	TX Threshold Register	R/W
0x34	RX Threshold Register	R/W
0x38	Write Completion Control Register	R/W
0x3C	Polling Expiration Register	R/W
0x40	Interrupt Status Register	R/W
0x44	Interrupt Mask Register	R/W
0x50	Lower Write Protection Register	R/W
0x54	Upper Write Protection Register	R/W

Offset	Function	R/W
0x58	Write Protection Register	R/W
0x60	Indirect Read Transfer Register	R/W
0x64	Indirect Read Transfer Watermark Register	R/W
0x68	Indirect Read Transfer Start Address Register	R/W
0x6C	Indirect Read Transfer Number Bytes Register	R/W
0x70	Indirect Write Transfer Register	R/W
0x74	Indirect Write Transfer Watermark Register	R/W
0x78	Indirect Write Transfer Start Address Register	R/W
0x7C	Indirect Read Transfer Number Bytes Register	R/W
0x80	Indirect AHB Trigger Address Range Register	R/W
0x8C	Flash Command Control Memory Register	R/W
0x90	Flash Command Control Register	R/W
0x94	Flash Command Address Register	R/W
0xA0	Flash Command Read Data Register (Lower)	RO
0xA4	Flash Command Read Data Register (Upper)	RO
0xA8	Flash Command Write Data Register (Lower)	R/W
0xAC	Flash Command Write Data Register (Upper)	R/W
0xB0	Polling Flash Status Register	R/W
0xFC	Module ID Register	RO

Table 1.14. PHY Register Map

Offset	Function	R/W
0xB4	PHY Configuration Register	R/W
0xB8	PHY DLL Master Configuration Register	R/W
0xBC	DLL Observable Register (Lower)	RO
0xC0	DLL Observable Register (Upper)	RO

1.5. Physical Estimates

Approximate two input NAND equivalent gate counts (TSMC28HPM):	92k

2. Programming the QSPI Flash Controller

The following topics are discussed in this chapter:

- Functional Description and Operation
- Programmer's Guide
- Programming Interface
- Performance Characteristics

2.1. Functional Description and Operation

2.1.1. AHB Control Interface

The AHB slave controller validates incoming AHB accesses, responds to invalid requests, performs any required byte and half-word reordering, blocks writes that violate the programmed write protection rules (only for direct access) and forwards the transfer request to either the direct access controller or the indirect access controller.

2.1.1.1. AHB Interface

The AHB interface conforms to the AMBA 3 AHB-Lite protocol specification from ARM. Locked transfers (HMASTLOCK) and transfers using the protection control signals (HPROT) are not supported. The AHB data-width is 32 bits wide. Therefore only byte, half-word and word accesses will be permitted. For writes, only incrementing bursts are supported, wrapping write bursts received will generate an error i.e. the access completes with HRESP set to error. INCR16, INCR8, INCR4, INCR and SINGLE type bursts are permitted. For reads all burst types, including WRAPs, are supported. The slave will operate correctly if a burst is terminated early due to interconnect burst termination or if the slave errors an access within the burst.

2.1.1.2. Remapping the AHB address

The QSPI Flash Controller does not implement any specific address decoding to error incoming addresses that may lie outside the connected FLASH memory space but the AHB address decoder is implemented. If it is enabled, the QSPI Flash Controller is able to detect address range of individual device to give an option to auto-switching of active devices based on address from AHB. The incoming AHB address, by default, maps directly to the address sent serially to the FLASH device. If the FLASH device has a 24-bit address, then the 24 LSB’s of the AHB address will be forwarded.

A remap feature is available to remap all incoming AHB addresses to ADDRESS+N, where N is the value store in the Remap Address Register (0x24). It is enabled via the QSPI Configuration Register. This feature could be used when software needs to move boot code to another FLASH region.

2.1.1.3. Write Protection

In order to protect the FLASH device, a write protection feature is implemented in hardware and controlled from software. Any AHB write detected (by using DAC controller), pointing to an area of FLASH that is protected, will generate an error, indicated on the HRESP error output.

A programmable region of the FLASH device, defined as a number of FLASH "blocks" starting from a particular block number can be protected. Three programmable registers (Lower Write Protection Register, Upper Write Protection Register and Write Protection Register) are provided. The first register defines the FLASH block that is located at the bottom of the region to be protected. The second register defines the FLASH block that is located at the top of the region

to be protected. The third register is a control register consisting of 2 bits. Bit 0 allows software to invert the region that is being protected, causing the programmed region to become the only areas of FLASH memory that is not protected from writes. The bit 1 is the write protection enable bit. When low, the FLASH device is unprotected.

For implementation, the AHB controller must map the incoming AHB address into its associated FLASH block. A block can be between 1 and 65K bytes, programmed via the Device Size Configuration Register register.

2.1.1.4. Access Forwarding

For legal accesses, the AHB controller will forward all AHB accesses to one of two access controllers: the direct access and indirect access controllers. Assuming the direct access controller has been enabled via the QSPI Configuration Register (0x00), then by default all AHB accesses will be forwarded to the direct access controller. Before any accesses can be forwarded to the indirect access controller, it must first be configured by software. This process is fully explained in Section 2.1.3, *"Indirect Access Controller (INDAC)"*. If the direct access controller is disabled, any incoming AHB access that cannot be forwarded to the indirect access controller will be completed immediately with an HRESP error. If DAC is enabled, the same access will be forwarded and serviced by it.

2.1.1.5. Sequential Access Detection and Burst Length

In order to maximize performance, the AHB interface signal "htrans" is not used by the controller to help identify sequential and non-sequential accesses. This is because "htrans" identifies all accesses at the start of a new AHB burst as non-sequential, as well as all accesses after a 1K boundary even when the access may be sequential to the last. Instead non-sequential accesses are detected by comparing the address of the current access with the address of the previous access. A decision is then made as to whether the access is sequential or not. If the direction of the access (write/read) has changed, then the access is deemed non-sequential. If the size of the read access (byte/half-word/word) has changed, then the access is also non-sequential. If AHB address decoder is enabled and transition between devices has been detected, then the new SPI transfer has to be generated because currently selected device needs to be initialized. This is the only case when accesses are considered as non-sequential despite addresses are consecutive. Otherwise the access is sequential if the address of the current transfer is sequential when compared the previous transfer. In addition, the "hburst" signal from AHB is not used by the controller to identify burst lengths. Instead a burst continues until a non-sequential access is received.

2.1.1.6. AHB Address Decoder

It is possible to connect up to four devices to the controller. Emerging memories reached size of 2Gb and they are still increasing over this parameter. To ensure high performance in sequential transfer of data greater than a few Gb, AHB Address Decoder can be enabled. Address range from AHB standpoint is merged to continuous form. It allows the software driver to manage of accesses without additional steps to detect address boundaries between devices. Switching is performed automatically by the controller based on actual address from AHB interface. Boundaries of addresses are calculated by hardware, based on programmable configuration from Device Size Configuration Register. Each connected device can have another size ranged from 512Mb to 4Gb. For example, if each device was configured to have density of 512Mb, the inferred address range will be from address of 0x0000000 to 0x0FFFFFF what calculates into 4*512Mb = 2Gb of continuous address area. CS[0] is assigned to part from 0x0000000 to 0x03FFFFF, CS[1] to part from 0x0400000 to 0x07FFFFF and so on. AHB Address Decoder should be used only for direct read transfers. Data to read should be stable to avoid necessity of polling each device after switching between them if write access was issued before. To make sure about stability of the data, it is recommended to do first reads separately for each device without using of AHB decoder.

2.1.2. Direct Access Controller (DAC)

Direct access refers to the operation where AHB accesses directly trigger a read or write to FLASH memory. It is memory mapped and can be used to both access and directly execute code from external FLASH memory. Any incoming AHB access that is not recognized as being within the programmable indirect trigger region is assumed to be a direct access

and will be serviced by the direct access controller. Note that accesses that use the direct access controller do not use the embedded SRAM. The AHB will be throttled as the read or write burst is carried out, and the amount of wait states applied will be dependent on the latency through the controller. The latency has been carefully designed to be as small as possible. Latency is kept to a minimum when the use of XIP read instructions are enabled.

When servicing AHB reads, the DAC will always send one additional access downstream than was originally issued by the AHB in any single AHB burst. This is invisible from the system interface. It is a predicted read and ensures the underlying SPI core can operate at maximum bandwidth. The AHB burst defined here is not an AHB burst as defined by AMBA. It is defined as follows:

1. The first access of the AHB burst is defined by:
    a. An AHB access that is NON-SEQUENTIAL (Non-sequential based on address comparison, not based on htrans).
    b. An AHB access that is NON-SEQUENTIAL or SEQUENTIAL when the downstream modules are IDLE.
2. The last access of the AHB burst is defined by an AHB access that is SEQUENTIAL and precedes a new burst as defined in 1.
3. The size of an AHB burst is the number of AHB accesses, starting from the first access and ending with the last one
4. The number of DAC requests per AHB burst is equal to the size of the AHB burst+1

When AHB clock is very slow comparing to the SPI data rate, the sequential read transfer might be interrupted on SPI side and then continued just after slow AHB clock had accepted the last data. The limiting value is getting narrower as SPI clock divider and AHB access size (hsize) go down and number of data IOs (single, dual or quad) go up. If the system is not able to meet SPI data rate, it is recommended to use higher SPI clock divider what will decrease SPI data rate but makes the necessity of passing through opcode+address+dummy before each transfer part not to care what increases overall performance by keeping data transfer being continuous. There is not formal requirement covering marginal case value of AHB clock since phase relationship between this clock and reference one is unknown (they are asynchronous with respect to each other). The architecture of the controller was designed to be able to work on high data rates. The presence of predicted read can lead into the rubbish data during mixed AHB accesses on slow system data rate. Mixed accesses mean the Direct AHB sequence of single non-sequential access following a few sequential ones. The predicted read cannot be dropped because of SPI transfer interruption therefore it is sent as separate SPI transaction what creates rubbish on the interface for one access. To avoid this issue, it is recommended just to drop this data by system or just introduce wait states on AHB before issuing non-sequential direct AHB access. In spite of that, usage of Indirect Read Transfer is worth to consider instead for described working conditions.

For AHB writes, the direct access controller will trigger a sequence of write commands that mimic the way reads are processed, although the number of DAC requests for writes is equal to the number of AHB write requests received. For writes, the AHB controller will ensure FLASH bursts will not break a FLASH page boundary. When a page boundary is detected, only the number of byte accesses up to that boundary will be forwarded. A sequential direct write request that crosses a page boundary must be detected as non-sequential to downstream modules, causing the downstream controller to force the Flash device to enter a self timed page program cycle. The Controller supports splitting writes crossing page boundaries only for word aligned addresses.

The external AHB master supplying the data for writes should attempt to guarantee write data for a particular page can be provided to the controller in a timely fashion to avoid downstream starvation. If there is too much delay in the system issuing a sequential write, then the FLASH write cycle may be initiated prematurely, reducing the effective life of the device. Note that if this cannot be guaranteed and is of concern, then the indirect write operation can instead be used to avoid this issue.

FLASH erase operations, which may be required before a page write, are triggered by software using the documented programming interface Section 2.1.5, *"Software Triggered Instruction Generator (STIG)"* (detailed later). Once a page program cycle has been started, the QSPI Flash Controller will automatically poll for the write cycle to complete before

allowing any further AHB accesses to complete. This is achieved by holding any subsequent AHB direct accesses in wait state.

2.1.3. Indirect Access Controller (INDAC)

2.1.3.1. Indirect Read Controller

The aim of the indirect mode of operation is to read significant numbers of bytes from FLASH memory without requiring an AHB access to trigger it. Instead indirect operations are controlled and triggered by software via specific APB control/ configuration registers (Indirect Read Transfer Control Register, Indirect Read Transfer Watermark Register, Indirect Read Transfer Start Address Register and Indirect Read Transfer Number Bytes Register). This block will communicate with an embedded low level SPI protocol state machine module to perform an efficient and optimized FLASH read burst, placing the read data into the local SRAM module ready for fast and low latency delivery to any external AHB master.

By default, the indirect read controller is disabled. Before enabling it, software must configure how much data is required and starting from what address. The start address and total number of bytes to be fetched is defined in registers at addresses Indirect Read Transfer Start Address Register and Indirect Read Transfer Number Bytes Register respectively. Up to two indirect operations can be programmed at any one time. The second can be triggered while the first is in progress. Supporting two indirect operations allows a short turnaround time between the completion of one indirect operation and the start of the second. Refer to the Section 2.1.3.2.2, *"Indirect Access Queuing*" for further details.

The total number of bytes to read in an indirect operation is not limited by the size of the SRAM. The size of SRAM will only limit the size of requests made to the DMA (if the DMA peripheral interface module is enabled). In the case of an SRAM overrun, the controller will back pressure FLASH reads until space becomes available in the SRAM. Back pressuring the reads on the SPI interface is handled by completing any current read burst, waiting until space in the SRAM becomes available and then issuing a new read burst at the address where the previous terminated burst ended.

An external master will be able to fetch the data the controller has read from external FLASH memory by issuing AHB reads to the controller. The AHB address of the incoming read access must be in the range AHB Indirect trigger address (programmed via the Indirect AHB Address Trigger Register) to AHB Indirect trigger address + 2**(Indirect AHB trigger address range) - 1. Default value of the range is equal to 16 locations. This allows a 16-beat burst to be applied starting from the AHB Indirect trigger address. The smaller bursts are possible to handle effectively as well with this approach. Furthermore it is not strict requirement to push consecutive address sequence. Actual AHB address just has to be inside the Indirect Range to grant the SRAM as a source. Each valid AHB Indirect read will cause the internal SRAM to be popped, thereby decoupling the AHB address from the FLASH address – i.e. not direct mapped. Therefore AHB Indirect trigger address does not have any relationship with FLASH address. It is just to indicate that data should take SRAM as source instead of FLASH Memory array after triggering of any valid Indirect Read. FLASH address for Indirect Read is taken from the Indirect Read Transfer Start Address Register (0x68). Assuming the requested data is present in the SRAM at the point the AHB access is received by the QSPI controller, then the data will be fetched from SRAM and the response to the read burst will be achieved with minimum latency. Once the data has been read from SRAM, the QSPI controller will free up the associated resource in the SRAM.

If an AHB read access is received whose address is not within the range described above then that access will not be completed using the indirect controller. It will instead be serviced by the direct access controller.

If an AHB read access is received whose address is within the range described above but the requested data is not immediately present in the SRAM then AHB wait states will be applied until the data has been read from FLASH and pushed to the SRAM.

If an AHB read burst is received whose access elements traverse the AHB Indirect trigger range, then the accesses within the Indirect trigger range will be processed by the indirect controller and the rest will be taken by the direct access controller. Note this is likely to be a software configuration error.

The external master is only permitted to issue 32-bit AHB reads until the last word of an indirect transfer. This helps keep the SRAM control logic less complex. On the final read, the external master may issue a 16-bit (HalfWord) or byte access

to complete the transfer. It is also permitted for the external master to always issue a 32-bit Word read on the last indirect access. The controller will pad the upper bits of the response with zero.

The current expectation is that the SRAM will be kept fairly full while the read operation is carried out. The fill level of the SRAM is directly readable by software in SRAM Fill Level Register. If the DMA peripheral interface controller is enabled, this will automatically request an external DMA fetch the data from the SRAM via the AHB in efficient chunks of data, each chunk being a part of the overall indirect transfer.

An indirect operation may be cancelled at any time by setting Indirect Read Transfer Control Register bit[1].

Any bus master should be allowed to initiate an indirect access. The DMA bus peripheral interface has been included in the design to offload some of the necessary software overhead to efficiently manage data transfers when an external DMA supporting this interface is added. The alternative to this is for software to access the SRAM fill-level directly via APB registers and then decide for itself when the data should be fetched from the local SRAM. When the DMA peripheral interface is disabled, the fill level watermark register accessible via the Indirect Read Transfer Watermark Register (0x64) is provided. When the SRAM fill level passes this watermark, an interrupt is generated. If the watermark value is >0, the watermark interrupt is also generated when the final byte of data has been read by the QSPI controller and placed in the SRAM, even if the actual SRAM fill level has not risen above the watermark. This last feature is useful to avoid software tracking how much data has been read and resetting the watermark value for the last few bytes of an indirect read transfer. Note that this watermark register is a dual-use register. When the DMA peripheral interface is enabled, it is used by hardware to control the rate at which the DMA requests are issued. When the DMA peripheral interface is disabled, the behaviour is as described above.

Two further interrupt sources are provided to help understand the status of an indirect operation. Firstly, an interrupt is generated when an indirect operation has completed. Secondly, an interrupt is generated if an indirect read operation was requested but could not be accepted due to the fact 2 indirect operations have already been buffered by the QSPI controller.

Setting Indirect Read Transfer Control Register bit[0] starts an indirect read operation, and Indirect Read Transfer Control Register bit[2] will be available to check the status.

2.1.3.1.1. Indirect read transfer process

When the DMA peripheral controller is enabled, the following sequence can be followed:

1. Setup QSPI Configuration Register (0x00)
2. Setup the SRAM watermark in the Indirect Read Transfer Watermark Register (0x64)
3. Setup the indirect transfer's FLASH start address in the Indirect Read Transfer Start Address Register (0x68)
4. Setup the number of bytes to be transferred in the Indirect Read Transfer Number Bytes Register (0x6c)
5. Setup the indirect transfer’s AHB trigger address in the Indirect AHB Address Trigger Register (0x1c)
6. Setup the indirect transfer's AHB trigger address range in the Indirect Trigger Address Range Register (0x80)
7. Setup number of bytes for "single" and "burst" DMA peripheral burst type transfers in the DMA Peripheral Configuration Register (0x20)
8. Trigger Indirect Read access by setting Indirect Read Transfer Control Register bit[0]
9. The completion status of the indirect read operation can be polled via the Indirect Read Transfer Control Register bit[5]. Note this bit is write-to-clear. Alternatively, an Indirect Complete interrupt will be generated when the Indirect read operation has completed.

10. The number of indirect read operations completed will be readable in the same register - Indirect Read Transfer Control Register bits[7:6].

When the DMA peripheral controller is disabled, the following sequence can be followed:

1. Setup QSPI Configuration Register (0x00)
2. Setup the indirect transfer's FLASH start address in the Indirect Read Transfer Start Address Register (0x68)
3. Setup the number of bytes to be transferred in the Indirect Read Transfer Number Bytes Register (0x6c)
4. Setup the indirect transfer's AHB trigger address in the Indirect AHB Address Trigger Register (0x1c)
5. Setup the indirect transfer's AHB trigger address range in the Indirect Trigger Address Range Register (0x80)
6. If the watermark interrupt feature is to be used, set the Indirect Read Transfer Watermark Register (0x64) which will cause an interrupt to be generated when the fill level increases beyond the watermark level. Setting the watermark can be useful indication to software when to read the next part of the indirect read transfer. Note that if the watermark is set to a value other than zero, the watermark interrupt will always trigger once the final byte of indirect transfer has been fetched and placed in the embedded SRAM, even if the watermark value is higher than the actual completed fill level.
7. Trigger Indirect Read access by setting Indirect Read Transfer Control Register bit[0]
8. If the watermark interrupt feature is to be used, wait for watermark interrupt. Else poll the SRAM fill level to decide when sufficient data is in the SRAM to trigger AHB data fetches
9. Read the expected amount of data from SRAM. If there is still more data to fetch in order to complete the indirect read transfer, then loop back to 8. Otherwise continue to 10
10. The completion status of the indirect read operation can be polled via the Indirect Read Transfer Control Register bit[5].
11. An Indirect Complete interrupt will be generated when the Indirect read operation has completed

2.1.3.2. Indirect Write Controller

The aim of the indirect mode of operation is to perform bulk transfer of data from the processor or DMA into FLASH memory in the most efficient manner. The fewest possible write cycles inside the FLASH device will be carried out for the indirect transfer, thus maximizing the life of the device.

Indirect write operation can be thought of from a software perspective as the inverse of the indirect read. It is controlled and triggered by software via specific APB control/configuration registers (Indirect Write Transfer Control Register, Indirect Write Transfer Watermark Register, Indirect Write Transfer Start Address Register and Indirect Write Transfer Number Bytes Register). This block will await delivery of the write data via the external AHB master, placing it in the local SRAM before communicating with the existing legacy SPI IP core to perform an efficient and optimized FLASH write burst.

By default, the indirect write controller is disabled. Before enabling it, software must configure how much data is required and starting from what address. The start address and total number of bytes to be written is defined in registers at addresses Indirect Write Transfer Start Address Register and Indirect Write Transfer Number Bytes Register respectively. Up to two indirect operations can be programmed at any one time. The second can be triggered while the first is in progress. Supporting two indirect operations allows a short turnaround time between the completion of one indirect operation and the start of the second. Indirect write queuing is very similar to indirect read queuing. Refer to the Section 2.1.3.2.2, "*Indirect Access Queuing*" for further details.

The total number of bytes to write in an indirect operation is not limited by the size of the SRAM. The size of SRAM will only limit the amount of data that can be accepted from the external AHB master. When the external master is the DMA, the amount of data requested by the controller via the DMA peripheral interface will never exceed the current fill level of the SRAM. However, this does not guarantee in itself that the DMA, or other external master, if the DMA is not used, will not attempt to push more data to the SRAM than the SRAM can accept. In the case of an SRAM overrun, the controller will back pressure the AHB with wait states. Note the fill level of the SRAM is readable via SRAM Fill Level Register and this can be used to avoid this situation.

An external master will provide the write data and will transfer this to the QSPI controller by issuing AHB writes. The AHB address of the incoming write access must be in the range AHB Indirect trigger address (programmed via the Indirect AHB Address Trigger Register) to AHB Indirect trigger address + 2**(Indirect AHB trigger address range) - 1. Default value of the range is equal to 16 locations. This allows a 16-beat burst to be applied starting from the AHB Indirect trigger address. The smaller bursts are possible to handle effectively as well with this approach. Furthermore it is not strict requirement to push consecutive address sequence. Actual AHB address just has to be in the Indirect Range to grant SRAM as source. Each valid AHB Indirect write will cause the internal SRAM to be pushed, thereby decoupling the AHB address from the FLASH address – i.e. not direct mapped. Therefore AHB Indirect trigger address does not have any relationship with FLASH address. It is just to indicate that data should take SRAM as source instead of FLASH Memory array after triggering of any valid Indirect Write. FLASH address for Indirect Write is taken from the Indirect Write Transfer Start Address Register (0x78). Assuming the SRAM is not full at the point the AHB access is received by the QSPI controller, then the data will be pushed to the SRAM with minimum latency.

If an AHB write access is received whose address is not within the range described above then that access will not be completed using the indirect controller. It will instead be serviced by the direct access controller.

If an AHB write access is received whose address is within the range described above but the SRAM is full, then AHB wait states will be applied until some or all of the data has been pushed from SRAM onto the FLASH.

If an AHB write burst is received whose access elements traverse the AHB Indirect trigger range, then the accesses within the Indirect trigger range will be processed by the indirect controller, and the rest will be taken by the direct access controller. Note this is likely to be a software configuration error.

The external master is only permitted to issue 32-bit AHB writes until the last word of an indirect transfer. This helps keep the SRAM control logic less complex. On the final write, the external master may issue a 32-bit word, 16-bit (half-word) or a byte access to complete the transfer. If the number of bytes to write is less than 4 on the last transfer, the master is still permitted to issue a 32-bit transfer. In these cases, the extra bytes are discarded by the controller.

When the SRAM holds a number of bytes equal to or greater than the size of a FLASH page (which itself is programmable by the controller, with a default of 256 bytes) or when the SRAM holds all remaining bytes of the currently executing indirect transfer, the controller will initiate a write burst to the command generator.

An indirect operation may be cancelled at any time by setting Indirect Write Transfer Control Register bit[1].

Any bus master should be allowed to initiate an indirect access. The DMA bus peripheral interface has been included in the design to offload some of the necessary software overhead to efficiently manage data transfers when an external DMA supporting this interface is added. The alternative to this is for software to access the SRAM fill-level directly via APB registers and then decide for itself when the data should be written to the local SRAM. When the DMA peripheral interface is disabled, the fill level watermark register accessible via the Indirect Write Transfer Watermark Register (0x74) is provided. When the SRAM fill level falls below this watermark, an interrupt is generated. Note that this watermark register is a dual-use register. When the DMA peripheral interface is enabled, it is used by hardware to control the rate at which the DMA requests are issued. In this mode, since the QSPI will not initiate an write operation unless there is at least one FLASH page worth of data in the local SRAM, or the remaining bytes of the indirect transfer are in the SRAM, then it is important the watermark in DMA mode is programmed to a value >= 1 flash page. Two further interrupt sources are provided to help understand the status of an indirect operation. Firstly, an interrupt is generated when an indirect operation has completed. Secondly, an interrupt is generated if an indirect write operation was requested but could not be accepted due to the fact two indirect operations have already been buffered by the QSPI controller.

Two further interrupt sources are provided to help understand the status of an indirect operation. Firstly, an interrupt is generated when an indirect operation has completed. Secondly, an interrupt is generated if an indirect read operation was requested but could not be accepted due to the fact 2 indirect operations have already been buffered by the QSPI controller.

Setting Indirect Write Transfer Control Register bit[0] starts an indirect write operation, and Indirect Write Transfer Control Register bit[2] will be available to check the status.

2.1.3.2.1. Indirect write transfer process

When the DMA peripheral controller is enabled, the following sequence can be followed:

1. Setup QSPI Configuration Register (0x00)
2. Setup the indirect transfer’s FLASH start address in the Indirect Write Transfer Start Address Register (0x78)
3. Setup the number of bytes to be transferred in the Indirect Write Transfer Number Bytes Register (0x7c)
4. Setup the indirect transfer’s AHB trigger address in the Indirect AHB Address Trigger Register (0x1c)
5. Setup the indirect transfer's AHB trigger address range in the Indirect Trigger Address Range Register (0x80)
6. Setup number of bytes for "single" and "burst" DMA peripheral burst type transfers in the DMA Peripheral Configuration Register (0x20)
7. Optional: Set the Indirect Write Transfer Watermark Register (0x74) to control the rate at which DMA requests will be issued.
8. Trigger Indirect Write access by setting Indirect Write Transfer Control Register bit[0].
9. The QSPI controller will request the DMA to transfer data using the DMA request interface.
10. The completion status of the indirect write operation can be polled via the Indirect Write Transfer Control Register bit[5]. This will be write-to-clear. The number of indirect write operations completed will be readable in the same register.
11. An Indirect Complete interrupt will be generated when the Indirect write operation has completed.

When the DMA peripheral controller is disabled, the following sequence can be followed:

1. Setup QSPI Configuration Register (0x00)
2. Setup the indirect transfer’s FLASH start address in the Indirect Write Transfer Start Address Register (0x78)
3. Setup the number of bytes to be transferred in the Indirect Write Transfer Number Bytes Register (0x7c)
4. Setup the indirect transfer’s AHB trigger address in the Indirect AHB Address Trigger Register (0x1c)
5. Setup the indirect transfer's AHB trigger address range in the Indirect Trigger Address Range Register (0x80)
6. It is functionally valid for software to simply write all the data to the SRAM in one block transfer. However, if the total number of bytes to write is greater than the size of the partitioned SRAM, then it is quite likely the SRAM will become full causing the QSPI to back-pressure the system AHB bus for a considerable time. This time is based on the FLASH data-rate and the page-write time of the device. To avoid sending all the write data in one block transfer,

software can make use of the watermark interrupt to identify a convenient time to send data a page at a time to the SRAM module. Alternatively, software can poll the SRAM fill level register directly to identify how empty the SRAM is at any one time in order to make a judgement as to when the most practical time to send the next part of the transfer.

7. If the watermark interrupt feature is to be used, set the Indirect Write Transfer Watermark Register (0x74) which will cause an interrupt to be generated when the fill level falls below the watermark. The watermark should be set to a number between zero and a page size. I.e. if the page size is 256 bytes, then setting the watermark to a value between 10 and 250 is reasonable and will cause the interrupt to trigger when the fill level drops below the programmed number. Setting the watermark can be useful to provide an indication to software when to write the next page of data to the SRAM.

8. Trigger Indirect Write access by setting Indirect Write Transfer Control Register bit[0].

9. If the remaining number of bytes still to be transferred into the SRAM for the current indirect transfer is greater than a FLASH page, then write 1 FLASH page worth of data to the SRAM. Otherwise send the remaining data from the indirect transfer to SRAM.

10. If all the data in the indirect transfer has now been sent to the SRAM, then go to 12. and await indirect complete status. Otherwise if there is more data still to be transferred then either:

    a. If the watermark interrupt feature is being used, then wait for watermark interrupt.

    b. Alternatively the SRAM fill level can be interrogated to identify a convenient time to send more data.

11. Loop back to 9.

12. Optional: The completion status of the Indirect write operation can be polled via Indirect Write Transfer Control Register bit[5].

13. An Indirect Complete interrupt will be generated when the Indirect write operation has completed.

2.1.3.2.2. Indirect Access Queuing

Software is permitted to queue up to two indirect transfers for both the indirect write controller and the indirect read controller. Supporting two indirect operations allows a short turnaround time between the completion of one indirect operation and the start of the second. Any attempt to queue more than two operations will cause an interrupt to be generated. To take advantage of this feature, software should attempt to keep both indirect programming slots full at all times.

From the software perspective, indirect access queuing is achieved by triggering bit 0 of the indirect transfer control register (Indirect Read Transfer Control Register or Indirect Write Transfer Control Register) twice in short succession. The indirect number of bytes register and the indirect FLASH start address register must be setup with the relevant transfer data before bit 0 can be triggered for each transfer. Since these registers will change regularly, the hardware must keep sampled versions of these registers for the duration of the indirect transfer.

The internal register block will only issue an indirect start trigger to the key underlying datapath blocks one at a time. There are 2 independent datapath blocks in the indirect access controller that will receive and independently sample this information. The first is the datapath block on the AHB side of the SRAM. For indirect reads, this is a read interface, for indirect writes, it is a write interface. The second is the datapath block on the FLASH side of the SRAM. For indirect reads, this is a write interface, for indirect writes, it is a read interface. Both blocks will process the indirect transfers at different times. For example, for an indirect read operation, the datapath block on the FLASH side of the SRAM will be able to start processing the second queued transfer as soon as the last byte of the first transfer has been written to SRAM. Before commencing the second transfer, this block must re-sample the number of bytes and the indirect FLASH start address register. Similarly, the datapath block on the AHB side will re-sample the same registers locally when it has forwarded all the FLASH data associated with the first indirect transfer from SRAM onto AHB.

2.1.3.2.3. Consecutive Writes and Reads using Indirect Transfers

It is permitted for software to trigger an indirect read operation while an indirect write operation is in progress. Similarly it is permitted to trigger an indirect write while an indirect read operation is in progress. Indirect write operations will take overall precedence.

2.1.3.2.4. Accessing the SRAM

Physically, the SRAM will be a single port macro. The depth of this SRAM is a configuration choice at compile time. It will be segmented into two halves. The lower half is reserved for indirect read use. The upper is for indirect write use only. The size of each half will be programmable via the SRAM Partition Configuration Register (0x18). This allows the programmer to allocate how many bits of the SRAM address bus are allocated to indirect read. By default, this is set so that exactly half of the SRAM is portioned for use by the indirect read controller. To ensure the AHB read data bus is not directly fed by the SRAM read data through combinatorial logic, an extra bank of holding registers is included in the indirect read data path. These registers act as an extra location to be added to the allocated number of SRAM locations for indirect read.

To illustrate how the SRAM (and the extra bank of holding registers) can be allocated between indirect read and write, the following example is provided. The depth of the SRAM in this example is configured to be 8 bits, equaling 256 locations...

- If the SRAM Partition Register is set to 0x00, then 256 locations are allocated to indirect writes and 1 location to indirect reads.
- If the SRAM Partition Register is set to 0x01, then 255 locations are allocated to indirect writes and 2 locations to indirect reads.
- If the SRAM Partition Register is set to 0x02, then 254 locations are allocated to indirect writes and 3 locations to indirect reads.
- (...)
- If the SRAM Partition Register is set to 0xfd, then 3 locations are allocated to indirect writes and 254 locations to indirect reads.
- If the SRAM Partition Register is set to 0xfe, then 2 locations are allocated to indirect writes and 255 locations to indirect reads.
- If the SRAM Partition Register is set to 0xff, then 1 locations are allocated to indirect writes and 256 location to indirect reads.

Note that a value of 0xFF or 0x00 in the SRAM Partition Register should be avoided by software, as only the bottom 8 bits of the SRAM fill level are accessible through software (i.e. up to 255 limit) via the SRAM Fill Level Register. I.e. if the fill level reaches 256 on either the indirect read or write side, it will appear when reading the Fill Level to be 0.

There are four SRAM sources that are arbitrated and muxed onto the single SRAM port. Up to three sources may be accessing this port at any one time. The sources are described as follows:

1. Indirect Write, Write source. This is located on the AHB side of the SRAM.
2. Indirect Write, Read source. This is located on the FLASH side of the SRAM.
3. Indirect Read, Write source. This is located on the FLASH side of the SRAM.
4. Indirect Read, Read source. This is located on the AHB side of the SRAM.

A fixed priority arbitration scheme is implemented. The priority allocated to these sources is set as follows:

Table 2.1. SRAM Access Priority

Transfer	Operation	Priority
Indirect Write	Write to SRAM (from System AHB)	3rd (exclusive with AHB read request)
Indirect Write	Read from SRAM (from QSPI Controller)	2nd
Indirect Read	Write to SRAM (from QSPI Controller)	1st
Indirect Read	Read from SRAM (from System AHB)	3rd (exclusive with AHB read request)

With the exception of the write port during an Indirect Read operation (on the FLASH side of the SRAM), the logic driving all four sources must not assume single cycle completion. Writes to the SRAM during an indirect read must be allowed to complete immediately to avoid data loss. Therefore this port is given maximum priority.

2.1.4. DMA Peripheral Controller

The peripheral interface will be used to trigger an external DMA into performing low-latency AHB data burst accesses to the controller. The DMA the peripheral interface is directly meant for is the CoreLink DMA Controller DMA-330 core from ARM. The DMA peripheral interface is only used for the Indirect modes of operation where data is buffered in local SRAM in order to quickly service incoming AHB requests and have the core perform the low level FLASH transfers over a longer period of time. There are two identical DMA interfaces, one for each indirect controller. For the indirect read controller, the QSPI controller will only issue DMA requests after the data has been retrieved from FLASH and written to local SRAM. For the indirect write controller, the QSPI controller will issue DMA requests immediately after the transfer is triggered and will continue to do so until the entire indirect write transfer has been transferred. The rate at which these requests are made can be altered using the watermark registers.

2.1.4.1. DMA Interface

The following is true for both DMA interfaces.

The peripheral controller is responsible for driving the DR (peripheral to DMA) request bus and for acknowledging the DA (DMA to peripheral) acknowledge bus. The two buses are simple AXI like interfaces, which consist of a two-way VALID and READY handshake mechanism.

For the DR request bus, the controller drives the "drvalid" signal when valid information is available on "drtype" and "drlast" signals. This information will remain set until the external DMA drives the "drready" signal, indicating it can accept the request.

For the DA acknowledge bus, the DMA drives the "davalid" signal high when valid information is available on "datype" signal. This information will remain set until the DMA peripheral controller drives the "daready" signal high, indicating it has accepted the acknowledgement made by the DMA.

The controller will issue a request via the DR bus to the DMA when it requires the DMA (or other AHB master) to transfer data from the controller via its AHB slave port. The controller will issue two types of requests to achieve this, single or burst. Each of these request types will correspond to a programmable but fixed number of bytes. By default, both of these types will reset to a request size of just 1 byte. The DR bus is also used to acknowledge a Flush request which is described in more detail below. The DR request uses the "drtype" bus to indicate the request type.

The DMA uses the DA bus to indicate when the DMA has completed the data transfer. This is acknowledged by the controller by driving the "daready" signal, but no other action is taken. The DMA also uses the DA bus to request that the controller perform a flush operation. When this occurs, the controller will re-evaluate the FIFO levels and replay all DMA

requests that may have been sent but have yet to be acknowledged. Note if a request on the DR channel is valid at the point the flush is received, the controller must hold the "drvalid" and "drtype" signals steady until the DMA acknowledges the request via "drready". Once the flush operation has completed, the controller will acknowledge the flush by sending a unique request on the DR channel, setting drtype to 2'b10.

Table 2.2. DMA Encoding

drtype/datype	Action
2'b00	Single Request/Acknowledge
2'b01	Burst Request/Acknowledge
2'b10	Flush Acknowledge/Request

Flushing a write request has no effect on writes that may have been already sent to the FLASH device.

2.1.4.2. DMA Order of operation

When an indirect operation is triggered, the total amount of data to be transferred from/to FLASH memory (in bytes) will be visible by the DMA peripheral controller. The controller will split this into a number of DMA burst and single requests by dividing the total number of bytes by the number of bytes programmed in the burst request, and then dividing the remainder by the number of bytes in a single request. It is for software to ensure there is no remainder following these divisions. To ensure the logic is kept simple, the divisions will always be a factor of 2, and the programmable registers defining them must be defined such. These programmable registers are defined in the DMA Peripheral Configuration Register. As an example, if the total amount of data selected to be read from FLASH using the low-latency Indirect mode is 512 bytes, the size of the SRAM is a fixed 256 bytes and software configures the burst type transfer size to be 256 bytes, then the SPI controller will trigger a DMA burst request when the first 256 bytes have been locally buffered. It can only then trigger the 2nd burst request when a further 256 bytes have been buffered in the local SRAM. Since the SRAM size was only 256 bytes itself, this means the DMA will have had to fetch the entire contents of the SRAM before the next request can be issued.

The SRAM fill level will be made visible to the DMA peripheral controller. For indirect reads, requests will be issued to the external DMA based on the following state machine:

1. Wait for START trigger
2. If the next request to send is a BURST request, wait until both of the following is TRUE
    a. SRAM fill level is greater than or equal to the value stored in the burst_size field of DMA Peripheral Configuration Register bits[11:8]
    b. SRAM fill level is greater than or equal to the value stored in the Indirect Read Transfer Watermark Register

    ACTION when TRUE:

    i. SEND BURST.
    ii. Calculate local expected version of fill level: (SRAM fill level - num_bytes_requested_to_DMA)
    iii. If the next request to send is a BURST request, move to 4
    iv. If the next request to send is a SINGLE request, move to 5
    v. Else move to 1
3. If the next request to send is a SINGLE request, wait until both of the following is TRUE

a. SRAM fill level is greater than or equal to the value stored in the single_size field of DMA Peripheral Configuration Register bits[3:0]

   ACTION when TRUE:

   i. SEND SINGLE.

   ii. Calculate local expected version of fill level: (SRAM fill level - num_bytes_requested_to_DMA)

   iii. If the next request to send is a SINGLE request, move to 5

   iv. Else move to 1

4. The next request to send is a BURST request. If the (SRAM fill level - num_bytes_requested_to_DMA) is less than the value stored in the burst_size field of DMA Peripheral Configuration Register bits[11:8] then goto 2, else:

   a. SEND BURST.

   b. Calculate local expected version of fill level: (SRAM fill level – num_bytes_requested_to_DMA)

   c. If the next request to send is a BURST request, move to 4

   d. If the next request to send is a SINGLE request, move to 5

   e. Else move to 1

5. The next request to send is a SINGLE request. If the (SRAM fill level - num_bytes_requested_to_DMA) is less than the value stored in the single_size field of DMA Peripheral Configuration Register bits[3:0] then goto 3, else:

   a. SEND SINGLE.

   b. Calculate local expected version of fill level: (SRAM fill level – num_bytes_requested_to_DMA)

   c. If the next request to send is a SINGLE request, move to 5

   d. Else move to 1

For indirect writes, requests will be issued to the external DMA based on the following state machine:

1. Wait for START trigger

2. If the next request to send is a BURST request, wait until both of the following is TRUE

   a. Remaining space in SRAM is greater than or equal to the value stored in the burst_size field of DMA Peripheral Configuration Register bits[11:8]

   b. SRAM fill level is less than or equal to the value stored in the Indirect Write Transfer Watermark Register

      ACTION when TRUE:

      i. SEND BURST.

      ii. Calculate local expected version of fill level: (SRAM fill level + num_bytes_requested_to_DMA)

      iii. If the next request to send is a BURST request, move to 4

      iv. If the next request to send is a SINGLE request, move to 5

v. Else move to 1

3. If the next request to send is a SINGLE request, wait until both of the following is TRUE

   a. Remaining space in SRAM is greater than or equal to the value stored in the single_size field of DMA Peripheral Configuration Register bits[3:0]

      ACTION when TRUE:

      i. SEND SINGLE.

      ii. Calculate local expected version of fill level: (SRAM fill level + num_bytes_requested_to_DMA)

      iii. If the next request to send is a SINGLE request, move to 5

      iv. Else move to 1

4. The next request to send is a BURST request. If the (SRAM fill level - num_bytes_requested_to_DMA) is less than the value stored in the burst_size field of DMA Peripheral Configuration Register bits[11:8] then goto 2, else:

   a. SEND BURST.

   b. Calculate local expected version of fill level: (SRAM fill level + num_bytes_requested_to_DMA)

   c. If the next request to send is a BURST request, move to 4

   d. If the next request to send is a SINGLE request, move to 5

   e. Else move to 1

5. The next request to send is a SINGLE request. If the (SRAM fill level - num_bytes_requested_to_DMA) is less than the value stored in the single_size field of DMA Peripheral Configuration Register bits[3:0] then goto 3, else:

   a. SEND SINGLE.

   b. Calculate local expected version of fill level: (SRAM fill level + num_bytes_requested_to_DMA)

   c. If the next request to send is a SINGLE request, move to 5

   d. Else move to 1

For the indirect read operation, a programmable Indirect Read Transfer Watermark Register (0x64) will define the minimum fill level that the controller can issue the first DMA request – the higher this figure is, the more data that must be buffered in SRAM before the DMA will collect it. For the indirect write operation, a programmable Indirect Write Transfer Watermark Register (0x74) will define the maximum fill level that the controller can issue the first DMA burst/ single request. The watermark registers allow the system to optionally concentrate AHB transfers over a short period of time. By default, the watermark registers are reset to zero, meaning the peripheral controller may issue DMA request as soon as possible. For the indirect read case, this means that there is sufficient data in the SRAM to perform a burst type request, or a single type request if there less data left to transfer than the number of bytes programmed in the burst type request. Note the DMA peripheral controller must never send an indirect read DMA request if the SRAM is empty, or an indirect write DMA request if the SRAM is full.

The DMA peripheral controller will send a trigger event to the external DMA by setting drtype[1:0] to the required type of access (single, burst or flush) along with drvalid for a single hclk cycle to tell the DMA that there is READ data available. drlast is asserted at the same time as drtype to indicate the last request of the current transfer sequence. drready is driven by the DMA to indicate that it can accept requests from the Peripheral Controller.

The DMA will indicate completion of a previously requested data transfer which takes place on the AHB bus by asserting ACK on datype along with davalid. daready is used to indicate to the DMA that the Peripheral Controller is available to accept status information on datype. A series of requests will be sent on the DA bus until all data read/written from/to the FLASH device that is stored in the SRAM has been acknowledged as being transferred to/from the DMA.

The peripheral interface only acts on the DA acknowledge channel if the DMA issues an abort request. All other requests are simply accepted using daready as required. This means the controller is free to pipeline the DR requests as much as it likes, and will be limited only by the pipelining requirements of the DMA.

The DMA peripheral interface may be disabled via software. When the interface is disabled, no DR channel request will ever be issued. An incoming flush request on the DA channel will be internally ignored, although the daready must still be set to acknowledge the flush request. Note if an AHB master other than the DMA will perform the data transfer for indirect operations, the DMA peripheral interface must be disabled.

2.1.5. Software Triggered Instruction Generator (STIG)

The direct and indirect access controllers are used to transfer data. In order to access the volatile and non-volatile configuration registers, the legacy SPI Status register, other status/protection registers as well as to perform ERASE functions, a separate software controller is required. The software triggered instruction generator, or STIG, is controlled using the Flash Command Control Register (Using STIG) (0x90) by setting up the command to issue to the FLASH device. This is a generic controller and can be used to perform any instruction that the FLASH device supports from the extended SPI protocol. Configuring of instructions which are not compliant with the specification of FLASH Devices could cause unpredicted behaviour of the controller. Bit 0 is used to trigger the command, bit 1 used by software to poll the status of the command execution. For reads, when the command has been serviced (bit 1 toggles from "1" to "0"), up to 8 bytes of read data will be placed in the Flash Command Read Data Register (Flash Command Read Data Register (Lower) and Flash Command Read Data Register (Upper)). For writes, the write data should be placed in the Flash Command Write Data Register (Flash Command Write Data Register (Lower) and Flash Command Write Data Register (Upper)). The completion of the STIG request could be also checked by the corresponding interrupt. The occurrence of the interrupt indicates that the controller is ready for accepting a new STIG request. It is important to notice that completion of the STIG request is not equivalent to completion it on SPI side. E.g. if STIG is configured to the command composed of data to transmit only, the data is taken from the corresponding STIG register fields and put into TX FIFO. Since all bytes to write are known, another STIG can be queued before serialization of the current one is completed.

There are some commands which require more data to read than 8 bytes (for example READ ID command). The additional STIG Memory Bank was implemented in order to accommodate these data if needed. The STIG Memory Bank is controlled by the Flash Command Control Register (Using STIG) Register bit[2]. If enabled, the number of bytes to read in the STIG is extended to the value ranging from 16 to 512 as defined in Flash Command Control Memory Register (Using STIG) Register bit[18:16]. It should be noticed that there are very few commands (excluding Read Array ones which are not intended to handle effectively in STIG Mode but in Direct/Indirect Modes) which return more than 8 bytes to the controller. If the maximum number of bytes to Read using STIG in target application is less than 512, the depth of the STIG Memory Bank can be set smaller what will result in saving noticeable part of the area (STIG Memory Bank is a internal component of the controller). The decision is made in pre-compilation time. The depth is configurable in *hdl/hdl_src/cdns_qspi_flash_ctrl_defs_default.v* by changing parameter called *cdns_qspi_stig_mem_bank_depth*. The default depth is set for the limited size of this parameter, namely 4. If number of bytes to Read in the STIG as defined in Flash Command Control Memory Register (Using STIG) Register bit[18:16] exceeds the Memory Bank Depth, remaining data will overwrite the STIG Memory Bank locations starting from its first address. Flash Command Read Data Register (Lower) and Flash Command Read Data Register (Upper) keep the last 8 bytes read from the Flash Device by STIG when Memory Bank is enabled. Therefore, for example if the user wants to get just a single byte from the last eight bytes from long continuous read SPI data chain, there is no need to access the STIG Memory Bank since data can be taken from suitable Flash Command Read Data Register. In order to access more data, STIG Memory Bank data request should be triggered. It is controlled by the Flash Command Control Memory Register (Using STIG) and works analogously for triggering STIG from the functional standpoint. Bit 0 is used to trigger the command, bit 1 used by software to poll the status of the command execution. When bit 1 toggles from "1" to "0", the byte of data (Flash Command Control Memory Register (Using STIG) Register bit[15:8]) from corresponding address (Flash Command Control Memory Register (Using STIG) Register bit[28:20]) is valid. The address should be set before triggering the STIG Memory Bank access. Each

consecutive STIG access overwrites the previous one so that the data in the Bank always fit into byte index fetched by the last STIG access configured to use the Memory Bank (first incoming byte equals first address of the Memory Bank, second one equals the second address and so on).

2.1.5.1. Servicing a STIG request

A STIG request will cause the QSPI Flash Controller to interrogate the Flash Command Control Register (Using STIG) (0x90) to determine what and how many bytes it should send to the FLASH device. Bits [31:24] of this register indicate the instruction to be sent and is always pushed first. If there is an address to send, then the address (the size of which is also programmed in the same register) is sent next. The address itself is stored in the Flash Command Address Register (0x94). If there are any dummy cycles to send (the size of which is also programmed in Flash Command Control Register (Using STIG)) then those are sent next. If there is data to write or read (the size of which is also programmed in Flash Command Control Register (Using STIG)) then for the case of writes, up to 8 bytes can be sent (as stored in the Flash Command Write Data registers, Flash Command Write Data Register (Lower) and Flash Command Write Data Register (Upper)). In the read case, when the read data has been collected from the FLASH device, the QSPI Flash Controller stores that in the Flash Command Read Data Registers (Flash Command Read Data Register (Lower) and Flash Command Read Data Register (Upper)). Up to 8 bytes can be get if Flash Command Control Register (Using STIG) Register bit[2] is disabled or up to 512 when enabled.

When the QSPI Flash Controller starts to service a STIG request, it sets Flash Command Control Register (Using STIG) Register bit[1] to indicate a command execution is in progress.

When the QSPI Flash Controller is in the auto-polling state, servicing a STIG request is slightly different. Most of devices are largely inaccessible after a program operation until the device has completed that write. Some group of them has a possibility to suspend programming page. It can be controlled by Polling Flash Status Register bit[8] which indicates active auto-polling phase. After requesting a STIG, the QSPI Flash Controller immediately issues appropriate OPCODE to Memory. During servicing a STIG (in auto-polling phase) the status bit of command execution remains steady and other parts of transfer such as ADDRESS or DUMMY BITS etc. are disabled (to issued Program Suspend Command is needed OPCODE only).

There is a programmable option to add delay between every repetitive poll operation (delay is defined by Write Completion Control Register bits[31:24]). This feature is implemented to free up SPI bandwidth if needed.

2.1.6. Arbitration between Direct/Indirect Access Controller and STIG

When multiple controllers are active simultaneously, a simple fixed-priority arbitration scheme is used to arbitrate between each interface and access the external FLASH. The fixed priority is defined as follows, highest priority first:

1. The Indirect Access Write
2. The Direct Access Write
3. The STIG
4. The Direct Access Read
5. The Indirect Access Read

Each controller is back pressured while waiting to be serviced.

2.1.7. SPI Command Translation

Requests issued by the direct access controller, the indirect access controller or the STIG will be translated into a sequence of byte transfers to send downstream (before serialization to the FLASH device). These sequences depend on the requested transfer but an example of a typical 1-byte non sequential READ is shown below:

- INSTRUCTION OPCODE -> ADDRESS -> Mode Byte -> Dummy Cycles -> 1 byte of don’t care

For sequential accesses, an extra byte of data per read is pushed to the FLASH device on the back of the above sequence assuming it can be done so with no gap between each transferred byte.

When PHY Mode is enabled and consequently no clock divider is configured, latency caused by multi domain synchronization may make an extra byte insufficient to avoid the transfer gap. To ensure the sequential access non-interrupted and keep the maximum performance of the controller, PHY Pipeline Mode was implemented. When enabled, number of don't care bytes is calculated based on the configuration.

The actual sequence sent to the FLASH device depends on the requested transfers, whether the transfer is non-sequential or sequential, whether the device has been configured in XIP mode and the state of the main Device Instruction Type programmable registers (at offsets Device Read Instruction Register and Device Write Instruction Register).

For writes, the write enable latch (or WEL) within the FLASH device itself must be high before a write sequence can be issued. The QSPI Flash Controller will automatically issue the write enable latch command before triggering a write command via the direct or indirect access controllers (DAC/INDAC) – i.e the user does not need to perform this operation. For increasing flexibility and performance user can turn off this feature by setting Device Write Instruction Register bit[8]. The opcode for WREN is typically 0x06 and is common between devices.

When write requests from the direct or indirect access controllers are no longer being received and all outstanding requests have been sent, the FLASH device will automatically start the page program write cycle. Any incoming request at this time will be held in wait states until the cycle has completed. The QSPI Flash Controller will automatically poll the FLASH device legacy SPI status register to identify when the write cycle has completed. This is achieved by sending the RDSR opcode to the FLASH device and waiting until the device itself has indicated the write cycle has completed (until the Write in Progress bit has cleared to zero and the write enable latch bit has also cleared to zero or device is ready bit has set to one). The WREN and the RDSR device instructions are the only ones that are sent by the controller under the hood. For any other specific instruction that the user determines should be sent to the device (for example if the device needs to be unprotected before a write command is issued), these should be handled separately by issuing FLASH commands via the STIG.

2.1.8. Selecting the Flash Instruction Type

In order to send the correct READ and WRITE opcodes, software should initialize the Device READ Instruction Type Register (Offset Device Read Instruction Register) and the Device WRITE Instruction Type Register (Offset Device Write Instruction Register). These registers include fields to setup the required instruction opcodes that is intended to be used to access the FLASH (default is basic READ and basic page program) as well as the instruction type, edge mode (DDR or SDR) and whether the instruction uses single, dual or quad pins for address and data transfer. Providing this level of control to the user provides a future proofed generic solution. To ensure the controller can operate from a reset state, the registers will be reset to an opcode compatible with SIO devices what can be modified using BOOT feature.

Despite being applicable for both READs and WRITEs, the Instruction Type field in the Device Read Instruction Register bits[9:8] only appears once – it is not included in the Device Write Instruction Register. If software sets this to anything other than "0", then the address transfer type and the data transfer type bits of both Device Read Instruction Register and Device Write Instruction Register Registers become don't care. It is made available to allow software to support the less common FLASH instructions where the opcode, address and data are sent on 2, 4 or 8 lanes (the opcode from most instructions are sent serially to the FLASH device, even for dual/quad instructions). Also note that for devices that support instructions that can send the OPCODE over multiple lanes, the name for these instructions are not consistently named in the FLASH datasheets. One of the devices that support these instructions is the Numonyx (Micron) N25Q128. The extra READ instructions are called DCFR and QCFR. The WRITE instructions are DCPP and QCPP.

There are group of devices (e.g. Numonyx (Micron) N25Q512A) capable to handling Read Operations in Dual Data Rate Mode (DDR) (it is also called Dual Transfer Rate Mode (DTR) and these two will be used interchangeably down the document). That means they can issue and capture the data on both rising and falling edges during working with dedicated command type. This enables the controller to maintain throughput at twice lower frequency of spi_clk. The Device Read Instruction Register has DDR enable bit which inform QSPI Flash Controller that opcode written into Read Opcode field is capable with DDR command type. The another field defined in Read Data Capture Register bits[19:16] which enables

the controller to shift the transmitted data in DDR mode. By default, data are shifted by 1 clock cycle to ensure hold timing greater than 0 during DDR transactions. It may not be sufficient for high ref_clk frequency in accordance with the high dividers.

There is also MT25Q family of Micron devices where DTR protocol was implemented. It enables device to handle all commands in DTR Mode. DTR Read commands described above detect DTR Mode based on dedicated OPCODE. Therefore OPCODE has to be send as STR. When DTR protocol is enabled, device does not need OPCODE to detect edge Mode because can recognize it based on volatile or non-volatile bit from configuration register of itself.

For reference, below is a table illustrating how software should configure the controller for selected specific READ and WRITE instruction supported by the mentioned devices:

Table 2.3. Read examples

Command	Opcode (how many lanes/ edge mode)	Addr/ Dummy/ Mode (how many lanes/ edge mode)	Data(how many lanes/ edge mode)	Instruction Type (Device Read Instruction Register bits[9:8])	Address transfer type (Device Read Instruction Register bits[13:12])	Data transfer type (Device Read Instruction Register bits[17:16])	DDR bit enable (Device Read Instruction Register bits[10])
READ	1/SDR	1/SDR	1/SDR	0	0	0	0
FAST_READ	1/SDR	1/SDR	1/SDR	0	0	0	0
DTR_FAST_READ	1/SDR	1/DDR	1/DDR	0	0	0	1
DOFR (Dual O/p Fast Read)	1/SDR	1/SDR	2/SDR	0	0	1	0
DIOFR (Dual I/O Fast Read)	1/SDR	2/SDR	2/SDR	0	1	1	0
DDIOFR (DTR Dual I/O Fast Read)	1/SDR	2/DDR	2/DDR	0	1	1	1
QOFR (Quad O/p Fast Read)	1/SDR	1/SDR	4/SDR	0	0	2	0
QIOFR (Quad I/O Fast Read)	1/SDR	4/SDR	4/SDR	0	2	2	0
DQIOFR (DTR Quad I/O Fast Read)	1/SDR	4/DDR	4/DDR	0	2	2	1
DCFR (Dual Command Fast Read)	2/SDR	2/SDR	2/SDR	1	DON'T CARE	DON'T CARE	0
DDCFR (DTR Dual Command Fast Read)	2/SDR	2/DDR	2/DDR	1	DON'T CARE	DON'T CARE	1

Command	Opcode (how many lanes/ edge mode)	Addr/ Dummy/ Mode (how many lanes/ edge mode)	Data(how many lanes/ edge mode)	Instruction Type (Device Read Instruction Register bits[9:8])	Address transfer type (Device Read Instruction Register bits[13:12])	Data transfer type (Device Read Instruction Register bits[17:16])	DDR bit enable (Device Read Instruction Register bits[10])
QCFR (Quad Command Fast Read)	4/SDR	4/SDR	4/SDR	2	DON'T CARE	DON'T CARE	0
DQCFR (DTR Quad Command Fast Read)	4/SDR	4/DDR	4/DDR	2	DON'T CARE	DON'T CARE	1

NOTE1:

This data are applicable for both 3-byte or 4-byte address variants of the commands if did not indicated otherwise.

NOTE2:

In DTR protocol all transfer phases (including opcode) take DDR edge mode independently on the command under execution. DTR protocol is to be enabled by QSPI Configuration Register bit[24] It has higher priority than DDR Mode enable bit from Device Read Instruction Register bits[10] (0x04).

Table 2.4. Write examples

Command	Opcode (how many lanes)	Addr/ Dummy/ Mode (how many lanes)	Data(how many lanes)	Instruction Type (Device Read Instruction Register bits[9:8])	Address transfer type (Device Write Instruction Register bits[13:12])	Data transfer type (Device Write Instruction Register bits[17:16])
PP	1	1	1	0	0	0
DIFP (Dual Input Fast Program)	1	1	2	0	0	1
DIEFP (Dual Input Extended Fast Program)	1	2	2	0	1	1
QIFP (Quad Input Fast Program)	1	1	4	0	0	2
QIEFP (Quad Input Extended Fast Program)	1	4	4	0	2	2
DCPP (Dual Command Fast Program)	2	2	2	1	DON'T CARE	DON'T CARE

Command	Opcode (how many lanes)	Addr/ Dummy/ Mode (how many lanes)	Data(how many lanes)	Instruction Type (Device Read Instruction Register bits[9:8])	Address transfer type (Device Write Instruction Register bits[13:12])	Data transfer type (Device Write Instruction Register bits[17:16])
QCPP (Quad Command Fast Program)	4	4	4	2	DON'T CARE	DON'T CARE

NOTE1:

This data are applicable for both 3-byte or 4-byte address variants of the commands if did not indicated otherwise.

NOTE2:

In DTR protocol all transfer phases (including opcode) take DDR edge mode independently on the command under execution. DTR protocol is to be enabled by QSPI Configuration Register.

2.1.9. APB Interface and Register Module

The APB interface conforms to AMBA v3.0. It is used to configure the core and perform software controlled FLASH accesses using the Flash Command Control register (refer to Section 2.1.5, "*Software Triggered Instruction Generator (STIG)*") and section Section 2.2.3, "*Using the Flash Command Control Register (STIG Operation)*". The APB interface feeds a single register block containing the programmable register set. The register block is timed to the APB clock. All control or enable bits that are triggered by APB writes and cause an event to trigger, or an operation somewhere in the QSPI controller to be enabled are synchronized to the destination clock. Static configuration bits that have no effect while a separate enable bit is low do not require synchronization.

2.1.10. SRAM Macro

The memory macro is required for the indirect mode of operation. The pins for this memory is routed to the top level allowing the integrator to select whether to use SRAM memory, register arrays or simple banks of flops. The memory macro required will be a single port only. The depth of the memory macro will be a configurable option and is application specific. The SRAM is hclk timed.

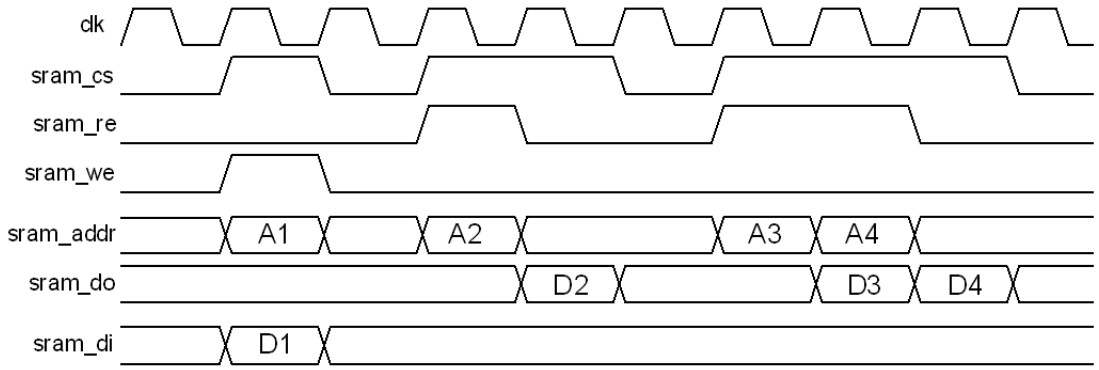
To keep maximum performance of SRAM interface, some combinational outputs from the controller left unregistered. These are intended to flop by SRAM Macro without any additional cells inserted before the SRAM inputs (pure wire connection). Since SRAM Macro is to be located on the chip level, destination flops are outside design under synthesis (QSPI Flash Controller) delivered by Cadence. To meet timings, output delays for these signals were relaxed. If SRAM Macro is placed close to the controller, the timings will be met with spare even for hclk close to defined maximum.

Write access to the SRAM memory takes one clock cycle.

Read access from the SRAM memory takes two clock cycles. Signal sram_re is used for latching the address. During the first cycle (of the single read access) sram_re is active (high) and memory is latching the address placed on sram_addr bus in internal address register. In the second cycle of the read access, SRAM memory is driving the outputs from memory cell selected by the latched address and then sram_re is inactive. For sequential read accesses the sram_re output is set active in order to read data from subsequent memory cells.

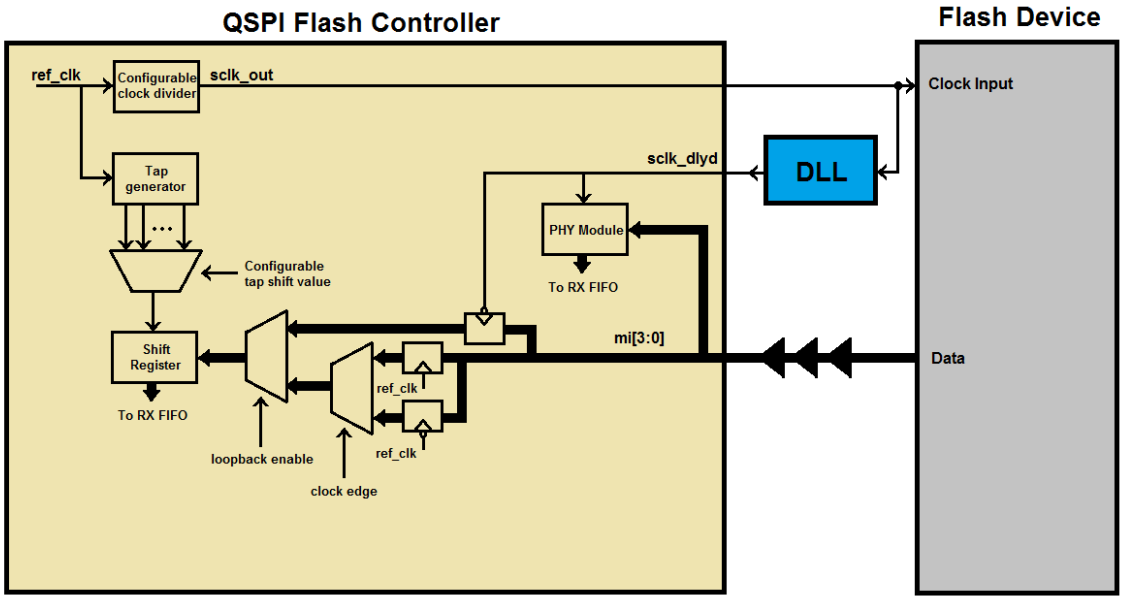
Figure 2.1 shows a timing diagram of the SRAM Interface

Figure 2.1. SRAM access example



2.1.11. Read Data Capturing

Figure 2.2. Sampling data diagram



There are three mechanisms of data capturing in the controller. They can be combined in some parts to ensure reliable sampling solution independent on the system requirements and the controller configuration. The mechanisms are like follows:

Section 2.1.11.1, "*Mechanism using taps*"	Describes the data capturing mechanism where sampling point is adjusted for one of the ref_clk edges inside divided SPI clock.
Section 2.1.11.2, "*Mechanism using external DLL*"	Describes the data capturing mechanism where valid data window is extended by flopping it with loopback SPI clock from external DLL.
Section 2.1.11.3, "*Mechanism using PHY Module*"	Describes all data capturing mechanisms implemented inside the PHY Module

2.1.11.1. Mechanism using taps

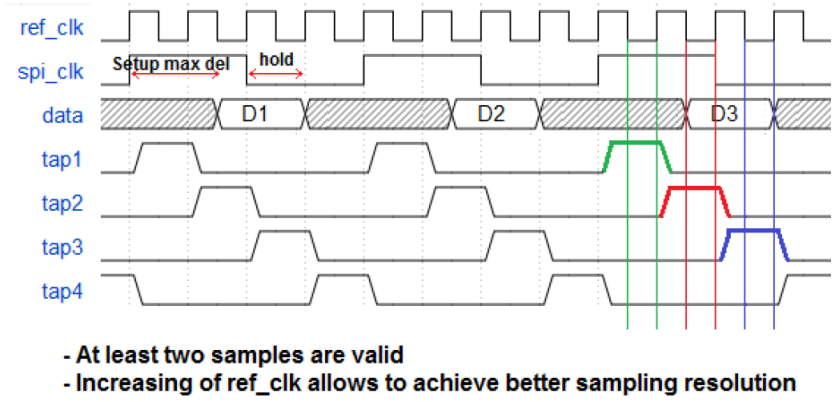
After POR, the adapted loopback clock circuit and the ref_clk delay register line both wake in a disabled state. This should be valid for most applications, and can be used for device enumeration, where the device clock will be configured slowly (default SPI clock divider = 32). The Read Data Capture Register (0x10) provides the control for the mechanism using taps.

Bit 5'th selects the edge of the ref_clk, on which data outputs from flash memory are sampled. There is the compilation option, to turn off capturing the data on negedge of the ref_clk. Some systems are very limited over implementation of both negative and positive edges of flip-flops located in single domain. Furthermore necessity of increasing of sampling resolution two times is required for very high performance reads only. When the system does not have such restrictive requirements, it is recommended to remove flip-flops (working on negedge of the ref_clk clock) from the project by change of the dedicated compilation parameter *(CDNS_QSPI_BOTH_EDGES_SAMPLING)* placed inside *cdns_qspi_spi_ctrl_core_defs.v* file.

Bits 4:1 controls the additional number of read data capture cycles (this is the fast ref_clk, running at least x4 of the device clock) that should be applied to the internal read data capture circuit. The large clock-to-out delay of the flash memory together with trace delays as well as other device delays may impose a maximum flash clock frequency which is less than the flash memory device itself can operate at. To compensate, software should set this register to a value that guarantees robust data captures.

Figure 2.3 shows a exemplary timing diagram of the capturing data with mechanism using taps in SDR edge mode.

Figure 2.3. Sampling mechanism using taps



2.1.11.2. Mechanism using external DLL

To further improve the input timing of the design, the master output clock (sclk_out) can be looped back in the logic that is instantiating the controller IP and then passed through an adjustable delay (not included in the QSPI IP core). This feature may be required when operating at high speeds with devices that have a very small guaranteed read data sampling window. For example, a device operating at 100MHz (ref_clk at 400MHz) where the clock to output valid time can be as high as 8ns or even 9ns, while the output HOLD time can be very low, even as low as just 1ns. This would give a valid sampling window of only a few ns. While this would remain valid across all operating conditions, that window applies to the device only. Extra delays associated with trace and pad in routing the read data back to the QSPI IP core must also be considered, and these will vary depending on operating conditions also. With such a small read data window, the ref_clk delay feature (allowing samples to be taken at 1.25ns (both edges) intervals when the ref_clk is 400MHz) may not be flexible enough for some purposes. Using the adapted delay line feature can provide a mechanism to allow the sampling point used by the QSPI IP to move beyond the ref_clk interval limitation and also may help compensate for operating condition variance. The adapted clock will be an input to the controller and, when enabled, will be used to clock the first register in the data-capture logic. Note that if the read data window is large enough, the ref_clk delay register described earlier will be sufficient to capture all data and the adapted loopback circuit will not be required. Clearing bit 0 enables

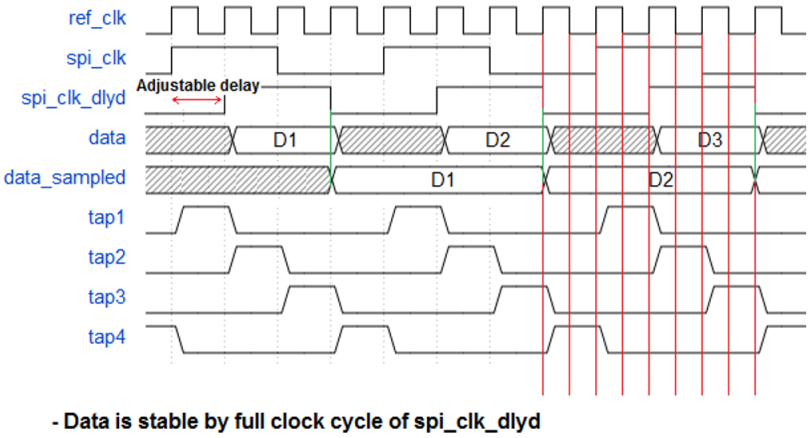
the adapted loopback clock circuit. Note that before the loopback clock is passed into the QSPI core, it must first be fed through a programmable delay line that is external to this IP. The user is responsible for implementing this delay line. The clock should be delayed such that the rising edge is in the center of the returned read data window.

The loopback clock input may be also used when it was decided to rely on the PHY sampling methods. The PHY Module has its own internal delay lines so there is no need to implement the external one. However loopback clock can be successfully used for estimating the external delay of the clock, especially when routing between data wires and loopback clock will be distributed adjacently. It may make tuning mechanism shorter and forces similar operating conditions fluctuations for chip areas where data and loopback clock are fed.

The decision which signal source is further forwarded into internal RX DLL depends on the configuration

Figure 2.4 shows a exemplary timing diagram of the capturing data with mechanism using external DLL:

Figure 2.4. Sampling mechanism using external DLL



2.1.11.3. Mechanism using PHY Module

When Cadence QSPI Controller and PHY are both integrated in single system, PHY Module is responsible for data capturing. More detailed description of all internal PHY sampling mechanisms is included in Section 4.2.2, "*Read Data Capturing by the PHY Module*".

2.1.12. BOOT feature description

Default BOOT feature assumes following:

- Baud Rate Divisor = /32 (sclk = ref_clk/32),
- Only CS[0] is active (Default BOOT feature allows to fetch data from single Flash Device only),
- Simple BOOT works in Extended SPI Protocol only (Single Mode),
- Simple BOOT feature uses simple Read Data Command (opcode: 0x03).

To further improve the BOOT feature the user has an option to change of default values of some programmable configuration fields to make BOOT feature more flexible. It is left for user to provide configuration compliant with the specification of the controller and Flash Device/Devices being used. The controller supports BOOT operation in XIP Mode that allows to achieve the maximum performance. The mentioned parameters can be changed in cdns_qspi_flash_ctrl_defs_default.v file before compilation process. Correct setting of them requires checking of structure of individual commands for connected Flash Device. The controller does not support booting in PHY Mode. The idea of BOOT assumes getting the data as soon as possible just after POR. To successfully work in PHY Mode, time consuming training process is needed. It is fully controllable by software but it was assumed that no programming sequence is needed

to fetch data in BOOT Mode. For reference, below is a table illustrating example of BOOT configuration for Fast Read instructions:

Table 2.5. Write examples

BOOT Mode	Parameters to change
100 MHz QUAD SDR	cdns_qspi_boot_baud_rate = 4'h1 cdns_qspi_boot_read_dummy = 5'h08 cdns_qspi_boot_data_type = 2'h2 cdns_qspi_boot_addr_type = 2'h2 cdns_qspi_boot_read_opcode = 8'heb
100 MHz DUAL SDR	cdns_qspi_boot_baud_rate = 4'h1 cdns_qspi_boot_read_dummy = 5'h08 cdns_qspi_boot_data_type = 2'h1 cdns_qspi_boot_addr_type = 2'h1 cdns_qspi_boot_read_opcode = 8'hbb
100 MHz SINGLE SDR	cdns_qspi_boot_baud_rate = 4'h1 cdns_qspi_boot_read_dummy = 5'h08 cdns_qspi_boot_data_type = 2'h0 cdns_qspi_boot_addr_type = 2'h0 cdns_qspi_boot_read_opcode = 8'h0b
50 MHz QUAD DDR	cdns_qspi_boot_baud_rate = 4'h3 cdns_qspi_boot_read_dummy = 5'h08 cdns_qspi_boot_data_type = 2'h2 cdns_qspi_boot_addr_type = 2'h2 cdns_qspi_boot_ddr_en = 1'h1 cdns_qspi_boot_read_opcode = 8'hed
100 MHz XIP QUAD SDR	cdns_qspi_boot_baud_rate = 4'h1 cdns_qspi_boot_xip = 1'h1 cdns_qspi_boot_read_dummy = 5'h08 cdns_qspi_boot_data_type = 2'h2 cdns_qspi_boot_addr_type = 2'h2 cdns_qspi_boot_read_opcode = 8'heb

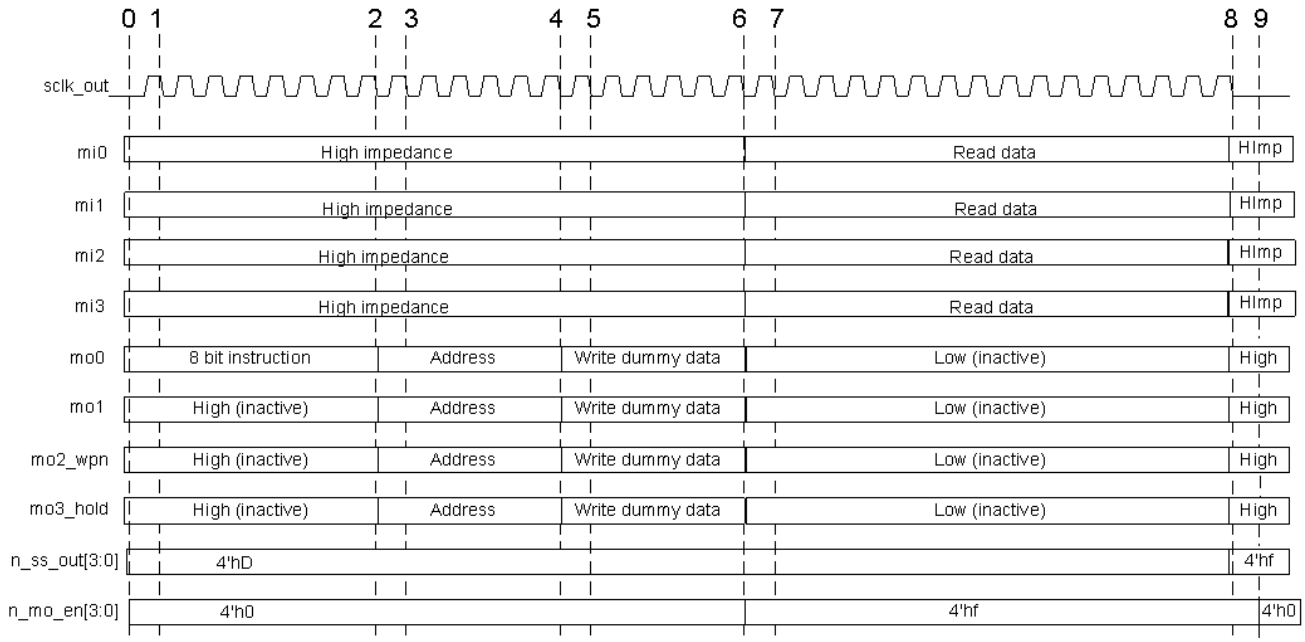
The cdns_qspi_boot_baud_rate values presented in the table were calculated based on assumption that frequency of the ref_clk is 400 MHz. Presented values can be slightly different for various devices. Furthermore the user should be aware that to executing BOOT reads in one of Multiple I/O Modes, the device has to be configured to handle these modes by non-volatile manner. Similarly, it has to be configured to XIP Mode to handle BOOT feature in XIP.

Since all clock domains are asynchronous for one another, there might be an scenario when the Master launches booting before flops from every domain are reset. For example with fast hclk and slow pclk, the BOOT read request may be issued before the configuration (APB domain) is reset and propagated. Since the system is aware of the clock values from all domains it is able to calculate safe delay when it is ensured to have all clock domains reset. The reset architecture of the controller is described in Section 1.1.5, “*Reset Diagram*”.

2.1.13. Example of an 8 byte Read Transfer

To help understand the high level behaviour of the Multiple-SPI interface, the following diagram shows the primary stages in a read transfer. Note this is just one type of read instruction (QUAD I/O READ) so there are many variations on this. Refer to the specification of particular Flash Device to get more combinations.

Figure 2.5. SPI transfer example



0. Start of transaction, chip select (n_ss_out) activates slave 1 (transitions from 4'hf to 4'hd).
1. Start of instruction phase. The first of 8 bits are output only on mo0. Other mo pins unused at this stage.
2. End of instruction phase. The last of the 8 instruction bits is output.
3. Start of address phase. The example uses a 3 byte address output across all 4 mo pins. One address nibble is output per clock cycle so 3 bytes takes 6 cycles.
4. End of address phase. The last address nibble is output using all 4 mo pins.
5. The example read transaction requires 3 bytes of dummy write data. This edge is where the first of 6 nibbles are output across all 4 mo pins.
6. Last nibble of dummy write data. This is the last write phase of the read transfer therefore the output enable (n_mo_en) switches from 4'h0 to 4'hf.
7. First cycle of read data on input mi pins. Example is reading 8 bytes which will take 16 cycles using all 4 mi pins. While read data is being received, the mo signals are inactive and low.
8. Last cycle of read data input. Mi input pins switch back to being high impedance. Mo output pins switch to high. Chip enable is de-asserted for slave1. Transaction complete.
9. Output enable switches back to 4'h0 read for next transaction.

2.2. Programmer's Guide

Software is responsible for configuring the QSPI controller before it can communicate with the FLASH device.

We assume that the controller’s static configuration bits are setup before the QSPI controller is "enabled" via QSPI Configuration Register bit[0]. This is necessary in order to avoid any clock boundary issues in paths where metastability protection has not been implemented. If the user wishes to change the controller's configuration, we recommend that the QSPI enable bit is first set to "disabled" before reconfiguring.

2.2.1. Configuring the QSPI Controller for use after reset

The QSPI Controller has been designed to wake up in a state that is suitable for performing basic reads and writes using the direct access controller. The BASIC read (opcode 0x03) and BASIC write (opcode 0x02) instructions are operations supported by all target devices. The Controller also wakes up with a baud rate divider setting of divide-by-32. Assuming the reference clock is operating at 400MHz after reset, then this means the effective SPI clock is just 12.5 MHz. This should be slow enough to meet all timing requirements of all target devices without any further device programming.

If the target device does not use 3 address bytes, the device size configuration register must be modified to the appropriate size.

If software plans to write to the device, and the number of bytes per device page is NOT equal to 256, then the device size configuration register must also be modified. All devices tested on this project use a page size of 256 bytes, in which case no register update is required.

While not a requirement, it is prudent for software to enable the write protect feature prior to enabling the QSPI controller. This will block any AHB writes from taking effect. To do so, the protection registers (Lower Write Protection Register, Upper Write Protection Register and Write Protection Register) should be setup and the number of bytes per device block in the device size configuration register should also be setup.

After POR, software can read from and write to the FLASH device (albeit slowly). Enabling/Disabling the controller and DAC is achieved with just one write to corresponding fields of the QSPI Configuration Register (0x00). Take note to maintain the default values of the baud rate divisor and the default state of CPOL/CPHA which also are located in this register. A write data value of 0x00780081 is recommended.

2.2.2. Configuring the QSPI Controller for optimal use

To access the flash optimally, software must configure the controller accurately:

1. Wait until any pending STIG or INDAC operation has completed or poll QSPI Configuration Register bit[31]

2. Disable the DAC – QSPI Configuration Register bit[7]. It is permitted, but not necessary to also disable the QSPI controller completely here via QSPI Configuration Register bit[0].

3. Update the Device Read Instruction Register and Device Write Instruction Register for the instruction type you wish to use for indirect and direct writes and reads.

4. Update the Mode Bit Configuration Register if mode bits have been enabled in the Device Read Instruction Register bit[20]

5. Update the device size configuration register if the contents are incorrect. Note parts or all of this register may have been updated after initialization. The number of address bytes is a key configuration setting required for performing reads and writes. The number of bytes per page is required for performing any write. The number of bytes per device block is only required if the write protect feature is used. If the default values are correct for the target device, or if some of the values (not including the number address bytes) were incorrect but device writes were not permitted.

6. Update the Device Delay Register This register allows the user to tweak how the chip select is driven after each FLASH access. This is required as each device may have different timing requirements. As the serial clock frequency is increased, these timing requirements become more important. Note the numbers programmed in this register are based on the period of ref_clk. Example: An ATMEL device needs 50ns minimum time before CS can be re-asserted after it has been de-asserted. By default, the controller will only provide a minimum of 1 SCLK period. When the device is operating at 100MHz, the SCLK period is only 10ns, so 40ns extra is required. Since the register defines the number of ref_clk cycles to add, and ref_clk is running at 400MHz (2.5ns period), then the user should program a value of at least 16 to the d_nss field of this register. This delay can be extended during auto-polling phase. There is possibility to define the polling repetition delay in the Write Completion Control Register bits[31:24]

7. Update the Remap Address Register, if required. Affects DAC path only. Refer to Section 2.1.1.2, "*Remapping the AHB address*" for further details.

8. Setup and enable write protection registers (Lower Write Protection Register, Upper Write Protection Register and Write Protection Register) if they are required and if they have not already been setup from post initialization.

9. Enable required interrupts via the Interrupt Mask Register.

10. Setup the baud rate divisor in the QSPI Configuration Register bit[22] to define the required clock frequency of the target device.

11. Update the Read Data Capture Register This register will delay when the read data is captured and can help when the read data path from the device to the controller is long and the device clock frequency is high. An update to this register may not be necessary.

12. Enable the QSPI Controller and the DAC via the QSPI Configuration Register

NOTE:

The configuration related to PHY Mode was described in PHY Programmer's Guide.

2.2.3. Using the Flash Command Control Register (STIG Operation)

The Flash Command Control register provides software means to access the FLASH device in a flexible and programmable manner. This is known as a STIG operation (Software Triggered Instruction Generator). The instruction opcode, number of address bytes (if any), the address itself, number of dummy cycles (if any), number of write data bytes (if any), the write data itself and the number of read data bytes (if any) can be programmed. Once these have been programmed, software can trigger the command via bit 0 and wait for its acceptance by polling bit 1. When this bit turns de-asserted, another STIG can be triggered. This method of accessing the FLASH is the typical mechanism that software would use to access the FLASH device's registers, as well as for performing ERASE operations. It can also be used to access the FLASH array itself, although the maximum of 8 data bytes may be read or written at any one time, defined in the Flash Command Write and Read Data registers (Flash Command Read Data Register (Lower), Flash Command Read Data Register (Upper), Flash Command Write Data Register (Lower) and Flash Command Write Data Register (Upper)). This number of bytes can be extended for Read Data commands using additional STIG Memory Bank controlled by Flash Command Control Register (Using STIG) Register bit[2] and Flash Command Control Memory Register (Using STIG).

Commands issued using this interface have a higher priority than all other READ accesses coming from AHB, and will therefore interrupt any READ commands being requested by the indirect or direct controllers.

2.2.4. Using SPI Legacy Mode

SPI legacy mode allows software to access the internal TX-FIFO and RX-FIFO directly, thus bypassing the direct, indirect and STIG controllers.

Legacy mode allows the user to issue any FLASH instruction to the device, but does place a heavy software overhead in order to manage the fill levels of the FIFO's effectively. This is because the legacy SPI core is bi-directional in nature, with data continuously being transferred in either direction while the chip select is enabled. Even if the driver only wishes to read data from the FLASH device, dummy data must be written out to ensure the chip select stays active, and vice versa for write transactions. This means that to perform a basic READ of 4 bytes to a device that has 3 address bytes, software would have to write a total of 8 bytes to the TX FIFO. The first byte would be the instruction opcode, the next 3 bytes are the address, and the final 4 bytes would be dummy data to ensure the chip select stays active while the read data is returned. Similarly, since 8 bytes were written to the TX-FIFO, software should expect 8 bytes to be returned in the RX-FIFO. The first 4 bytes of this would be discarded, leaving the final 4 bytes holding the required READ data.

Since the TX-FIFO and RX-FIFO are of limited depth, software has a responsibility to maintain the FIFO levels to ensure the TX-FIFO does not become exhausted during the instruction execution and the RX-FIFO doesn't overflow. This can place a lot of overhead on software. Interrupts are provided to indicate when the fill levels pass programmable

watermarks, which are themselves programmable registers: TX Threshold Register and RX Threshold Register. The limited depth may impose the limitation over execution of some specific SPI commands in legacy mode. It should be noticed that the controller interprets all transmitted bytes as valid. For example, if the Flash Device was configured to return valid data after many dummy cycles, the TX FIFO could become full before the controller sends all of dummy data.

Software accesses the TX-FIFO by writing any value to any address via the AHB interface to the QSPI controller while legacy mode is enabled. Software accesses the RX-FIFO by reading any address via the AHB interface to the QSPI controller while legacy mode is enabled.

2.2.5. Using XIP Mode

Since there is no consistent standard of servicing XIP read operations among FLASH Devices it is not ensured that described approaches cover all cases supported by FLASH Devices present on the market. Most common approaches have been listed.

2.2.5.1. Entering XIP mode from POR

XIP is a mode that can be entered in a non-volatile way if the device has XIP enabled as a non-volatile configuration setting. Software will not be able to discover the state of XIP from POR via FLASH status register reads as the only operation a FLASH device will recognize when XIP mode is enabled is an XIP read operation.

If it is already known that the device will enter XIP from POR, then the Mode Bit Configuration Register and QSPI Configuration Register bit[18] should be set in initial boot.

If it is not already known that the device will enter XIP from POR, and XIP from POR may be supported by the attached FLASH device, then software can attempt to exit XIP mode by issuing an XIP exit command using a STIG command (via the Flash Command Control Register (Using STIG)). To do this, software must be aware of the mode bit requirements of that device, as XIP entry and exit changes per device. Devices that support XIP from POR from the list of tested devices are Micron N25Q and MT25Q devices. For these devices, exiting XIP mode can be performed by configuring the opcode to 8'h00, the number of address bytes to 3, the number of dummy cycles to 16 and the number of read bytes to 1. The address that should be programmed should be {8 mode bits for exiting XIP mode, 16'h0000}. The mode bits for the Micron device must be 8'b10000000.

Note the XIP from POR mode can be enabled by issuing a NVCR write command via the Flash Command Control Register (Using STIG). As this does not take effect until the next POR sequence, the effect of this will be picked up during the next enumeration sequence.

2.2.5.2. Entering XIP mode otherwise

XIP mode is supported in most FLASH devices. However, FLASH manufacturers do not have a consistent standard approach for achieving this. Most use signature bits that are sent to the device immediately following the address bytes. Some, like the Micron devices, use signature bits and also require a FLASH device configuration register write to enable XIP. For the FLASH devices that must be compliant to this controller, the following steps can be taken by software to enter XIP mode:

2.2.5.2.1. Micron N25Q and MT25Q Devices (No support for Basic-XIP)

XIP mode must first be enabled by setting the corresponding field of VCR within the FLASH device. The VCR can be written to using the Flash Command Control Register to issue a VCR write command. The steps to follow are described below:

1. Disable the Direct Access Controller and Indirect Access Controller (QSPI Configuration Register bit[7]) to ensure no new AHB read accesses will be sent to the FLASH device
2. Configure the Flash Command Control Register (Using STIG) to issue a VCR write to FLASH memory
3. Set the XIP mode bits in the Mode Bit Configuration Register (0x28) to 8'b00000000

4. Enable the local controllers XIP mode by setting QSPI Configuration Register bit[17]
5. Re-enable the Direct Access Controller and, if required, the Indirect Access Controller

2.2.5.2.2. Other Micron Devices (Supporting Basic-XIP)

1. Disable the Direct Access Controller and Indirect Access Controller (QSPI Configuration Register bit[7]) to ensure no new AHB read accesses will be sent to the FLASH device
2. Set the XIP mode bits in the Mode Bit Configuration Register (0x28) to 8'b10000000
3. Enable the local controllers XIP mode by setting QSPI Configuration Register bit[17]
4. Re-enable the Direct Access Controller and, if required, the Indirect Access Controller

2.2.5.2.3. Winbond Devices

1. Disable the Direct Access Controller and Indirect Access Controller (QSPI Configuration Register bit[7]) to ensure no new AHB read accesses will be sent to the FLASH device
2. Set the XIP mode bits in the Mode Bit Configuration Register (0x28) to 8'b00100000
3. Enable the local controllers XIP mode by setting QSPI Configuration Register bit[17]
4. Re-enable the Direct Access Controller and, if required, the Indirect Access Controller

2.2.5.2.4. Spansion Devices

1. Disable the Direct Access Controller and Indirect Access Controller (QSPI Configuration Register bit[7]) to ensure no new AHB read accesses will be sent to the FLASH device
2. Set the XIP mode bits in the Mode Bit Configuration Register (0x28) to 8'b10100000
3. Enable the local controllers XIP mode by setting QSPI Configuration Register bit[17]
4. Re-enable the Direct Access Controller and, if required, the Indirect Access Controller

2.2.5.3. Exiting XIP mode

To exit XIP mode, software should first disable the Direct Access Controller and Indirect Access Controller to ensure no new AHB read accesses will be sent to the FLASH device. It should then set the mode bits to anything other than the mode bits specified in the specification of corresponding Flash Device. These are dependent on the FLASH device and manufacturer. Software should then reset QSPI Configuration Register bit[17].

Note the FLASH device must see a READ instruction before it can disable its internal XIP mode state, so this means XIP mode will internally stay active until the next READ instruction is serviced. Care must be taken to ensure that XIP mode is disabled before the end of any READ sequence.

2.2.6. Using Indirect Data Transfer Mode

The software sequence that should be followed when using the Indirect read controller, including how the DMA peripheral interface is setup is specified earlier in this document (Section 2.1.3.2, "*Indirect Write Controller*" and Section 2.1.3.1, "*Indirect Read Controller*").

2.2.7. Remapping AHB addresses

The remap feature of the QSPI allows software to define how incoming AHB addresses translate to FLASH addresses when the access passes through the direct access controller. By default there is a 1:1 mapping. When QSPI Configuration

Register bit[16] is enabled, the incoming AHB address is remapped to address+N, where N is the value stored in the Mode Bit Configuration Register (0x24). It is recommended that the QSPI is disabled before configuring the remap registers.

2.2.8. Servicing Interrupts

The controller provides a single active high level sensitive interrupt pin. Refer to the Interrupt Status Register (0x40) for details of all the interrupt sources provided.

The interrupt source can be cleared when software reads the interrupt status register. Reported interrupts are cleaned by writing "1" (write-to-clear approach is implemented).

2.2.9. Using the AHB Protection Registers

On POR, the AHB protection mechanism wakes in a disabled state. Software can use the protection registers to block AHB writes to certain regions. Refer to Section 2.1.1.3, "*Write Protection*" for details. It is recommended that the QSPI is disabled before configuring the protection registers.

2.3. Programming Interface

2.3.1. QSPI Configuration Register

Description: This register contains basic configuration fields of the controller.

Table 2.6. QSPI Configuration Register

Offset	Bit	R/W	Description	Reset
0x00	31	RO	Serial Interface and QSPI pipeline is IDLE. This is a STATUS read-only bit. Note this is a re-timed signal, so there will be some inherent delay on the generation of this status signal. **Note** It is recommended to wait at least 4 cycles of ref_clk between getting IDLE bit asserted and switching any configuration field. This latency ensures that all low level synchronization is done, FIFOs are empty and the controller can be safely introduced into another mode of operation	1'h1
0x00	30:26	RO	Unused bits. Read as zero.	5'h0
0x00	25	R/W	Pipeline PHY Mode enable bit This bit is relevant only for configuration with PHY Module. It should be asserted to "1" between consecutive PHY pipeline reads transfers and de-asserted to "0" otherwise. Detailed description of pipeline feature is included in PHY Pipeline Mode.	1'h0
0x00	24	R/W	Enable DTR Protocol This bit should be set if device is configured to work in DTR protocol.	1'h0

Offset	Bit	R/W	Description	Reset
			Note DTR protocol provides all commands in DTR Mode. There are DTR Read commands which can be handled in STR Protocol (but in DTR Mode). This bit should be equal "0" to executing them. DTR commands in STR protocol are controlled by 10th bit of Device Read Configuration Register.	
0x00	23	R/W	Enable AHB Decoder (Direct Access Mode Only) When set to "1", QSPI Configuration Register bit[13:10] are not relevant. Active slave is based on actual AHB address. The partition for each device is calculated with respect to Device Size Configuration Register bits[28:21]	1'h0
0x00	22:19	R/W	Master mode baud rate divisor (2 to 32), SPI baud rate = (master reference clock)/BD Where BD is: 4'b0000 = /2 4'b0001 = /4 4'b0010 = /6 4'b0011 = /8 4'b0100 = /10 4'b0101 = /12 4'b0110 = /14 4'b0111 = /16 4'b1000 = /18 ... 4'b1111 = /32 Set this register up before enabling the QSPI controller.	4'hf
0x00	18	R/W	Enter XIP Mode immediately 0 = If XIP is enabled, then setting to "0" will cause the controller to exit XIP mode after the next READ instruction has completed. 1 = Operate the device in XIP mode immediately Use this register when the external device is introduced to XIP mode (as per the contents of its configuration register). The	1'h0

Offset	Bit	R/W	Description	Reset
			controller will assume the next READ instruction will be passed to the device as an XIP instruction, and therefore will not require the READ opcode to be transferred. **Note** To exit XIP mode, this bit should be set to "0". This will take effect in the attached device only AFTER the next READ instruction is executed. Software should therefore ensure that at least one READ instruction is requested after resetting this bit before it can be sure XIP mode in the device is exited. This bit is synchronized in hardware.	
0x00	17	R/W	Enter XIP Mode on next READ 0 = If XIP is enabled, then setting to "0" will cause the controller to exit XIP mode after the next READ instruction has completed. 1 = If XIP is disabled, then setting to "1" will inform the controller that the device is ready to enter XIP on the next READ instruction. The controller will therefore send the appropriate command sequence, including mode bits to cause the device to enter XIP mode. Use this register after the controller has ensured the attached FLASH device has been configured to be ready to enter XIP mode. **Note** To exit XIP mode, this bit should be set to "0". This will take effect in the attached device only AFTER the next READ instruction is executed. Software should therefore ensure that at least one READ instruction is requested after resetting this bit before it can be sure XIP mode in the device is exited. This bit is synchronized in hardware.	1'h0
0x00	16	R/W	Enable AHB Address Re-mapping (Direct Access Mode Only) When set to "1", the incoming AHB address will be adapted and sent to the FLASH device as (address + N), where N is the value stored in the remap address register. Use this register after the controller has ensured the attached FLASH device has been configured to be ready to enter XIP mode. This bit is synchronized in hardware.	1'h0

Offset	Bit	R/W	Description	Reset
0x00	15	R/W	Enable DMA Peripheral Interface Set to "1" to enable the DMA handshaking logic. When enabled the QSPI will trigger DMA transfer requests via the DMA peripheral interface. Set to '0' to disable DMA Peripheral Interface. This bit is synchronized in hardware.	1'h0
0x00	14	R/W	Set to drive the Write Protect pin of the FLASH device. This is resynchronized to the generated memory clock as necessary. Note that the WP pin is only valid in SINGLE or DUAL transfer modes. During QUAD transfers, the WP pin is used for transferring data and therefore any setting of this register bit will be ignored. This bit is synchronized in hardware.	1'h0
0x00	13:10	R/W	Peripheral chip select lines If bit 9 of this register = 0, *ss[3:0]* are output thus: *ss[3:0]* *n_ss_out[3:0]* *4'bxxx0* *4'b1110* *4'bxx01* *4'b1101* *4'bx011* *4'b1011* *4'b0111* *4'b0111* *4'b1111* *4'b1111 (no peripheral selected)* If bit 9 of this register = 1, *ss[3:0]* directly drives *n_ss_out[3:0]*	1'h0
0x00	9	R/W	Peripheral select decode 0 = only 1 of 4 selects *n_ss_out[3:0]* is active 1 = allow external 4-to-16 decode (*n_ss_out*=ss)	1'h0
0x00	8	R/W	Legacy IP Mode Enable 0 = Use Direct Access Controller/Indirect Access Controller or STIG interface for data transfers 1 = When set to "1", legacy mode is enabled. In this mode, any write to the controller via the AHB interface is serialized and sent to the FLASH device. Any valid AHB read will pop the internal RX-FIFO, retrieving data that was forwarded by the external FLASH device on the SPI lines. 4, 2 or 1 byte transfers are permitted and controlled via the HSIZE input. This bit is synchronized in hardware.	1'h0
0x00	7	R/W	Enable Direct Access Controller	1'h1

Offset	Bit	R/W	Description	Reset
			0 = disable the Direct Access Controller once current transfer of the data word is completed. 1 = enable the Direct Access Controller When the Direct Access Controller and Indirect Access Controller are both disabled, all AHB requests are completed with an error response. This bit is synchronized in hardware.	
0x00	6:4	RO	Unused bits. Read as zero.	3'h0
0x00	3	R/W	PHY Mode enable When enabled, the controller is informed that PHY Module is to be used for handling SPI transfers. Detailed description of this feature is included in the Chapter 4, *PHY Module Specification*	1'h0
0x00	2	R/W	Clock phase - This maps to the standard SPI CPHA transfer format. **Note** Keep this bit low when operating in DDR Mode or DDR Protocol to ensure the falling edge of SPI clock as the last one. If PHY Mode is enabled, it must be ensured that shift delay of SPI clock was adjusted to range from 180° to 360° of ref_clk period.	1'h0
0x00	1	R/W	Clock polarity outside SPI word - This maps to the standard SPI CPOL transfer format. **Note** Keep this bit low when operating in DDR Mode or DDR Protocol to ensure the falling edge of SPI clock as the last one. If PHY Mode is enabled, it must be ensured that shift delay of SPI clock was adjusted to range from 180° to 360° of ref_clk period.	1'h0
0x00	0	R/W	QSPI Enable 0 = disable the Multiple-SPI interface once current transfer of the data word is completed. 1 = enable the Multiple-SPI interface When set to low, all output enables are inactive and all pins are set to input mode. This bit is synchronized in hardware.	1'h1

2.3.2. Device Read Instruction Register

Description: This register defines the configuration of Multiple-SPI READ instruction. This register should be setup while the controller is idle.

Table 2.7. Device Read Instruction Register

Offset	Bit	R/W	Description	Reset
0x04	31:29	RO	Unused bits. Read as zero.	3'h0
0x04	28:24	R/W	Number of Dummy Clock Cycles required by device for Read Instruction.	5'h00
0x04	23:21	RO	Unused bits. Read as zero.	3'h0
0x04	20	R/W	Mode Bit Enable Set to "1" to ensure the mode bits as defined in the Mode Bit Configuration Register are sent following the address bytes.	1'h0
0x04	19:18	RO	Unused bits. Read as zero.	2'h0
0x04	17:16	R/W	Data Transfer Type for Standard SPI modes (as defined by the instruction type in bits [9:8]) 2'b00: SIO mode - data is shifted to the device on DQ0 only and from the device on DQ1 only 2'b01: Used for Dual Input/Output instructions. For data transfers, DQ0 and DQ1 are used as both inputs and outputs. 2'b10: Used for Quad Input/Output instructions. For data transfers, DQ0, DQ1, DQ2 and DQ3 are used as both inputs and outputs.	2'h0
0x04	15:14	RO	Unused bits. Read as zero.	2'h0
0x04	13:12	R/W	Address Transfer Type for Standard SPI modes (as defined by the instruction type in bits [9:8]) 2'b00: Addresses can be shifted to the device on DQ0 only 2'b01: Addresses can be shifted to the device on DQ0 and DQ1 only 2'b10: Addresses can be shifted to the device on DQ0, DQ1, DQ2 and DQ3	2'h0
0x04	11	RO	Unused bit. Read as zero.	1'h0
0x04	10	R/W	DDR Bit Enable Set to "1" when opcode from bits 7 to 0 is compliant with DDR command. Master output data is issued by the controller in DDR fashion starting of the address SPI transfer phase. It is not applicable for STIG Mode.	1'h0
0x04	9:8	R/W	Instruction Type 2'b00: Use Standard SPI mode (instruction always shifted into the device on DQ0 only)	2'h0

Offset	Bit	R/W	Description	Reset
			2'b01: Use DIO-SPI mode (Instructions, Address and Data always sent on DQ0 and DQ1) 2'b10: Use QIO-SPI mode (Instructions, Address and Data always sent on DQ0,DQ1,DQ2 and DQ3) **Note** These bits are relevant not just to READ transfers. They are global settings and will affect READ's, WRITES via the DAC and INDAC as well as any transfer sent using the STIG.	
0x04	7:0	R/W	Read Opcode to use when not in XIP mode	8'h03

2.3.3. Device Write Instruction Register

Description: This register defines the configuration of Multiple-SPI WRITE (Program Page) instruction. This register should be setup while the controller is idle.

Table 2.8. Device Write Instruction Register

Offset	Bit	R/W	Description	Reset
0x08	31:29	RO	Unused bits. Read as zero.	3'h0
0x08	28:24	R/W	Number of Dummy Clock Cycles required by device for Write Instruction.	5'h00
0x08	23:18	RO	Unused bits. Read as zero.	6'h00
0x08	17:16	R/W	Data Transfer Type for Standard SPI modes (as defined by the instruction type in Device Read Instruction Register bits[9:8]) 2'b00: SIO mode - data is shifted to the device on DQ0 only and from the device on DQ1 only 2'b01: Used for Dual Input/Output instructions. For data transfers, DQ0 and DQ1 are used as both inputs and outputs. 2'b10: Used for Quad Input/Output instructions. For data transfers, DQ0, DQ1, DQ2 and DQ3 are used as both inputs and outputs.	2'h0
0x08	15:14	RO	Unused bits. Read as zero.	2'h0
0x08	13:12	R/W	Address Transfer Type for Standard SPI modes (as defined by the instruction type in Device Read Instruction Register bits[9:8]) 2'b00: Addresses can be shifted to the device on DQ0 only 2'b01: Addresses can be shifted to the device on DQ0 and DQ1 only 2'b10: Addresses can be shifted to the device on DQ0, DQ1, DQ2 and DQ3	2'h0
0x08	11:9	RO	Unused bits. Read as zero.	3'h0

Offset	Bit	R/W	Description	Reset
0x08	8	R/W	WEL Disable This bit allows the user to turn off automatic issuing of WEL Command before write operation for DAC or INDAC.	1'h0
0x08	7:0	R/W	Write Opcode	8'h02

2.3.4. Device Delay Register

Description: This register is used to introduce relative delays into the generation of the master output signals. This allows the timings to be adjusted to meet the requirements of a specific flash device. All timings are defined in cycles of the SPI master ref clock. This register should be setup while the controller is idle.

For an illustration of the register fields, refer to the two diagrams below this table: Device Delay Register timing diagrams.

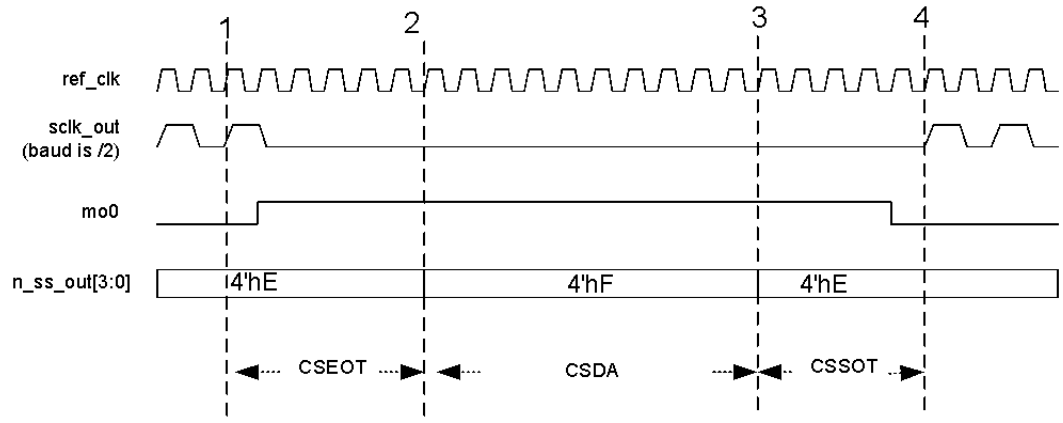
Table 2.9. Device Delay Register

Offset	Bit	R/W	Description	Reset
0x0C	31:24	R/W	CSDA - Chip Select De-Assert Added delay in master reference clocks (ref_clk) for the length that the master mode chip select outputs are de-asserted between transactions. The minimum delay for chip select to be de-asserted (CSDA=0) is: 1 sclk_out + 1 ref_clk to ensure the chip select is never re-asserted within an sclk_out period. If CSDA=N, then the chip select de-assert time will be: 1 sclk_out + 1 ref_clk + N ref_clks.	8'h00
0x0C	23:16	R/W	CSDADS – Chip Select De-Assert Different Slaves Delay in master reference clocks (ref_clk) between one chip select being de-activated and the activation of another. This is used to ensure a quiet period between the selection of two different slaves. CSDADS is only relevant when switching between 2 different external flash devices. An example scenario might be if you have 2 FLASH devices named A and B, each connected to the controller on a different chip select. You might have a requirement to have chip select inactive for a particular period of time between each successive non-sequential transfer of the same device. You would control this using bits CSDA. There may be a different requirement to hold chip select inactive for a longer period of time between the end of one transfer on device A and the start of the next transfer on device B. You would control this using CSDADS. The minimum delay (CSDADS=0) is: 1 sclk_out + 3 ref_clk.	8'h00

Offset	Bit	R/W	Description	Reset
			If CSDA=N, then the chip select de-assert time will be: 1 sclk_out + 1 ref_clk + N ref_clks.	
0x0C	15:8	R/W	CSEOT – Chip Select End Of Transfer Delay in master reference clocks between last bit of current transaction and de-asserting the device chip select (*n_ss_out*). By default (when CSEOT=0), the chip select will be de-asserted on the last falling edge of sclk_out in SPI Mode 0 or on the last rising edge + half spi clock cycle in SPI Mode 3 at the completion of the current transaction. If CSEOT=N, then chip selected will de-assert N ref_clks after the last falling edge of sclk_out in SPI Mode 0 or on the last rising edge + half spi clock cycle in SPI Mode 3.	8'h00
0x0C	7:0	R/W	CSSOT – Chip Select Start Of Transfer Delay in master reference clocks between setting *n_ss_out* low and first bit transfer. By default (CSSOT=0), chip select will be asserted half a SCLK period before the first rising edge of sclk_out. If CSSOT=N, chip select will be asserted half a sclk_out period before the first rising edge of sclk_out + N ref_clks.	8'h00

Figure 2.6 shows chip select behaviour between 2 consecutive transfers to the same slave

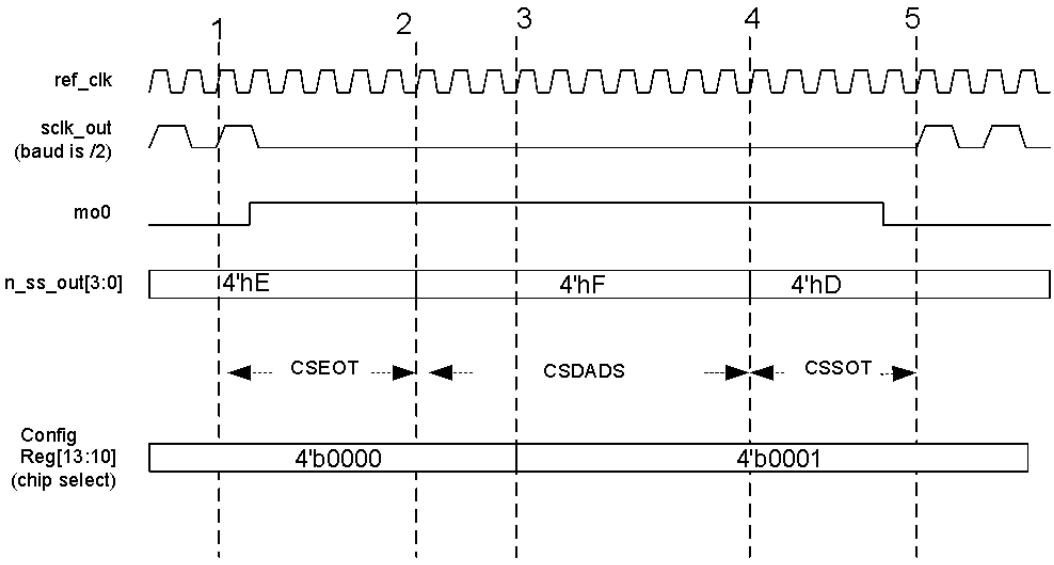
Figure 2.6. Device Delay Register timing diagram (single slave)



1. Last active sclk_out cycle in SPI transaction. This does not mark the end of the overall SPI transaction.
2. Chip select de-asserted, SPI transaction is now complete.
3. Chip select asserted, first step of a new SPI transaction.
4. This is the first rising edge of sclk_out for the new transaction.

Figure 2.7 shows chip select behaviour between 2 consecutive transfers to different slaves

Figure 2.7. Device Delay Register timing diagram (different slaves)



1. Last active sclk_out cycle in SPI transaction. This does not mark the end of the overall SPI transaction.
2. Chip select slave 0 de-asserted, SPI transaction is now complete.
3. While controller is idle, APB write to config register, selects slave 1 and de-selects slave 0.
4. Chip select for slave 1 asserted, first step of a new SPI transaction.
5. This is the first rising edge of sclk_out for the new transaction.

2.3.5. Read Data Capture Register

Description: This register is used to adjust SPI transfer conditions in order to fetch and capture data reliably. This register should be setup while the controller is idle.

Table 2.10. Read Data Capture Register

Offset	Bit	R/W	Description	Reset
0x10	31:20	RO	Unused bits. Read as zero.	12'h000
0x10	19:16	R/W	Delay the transmitted data (Master Output) by the programmable number of ref_clk cycles (in order to improve the hold timing during transfers in DDR Mode). **Note** This field is only relevant when DDR Read Command is executed. Otherwise can be ignored. delay_value = 1 x ref_clk + [field_value] x ref_clk	4'h0
0x10	5	R/W	Sample edge selection (of the flash memory data outputs). 0 = data outputs from flash memory are sampled on falling edge of the ref_clk	1'h0

Offset	Bit	R/W	Description	Reset
			1 = data outputs from flash memory are sampled on rising edge of the ref_clk	
0x10	4:1	R/W	Delay the read data capturing logic by the programmable number of ref_clk cycles	4'h0
0x10	0	R/W	Bypass of the adapted loopback clock circuit 0 = enable the adapted loopback clock circuit 1 = disable the adapted loopback clock circuit The usage of the loopback clock input for capturing data is different depending on the value of the QSPI Configuration Register bit[3]. If PHY Mode is configured, please refer to Section 4.2.2, *"Read Data Capturing by the PHY Module"*, otherwise, please refer to Section 2.1.11.2, *"Mechanism using external DLL"* to get more detailed information.	1'h1

2.3.6. Device Size Configuration Register

Description: This register allows the user to define the memory organization of using Flash Devices. This register should be setup while the controller is idle.

Table 2.11. Device Size Configuration Register

Offset	Bit	R/W	Description	Reset
0x14	31:29	RO	Unused bits. Read as zero.	2'h0
0x14	28:27	R/W	Size of Flash Device connected to CS[3] pin. This field is only valid when AHB Decoder is enabled. 2'b00 = size of 512Mb 2'b01 = size of 1Gb 2'b10 = size of 2Gb 2'b11 = size of 4Gb	2'h0
0x14	26:25	R/W	Size of Flash Device connected to CS[2] pin. This field is only valid when AHB Decoder is enabled. 2'b00 = size of 512Mb 2'b01 = size of 1Gb 2'b10 = size of 2Gb 2'b11 = size of 4Gb	2'h0
0x14	24:23	R/W	Size of Flash Device connected to CS[1] pin. This field is only valid when AHB Decoder is enabled. 2'b00 = size of 512Mb 2'b01 = size of 1Gb	2'h0

Offset	Bit	R/W	Description	Reset
			2'b10 = size of 2Gb 2'b11 = size of 4Gb	
0x14	22:21	R/W	Size of Flash Device connected to CS[0] pin. This field is only valid when AHB Decoder is enabled. 2'b00 = size of 512Mb 2'b01 = size of 1Gb 2'b10 = size of 2Gb 2'b11 = size of 4Gb	2'h0
0x14	20:16	R/W	Number of bytes per block This is required by the controller for performing the write protection logic. The number of bytes per block must be a power of 2 number. I.e. 0 = 1 byte 1 = 2 bytes 3 = 8 bytes (...) 16 = 65535 bytes etc.	5'h00
0x14	15:4	R/W	Number of bytes per device page This is required by the controller for performing FLASH writes up to and across page boundaries.	12'h100
0x14	3:0	R/W	Number of address bytes A value of 0 = 1 byte etc.	4'h2

2.3.7. SRAM Partition Configuration Register

Description: This register allows the user to allocate the SRAM area for indirect writes and reads. N in this register is the SRAM depth, configured in the cdns_qspi_flash_ctrl_defs_default.v. The default value of N is 8 what assumes that external SRAM hard macro has 2**8=256 memory locations with fixed data size of 32 bits. This register should be setup while the controller is idle.

Table 2.12. SRAM Partition Configuration Register

Offset	Bit	R/W	Description	Reset
0x18	31:N	RO	Unused bits. Read as zero.	N{1'b0}
0x18	N-1:0	R/W	Defines the size of the indirect read partition in the SRAM, in units of SRAM locations. By default, half of the SRAM is reserved for indirect read operation, and half for indirect write. The size of this register will scale with the depth of the SRAM.	{1'b1,{N-1{1'b0}}} Top bit is set, all other bits are zeroed

Offset	Bit	R/W	Description	Reset
			The number of locations allocated to indirect read = SRAM_PARTITION_REG+1 The number of locations allocated to indirect write = (2**N)-SRAM_PARTITION_REG Avoid setting SRAM_PARTITION_REG to N{1'b0} or (2**N)-1)	

2.3.8. Indirect AHB Address Trigger Register

Description: This register allows the user to define the address distinguishing DAC access from triggered INDAC one. This register should be setup while the controller is idle.

Table 2.13. Indirect AHB Address Trigger Register

Offset	Bit	R/W	Description	Reset
0x1C	31:0	R/W	Indirect Trigger Address This is the base address that will be used by the AHB controller. When the incoming AHB read access address matches a range of addresses from this trigger address to the trigger address + [configured range in Indirect Trigger Address Range Register], then the AHB request will be completed by fetching data from the Indirect Controllers SRAM.	32'h00000000

2.3.9. DMA Peripheral Configuration Register

Description: This register allows the user to define the parameters of DMA peripheral controller. This register should be setup while the controller is idle.

Table 2.14. DMA Peripheral Configuration Register

Offset	Bit	R/W	Description	Reset
0x20	31:12	RO	Unused bits. Read as zero.	20'h00000
0x20	11:8	R/W	Number of bytes in a burst type request on the DMA peripheral request. A programmed value of 0 represents a single byte. This should be setup before starting the indirect read or write operation. The actual number of bytes used is 2**[value in this register] which will simplify implementation.	4'h0
0x20	7:4	RO	Unused bits. Read as zero.	4'h0
0x20	3:0	R/W	Number of bytes in a single type request on the DMA peripheral request. A programmed value of 0 represents a single byte. This should be setup before starting the indirect read or write operation. The actual number of bytes used is 2**[value in this register] which will simplify implementation.	4'h0

2.3.10. Remap Address Register

Description: This register allows the user to define the address offset for DAC accesses. This register should be setup while the controller is idle.

Table 2.15. Remap Address Register

Offset	Bit	R/W	Description	Reset
0x24	31:0	R/W	Remapping of incoming AHB address to a different address used by the FLASH device.	32'h00000000

2.3.11. Mode Bit Configuration Register

Description: This register allows the user to define the mode bits for corresponding Flash Device. This register should be setup while the controller is idle.

Table 2.16. Mode Bit Configuration Register

Offset	Bit	R/W	Description	Reset
0x28	31:8	RO	Unused bits. Read as zero.	24'h000000
0x28	7:0	R/W	Mode bits These are the 8 bits that are sent to the device following the address bytes.	8'h00

2.3.12. SRAM Fill Level Register

Description: This register keeps the values of current fill levels of both SRAM partitions.

Table 2.17. SRAM Fill Level Register

Offset	Bit	R/W	Description	Reset
0x2C	31:16	RO	SRAM Fill Level (Indirect Write Partition) in units of SRAM Words (4 bytes).	16'h0000
0x2C	15:0	RO	SRAM Fill Level (Indirect Read Partition) in units of SRAM Words (4 bytes).	16'h0000

2.3.13. TX Threshold Register

Description: This register allows the user to define the TX FIFO level arousing the corresponding interrupt. This register should be setup while the controller is idle.

Table 2.18. TX Threshold Register

Offset	Bit	R/W	Description	Reset
0x30	31:5	RO	Unused bits. Read as zero.	27'h0000000
0x30	4:0	R/W	Defines the level at which the small TX FIFO not full interrupt is generated. This is only relevant when accessing the FLASH device in legacy mode. It can be otherwise ignored.	5'h01

2.3.14. RX Threshold Register

Description: This register allows the user to define the RX FIFO level arousing the corresponding interrupt. This register should be setup while the controller is idle.

Table 2.19. RX Threshold Register

Offset	Bit	R/W	Description	Reset
0x34	31:5	RO	Unused bits. Read as zero.	27'h0000000
0x34	4:0	R/W	Defines the level at which the small RX FIFO not empty interrupt is generated. This is only relevant when accessing the FLASH device in legacy mode. It can be otherwise ignored.	5'h01

2.3.15. Write Completion Control Register

Description: This register defines how the controller will poll the device following a write transfer. By default it is set to poll bit 0 of the device STATUS register (using opcode 0x05), which is common across all devices to indicate WRITE IN PROGRESS (or WIP). However, in some devices, notably the Micron devices that are >= 512MB in size, it is required that the controller polls a different bit of a different device register. This means the controller must issue a different command during this polling phase. For example the Micron N25Q devices that are >= 512MB in size, it is bit 7 of the FLAG STATUS register (opcode 0x70) instead of bit 0 of the normal STATUS register that must be polled. For this reason, the following register has been provided for software to determine the bit and the opcode to use to poll for write completion, and also the number of successive valid polls that should take place. In order not to killing the device by throttling it by continuously generated READ Status SPI transactions, it is possible to prolong the delay between them (bits[31:24]). Defining this delay is not always desired since the ready bit indication can come later what impacts the overall performance. This register should be setup while the controller is idle.

Table 2.20. Write Completion Control Register

Offset	Bit	R/W	Description	Reset
0x38	31:24	R/W	Polling repetition delay Defines additional delay for maintain Chip Select de-asserted during auto-polling phase.	8'h00
0x38	23:16	R/W	Polling count Defines the number of times the controller should expect to see a true result from the polling in successive reads of the device register. Adding additional cycles might be necessarily when the controller is working on high throughput.	8'h01
0x38	15	R/W	Enable polling expiration Set to "1" for enabling auto-polling expiration after number of auto-polling cycles defined in the Polling Expiration Register	1'h0
0x38	14	R/W	Disable polling This switches off the automatic polling function.	1'h0
0x38	13	R/W	Polling polarity Defines the polling polarity. 0 = The write transfer to the device will be complete if the polled bit is equal to "0" 1 = The write transfer to the device will be complete if the polled bit is equal to "1"	1'h0
0x38	12:11	RO	Unused bits. Read as zero.	2'h0

Offset	Bit	R/W	Description	Reset
0x38	10:8	R/W	Polling bit index Defines the bit index that should be polled. 3'b000 = bit 0 of the returned data will be polled for. 3'b001 = bit 1 of the returned data will be polled for. 3'b010 = bit 2 of the returned data will be polled for. (...) 3'b101 = bit 5 of the returned data will be polled for. 3'b110 = bit 6 of the returned data will be polled for. 3'b111 = bit 7 of the returned data will be polled for.	3'h0
0x38	7:0	R/W	Polling opcode Defines the opcode that should be issued by the controller when it is automatically polling for device program completion. This command is issued followed all device write operations. By default, this will poll the standard device STATUS register using opcode 0x05.	8'h05

2.3.16. Polling Expiration Register

Description: This register defines maximum number of poll cycles. If the expected value of the bit being polled is not gotten after number defined in this register, the auto-polling is done on the next phase. The value stored in this register matters only if Write Completion Control Register bits[15] is set to "1". If auto-polling is not disabled by Write Completion Control Register bits[14], at least two auto-polling phases are queued before execution. This register should be setup while the controller is idle.

Table 2.21. Polling Expiration Register

Offset	Bit	R/W	Description	Reset
0x3C	31:0	R/W	Defines the numbers of poll cycles after which auto-polling phase terminates and polling expiration interrupt is generated (if unmasked).	32'hffffffff

2.3.17. Interrupt Status Register

Description: The status fields in this register are set when the described event occurs and the interrupt is enabled in the mask register. When any of these bit fields are set, the interrupt output is asserted high. The fields are cleared using write-one-to-clear approach. Note that bit fields 7 through 11 are only valid when legacy SPI mode is active.

Table 2.22. Interrupt Status Register

Offset	Bit	R/W	Description	Reset
0x40	31:15	R/W	Unused bits. Read as zero.	17'h00000
0x40	14	RWC	The STIG request completion interrupt. The controller is ready for getting another STIG request.	1'h0

Offset	Bit	R/W	Description	Reset
0x40	13	RWC	The maximum number of programmed polls cycles is expired.	1'h0
0x40	12	RWC	Indirect Read partition of SRAM is full and unable to immediately complete indirect operation.	1'h0
0x40	11	RWC	Small RX FIFO full (current FIFO status). Can be ignored in non-SPI legacy mode. 0 = FIFO is not full 1 = FIFO is full	1'h0
0x40	10	RWC	Small RX FIFO not empty (current FIFO status) Can be ignored in non-SPI legacy mode. 0 = FIFO has less than RX THRESHOLD entries 1 = FIFO has at least RX THRESHOLD entries	1'h0
0x40	9	RWC	Small TX FIFO full (current FIFO status). Can be ignored in non-SPI legacy mode. 0 = FIFO is not full 1 = FIFO is full	1'h0
0x40	8	RWC	Small TX FIFO not full (current FIFO status). Can be ignored in non-SPI legacy mode. 0 = FIFO has more than TX THRESHOLD entries 1 = FIFO has at most TX THRESHOLD entries	1'h0
0x40	7	RWC	Receive Overflow This should only occur in Legacy SPI mode. Set if an attempt is made to push the RX FIFO when it is full. This bit is reset only by a system reset and cleared only when this register is read. If a new push to the RX FIFO occurs coincident with a register read this flag will remain set. 0 = no overflow has been detected. 1 = an overflow has occurred.	1'h0
0x40	6	RWC	Indirect Transfer Watermark Level Breached To use this interrupt effectively, it is acceptable to mask/unmask it during the SPI transfer in order not to interrupt it what would impact the performance. It breaks the rule that Interrupt Mask Register (6th bit) should be modified when the controller is in the IDLE state. It may be useful to block the interrupt after it occurs and then unblock after AHB activity is completed. If watermark level is set to value which ensures interrupt generation with at least a few AHB and APB clock cycles between, modifying the Interrupt Mask Register (6th bit) is safe, because the information is synchronized and propagated on time.	1'h0

Offset	Bit	R/W	Description	Reset
0x40	5	RWC	Illegal AHB Access Detected. AHB write wrapping bursts and the use of SPLIT/RETRY accesses will cause this interrupt to trigger. This interrupt is also triggered, if AHB access is performed when the DAC controller is disabled.	1'h0
0x40	4	RWC	Write to protected area was attempted and rejected.	1'h0
0x40	3	RWC	Indirect operation was requested but could not be accepted. Two indirect operations already in storage.	1'h0
0x40	2	RWC	Controller has completed last triggered indirect operation.	1'h0
0x40	1	RWC	Underflow Detected 0 = no underflow has been detected 1 = underflow is detected and an attempt to transfer data is made when the small TX FIFO is empty. This may occur when AHB write data is being supplied too slowly to keep up with the requested write operation	1'h0
0x40	0	RWC	Mode fail Legacy interrupt which is not being used any more.	1'h0

2.3.18. Interrupt Mask Register

Description: This register allows the user to mask/unmask particular interrupt sources. This register should be setup while the controller is idle.

Table 2.23. Interrupt Status Register

Offset	Bit	R/W	Description	Reset
0x44	31:15	R/W	Unused bits. Read as zero.	17'h00000
0x44	14:0	RWC	0 = the interrupt for the corresponding interrupt status register bit is disabled. 1 = the interrupt for the corresponding interrupt status register bit is enabled.	15'h0000

2.3.19. Lower Write Protection Register

Description: This register allows the user to define lower boundary of the write protection area. This register should be setup while the controller is idle.

Table 2.24. Lower Write Protection Register

Offset	Bit	R/W	Description	Reset
0x50	31:0	R/W	The block number that defines the lower block in the range of blocks that is to be locked from writing. The definition of a block in terms of number of bytes is programmable via the Device Size Configuration Register.	32'h00000000

Offset	Bit	R/W	Description	Reset
			Modify this bit only when write protection feature (via bit 1 of the write protection register) is low.	

2.3.20. Upper Write Protection Register

Description: This register allows the user to define upper boundary of the write protection area. This register should be setup while the controller is idle.

Table 2.25. Upper Write Protection Register

Offset	Bit	R/W	Description	Reset
0x54	31:0	R/W	The block number that defines the upper block in the range of blocks that is to be locked from writing. The definition of a block in terms of number of bytes is programmable via the Device Size Configuration Register. Modify this bit only when write protection feature (via bit 1 of the write protection register) is low.	32'h00000000

2.3.21. Write Protection Register

Description: This register allows the user to define the configuration of write protection settings. This register should be setup while the controller is idle.

Table 2.26. Write Protection Register

Offset	Bit	R/W	Description	Reset
0x58	31:2	RO	Unused bits. Read as zero.	30'h00000000
0x58	1	R/W	Write Protection Enable Bit When set to "1", any AHB write access with an address within the protection region defined in the lower and upper write protection registers is rejected. An AHB error response is generated and an interrupt source triggered if unmasked. When set to "0", the protection region is disabled. This bit is internally synchronized in hardware	1'h0
0x58	0	R/W	Write Protection Inversion Bit When set to "1", the protection region defined in the lower and upper write protection registers is inverted meaning it is the region that the system is permitted to write to. When set to "0", the protection region defined in the lower and upper write protection registers is the region that the system is not permitted to write to. Modify this bit only when write protection feature (via bit 1 of the write protection register) is low.	1'h0

2.3.22. Indirect Read Transfer Control Register

Description: This register allows the user to control of the Indirect Read Transfer logic.

Table 2.27. Indirect Read Transfer Control Register

Offset	Bit	R/W	Description	Reset
0x60	31:8	RO	Unused bits. Read as zero.	24'h000000
0x60	7:6	RO	Number of indirect operations completed. This is used in conjunction with bit 5. Incremented by H/W when an indirect operation has completed, decremented when bit 5 is written with a "1"	2'h0
0x60	5	R/W	Indirect Completion Status (status) Set by H/W when an indirect operation has completed, cleared by writing a "1"	1'h0
0x60	4	RO	Two indirect read operations have been queued (status) This bit is set and cleared by H/W.	1'h0
0x60	3	R/W	SRAM full and unable to immediately complete indirect operation (status) Set by H/W, cleared by writing a "1"	1'h0
0x60	2	RO	Indirect read operation in progress (status) This bit can only be cleared by H/W.	1'h0
0x60	1	WO	Cancel indirect read (control) Writing a "1" to this bit will cancel an ongoing Indirect READ operation. This will cancel all indirect operations currently in progress. This bit is internally synchronized in hardware.	1'h0
0x60	0	WO	Start indirect read (control) Writing a "1" to this bit will trigger an indirect READ operation. The assumption is that the indirect start address and Indirect number of bytes register is setup before triggering the indirect READ operation. This bit is internally synchronized in hardware.	1'h0

2.3.23. Indirect Read Transfer Watermark Register

Description: This register allows the user to define watermark level for Indirect read transfers. This register should be setup before an indirect read transfer is triggered.

Table 2.28. Indirect Read Transfer Watermark Register

Offset	Bit	R/W	Description	Reset
0x64	31:0	R/W	Watermark value	32'h00000000

Offset	Bit	R/W	Description	Reset
			This represents the minimum fill level of the SRAM before a DMA peripheral access is permitted. When the SRAM fill level passes the watermark, an interrupt source is also generated if unmasked. This can be disabled by writing a value of all zeroes. In units of Bytes.	

2.3.24. Indirect Read Transfer Start Address Register

Description: This register allows the user to define start address of indirect read transfer which is about to be triggered. This register should be setup before an indirect read transfer is triggered.

Table 2.29. Indirect Read Transfer Start Address Register

Offset	Bit	R/W	Description	Reset
0x68	31:0	R/W	Start of Indirect Access This is the FLASH start address from which the indirect access will commence its READ operation.	32'h00000000

2.3.25. Indirect Read Transfer Number Bytes Register

Description: This register allows the user to define number of bytes to be read of indirect read transfer which is about to be triggered. This register should be setup before an indirect read transfer is triggered.

Table 2.30. Indirect Read Transfer Number Bytes Register

Offset	Bit	R/W	Description	Reset
0x6C	31:0	R/W	Indirect Number of Bytes This is the number of bytes that the indirect access will consume. This can be bigger than the configured size of SRAM.	32'h00000000

2.3.26. Indirect Write Transfer Control Register

Description: This register allows the user to control of the Indirect Write Transfer logic.

Table 2.31. Indirect Write Transfer Control Register

Offset	Bit	R/W	Description	Reset
0x70	31:8	RO	Unused bits. Read as zero.	24'h000000
0x70	7:6	RO	Number of indirect operations completed. This is used in conjunction with bit 5. Incremented by H/W when an indirect operation has completed, decremented when bit 5 is written with a "1"	2'h0
0x70	5	R/W	Indirect Completion Status (status) Set by H/W when an indirect operation has completed, cleared by writing a "1"	1'h0
0x70	4	RO	Two indirect read operations have been queued (status)	1'h0

Offset	Bit	R/W	Description	Reset
			This bit is set and cleared by H/W.	
0x70	3	RO	Unused bit. Read as zero.	1'h0
0x70	2	RO	Indirect write operation in progress (status) This bit can only be cleared by H/W.	1'h0
0x70	1	WO	Cancel indirect write (control) Writing a "1" to this bit will cancel an ongoing Indirect WRITE operation. This will cancel all indirect operations currently in progress. This bit is internally synchronized in hardware.	1'h0
0x70	0	WO	Start indirect write (control) Writing a "1" to this bit will trigger an indirect WRITE operation. The assumption is that the indirect start address and Indirect number of bytes register is setup before triggering the indirect WRITE operation. This bit is internally synchronized in hardware.	1'h0

2.3.27. Indirect Write Transfer Watermark Register

Description: This register allows the user to define watermark level for Indirect write transfers. This register should be setup before an indirect write transfer is triggered.

Table 2.32. Indirect Write Transfer Watermark Register

Offset	Bit	R/W	Description	Reset
0x74	31:0	R/W	Watermark value This represents the maximum fill level of the SRAM before a DMA peripheral access is permitted. When the SRAM fill level falls below the watermark, an interrupt source is also generated if unmasked. This can be disabled by writing a value of all zeroes. In units of Bytes.	32'h00000000

2.3.28. Indirect Write Transfer Start Address Register

Description: This register allows the user to define start address of indirect write transfer which is about to be triggered. This register should be setup before an indirect write transfer is triggered.

Table 2.33. Indirect Write Transfer Start Address Register

Offset	Bit	R/W	Description	Reset
0x78	31:0	R/W	Start of Indirect Access This is the FLASH start address from which the indirect access will commence its READ operation.	32'h00000000

2.3.29. Indirect Write Transfer Number Bytes Register

Description: This register allows the user to define number of bytes to be written of indirect read transfer which is about to be triggered. This register should be setup before an indirect read transfer is triggered.

Table 2.34. Indirect Write Transfer Number Bytes Register

Offset	Bit	R/W	Description	Reset
0x7C	31:0	R/W	Indirect Number of Bytes This is the number of bytes that the indirect access will consume. This can be bigger than the configured size of SRAM.	32'h00000000

2.3.30. Indirect Trigger Address Range Register

Description: This register allows the user to define the indirect trigger address range. If the configured range exceeds number of bytes programmed for particular indirect transfer, there is no need to detect indirect trigger address boundaries by software. This register should be setup before an indirect read transfer is triggered.

Table 2.35. Indirect Trigger Address Range Register

Offset	Bit	R/W	Description	Reset
0x80	31:4	RO	Unused bits. Read as zero.	28'h0000000
0x80	3:0	R/W	Indirect Range Width This is the address offset of the Indirect AHB Address Trigger Register (0x1C). When any valid Indirect Access is triggered and AHB address fits to the range, the request is forwarded into Indirect Write Controller or Indirect Read Controller depending on which one was requested by valid trigger. The value is given as power of 2. The default one is 2**4 = 16 what allows 16-byte bursts to be performed. This field reflects width of the range so number of locations of valid addresses (single location has 32 bits) ranges from Indirect Trigger Address to Indirect Trigger Address + [Indirect Range With - 1] (Indirect Trigger Address -> Indirect Trigger Address + 15 [by default]).	4'h4

2.3.31. Flash Command Control Memory Register (Using STIG)

Description: This register controls the Memory Bank accesses. It also defines the number of bytes intended to get by STIG access configured to use the STIG Memory Bank.

Table 2.36. Flash Command Control Memory Register (Using STIG)

Offset	Bit	R/W	Description	Reset
0x8C	31:29	RO	Unused bits. Read as zero.	3'h0
0x8C	28:20	R/W	Memory Bank Address The address of the Memory Bank which data will be read from. It is equivalent to the index value of the byte read by the last	9'h000

Offset	Bit	R/W	Description	Reset
			STIG access configured to work with STIG Memory Bank. For example, setting this field for 255 (8'hff) will return (after triggering Memory Bank access) 255th byte read by the last STIG access configured to work with STIG Memory Bank. This should be setup before triggering the command via bit 0 of this register.	
0x8C	19	RO	Unused bit. Read as zero.	1'h0
0x8C	18:16	R/W	Number of STIG Memory Bank Read Bytes It defines the number of read bytes for the STIG configured to work with STIG Memory Bank as follows: 3'b000 = 16 bytes 3'b001 = 32 bytes 3'b010 = 64 bytes 3'b011 = 128 bytes 3'b100 = 256 bytes 3'b101 = 512 bytes 3'b11x = unused This should be setup before triggering the command via bit 0 of Flash Command Control Register (Using STIG)	3'h0
0x8C	15:8	RO	Memory Bank Read Data Last requested data from the STIG Memory Bank. It turns to be stable when bit 1 of this register toggles from "1" to "0".	8'h00
0x8C	7:2	RO	Unused bit. Read as zero.	6'h00
0x8C	1	RO	Memory Bank data request in progress	1'h0
0x8C	0	WO	Trigger the Memory Bank data request This bit is internally synchronized.	n/a

2.3.32. Flash Command Control Register (Using STIG)

Description: This register controls SPI transactions generated by STIG. It allows the user to define corresponding SPI frame to particular command, triggering the transfer and polling for its completion.

Table 2.37. Flash Command Control Register (Using STIG)

Offset	Bit	R/W	Description	Reset
0x90	31:24	R/W	Command Opcode The command opcode field should be setup before triggering the command. For example, 0x20 maps to SubSector Erase. Writing to the execute field (bit 0) of this register launches the command.	8'h00

Offset	Bit	R/W	Description	Reset
			Note Using this approach to issue commands to the device will make use of the instruction type of the device instruction configuration register. If this field is set to 2'b00, then the command opcode, command address, command dummy cycles and command data will all be transferred in a serial fashion. If this field is set to 2'b01, then the command opcode, command address, command dummy cycles and command data will all be transferred in parallel using DQ0 and DQ1 pins. If this field is set to 2'b10, then the command opcode, command address, command dummy cycles and command data will all be transferred in parallel using DQ0, DQ1, DQ2 and DQ3 pins.	
0x90	23	R/W	Read Data Enable Set to "1" if the command specified in bits 31:24 requires read data bytes to be received from the device.	1'h0
0x90	22:20	R/W	Number of Read Data Bytes Up to 8 Data bytes may be read using this command (Set to 0 for 1 byte, 7 for 8 bytes). In case of Flash Command Control Register (Using STIG) Register bit[2] is enabled, this field is not to care and Number of Read Data Bytes is as specified in Flash Command Control Memory Register (Using STIG) Register bit[15:8].	3'h0
0x90	19	R/W	Command Address Enable Set to "1" if the command specified in bits 31:24 requires an address. This should be setup before triggering the command via bit 0 of this register.	1'h0
0x90	18	R/W	Mode Bit Enable Set to "1" to ensure the mode bits as defined in the Mode Bit Configuration Register are sent following the address bytes.	1'h0
0x90	17:16	R/W	Number of Address Bytes Set to the number of address bytes required (the address itself is programmed in the Flash Command Address Register). This should be setup before triggering the command via bit 0 of this register. 2'b00 = 1 address byte 2'b01 = 2 address bytes 2'b10 = 3 address bytes	2'h0

Offset	Bit	R/W	Description	Reset
			2'b11 = 4 address bytes	
0x90	15	R/W	Write Data Enable Set to "1" if the command specified in bits 31:24 requires write data bytes to be sent to the device.	1'h0
0x90	14:12	R/W	Number of Write Data Bytes Up to 8 Data bytes may be written using this command (Set to 0 for 1 byte, 7 for 8 bytes).	3'h0
0x90	11:7	R/W	Number of Dummy Cycles Set to the number of dummy cycles required for command specified in bits 31:24. This should be setup before triggering the command via bit 0 of this register.	5'h00
0x90	6:3	RO	Unused bits. Read as zero.	4'h0
0x90	2	R/W	STIG Memory Bank enable bit. This should be setup before triggering the command via bit 0 of this register.	1'h0
0x90	1	RO	STIG command execution in progress.	1'h0
0x90	0	WO	Execute the command This bit is internally synchronized.	n/a

2.3.33. Flash Command Address Register

Description: This register allows the user to define the address of the command using by the STIG controller. This register should be setup before an indirect read transfer is triggered.

Table 2.38. Flash Command Address Register

Offset	Bit	R/W	Description	Reset
0x94	31:0	R/W	Command Address It is the address used by the command specified in Flash Command Control Register (Using STIG) Register bits[31:24].	32'h00000000

2.3.34. Flash Command Read Data Register (Lower)

Description: This register keeps the last 4 bytes read by STIG SPI access.

Table 2.39. Flash Command Read Data Register (Lower)

Offset	Bit	R/W	Description	Reset
0xA0	31:0	RO	Command Read Data (Lower) This is the data that is returned by the FLASH device for any status or configuration read operation carried out by triggering	32'h00000000

Offset	Bit	R/W	Description	Reset
			the event in the control register. The register will be valid when the Flash Command Control Register (Using STIG) Register bit[1] is low.	

2.3.35. Flash Command Read Data Register (Upper)

Description: This register keeps the last but 4 bytes read by STIG SPI access. This register in conjunction with the Flash Command Read Data Register (Lower) enables the controller to keep 8 last bytes read from the Flash Device using STIG.

Table 2.40. Flash Command Read Data Register (Upper)

Offset	Bit	R/W	Description	Reset
0xA4	31:0	RO	Command Read Data (Upper) This is the data that is returned by the FLASH device for any status or configuration read operation carried out by triggering the event in the control register. The register will be valid when the Flash Command Control Register (Using STIG) Register bit[1] is low.	32'h00000000

2.3.36. Flash Command Write Data Register (Lower)

Description: This register takes the first 4 bytes to be written by STIG.

Table 2.41. Flash Command Write Data Register (Lower)

Offset	Bit	R/W	Description	Reset
0xA8	31:0	R/W	Command Write Data (Lower) This is the data that is to be written to the FLASH device for any status or configuration write operation carried out by triggering the event in the control register.	32'h00000000

2.3.37. Flash Command Write Data Register (Upper)

Description: This register takes the bytes ranging from 5 to 8 to be written by STIG.

Table 2.42. Flash Command Write Data Register (Upper)

Offset	Bit	R/W	Description	Reset
0xAC	31:0	R/W	Command Write Data (Upper) This is the data that is to be written to the FLASH device for any status or configuration write operation carried out by triggering the event in the control register.	32'h00000000

2.3.38. Polling Flash Status Register

Description: This register provides auto-polling data. It acts as the extension for the Write Completion Control Register where full status is not available and any action can be taken only relying on the indication of single bit being polled for.

Table 2.43. Polling Flash Status Register

Offset	Bit	R/W	Description	Reset
0xB0	31:20	RO	Unused bits. Read as zero.	12'h000
0xB0	19:16	R/W	Number of dummy cycles for auto-polling This field enables the user to define additional dummy cycles for read status during auto-polling state. Some cycles may be necessary to add if external delay of read data path shifts it outside the defined clock cycle. An option of adding dummy cycles may not force the target clock rate being degraded.	4'h0
0xB0	15:9	RO	Unused bits. Read as zero.	7'h00
0xB0	8	RO	Polling Status Valid This bit is set when value in bits from 7 to 0 is valid.	1'h0
0xB0	7:0	RO	Flash Status Defines the Status of Device returned by the last auto-polling access.	8'h00

2.3.39. Module ID Register

Description: This register provides the IP release number and the configuration data.

Table 2.44. Module ID Register

Offset	Bit	R/W	Description	Reset
0xFC	31:24	RO	Fix/patch number related to the Module/Revision ID	8'h01
0xFC	23:8	RO	Module/Revision ID	16'h0001
0xFC	7:2	RO	Unused bits. Read as zero.	6'h00
0xFC	1:0	RO	Configuration ID number (QSPI + PHY)	2'h2

2.4. Performance Characteristics

For the direct read operations, currently Multiple-SPI Flash devices support a maximum clock speed of 133 MHz (65 Mbytes/sec). Burst access to Multiple-SPI Flash devices incurs approximately 10% overhead.

The basic performance test will be the time to read a 60KB region of FLASH memory. The specific location of this region is not considered, and will remain constant through all test runs.

When using direct access controller and issuing constant bursts of 16 to the controller, the following numbers were measured. No IDLE or BUSY accesses were issued to the controller during the 60KB fetch – it should be noted that in a real system the numbers of these IDLE or BUSY accesses applied to the controller during a large fetch will only affect the overall time for retrieving the data when the AHB data rate is reduced so significantly that it causes the controller to back-pressure – i.e. the average AHB data rate becomes close or less than the FLASH data rate programmed.

Table 2.45. Performance data (Direct Access)

Flash Clock Frequency (MHz)	Flash Instruction used	Controller Used	MAX theoretical FLASH Data-rate (not inclusive of overhead) Mbps	Best case number of cycles to fetch and forward to AHB (Theoretical no overhead)	Best case number cycles to fetch and forward to AHB (Actual)
12.5	SLOW READ	DIRECT	12.5	3932165	3932423
50	SLOW READ	DIRECT	50	983040	983112
133	FAST READ	DIRECT	133	369564	369662
12.5	QIO READ	DIRECT	50	983040	983208
50	QIO READ	DIRECT	200	245760	245808
100	QIO READ	DIRECT	400	122880	122908
133	QIO READ	DIRECT	520	92391	92428
66	QIO DDR READ	DIRECT	520	92391	92438

The actual numbers are very close to the theoretical best. This is because the controller will automatically detect that each AHB read request issued is sequential to the last. This is despite the fact the requests themselves are issued in accordance with AHB rules and restrictions which will give indications of non-sequential accesses even when the access is sequential to the last (such as when an access crosses a 1K boundary or when a fixed burst such as INCR16 is completed).

The numbers reported in the table assume the AHB requests were all made in WORD (32-bit) transfers. If the master issuing the requests sent the full bulk transfer using different sized requests, the controller would have separated the full transfer into multiple smaller bursts, a burst split occurring whenever the AHB request size changed. For each split burst, the controller would require to send overhead bytes in the form of opcode+address+dummy phases.

The AHB clock frequency was carefully selected to ensure continuous SPI data transfer. The tests shown that all measured transfers up to 100 MHz of Flash Clock frequency managed to work without burst spliting for AHB clock equals also 100 MHz. All higher data rate performance tests were executed with AHB clock equals 400MHz. The performance results are formalized into AHB cycles what ensures concise form of the controller latency.

The equivalent test carried out with XIP mode enabled yields the following results:

Table 2.46. Performance data (XIP Mode)

Flash Clock Frequency (MHz)	Flash Instruction used	Controller Used	MAX theoretical FLASH Data-rate (not inclusive of overhead) Mbps	Best case number of cycles to fetch and forward to AHB (Theoretical no overhead)	Best case number cycles to fetch and forward to AHB (Actual)
12.5	SLOW READ	DIRECT	12.5	3932165	-
50	SLOW READ	DIRECT	50	983040	-
12.5	QIO READ	DIRECT	50	983040	983145
50	QIO READ	DIRECT	200	245760	245793
100	QIO READ	DIRECT	400	122880	122901
133	QIO READ	DIRECT	520	92391	92422
66	QIO DDR READ	DIRECT	520	92391	92437

The results are almost the same as the non-XIP data. This is because the instruction opcode itself is only sent once through the entire bulk transfer. For the XIP test, this means a reduction in time of only 8 cycles.

Note that Basic READ instruction cannot operate in XIP mode and so is omitted.

When using "Indirect" mode, the results are also very close to the theoretical best. This is because in indirect mode the accesses sent to the FLASH device are handled internally by the controller and are always issued sequential throughout the block fetch, meaning there is no requirement from the AHB interface to maintain maximum performance at this stage. Even less much less overhead and the data is fetched from FLASH into SRAM quicker. There is extra overhead required to move the data from the local SRAM to the external AHB master requesting the data.

The time to read the data from FLASH memory using an indirect access and push this to SRAM is as follows:

Table 2.47. Performance data (Indirect Access)

Flash Clock Frequency (MHz)	Flash Instruction used	Controller Used	MAX theoretical FLASH Data-rate (not inclusive of overhead) Mbps	Best case number of cycles to fetch and forward to AHB (Theoretical no overhead)	Best case number cycles to fetch and forward to AHB (Actual)
12.5	SLOW READ	INDIRECT	12.5	3932165	3932425
50	SLOW READ	INDIRECT	50	983040	983113
12.5	QIO READ	INDIRECT	50	983040	983203
50	QIO READ	INDIRECT	200	245760	245803
100	QIO READ	INDIRECT	400	122880	122903
133	QIO READ	INDIRECT	520	92391	92483
66	QIO DDR READ	INDIRECT	520	92391	92483

The time taken to fetch this data from SRAM and forward to AHB is system dependent and not included in this table. However, the QSPI controller will only insert wait states to the AHB when fetching this data from SRAM when the SRAM itself is busy being accessed. This is dependent on how many other ports are trying to access the SRAM. If the indirect write operation is idle, at worst, this is every 8 AHB clocks. Obviously this is very system dependent on the number of IDLE/BUSY accesses sent to the controller.

The AHB clock frequency was carefully selected to ensure continuous SPI data transfer. The tests shown that all measured transfers up to 100 MHz of Flash Clock frequency managed to work without burst spliting for AHB clock equals also 100 MHz. All higher data rate performance tests were executed with AHB clock equals 400MHz. The performance results are formalized into AHB cycles what ensures concise form of the controller latency. The additional latency strongly depends on how much data left to fetch from SRAM after Indirect Transfer is completed.

Achieving performance very close to theoretical maximum defined by Multiple-SPI Flash Memory Vendors is only possible along with the integrated PHY Module. The performance tests were executed for PHY Pipeline Mode which is intended to high data rate transfers with software overhead as little as possible. There are just some regulations which must be met as specified in PHY Pipeline Mode section. The AHB clock value for PHY performance tests was set to 400MHz.

Table 2.48. Performance data (PHY Pipeline access)

Flash Clock Frequency (MHz)	Flash Instruction used	Controller Used	MAX theoretical FLASH Data-rate (not inclusive of overhead) Mbps	Best case number of cycles to fetch and forward to AHB (Theoretical no overhead)	Best case number cycles to fetch and forward to AHB (Actual)
133	QIO READ	DIRECT	520	92391	92423
80	QIO DDR READ	DIRECT	640	98304	98327

The latency is caused by the necessity of dropping pipeline accesses. Number of accesses to queue is calculated based on the IO configuration what ensures the latency to be as little as possible for all Multiple-SPI variants.

Note

Performance results presented in this section were achieved in RTL simulation with Memory Models provided by 3rd Party Vendors.

3. Integrating the QSPI Flash Controller

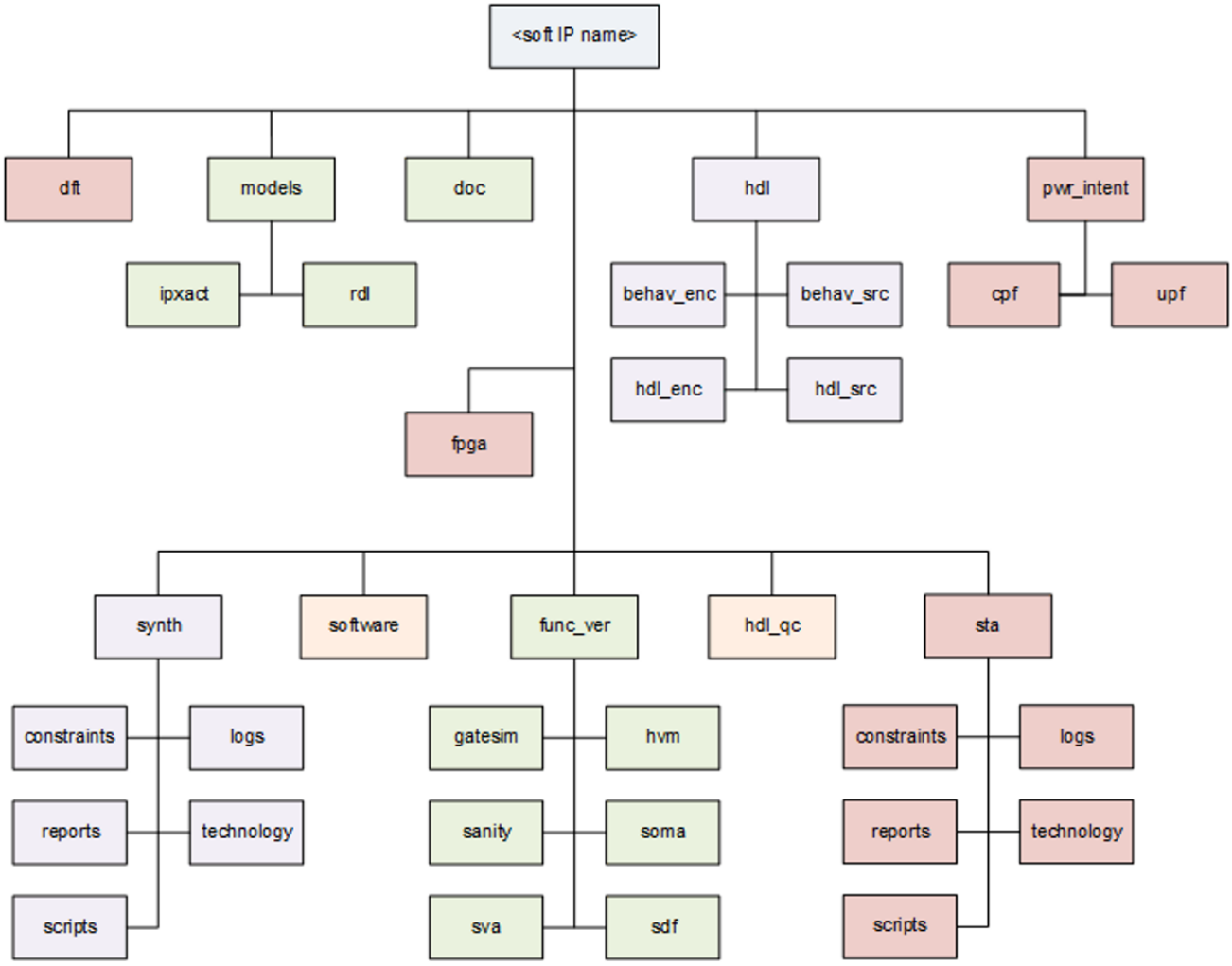
The following topics are discussed in this chapter:

3.1. IP Content

3.1.1. Delivery Structure

The delivery directory structure for all IPG soft IP blocks is depicted below in Figure 3.1, “Soft IP Delivery Structure”. Colours in the diagram are intended only to make it easier to identify primary directories and the subdirectories that accompany them – no further relationship between content of similarly-shaded directories should be inferred. All soft IP deliveries originating from Cadence design IP business units will follow this structure, even if certain directories are not populated for a given IP block.

Figure 3.1. Soft IP Delivery Structure



Customer delivery subdirectory contents are based on a set of directory definitions intended to accommodate each possible type of design view or data set that could generally apply to Cadence IP. Hard and soft IP deliveries each use a unique subset of these directories, with these subsets being common across all IPG business units. This means that hard and soft IP deliveries from all groups will have a consistent look and feel, however, not all directories are applicable for individual designs hence they are left intentionally empty.

The following table describes the meaning of the delivery directory structure for the QSPI Flash Controller and indicates the empty directories.

Table 3.1. Design Delivery Definitions

Directory/File	Description
dft	This directory is left empty.
doc	Directory containing all relevant IP documentation. The Following documents are delivered: • The User Guide

Directory/File	Description
	• The Virtual Cadence Flash Memory Specification • The BUG Release info
fpga	This directory is left empty.
func_ver	Encapsulating directory containing a variety of different collateral sets related to functional verification of a given IP block. This directory contains the following subdirectories: gatesim Left empty hvm Left empty sanity Demonstration sanity-level test environment described in Verification and Test Application sdf Left empty soma Left empty sva System Verilog assertions for sanity-level test environment
hdl	Encapsulating directory for design source files. This directory contains the following subdirectories: behav_enc Left empty behav_enc Left empty hdl_enc Left empty hdl_src Unencrypted synthesizable design source files
hdl_qc	This directory is left empty.
models	Encapsulating directory containing various models/views of a given IP block. This directory contains the following subdirectories, each representing a different design model: ipxact Register abstraction models in IP-XACT format rdl Left empty
pwr_intent	This directory contains the following subdirectories: cpf Left empty upf Left empty
software	This directory is left empty.
sta	This directory contains the following subdirectories: constraints Left empty logs Left empty

Directory/File	Description	
	reports	Left empty
	technology	Left empty
	scripts	Left empty
synth	This directory contains the following subdirectories:	
	constraints	Constraint files
	logs	Left empty
	reports	Left empty
	technology	Left empty
	scripts	Synthesis and simulation scripts
work	This directory does not exist within delivery. The user is responsible for create it.	

3.1.2. Reference Environment

The following tools were used for the most quotable branches of the design flow:

RTL Simulator: Cadence Incisive Simulator 14.10.002

Code Coverage: Cadence Incisive Simulator 14.10.002

Logic and Test Synthesis: Cadence RTL Compiler 14.10.000

Perl: v5.6.1 built for i686-linux

3.2. Implementation Application Notes

3.2.1. Micron N25Q Family of Devices

Micron N25Q devices >= 512Mb in size contain the Flag Status Register, which requires to be polled in order to determine whether the program or erase controller bit is Busy or Ready. The QSPI controller has an automatic method to poll the device for write completion and it does this by default polling bit 0 of the device STATUS register using opcode 0x05. This is the write in progress (or WIP as it is referred in many device datasheets). For these larger Micron devices however, the FLAG STATUS register needs to be polled instead. This can be achieved by carefully programming the Write Completion Control Register.

3.2.2. Micron MT25Q Family of Devices

Micron MT25Q devices can achieve the clock frequency up to 133 MHz (in STR) and 66 MHz (in DTR) what results in throughput up to 65 MB/s. Transfer rate has been increased comparing with N25Q ones which can handle data rate of 54 MB/s.

Adding support for DTR protocol is another significant enhancement. It can be activated by volatile or non-volatile way. It allows device to work in DTR Mode for each possible command (not only DTR READ ones). Additional benefit over this feature is possibility to handle OPCODE phase in DTR Mode what improves overall performance.

3.3. Implementation Guidelines

3.3.1. Clocking

There are four clock sources for the QSPI Controller and one generated clock. Additionally, there are two clocks generated by DLL. All of them are described individually below:

- AHB Clock (hclk)
- APB Clock (pclk)
- Reference Clock (ref_clk)
- Loopback Clock (spi_dlyd_i)
- Generated Clock (sclk_out)
- SPI Clock from DLL (tx_dll_clk)
- Sampling clock from DLL (rx_dll_clk)

3.3.1.1. AHB Clock (hclk)

The AHB clock is the main system clock used to transfer data over the AHB bus between an external master and the QSPI controller. This clock is only used at the front end of the controller and must be fast enough to guarantee the overall bandwidth can be met. I.e. there is a minimum frequency, but no maximum. The maximum allowed frequency is only limited by the speed of the technology used and the required clock relationships as defined in the Section 3.3.2, "*Clock Relationships*".

The QSPI controller only uses the rising edge of this clock and there are therefore no duty cycle requirements. The use of the AHB clock in the design has been implemented in a way that means it can be considered asynchronous to all the other input clocks.

3.3.1.2. APB Clock (pclk)

The APB clock is used to access the register map of the QSPI controller, perform controller and device configuration, service interrupts and control certain run time modes. The allowed frequency is limited by the speed of the technology used and the required clock relationships. It is acceptable, but not a requirement, for this signal to be tied to *hclk*. The QSPI controller only uses the rising edge of this clock and has therefore no duty cycle requirements. The use of the APB clock in the design has been implemented in a way that means it can be considered asynchronous to all the other input clocks.

3.3.1.3. Reference Clock (ref_clk)

The reference clock is used to serialize the data and drive the external SPI interface. It decouples the system clocks (AHB and APB) from the SPI device. The allowed frequency is limited by the speed of the technology used and the required clock relationships. The QSPI controller generally uses the rising edge of this clock and there are therefore no duty cycle requirements. When CDNS_QSPI_BOTH_EDGE_SAMPLING parameter is uncommented, the controller has also falling edge flops which only mediate in sampling data from device and do not have any further impact on control logic of the core. The use of the reference clock in the design has been implemented in a way that means it can be considered asynchronous to all the other input clocks. The selected frequency is linked directly (via the baud rate divider) to the required FLASH clock rate.

The PHY Module interfaces with low-level SPI block on ref_clk. Neither AHB nor APB clock is used in PHY Module. Control logic for PHY is generated by SPI Control logic module included within the low-level SPI block.

3.3.1.4. Loopback Clock (spi_dlyd_i)

This clock is used for prolonging the read data window to ensure more valid samples for tap data capturing mechanism. It comes from Delay Line located externally for the controller.

This clock is optional and the input may be set to high if loopback feature is not intended to using. Generation path for this clock was presented in the Section 1.1.5, "*Reset Diagram*". More detailed description of mechanism using this clock is included in Section 2.1.11.2, "*Mechanism using external DLL*".

3.3.1.5. Generated Clock (sclk_out)

The *sclk_out* pin is used as the clock source of external SPI slave FLASH devices. As far as the QSPI controller is concerned, the generated clock is not a clock, but a *ref_clk* timed output from the registers block. For synthesis, it will be treated as a data output of the core. It is synchronous to the reference clock, divided down via the internal baud rate dividers. The master mode baud rate divisor allows division by any even number up to 32 as follows:

Table 3.2. Baud Rate Juxtaposition

Divisor	Baud Rate
4'b0000	/2
4'b0001	/4
4'b0010	/6
4'b0011	/8
4'b0100	/10
4'b0101	/12
4'b0110	/14
4'b0111	/16
...	...
4'b1111	/32

The baud_rate along with the polarity and phase settings (CPOL and CPHA) for this clock are programmable bits in the QSPI Configuration Register (0x00).

If PHY Mode is enabled, generated clock is not forwarded further to Flash Device but delayed version of gated *ref_clk*. There is the output multiplexer for configuration with PHY which selects corresponding path.

3.3.1.6. SPI Clock from DLL (tx_dll_clk)

While the PHY Mode is enabled, this clock is granted and consequently forwarded into external FLASH Device. Its frequency is equal to *ref_clk* but phase could be adjusted by programmable number of delay elements.

3.3.1.7. Sampling clock from DLL (rx_dll_clk)

While PHY Mode is enabled, this clock is used for capturing data. Its frequency is equal to *ref_clk* but phase could be adjusted by programmable number of delay elements. It allows the user to adjust safe sampling point after software calibration.

The source of this clock is selected by MUX before it passes through DLL. In PHY Mode, by default, gated ref_clk is forwarded into DLL. If Read Data Capture Register bit[0] is set, the loopback clock is used for sampling data.

3.3.2. Clock Relationships

The AHB, APB and reference clock can be considered asynchronous to one another, although there will be some constraints that must be adhered to with respect to minimum frequencies.

The reference clock frequency must be at least 4X the generated memory clock frequency (for SDR commands) and at least 8X (for DDR commands).

If PHY Mode is enabled, *spi_clk* = *ref_clk* in terms of frequency hence this requirement is not applicable.

The *hclk* clock frequency must be chosen such that the bandwidth requirements of the required data rates can be met.

The maximum allowed *hclk* frequency is only limited by the speed of the technology used and the required clock relationships with *ref_clk*. If *hclk* is faster than *ref_clk*, there might be a scenario where two AHB accesses occur within single *ref_clk* cycle. With vector changing faster than destination clock, the read side of the FIFO is exposed on capturing metastable values which may impact on expected FIFO levels. In order not to define an upper limitation over *hclk* clock frequency since it is not needed, the minimum number of AHB wait states to add after getting *hready_out* was defined. These wait states introduce the sufficient gap between consecutive push accesses ensuring FIFO levels being stable while taken by the corresponding control logic.

The following formula defines the number of wait states needed to add:

*number_of_wait_states = (hclk*bytes_in_spi_cycle)/(ahb_data_size*ref_clk*spi_clk_divider)*

Where:

- *hclk* - AHB clock frequency
- *bytes_in_spi_cycle* - Number of bytes expected to get in a single SPI cycle (as configured by software). E.g. if data to write/read via SPI was configured in Quad DDR fashion, there is 1 byte transmitted within single SPI cycle so the value of this parameter should be 1. If data to write/read via SPI was configured in Dual SDR Mode, just a part (1/4) of byte is collected within single SPI cycle so the value of this parameter should be 0.25.
- *ahb_data_size* - AHB data size as defined by *hsize*. For mixed accesses the highest value should be picked up. This parameter does not correspond directly to the *hsize* value but the number of bytes defining by this vector.
- *ref_clk* - reference clock frequency
- *spi_clk_divider* - SPI clock divider as defined in QSPI Configuration Register bit[22]. If PHY Mode is enabled, the value of divider equals to 1.

Examples:

- AHB side: *hsize* = 2'b10 => *ahb_data_size* = 4 bytes, *hclk* = 400 MHz (fast system clock)

  SPI side: *bytes_in_spi_cycle* = 0.25 (Dual SDR Mode), *ref_clk* = 200 MHz, *spi_clk_divider* = 4 (to meet the requirement that the reference clock frequency must be at least 4X the generated memory clock frequency (for SDR commands)).

  *number_of_wait_states = (hclk*bytes_in_spi_cycle)/(ahb_data_size*ref_clk*spi_clk_divider) = (400*0.25)/(4*200*4) = 0.03125 = 0 (round down)*

  Conclusion: There is no need to introduce additional AHB wait states for working conditions defined in this example. The spare range for upper value AHB clock is very wide.

- AHB side: *hsize* = 2'b10 => *ahb_data_size* = 4 bytes, *hclk* = 400 MHz (fast system clock)

  SPI side: *bytes_in_spi_cycle* = 1 (Quad DDR Mode), *ref_clk* = 200 MHz, *spi_clk_divider* = 8 (to meet the requirement that the reference clock frequency must be at least 8X the generated memory clock frequency (for DDR commands)).

  *number_of_wait_states = (hclk*bytes_in_spi_cycle)/(ahb_data_size*ref_clk*spi_clk_divider) = (400*1)/(4*200*8) = 0.06125 = 0 (round down)*

  Conclusion: There is no need to introduce additional AHB wait states for working conditions defined in this example. The spare range for upper value AHB clock is very wide.

- AHB side: *hsize* = 2'b10 => *ahb_data_size* = 4 bytes, *hclk* = 400 MHz (fast system clock)

  SPI side: *bytes_in_spi_cycle* = 1 (Quad DDR Mode), *ref_clk* = 133 MHz (limited by Flash), *spi_clk_divider* = 1 (PHY Mode)

  *number_of_wait_states = (hclk*bytes_in_spi_cycle)/(ahb_data_size*ref_clk*spi_clk_divider) = (400*1)/(4*133*1) = 0.75 = 0 (round down)*

  Conclusion: There is no need to introduce additional AHB wait states for working conditions defined in this example, however, decreasing ref_clk (and consequently SPI clk in PHY Mode) into 100 MHz will make the *number_of_wait_states* = 1.

The minimum allowed *hclk* frequency is only limited by the relationships with *ref_clk* or being precise with SPI clock which is always generated based on *ref_clk*. Minimum *hclk* must be chosen such that the bandwidth requirements of the required data rates can be met.

3.3.3. Timing and Load Parameters

For all internal interfaces (including AHB and APB), all inputs must be setup a minimum of 40% of the clock period before the rising edge of the related clock and all outputs will be valid at a maximum of 40% of the clock period after the rising edge of the related clock. Given that there are combinational input-output paths over SRAM interface, in order to minimize latency, these requirements could be slightly tighten (especially for high *hclk* value). If Static Timing Analysis failed for these paths – we recommend to consider keep *sram_cs* signal continuously enabled and treat de-asserted *sram_we* as *sram_re*. In this case these paths could be specified as false ones.

3.3.4. Designing Across Clock Boundaries

This section defines the rules governing designing across the 3 asynchronous clock boundaries (*ref_clk*, *hclk* and *pclk*).

3.3.4.1. Static Configuration

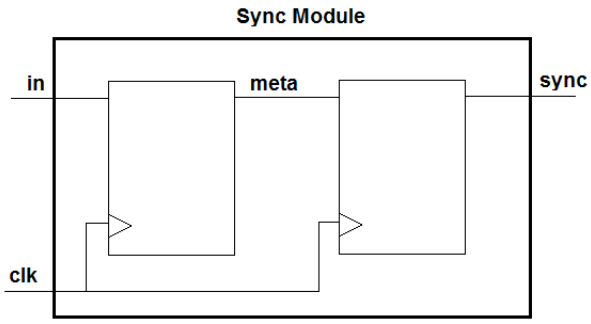
Global static configuration signals driven in the APB clock domain while the QSPI controller is disabled, or configuration signals specific to a certain mode of operation that are driven while that mode is disabled do not need to be synchronized.

3.3.4.2. Metastability

To address the problem of metastability, all critical signals not covered specifically in this document that pass between clock domains are synchronised by passing through at least two ganged flip-flops. Because *hclk*, *pclk* and *ref_clk* can all be asynchronous with respect to each other. Two stages synchronizer is randomized for verification purposes.

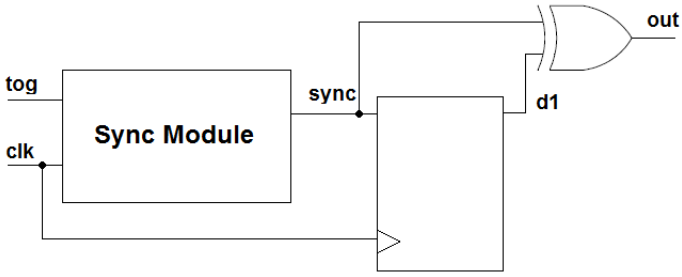
Figure 3.2 shows a 2 flop stage synchronization module

Figure 3.2. Sync Module



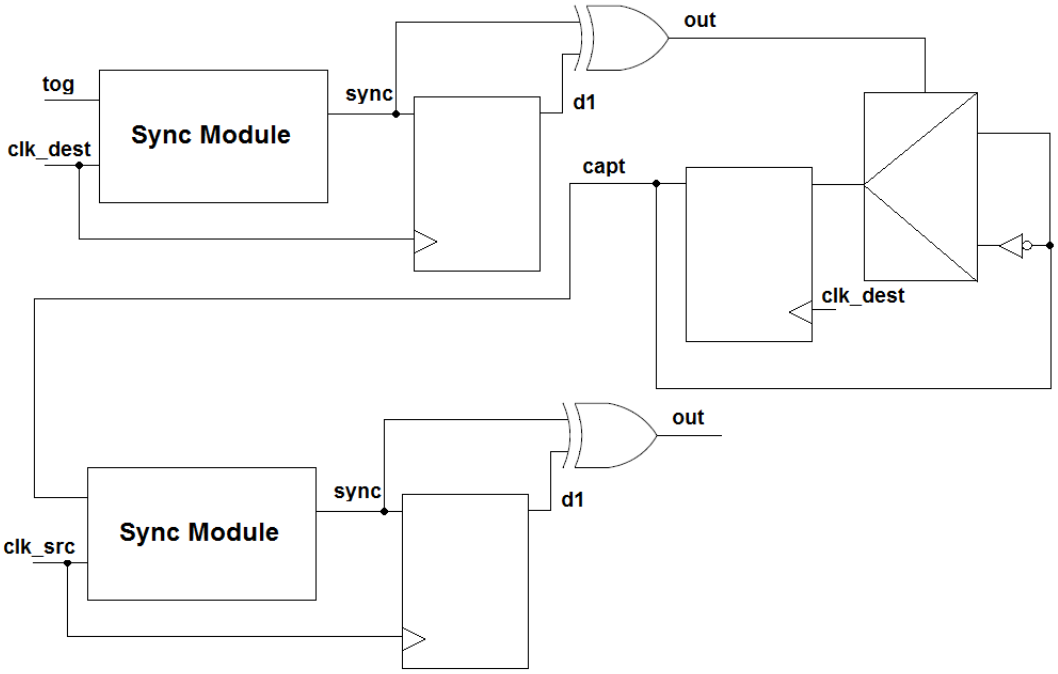
Some group of signals are synchronized with "enable pulse synchronization mechanism". Input signal toggles between occurrences.

Figure 3.3. Enable pulse synchronization mechanism



For signals which require acknowledge signal from destination domain before unblocking for toggling, synchronization scheme is extended.

Figure 3.4. Enable pulse synchronization mechanism with acknowledge



3.3.4.3. Data Path Transfer Between ref_clk and hclk

The legacy SPI core handles the key clock boundary between the AHB domain and the reference clock domain. This is part of the existing silicon proven IP and is handled using carefully implemented transmit and receive FIFO's and handshaking logic using the standard metastability rules.

3.3.4.4. Synchronization in the PHY Module

The *tx_dll_clk* is considered as output port because it does not flop any modules inside the chip containing the controller. Therefore, there are just two clock domains within PHY Module – *ref_clk* and *rx_dll_clk*. Synchronization between these domains is performed using small asynchronous FIFO embedded inside the PHY Module.

3.3.4.5. Reset

There are three asynchronous resets present in the design, one for each of the main clock domains (*hclk*, *pclk* and *ref_clk*). These resets are used directly to reset the synchronous parts of the design. The reset signals may be set low asynchronously with respect to the clock, but must be set high synchronously.

Given that *rx_dll_clk* is the delayed (and gated) variant of *ref_clk* for the configuration with the PHY Module, it has been decided that these domains have the same reset (*n_ref_rst* one). This allows the controller to keep a consistent netlist over both configurations.

3.3.5. Installation

All the information needed for a quick start with the QSPI IP are included in the Verification and Test Application.

Presence PHY Module is configured by the parameter *cdns_qspi_include_phy* included in *cdns_qspi_flash_ctrl_defs_default.v* file.

3.3.6. Compilation

Presence of the PHY Module does not affect compilation scripts. Detailed description of usage and run guidelines was included in Verification and Test Application.

3.3.7. Simulation

Presence of the PHY Module does not affect simulation scripts. Detailed description of usage and run guidelines was included in Verification and Test Application.

3.3.8. Synthesis

The SDC file for QSPI Flash Controller configuration with PHY was designed to be simply modified in terms of clock frequencies. Given that the PHY Module is clocked by *ref_clk* – frequency of this clock is critical over Static Timing Analysis for the PHY Module. Maximum synthesis proven frequency of *ref_clk* when the controller is integrated with PHY is equal to 200MHz. This is a safe margin since the current Flash Devices available in the Market can operate slightly above 100MHz. There are also additional constrains for PHY Module itself.

The SDC file for QSPI Flash Controller configuration without PHY incurs more restrictive requirements over frequencies (comparing with complete controller and PHY solution) to achieve maximum performance.

Target technology for synthesis is TSMC_28_HPM.

To run synthesis, use: *./run_phys.csh -synth* from *synth/scripts* directory. Be ensured that *cdns_qspi_synth* is uncommented in *cdns_qspi_flash_ctrl_defs_default.v*. It is assumed that Cadence RTL Compiler is correctly installed.

3.3.9. Gated Clocks

Both *tx_dll_clk* and *rx_dll_clk* are gated ones. The gating mechanism was described in the Chapter 4, *PHY Module Specification*.

3.3.10. Asynchronous and unregistered inputs

Because of the asynchronous nature of Serial Flash Devices, mi_4bits vector is an unregistered input. It determines Master Input Data from external memory.

3.3.11. Combinatorial Outputs

There are combinational outputs of QSPI Flash Controller for SRAM Interface to ensure maximum performance of Indirect transfers. It is recommended to locate the SRAM hard macro close enough to the controller for ensure the timing closure.

For PHY Module datapath, after being passed of gated *ref_clk* through TX DLL, *tx_dll_clk* turns to be an asynchronous signal. There is additional combinational delay on MUX selecting the source of SPI clock to device (one from the PHY or one divided generated by the low-level SPI Module). Regarding limitations over maximum frequency of actual Serial Flash Devices which is just slightly greater than 100 MHz – combinational delay of this clock is not critical for timing closure, as proven by synthesis.

3.3.12. Power Management

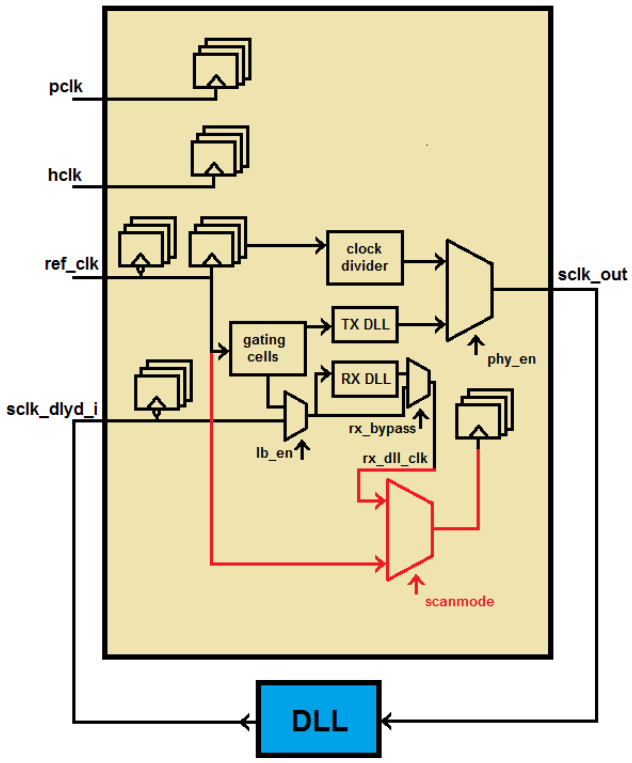
There are not any dedicated low-power mechanisms implemented in the PHY Module. When operating in 1x DDR Mode instead of 8x DDR Mode (as is limited in legacy architecture) it is possible to achieve the same throughput with 8 times lower clock value, what is very effective from the dynamic power standpoint.

3.3.13. Testability

Generally scan insertion for DFT is straightforward.

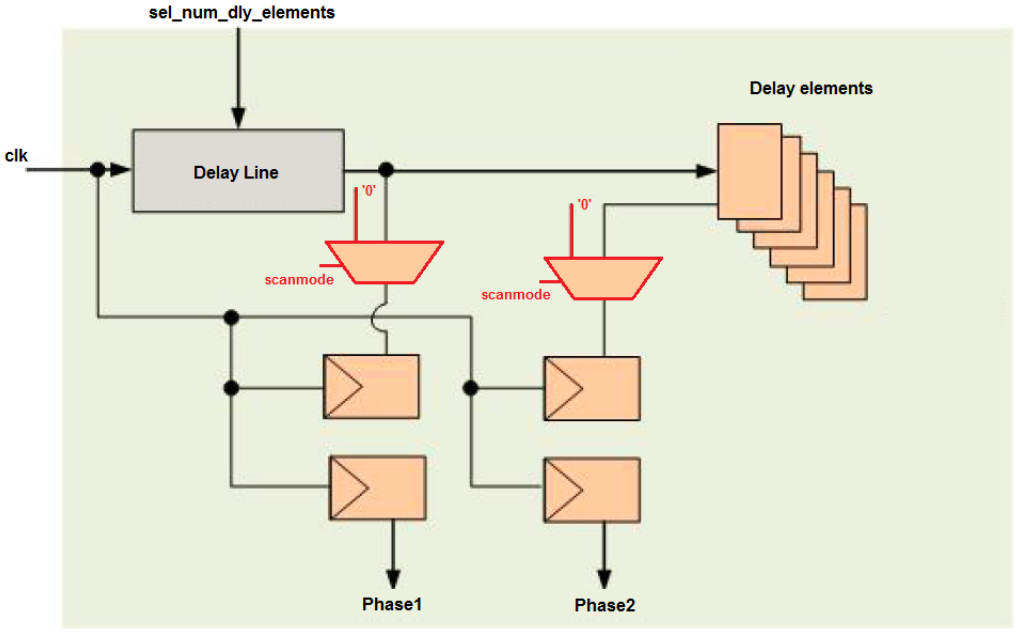
There is gated clock within PHY which is used for sampling data from Flash Device during normal working mode. This is functionally ensured that the clock is equal to *ref_clk* in terms of frequency, therefore to make its flops scannable for the test mode, dedicated MUX is implemented as depicted below in the red part:

Figure 3.5. Scan MUX for gated sampling clock



The first level phase detect flops take accordingly the *ref_clk* and its delayed variant during normal operation corner. In the scan mode, default delay of DLL is set to minimum. It could be assumed that both inputs take value of 0 (the shift is smaller than half *ref_clk* cycle). Pulling these to low makes the first level phase detect flops being scannable for DFT:

Figure 3.6. Scan MUXes for phase detect flops



3.3.14. Latches

There are no LATCH registers in this IP.

3.3.15. SDC Constraints

SDC Constraints are located in the *synth/constraints* directory of the project.

3.3.16. Debug features

There are no on-chip tests and debug features in this IP.

3.4. Verification and Test Application

There are two verification environments for the controller. The advanced UVM 'e' based environment and directed verilog environment also called "sanity testbench". The sanity testbench is the part of the customer delivery hence following sections describes it in detail.

3.4.1. Testbench Structure

There is one supplied verilog based test-bench in the *func_ver/sanity* directory. It has been developed around the RTL description, and has been used with a synthesised net-list (not provided). Numerous tests are provided with the test bench.

This verilog based verification environment uses memory models downloaded from various memory manufacturers. These cannot be delivered to IP customers due to copyright restrictions and therefore must be downloaded separately from the manufacturer’s website if the user wishes to rerun tests on the delivered test-bench.

The following models were used in this environment. Note that these models may have been updated or no longer available since they were used initially to help verify the QSPI design IP.

1. Atmel AT25DQ161.v - Note Atmel memory have been bought by Adesto and so this model should be requested by Adesto.

2. Atmel AT25DQ321.v - Note Atmel memory have been bought by Adesto and so this model should be requested by Adesto.

3. Micron N25Q1024A13E.v

4. Micron N25Qxxx.v (MT25Q)

5. Spansion S25FL129P.v

6. Spansion s25fs512s.v

Since Flash Devices generally keep their backward compatibility over the interface, integration of emerging devices not yet connected into testbench is not very time consuming task. The Models are being developed by third party vendors therefore Cadence cannot guarantee that existing tests get passed state even for updated versions of the same devices from the list pointed above. Delivered test cases should be treated as functional guideline and good template for particular user scenarios. After investigating Testbench syntax description combined with the examples, the user should have clear idea how to start with the verification of the controller using 3rd Party Models.

Additionally, Cadence has been developing virtual Multiple-SPI Flash Memory Model. The idea of this model is to support the most notable features of all listed 3rd Party Flash Devices in single one. Presence of this Model ensures all provided test passed (if Test-Bench is correctly set-up) and enables quicker start with the controller evaluation. However, it should be noticed that the Model is virtual hence it is left for a customer choice which verification approach is more suitable for their needs (3rd Party Simulation Memory Model of real device or Cadence Virtual Memory Model). More detailed information about Cadence Virtual Memory Model are covered in Cadence Virtual Flash Memory Model.

Simulation must be performed in the *work* directory (the user is responsible for creating it), which must contain a *results* and a *files* subdirectory. These are created automatically by the delivered scripts.

Test scenarios are stored in the *func_ver/sanity/tests_[configuration]* directories; these can be used as a starting point for development of further test scenarios. These test scenarios are translated using a Perl program called *trans.pl*, stored in the *func_ver/sanity/tb* directory. The output is test stimulus and comparison data stored in the *work/files* directory.

Simulation scripts are supplied in the *synth/scripts/sim* directory; these are configured to use the Cadence Incisive NC Verilog Simulator tool.

To run the simulation scripts an environment variable called *$KITSLIB* must be initialised. This is used during gate level simulations to point to the required libraries. If only performing RTL simulations *$KITSLIB* can be set to a dummy value.

To simulate the test scenarios, the RTL and testbench code must first be compiled, this is accomplished using the *compile.pl* script held in the *synth/scripts/sim* directory and again assumes that Cadence Incisive NC Verilog Simulator has been properly installed.

For example from the *work* directory use: *../synth/scripts/sim/compile.pl*

Once the RTL and testbench code has been compiled, the *simulate.pl* script should be called. To run all the tests, call the *simulate.pl* script with the option *-all* (tests will be running in serial fashion), otherwise use *-run [test case]*.

For example from the *work* directory use: *../synth/scripts/sim/simulate.pl -run basic_rw*

If the *work* directory is not the current directory, an error message will be displayed.

The script called *regression* held also in the *synth/scripts/sim* compiles and executes all tests in parallel. To use it effectively, the server farm is recommended.

To run this script, from the *work* directory use: *../synth/scripts/sim/regression -gui*

The delivered scripts have *–man* option which allows to enroll all another switches within particular script i.e. *../synth/scripts/sim/simulate.pl –man*. Only switches described in this section were verified for this IP.

After a test scenario has been simulated a brief pass or fail status is written to the *results* directory. This status is written to a file with the same name as the test scenario with an extension of *.res*. A summary of previous simulation results can be returned by executing the Unix command *cat results/** from the work directory.

Note

If user does not use the provided compilation script (*compile.pl*), then they should change name of the *hdl/hdl_src/cdns_qspi_flash_ctrl_defs_default.v* file to *hdl/hdl_src/cdns_qspi_flash_ctrl_defs.v* before run compilation.

3.4.2. Testbench setting up (Using Cadence Virtual Memory Model)

Exemplary list of steps needed to set the environment up is expanded below:

1. It is assumed that Cadence Incisive NC Verilog Simulator has been properly installed and design has been unpacked from delivered *tar*.
2. type (from console window): *cd cdns_qspi_flash_ctrl/*
3. create "work" directory: *mkdir work*
4. type: *cd work*
5. run compilation script: *../synth/scripts/sim/compile.pl*
6. Compilation should finish without errors.
7. run one of the test cases in GUI:: *../synth/scripts/sim/simulate.pl –run [file_name] -gui*
8. Press F2 to run simulation after ncsim GUI environment is loaded.

3.4.3. Testbench setting up (Using 3rd Party Memory Models)

Exemplary list of steps needed to set the environment up is expanded below:

1. It is assumed that Cadence Incisive NC Verilog Simulator has been properly installed and design has been unpacked from delivered *tar*.
2. type (from console window): *cd cdns_qspi_flash_ctrl/*
3. create "work" directory: *mkdir work*
4. type: *cd work*
5. Download Models from network (e.g. Micron ones):
   a. Go to: www.micron.com [http://www.micron.com]
   b. Register and log in there
   c. Go to: https://www.micron.com/products/nor-flash/serial-nor-flash and select desired device from the list

d. Click on device density and then name e.g. 2Gb/MT25QL02GCBA8E12-0SIT

e. Find the Verilog model and download it

6. Download Models from network (e.g. Spansion ones):

   a. Go to: http://www.spansion.com/Products/memory/Serial-Flash/Pages/Spansion%20FL.aspx

   b. Click on "design model" card and download desired model from the list

7. Download Models from network (e.g. Adesto ones):

   a. Go to: http://www.adestotech.com/simulation-tools/ and select any verilog model from the list e.g. AT45DQ161

8. It is not mandatory to instantiate more than one memory model. Example shows how to run the simulation with just downloaded Spansion S25FL512S Model. Integration of other ones is equivalent.

9. Copy all files (Spansion S25FL512S has just one Verilog file) from downloaded package into *cdns_qspi_flash_ctrl/func_ver/sanity/Memory_Models*

10. Open in editor: *cdns_qspi_flash_ctrl/func_ver/sanity/tb/cdns_qspi_flash_ctrl.v* and exclude all instantiated Cadence Virtual Memory Models from the project:

```
/*cdns_generic_memory_model #(
  .MEMORY_NAME("[CDNS_GENERIC_MEM_MODEL_0]")
  ) cdns_generic_memory_model_0 (
  .sclk (sclk_out),
  .cs_n (n_ss_out[0]),
  .dq0 (flash_si[0]),
  .dq1 (flash_so[0]),
  .dq2 (flash_wpn[0]),
  .dq3 (flash_holdn[0]),
  .dq4 (flash_dq4[0]),
  .dq5 (flash_dq5[0]),
  .dq6 (flash_dq6[0]),
  .dq7 (flash_dq7[0])
);
(...)
cdns_generic_memory_model #(
  .MEMORY_NAME("[CDNS_GENERIC_MEM_MODEL_3]")
  ) cdns_generic_memory_model_3 (
  .sclk (sclk_out),
  .cs_n (n_ss_out[3]),
  .dq0 (flash_si[3]),
  .dq1 (flash_so[3]),
  .dq2 (flash_wpn[3]),
  .dq3 (flash_holdn[3]),
  .dq4 (flash_dq4[3]),
  .dq5 (flash_dq5[3]),
  .dq6 (flash_dq6[3]),
  .dq7 (flash_dq7[3])
);*/
```

It is worth to notice that *cdns_qspi_flash_ctrl/func_ver/sanity/tb/cdns_qspi_flash_ctrl.v* includes commented instances of 3rd Party Devices. This code part may act as guideline in integration them. Some examples are depicted below:

```
// ATMEL AT25DQ321 FLASH MODEL - 32Mbit
/*  AT25DQ321 i_atmel_flash_model (
    .CSB    (n_ss_out[0]),
    .SCK    (sclk_out),
    .SI     (flash_si[0]),
    .WPB    (flash_wpn[0]),
    .SO     (flash_so[0]),
    .HOLDB  (flash_holdn[0]),
    .VCC    (1'b1),
    .GND    (1'b0)
  );*/

  // SPANSION S25FL129P 128Mbit Flash Model
/*  s25fl129p00 i_s25fl129p_spansion_flash_model (
    .SCK      (sclk_out),
    .SI       (flash_si[1]),
    .CSNeg    (n_ss_out[1]),
    .HOLDNeg  (flash_holdn[1]),
    .WPNeg    (flash_wpn[1]),
    .SO       (flash_so[1])
);*/
```

11. Instantiate downloaded model instead of removed ones:

```
s25fs512s i_s25fs512s_spansion_flash_model (
    .SCK      (sclk_out),
    .SI       (flash_si[1]),
    .CSNeg    (n_ss_out[1]),
    .RESETNeg (flash_holdn[1]),
    .WPNeg    (flash_wpn[1]),
    .SO       (flash_so[1])
);
```

12. Add new needed files and remove non using models from *cdns_qspi_flash_ctrl/func_ver/sanity/tb/cdns_qspi_flash_ctrl.f*

    Exemplary model to remove from the file:

    ../Memory_Models/cdns_generic_memory_model.v

    Line to add:

    ../Memory_Models/s25fs512s.v

13. run compile script from work directory: *../synth/scripts/sim/compile.pl*

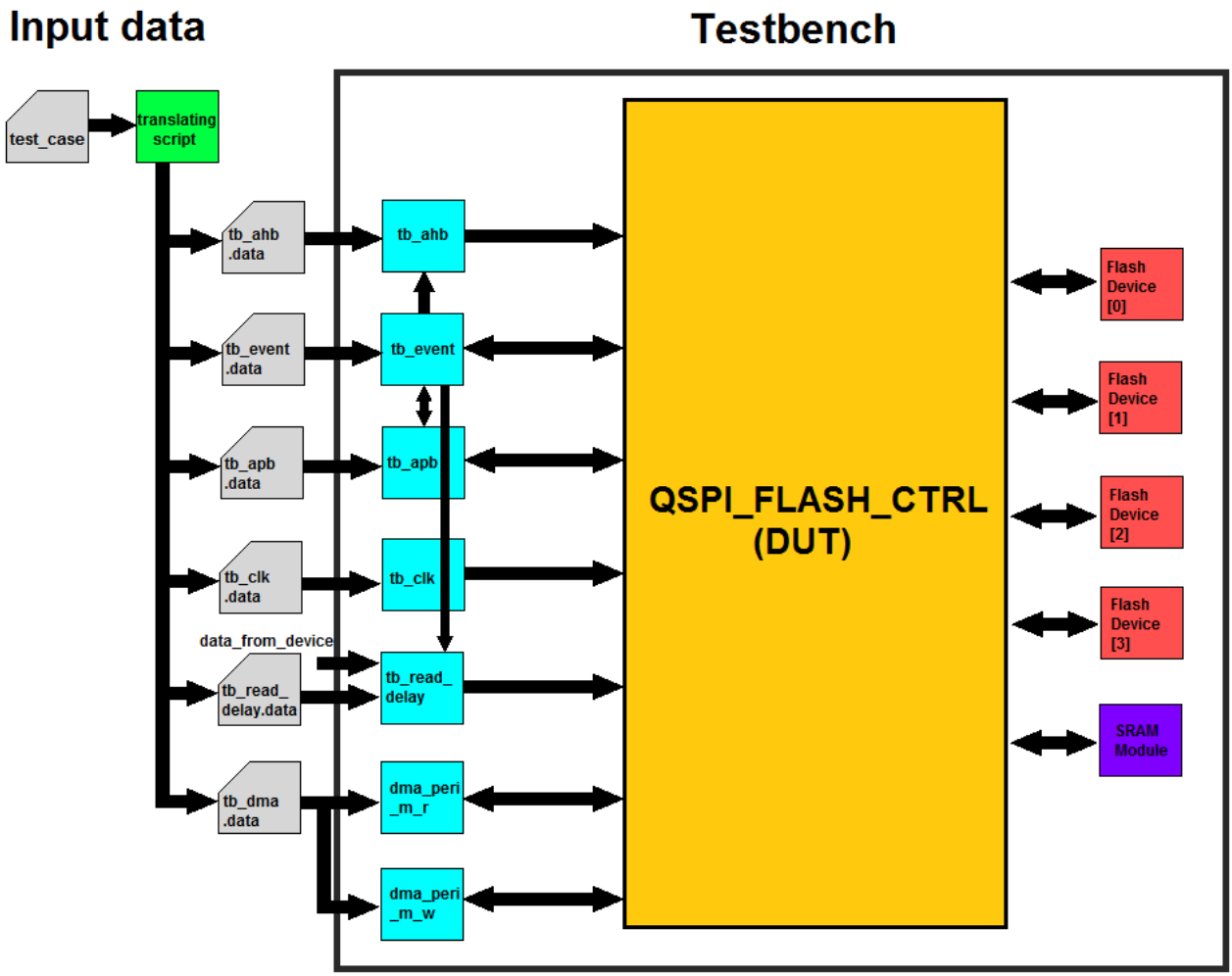
14. Compilation should finish without errors.

15. run one of the test cases in GUI: *../synth/scripts/sim/simulate.pl –run [file_name] -gui*

16. Press F2 to run simulation after ncsim GUI environment is loaded.

3.4.4. Testbench blocks' description

Figure 3.7. Testbench Block Diagram



The sub-blocks shown on the picture can be described as follows:

3.4.4.1. "test_case" block

This block represents file with test case selected to being run.

3.4.4.2. "translating_script" block

This block represents the perl script file which translates tb commands into data chain files.

3.4.4.3. "*.data" blocks

These files contain separated data chains for particular command groups described in Testbench syntax description.

3.4.4.4. "tb_ahb" block

This block translates data chain into actual AHB Interface.

3.4.4.5. "tb_apb" block

This block translates the data chain into actual APB Interface. It communicates with the controller in both directions. This is caused by the fact that interrupt logic (generated by the controller) was implemented in APB domain.

3.4.4.6. "tb_event" block

This block controls simulation time of all interface stimuli scheduled to execute within the test case.

3.4.4.7. "tb_read_delay" block

This block emulates additional read data delay between the Flash Device data outputs and the controller data inputs.

3.4.4.8. "dma_peri_m_r" block

This block emulates external DMA Module Interface for Indirect Read Mode.

3.4.4.9. "dma_peri_m_w" block

This block emulates external DMA Module Interface for Indirect Write Mode.

3.4.4.10. "QSPI_FLASH_CTRL (DUT)" block

Device under test.

3.4.4.11. "Flash Device [3:0]"

Memory Models from third party vendors.

3.4.4.12. "SRAM Module"

Verification Model of SRAM.

3.4.5. Testbench syntax description

3.4.5.1. APB Interface commands

3.4.5.1.1. APB Write

Syntax:

apbwr([simulation time]) [address] [data]

Description:

Address parameter is size of 1 byte and data parameter is size of 4 bytes. Both need to be given in hex base. Simulation time is calculated based on configured resolution of it. When simulation time parameter is not specified, valid access is queued to handle just after the last one from actual clock domain.

Example:

apbwr(1000) 00 00080481 --Write data into configuration register

3.4.5.1.2. APB Read

Syntax:

apbwr([simulation time]) [address] [expected_data]

Description:

Address parameter is size of 1 byte and data parameter is size of 4 bytes. Both need to be given in hex base. Simulation time is calculated based on configured resolution of it. When simulation time parameter is not specified, valid access is queued to handle just after the last one from actual clock domain. If data was not predicted, it would be filled with "xxxxxxxx".

Example:

apbrd(1000) 00 00080481 --Read configuration register and check its value

3.4.5.2. AHB Interface commands

3.4.5.2.1. AHB Write

Syntax:

ahbwr([simulation time]) [hburst] [htrans] [hsize] [address] [data]

Description:

First three parameters (hburst, htrans and hsize) need to be given in dec base. Both Address parameter and data parameter are size of 4 bytes. They need to be given in hex base. Simulation time is calculated based on configured resolution of it. When simulation time parameter is not specified, valid access is queued to handle just after the last one from actual clock domain.

Example:

ahbwr(1000) 3 3 2 00000000 12345678 --Perform sequential write 4-byte burst access

3.4.5.2.2. AHB Read

Syntax:

ahbrd([simulation time]) [hburst] [htrans] [hsize] [address] [data]

Description:

First three parameters (hburst, htrans and hsize) need to be given in dec base. Both Address parameter and data parameter are size of 4 bytes. They need to be given in hex base. Simulation time is calculated based on configured resolution of it. When simulation time parameter is not specified, valid access is queued to handle just after the last one from actual clock domain. If data was not predicted, it would be filled with "xxxxxxxx".

Example:

ahbrd(1000) 3 3 2 00000000 12345678 --Perform sequential read 4-byte burst access and check correctness of expected read data

3.4.5.3. AHB bulk transfer commands

3.4.5.3.1. AHB bulk write

Syntax:

do_bulk_write([simulation time]) [no_of_AHB_words] [address] [data] 1/[no_of_wait_states]

Description:

This command handles large data transfers and calculates their performance. Interface values are automatically set as following:

- hburst = 1, htrans = 2, hsize = 2

Address parameter refers to initial address. Data parameter can be set as following:

- incrementing, decrementing, [size_of_4_bytes_hex_value]

Simulation time is calculated based on configured resolution of it. When simulation time parameter is not specified, valid access is queued to handle just after the last one from actual clock domain.

Examples:

do_bulk_write(1000) 1024 00000000 ffffffff 1/1

do_bulk_write(2000) 1024 00000000 incrementing 1/1

3.4.5.3.2. AHB bulk read

Syntax:

do_bulk_read([simulation time]) [no_of_AHB_words] [address] [expected_data] 1/[no_of_wait_states]

Description:

This command handles large data transfers and calculates their performance. Interface values are automatically set as following:

- hburst = 1, htrans = 2, hsize = 2

Address parameter refers to initial address. Data parameter can be set as following:

- incrementing, decrementing, [size_of_4_bytes_hex_value]

Simulation time is calculated based on configured resolution of it. When simulation time parameter is not specified, valid access is queued to handle just after the last one from actual clock domain.

Examples:

do_bulk_read(1000) 1024 00000000 ffffffff 1/1

do_bulk_read(2000) 1024 00000000 decrementing 1/1

3.4.5.4. Other commands

3.4.5.4.1. Data path delay

Syntax:

set_read_data_path_delay([simulation time]) [delay_value]ns

Description:

This command models additional delay of data from device.

Example:

set_read_data_path_delay(100) 3ns --calculated data path delay for individual purpose is different than one returned by Simulation Model of the device. Therefore adjustment is needed.

3.4.5.4.2. DMA interface settings

Syntax:

set_tb_dma [time_in_ns] [drready] [davalid] [datype]

Description:

This command emulates interface signals from external DMA Module. All parameters need to be given in dec base.

Example:

set_tb_dma 25200000 0 1 2

3.4.5.4.3. Clock values settings

Syntax:

clk_values [apb_clk] [ahb_clk] [ref_clk]

Description:

This command must be the first one in each test and cannot be used more than once within single test case. It configures clock values. Each parameter is interpreted by TB as given in MHz.

Example:

clk_values 100 100 400

3.4.5.4.4. Special characters

Each test should be ended with end([simulation time]) keyword. Any comments should be preceded by --.

Example:

end(10000) -- End of the test

3.4.5.4.5. Interrupt indication

Each command related with APB or AHB interfaces which takes [simulation time] as a parameter, can also be triggered relying on any incoming interrupt. It should be noticed that interrupt output is located in APB domain hence some latency caused by synchronization is imposed on AHB domain before TB gets the information about interrupt event.

Example:

apbrd(int) 40 00000004 -- INDAC complete interrupt is expected to be asserted by the test just after interrupt output is high

3.4.6. CPU Based Testing

No CPU based tests are supplied with this release of the IP.

3.4.7. Cadence Virtual Flash Memory Model

The Cadence Virtual Flash Memory Device is a high-performance serial flash memory device which also includes one-time programmable (OTP) memory area. It features a high speed SPI-compatible bus interface with DDR clock frequency using up to eight I/O signals. The memory is organized as uniform 64kB sectors with 4kB and 32kB sub-sectors, and 256 byte pages. The device offers an advance security protection scheme where each sector can be independently locked.

The detailed specification of the Cadence Virtual Flash Memory Model is available as the separate document cdns_virtual_flash_memory_model.pdf which is a part of the delivery.

4. PHY Module Specification

The Cadence QSPI PHY Module IP, subsequently referred to in this document also as the "PHY Module", was designed in order to extend the architecture of the Cadence QSPI Flash Controller IP. The controller itself assumes that SPI transfer clock should be at least 4X times lower than the reference clock to ensure correct data transactions in SDR Mode and 8X times for the DDR one. This provides comfortable regulation of data synchronization but at the same time it is not effective from a dynamic power efficiency standpoint. Furthermore, emerging devices are getting better and better over their performance. The higher is the operation frequency (SPI clock) and along with it the reference frequency, the narrower are the delay windows for synthesis. With the assumption that emerging Flash Devices can operate on frequencies close to 200 MHz, special care should be taken to synthesize the system-on-chip (SoC) on such frequency. Moreover, specific requirements for actual systems might not be able to operate properly on such a frequency. Implementation of the PHY Module dedicated to work with the QSPI Flash Controller enables performing either DDR or SDR transfers with these clocks equal to each other in terms of frequency. It means that even if a given Flash Device operates on 200MHz, the reference clock is also 200MHz, and as a result the synthesis problem for critical paths is solved. Major part of low-level SPI logic had to be modified to adapt the PHY Module to the logic of existing IP. More details are presented in the sections provided below.

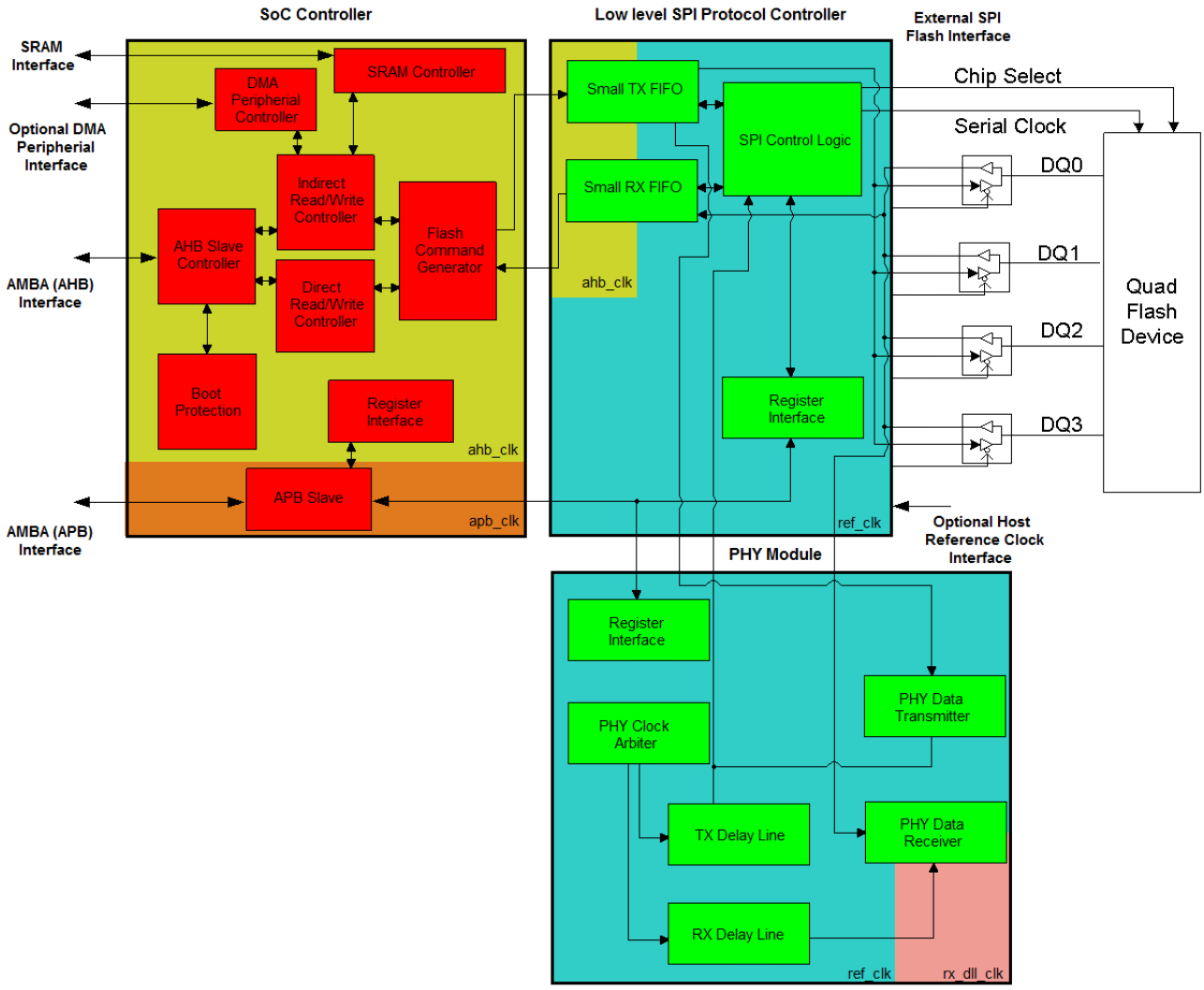
The following topics are discussed in this chapter:

4.1. Controller and PHY solution architecture

High-level architecture of the Cadence QSPI IP can be divided into two blocks. The first one is the SoC Controller which interfaces with an external Master. It also allows to queue Flash transactions based on data from ARM AMBA AHB and APB bus interfaces. The second module is the low-level SPI Protocol Controller. It is used for serializing data from SoC Controller Module and translating them into SPI Transfer Protocol. There are two possible data paths here. The first one does not use the PHY Module for formulating SPI Transactions. Using it is recommended when there is no strict requirement of high performance (clock divider is greater or equal 4). This path preserves backward compatibility with the previous solution and bypasses the PHY Module. The second one is the data path with PHY Module for dividers lower than 4. The configuration is set by software.

Figure 4.1 shows a block diagram of the QSPI Flash Memory Controller and PHY.

Figure 4.1. Block Diagram of the QSPI IP with the PHY Module

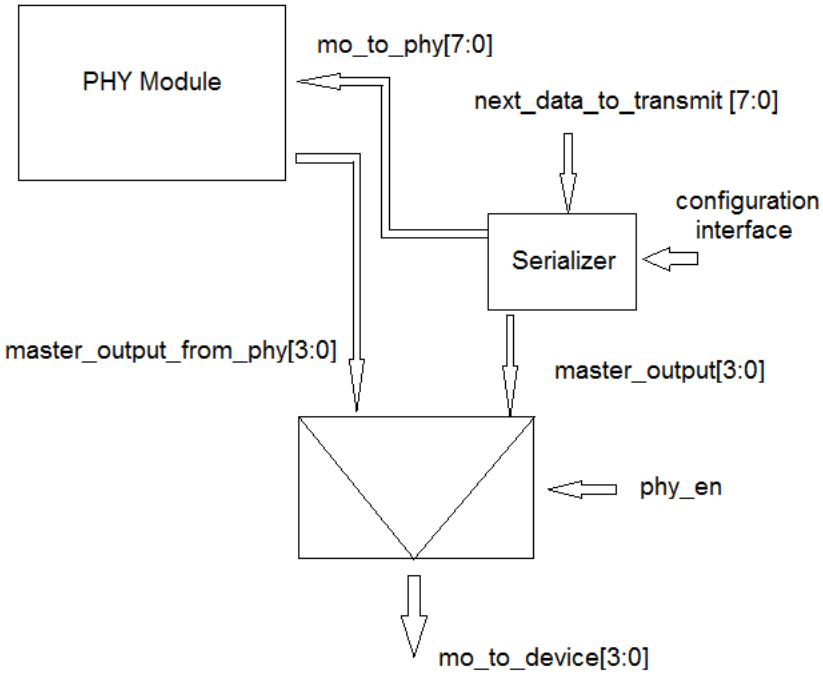


The controller is waiting for valid access from AHB bus or for software trigger from the APB one. When such an event occurs, the corresponding internal controller is selected (Direct Controller, Indirect Controller or Software-Triggered Instruction Generator). The block called Flash Command Generator arbitrates between accesses and forwards control into low-level SPI Module. This block works on *ref_clk*. Data to transmit are synchronized from *ahb_clk* domain (clock of Flash Command Generator) to the *ref_clk* one in TX FIFO and then serialized to SPI Interface. When direction of transfer changes and device returns data to the controller, these are sent into RX FIFO and then synchronized from *ref_clk* domain to the *ahb_clk* one and then they are ready to be put on the AHB Bus. The SPI control logic selects source of data path (with or without PHY) based on configuration. Note, *ahb_clk* and *apb_clk* are not directly used for low-level SPI transfer. Instead, the reference clock and its delayed variants are relevant for this type of transfer, with the PHY Mode being enabled. The Section 4.1.1.1, "*Input data path*" and Section 4.1.1.2, "*Output data path*" describes TX and RX data paths in more detail.

4.1.1. Communication between QSPI Flash Controller and PHY

4.1.1.1. Input data path

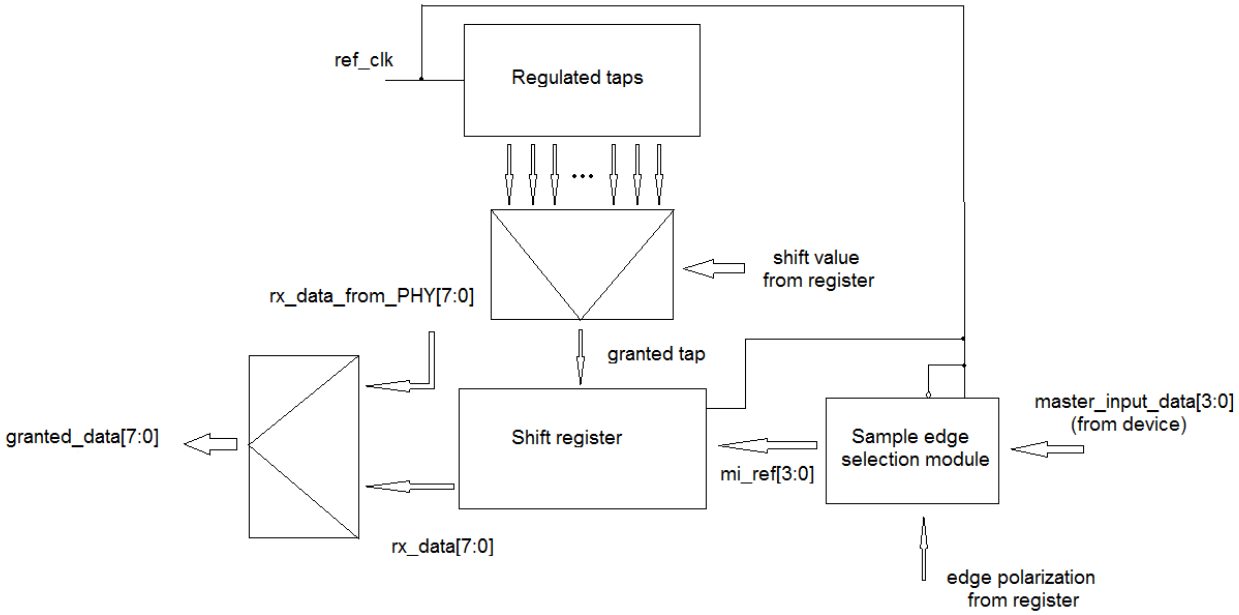
Figure 4.2. Input Data Path Diagram



The SPI Control Module generates next byte to translate on SPI Interface (this is the last data popped from TX FIFO or one of higher prioritized self-generated data phase) and then it is ready to be sent on SPI I/Os. Depending on the configured IO Mode (Single, Dual or Quad), the Serializer translates input data into Multiple SPI interface-compliant data. When the PHY Mode is enabled, data can be toggled on both edges of *ref_clk* (range of master output register is doubled due to this). The PHY Module handles such transaction and generates half-clock toggling translation on Master Output Interface which is then sampled by *tx_dll_clk* adjustable to meet setup and hold timing conditions. PHY Mode enable bit determines which master output path is selected. Detailed description of delay and arbitration of SPI clock is presented in the PHY Module Architecture section Section 4.1.2, "*PHY Module Architecture*".

4.1.1.2. Output data path

Figure 4.3. Output Data Path Diagram



When the device returns data and the PHY Mode is disabled, the data is captured by a mechanism that using taps.

Given that the above-mentioned limitations (/4 SDR and /8 DDR) over clock dividers have to be met, there are at least 8 samples that need to be adjusted (half clock resolution of ref_clk is implemented and configured by default). More detailed description of this mechanism is provided in Section 2.1.11.1, *"Mechanism using taps"*. The data is then formulated in shift register and when 1 byte is completed, it is pushed into RX FIFO and synchronized into ahb_clk to be ready for putting on the AHB bus.

When the PHY Mode is enabled, data is sampled using *rx_dll_clk* adjusted by the user. Then it is synchronized to reference domain, pushed into RX FIFO, and synchronized into *ahb_clk* domain to be ready for putting on the AHB bus.

4.1.2. PHY Module Architecture

The PHY Module can be basically divided into four sub-blocks: Data Transmitter, Data Receiver, Delay Line Module, and Clock Arbiter.

The Data Transmitter serializes the data to transmit in order to assign complaint signal values into Master Output Interface. The input data is byte-sized. The configuration interface covers among others selected SPI Protocol (Single, Dual or Quad) or type of command (DDR, SDR) – these values are needed to calculate how many bits fit into a clock cycle and consequently what value of step should be set by the hardware.

The Data Receiver block samples data from Flash Device on *rx_dll_clk* and synchronizes them to reference clock domain which Low-level SPI RX FIFO is flopping by.

There is also the Delay Line Module block which provides an appropriate delay of input clock to ensure correct data transmitting and sampling. This block contains two separated delay paths for SPI clock feeding external FLASH device and internal PHY sampling clock (*rx_dll_clk*) which is used for capturing data from the device when the PHY Mode is enabled. Delay values of each line are configurable by software.

The last module is the Clock Arbiter. It handles gating clock logic for clocks which are further forwarded to DLLs' inputs. Additionally, it selects the source of the *rx_dll_clk* - gated *ref_clk*and loopback clock input signals are possible to forward into RX DLL. The decision is made based on software configuration.

Figure 4.4. PHY Module Diagram

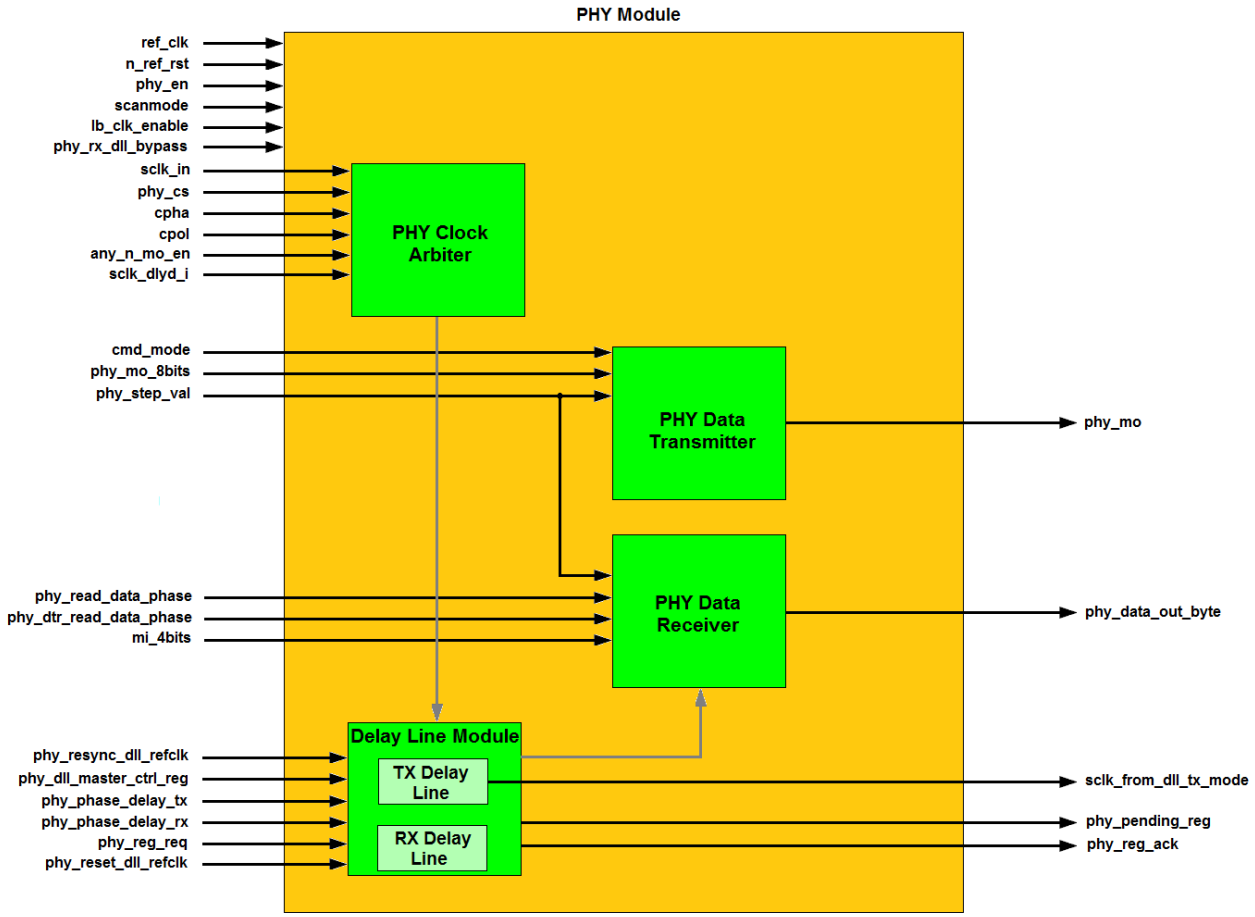


Table 4.1. PHY Module Signals' Description

Signal Name	I/O	Description
ref_clk	I	Reference clock (*ref_clk*).
n_ref_rst	I	Active low *n_ref_rst* reset. This signal must be asserted low asynchronously, and de-asserted high synchronously with *n_ref_rst*. Resets all registers in the reference clock domain.
phy_en	I	Bit from APB Register Module synchronized into reference domain. It determines if PHY Module is enabled.
scanmode	I	DFT Scan Mode input
lb_clk_enable	I	This bit comes from Register Module and determines if loopback clock is to be used for sampling data from Flash Device
phy_rx_dll_bypass	I	This bit comes from Register Module and determines if external delay of loopback clock is large enough to capture valid data without further delaying by RX DLL.

Signal Name	I/O	Description
sclk_in	I	Divided clock derived from the low-level SPI Control Module. It is applicable for PHY Module when DDR 2X Mode is granted (only scenario when divided clock needs to passing through DLL).
phy_cs	I	This signal is asserted high when active SPI transfer is ongoing. It is generated by the low-level SPI Control Module.
cpha	I	This bit comes from Register Module and determines configured SPI clock phase.
cpol	I	This bit comes from Register Module and determines configured SPI clock polarity.
phy_read_data_phase	I	This signal is asserted high when active SPI transfer is ongoing and its direction is set as input for the controller. It is generated by the low-level SPI Control Module.
any_n_mo_en	I	This signal determines whether any data lines are active for Read phase of current SPI transfer
sclk_dlyd_i	I	Loopback SPI clock
cmd_mode[1:0]	I	Determines Edge Mode as follows: • 2'b00 - Inactive state • 2'b01 - SDR command • 2'b10 - DDR command • 2'b11 - Unused
phy_mo_8bits[7:0]	I	Next byte to serialize and translate for the SPI Interface.
phy_step_val[2:0]	I	This signal determines value of step to complete single FIFO data word. Its value depends on the SPI transfer Mode (Quad, Dual or Single) and the Edge Mode (SDR or DDR).
phy_dtr_read_data_phase	I	This signal is asserted high when active DDR SPI transfer is ongoing and its direction is set as input for the controller. It is generated by the low-level SPI Control Module.
mi_4bits[3:0]	I	Input data interface from the Flash Device.
phy_resync_dll_refclk	I	This signal comes from the Register Interface and is used for resynchronisation of both Delay Lines to enable them to upgrade their configuration.
phy_dll_master_ctrl_reg[31:0]	I	This signal comes from the Register Interface and determines control logic for the Master DLL.
phy_phase_delay_tx[6:0]	I	This signal comes from the Register Interface and determines the number of delay elements of SPI clock.
phy_phase_delay_rx[6:0]	I	This signal comes from the Register Interface and determines the number of delay elements of sampling clock.
phy_reg_req	I	This signal mediates in synchronisation mechanism between internal PHY registers and the Controller APB Register interface. Asserted when valid read request of any PHY register field occurs.

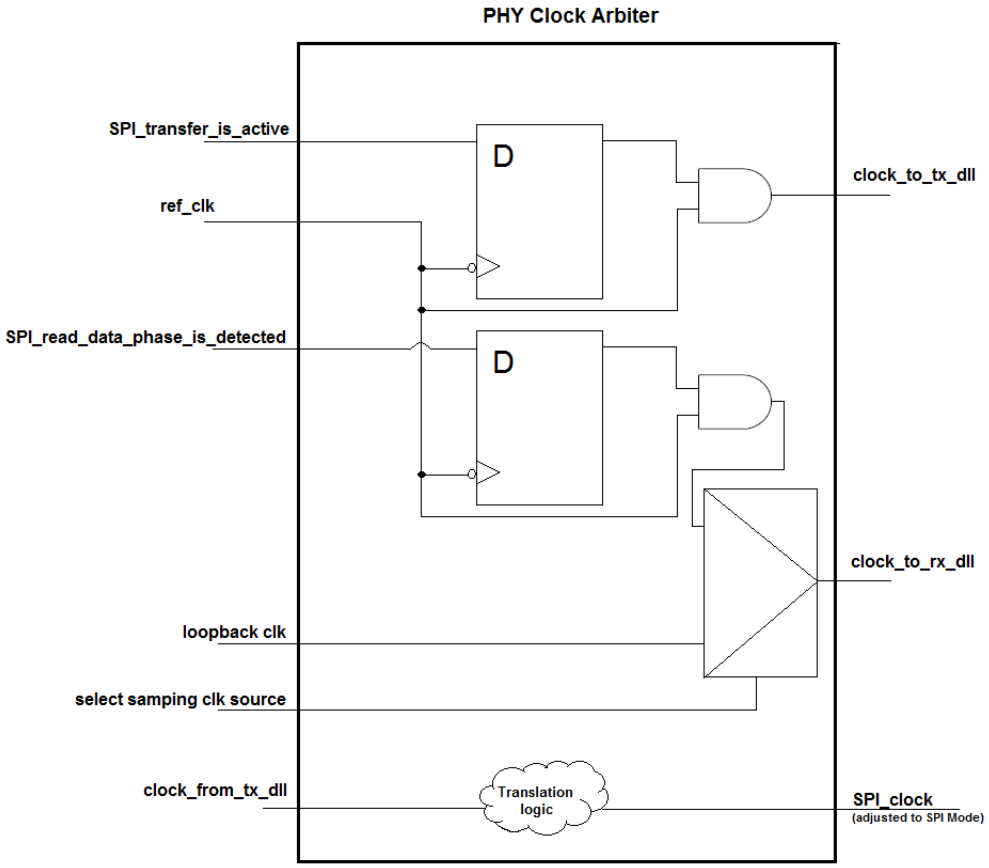
Signal Name	I/O	Description
phy_reset_dll_refclk	I	This bit is used for reset of Delay Lines by software. The reset is active when de-asserted.
phy_mo[3:0]	O	Data to transmit on SPI Master Output interface generated by the PHY Module.
sclk_from_dll_tx_mode	O	Delayed reference clock which acts as SPI clock in the PHY Mode adapted into configured SPI Mode.
phy_pending_reg[63:0]	O	Merged form of all observable Delay Line fields possible to read by software.
phy_reg_ack	O	This signal mediates in synchronisation mechanism between internal PHY registers and the Controller APB Register interface. Asserted when valid read request of PHY register fields is ready to acknowledge.

4.1.2.1. PHY Clock Arbiter

The SPI Clock is generated only when the SPI Control Logic FSM indicates that SPI transfer is ongoing. Conversely, *ref_clk* is generated continuously. When 1X case is configured *ref_clk* has to be gated. It is implemented by using common solution with single flip flop and AND gate. The actual clock is selected based on the PHY enable bit from the Register Interface.

This Module generates also a sampling clock ready to be passed through the RX delay line. It is generated when Read Data phase of SPI transfer is detected based on information from the SPI Control Module. The sampling clock can be optionally sourced from loopback input. The figure below presents a simplified scheme of the implemented solution.

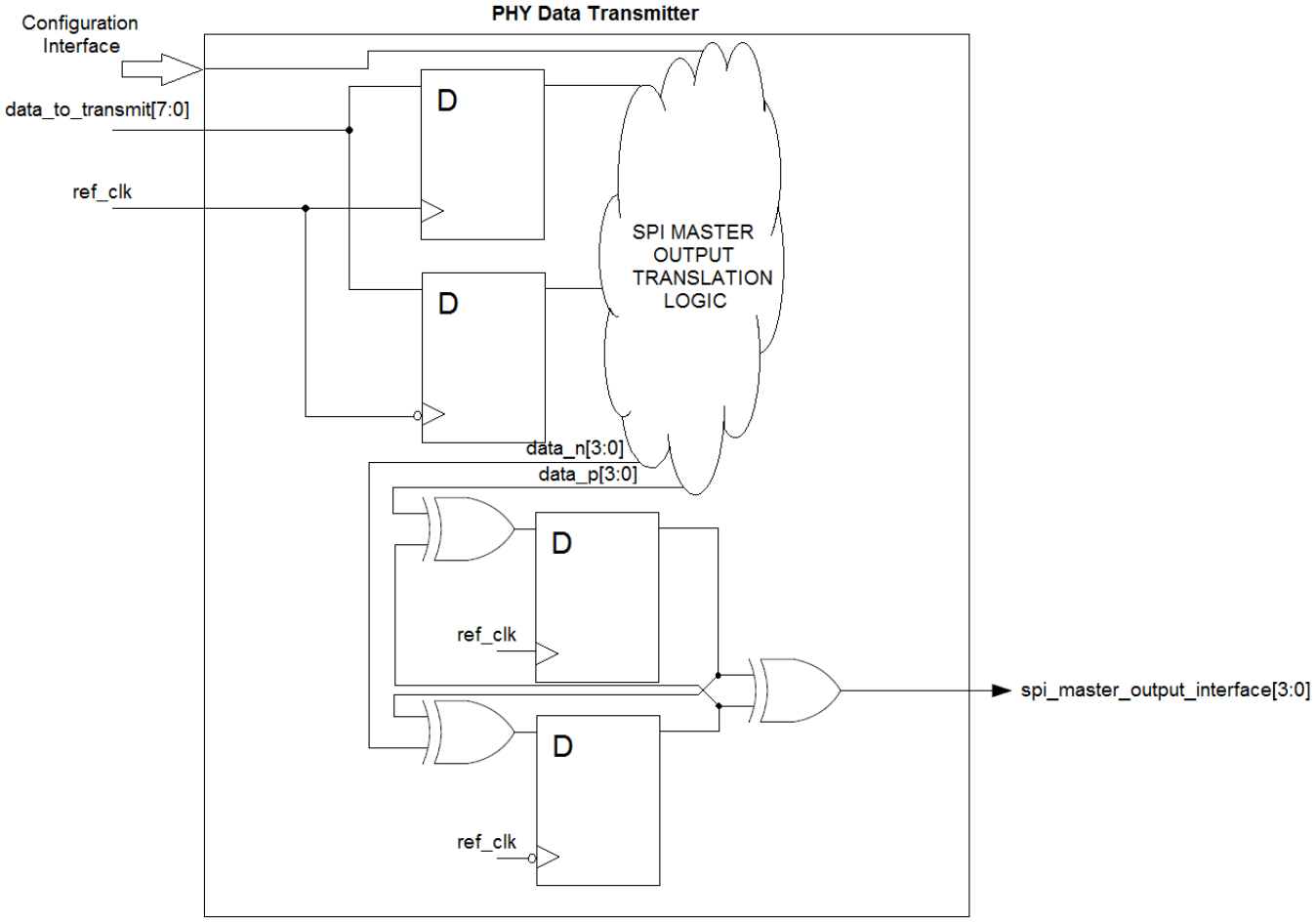
Figure 4.5. PHY Clock Arbiter



4.1.2.2. PHY Data Transmitter

The Data Transmitter latches queued data to transmit on both negative and positive edges of *ref_clk*. Then data is chopped to fit into Master Output Interface. In the last stage of TX data path with PHY Module, data from both edges are connected, forwarded back to low-level SPI Module and then put on SPI Master Outputs (synchronous multiplexing technique has been used) if PHY enable bit is asserted. Otherwise, the Master Outputs generated by the low-level SPI Module are selected.

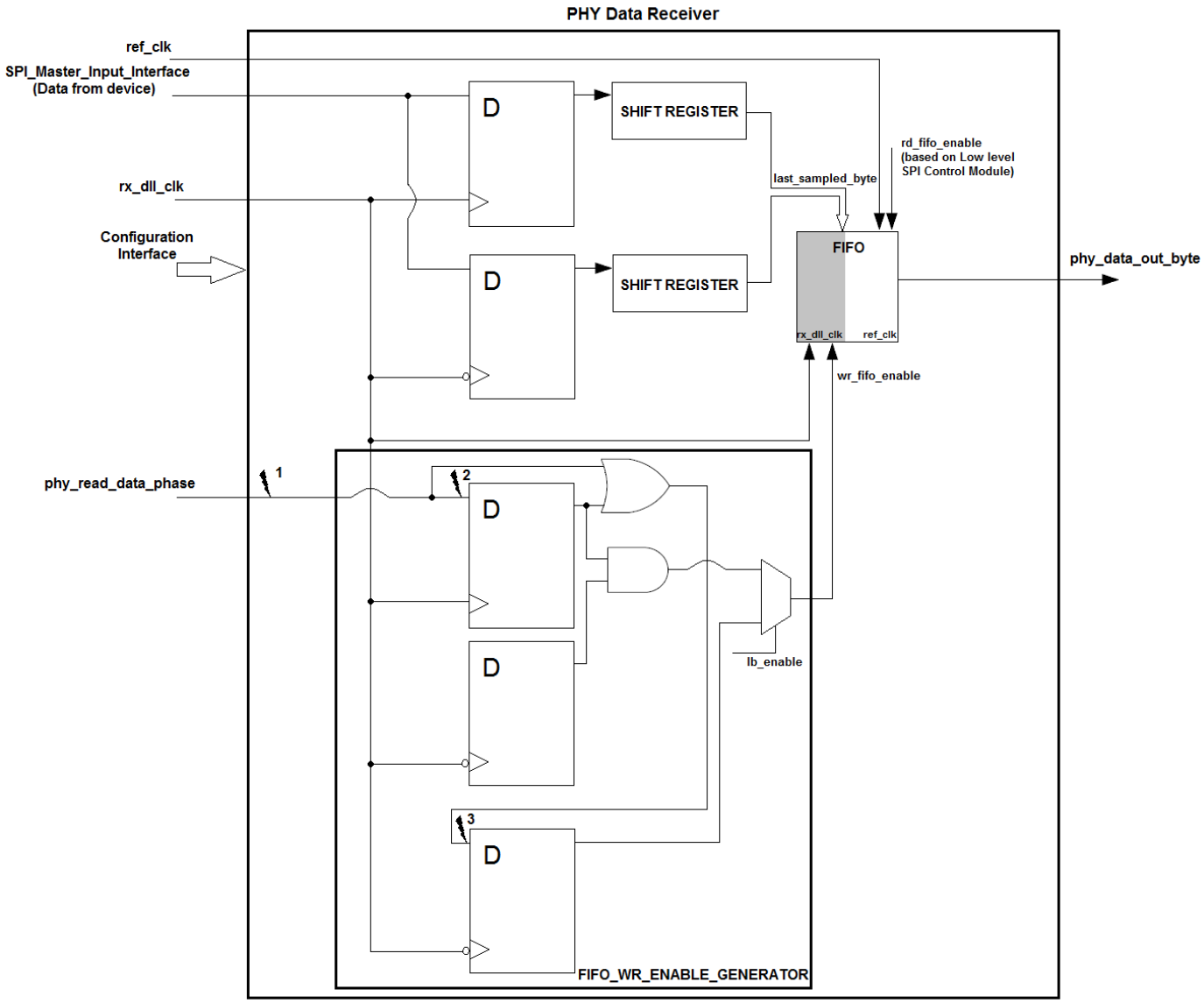
Figure 4.6. PHY Data Transmitter



4.1.2.3. PHY Data Receiver

The idea of Data Receiver Module is quite simple. Data is sampled on both edges of delayed reference clock (sampling clock) which ensures the wide range of the sampling window. Read data are gathered in shift registers and then merged into byte-sized input FIFO location. Given that this data is forwarded further into RX FIFO of low-level SPI Module with write side flopped by *ref_clk*, another internal PHY FIFO was implemented to synchronize data from *rx_dll_clk* domain to the *ref_clk* one. These clocks are asynchronous between each other but equal over their frequency. To keep maximum performance, a dedicated PHY FIFO write enable generator was designed. Read phase detection signal comes from the low-level SPI Module and number of cycles needed to synchronize this signal would affect performance because of Read Data saturation. Latching *phy_read_data_phase* signal on following positive edge of *rx_dll_clk* (which could occur very close to the last positive edge of *ref_clk*) incurs the risk of incorrect data being flopped due to the setup timing requirement. The implemented solution ensures additional half clock setup timing window. There are constrains added to delivered SDC file inserting maximum delay from 1 to 2 and 1 to 3 equal to 50% *ref_clk* (refer to the figure below). Not violated synthesis results prove reliability of this solution. The configuration interface controls transmission phase in order to detect completion of data. If the loopback clock has been selected by the PHY clock arbiter module, *rx_dll_clk gets* the PHY Receiver Module as inverted delayed loopback clock. The inverse ensures the last edge being positive what is needed to write the last captured data into the FIFO. Additionally, "AND" gate is added for de-asserting the *wr_fifo_enable* on the same edge in loopback sampling Mode. If the gated *ref_clk* is selected, the inverse is not needed since gating logic generates one more clock cycle using for writing the last data into the FIFO.

Figure 4.7. PHY Data Receiver



4.1.2.4. Delay Line Module

Due to the asynchronous nature of the Flash devices, the timing requirements for capturing and receiving data between the ASIC and the Flash devices must be addressed in any DDR PHY design. The PHY Module contains a circuit that, in conjunction with I/O cell circuitry, can be used to meet the timing requirements for an ASIC design. The delay compensation circuit was designed with the following features:

- Programmable SPI clock delay specified as a percentage of a clock cycle
- Programmable sampling clock delay specified as a percentage of a clock cycle
- Delay compensation circuit re-sync circuitry activated during refresh cycles to compensate for temperature and voltage drift

The delay compensation circuitry relies on a master/slave approach. There is a master delay line which is used to determine how many delay elements constitute a complete cycle. This count is used, along with the programmable fractional delay settings, to determine the actual number of delay elements to program into the slave delay lines. The master and slave delay lines are identical. This approach allows the memory controller to observe a clock and then delay other signals a fixed percentage of that clock.

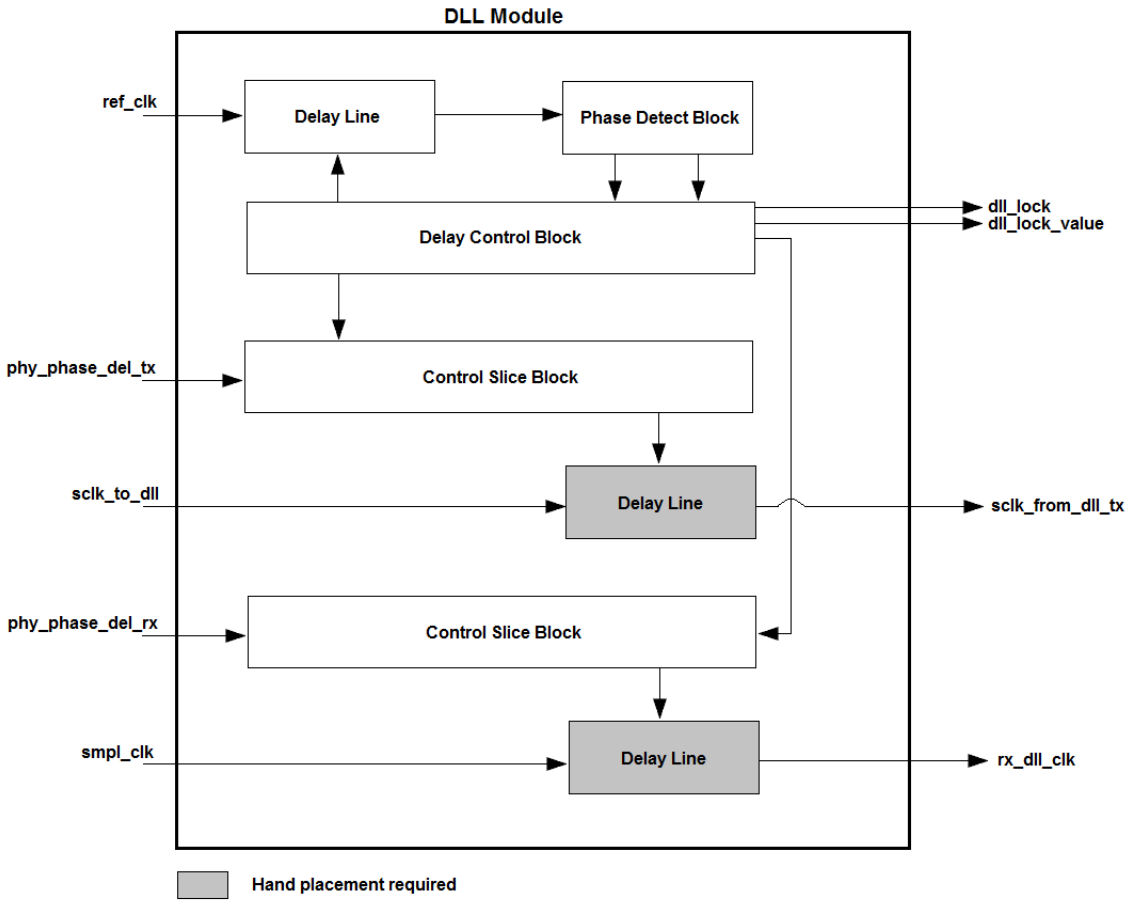
The actual delay element for the delay chains is user-selectable. The RTL code for the DLL is structured in a netlist fashion so this cell can be easily inserted into the design. This allows this delay compensation circuit implementation to be migrated to several different process technologies.

The design of the delay compensation circuit can be described as three steps:

- Determine the number of delay elements needed to capture an entire clock cycle
- Determine the number of delay elements needed to delay the reference clock for both TX and RX delay lines
- Configure and design the delay chains

The block diagram for the delay compensation circuit is shown in the figure below:

Figure 4.8. DLL Module



This process begins by centring on a one-hot counter and a pair of flip-flops. These devices determine when the rising edge of the master clock meets the rising edge of the master clock after it is sent through its delay line. The one-hot

counter can insert or remove a delay element each cycle until the rising edges of the two signals are aligned. Once this is achieved, the number of delay elements needed to capture a clock cycle is known. This number determines the number of select lines needed in the encoder to control the delay and the maximum number of cycles that the delay compensation circuit needs for initialization.

The number of elements that are needed to capture an entire clock cycle is then converted into an unsigned integer named *encoder[6:0]*. This integer is used as the dividend for the read and write delay parameters. The actual delay setting for the delay lines is calculated by multiplying the *encoder[6:0]* integer by the parameter settings for each delay line and then dividing by 128 and rounding. These values are then encoded into a one-hot counter and updated at initialization and at every slave update interval.

The *encoder[6:0]* value can be used to determine if the DLL has locked on to a harmonic of the clock signal. To check this, the user must have knowledge of the approximate delay of a single delay element in the DLL at the current operating condition. Multiplying the single element delay by the *encoder[6:0]* value will roughly equal the period of the clock. If this product is a multiple of the clock, then the DLL has locked onto a harmonic.

DLL Locking is controlled through the Register Interface. When the DLL is reset by asserting the *dll_rst_n* signal, the master DLL will perform a locking procedure starting with the programmed value in the *dll_start_point* field - PHY DLL Master Control Register bits[6:0]. and the current frequency of operation. The *dll_start_point* field should be programmed with a value that does not exceed a delay of 1 and 1/4 cycles of delay as calculated by estimating the worst case timing through a delay element at the highest frequency the DLL will be operating. For example, if the maximum operating frequency is 200MHz (period of 5ns) and the worst case delay element has a 85ps delay, then the dll_start_point should be programmed to 5 / 0.085 = 58.8 = 59 elements = 0x3B.

With this setting, the master DLL is guaranteed to correctly lock for all frequencies below 200MHz and in any process corner. If the delay provided by the delay line is enough to cover a full clock cycle, the master DLL is in full clock mode. In this case, the *dll_lock_value* field - DLL Observable Register Lower bits[14:8] reports the number of delay element in one full clock cycle. This number is used by the slave delay line fractional settings to determine the number of elements of delay to add to the slave delay lines. For example, if the *dll_lock_value* = 50 = 0x32, then the slave delay line percentage = 25% = 0x20, and the slave delay line delay value = 50 * 0.25 = 12.5 elements. The DLL Module will round the calculated value into integer number of delay elements.

If the frequency of operation is such that the delay line is not long enough to accommodate a full cycle of delay, the master DLL will automatically detect this and switch to half clock mode. In this mode, the master DLL attempts to lock when the delay in the delay line reaches a half-clock cycle. If lock is achieved in half clock mode, the slave delay lines are automatically adjusted to program a fractional delay of a full cycle. There is no need to change the slave delay settings based upon the lock mode of the master DLL. For example, if the *dll_lock_value* = 50 = 0x32, then the slave delay line percentage = 25% = 0x20, and the slave delay line delay value = (50*2) * 0.25 = 25 elements. It is not recommended to work in half clock mode when using loopback sampling method or SPI clock is configured to SPI Mode 3.

If the frequency of operation is such that the delay line is not long enough to capture a half-clock cycle of delay, the master DLL will indicate lock and set the number of delays to the maximum length of the delay line. This is called saturation mode. There is no need to change the slave delay settings in this mode. The slave delay settings will be fixed at the fractional delay based upon the maximum delay of the delay line times 2. For example, if the *dll_lock_value* = MAX = 128 = 0x80, and the slave delay line percentage = 25% = 0x20, then the slave delay line delay value = (128*2) * 0.25 = 64 elements.

Once locked, the master DLL will remain locked until it is reset. The master DLL will achieve an initial lock under the following conditions:

1. The *dll_bypass_mode* field - PHY DLL Master Control Register bit[23] is set to 1'b1.

2. The master DLL phase detect block indicates that the current number of delay elements in the delay line is such that the edge leaving the delay line is within the delta delay generated by the phase detect delay line.

Once one of these situations is achieved, the master DLL is considered locked.

Ideally the phase detect delay should be just wide enough so that when in full clock mode, the first flip-flop in the phase detect captures a 1 and the second phase detect flip-flop captures a 0. The phase detect should not be as wide as to create a situation where there are multiple delay values of the slave delay line that have the first flip-flop read a 0 and the second flip-flop read a 1. This would create more errors in the slave delay lines. On the other hand, if the phase detect is too narrow, the first and second phase detect flip-flops would always return the same value. This would create an oscillation between adding or subtracting multiple increments or decrements. This would also increase the error in the slave delay line settings since there would not be an accurate value of the number of delay elements in one cycle.

The master DLL locking logic includes a debug feature to observe the increment and decrement profile of the DLL lock logic over an 8 sample running window. The *lock_dec_dbg* (DLL Observable Register Lower bits[23:16]) and *lock_inc_dbg* (DLL Observable Register Lower bits[31:24]) hold the last 8 results of the master DLL phase testing. Bit 0 corresponds to the latest sample, bit 1 corresponds to the previous sample, etc. and bit 7 hold the 8th previous sample of the increment and decrement procession of the master DLL locking function. Refer to the table below for the description of these bits:

Table 4.2. DLL Locking Bit Settings

dll_lock_dec_debug	dll_lock_inc_debug	Description
0	0	The master DLL did not change the tap setting of the delay line for this sample period.
0	1	The master DLL added one tap setting to the delay line for this sample period.
1	0	The master DLL subtracted one tap setting from the delay line for this sample period.
1	1	Error condition. This combination should never occur.

The master DLL functions with the highest accuracy when the phase detect window is just large enough to cause the locking mechanism to respond with a "00" setting indicating no change in the tap setting. If the phase detect window is too small, the increment/decrement debug fields will never respond with a "00" in the same cycle. As elements are added to the phase detect logic, the *phase_detect_sel* field - PHY DLL Master Control Register bit[22:20] will increase and the increment/decrement debug fields will start responding with "00".

A debug feature is also included to provide visibility into the magnitude of the local drift of the master DLL once it is locked. The same increment/decrement debug fields can be sampled to detect N consecutive increments of decrements in an 8 sample window. The value of N is determined by the setting of the *dll_lock_num* field - PHY DLL Master Control Register bit[18:16]. Each time the increment/decrement debug fields detect N consecutive increment of decrement functions, the value of the *dll_unlock_cnt* field - DLL Observable Register Lower bits[7:3] is incremented. The *dll_lock_num value* is reset by asserting and de-asserting the *dll_reset_n* signal to the PHY. The *dll_lock_num value* will saturate at a value of 0x1F. Any time the *dll_unlock_cnt* field is triggered to increment, the current state of the *lock_inc_dbg* and *lock_dec_dbg* are stored in these fields.

4.1.2.4.1. Phase Detect Block

The Phase Detect Block is connected to the master delay line and contains the phase detect circuitry. It is important to note that the clock to the delay line and the clocks to the phase detect modules should be driven off the same clock driver and be the same net to minimize skew.

In operation, the phase detect flip-flops can be subjected to transitions which could promote metastability and therefore the primary phase detectors are protected by a second level flip-flop before the output signals phase1 and phase2 signals are sent to the control logic. Since the phase detect flip-flops are subject to potential metastability, the selection of these

flip-flops is not left to synthesis, but instead they should be hand-selected from the target library. For this reason, the RTL includes a hand-instantiated flop cell (*cdns_qspi_phy_hic_dll_sync_flop.v*) which is referenced in the RTL of the phase detector block (*cdns_qspi_phy_dll_phase_detect.v*) When selecting a flip-flop from the target library to replace the behavioural model in the *cdns_qspi_phy_hic_dll_sync_flop.v* file, the following should be considered:

- In general, faster flip-flops are more metastable resistant
- LVT flip-flops are orders of magnitude better than HVT flip-flops; HVT flip-flops should never be used
- Use flip-flops with the smallest setup-hold window
- Flip-flops without scan may have better characteristics

In the actual layout, place the phase detect flip-flops very close to each other and with the shortest amount of wiring between them. They should also be placed right next to the end of the delay line.

Figure 4.9. Phase Detect Block

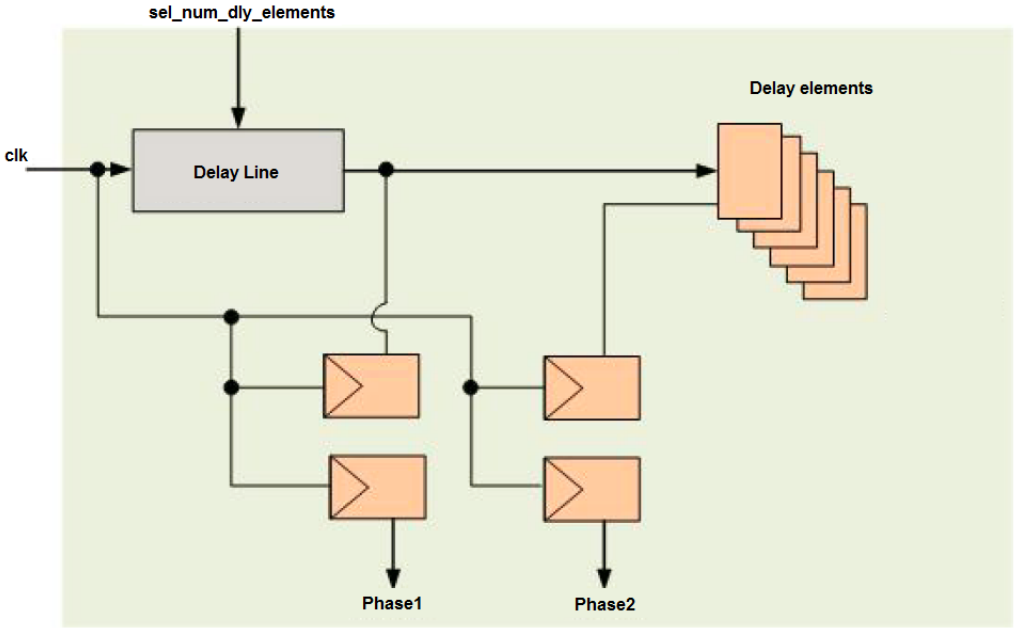


Table 4.3. Phase Detect Block Connections

Signal Name	I/O	Description
clk	I	Master clock. **Note** Clock loads in this module should originate from the same clock driver to minimize skew.
sel_num_dly_elements[127:0]	I	One-hot counter select line from the Delay Control block.
phase1	O	Phase detect signal for early clock.
phase2	O	Phase detect signal for late clock.

4.1.2.4.2. Delay Control Block

The Delay Control Block is responsible for locking on a frequency and distributing this lock value to the slave delay lines. Those lines use this information, along with the delay parameters, to set the delay for each individual delay line.

Figure 4.10. Delay Control Block

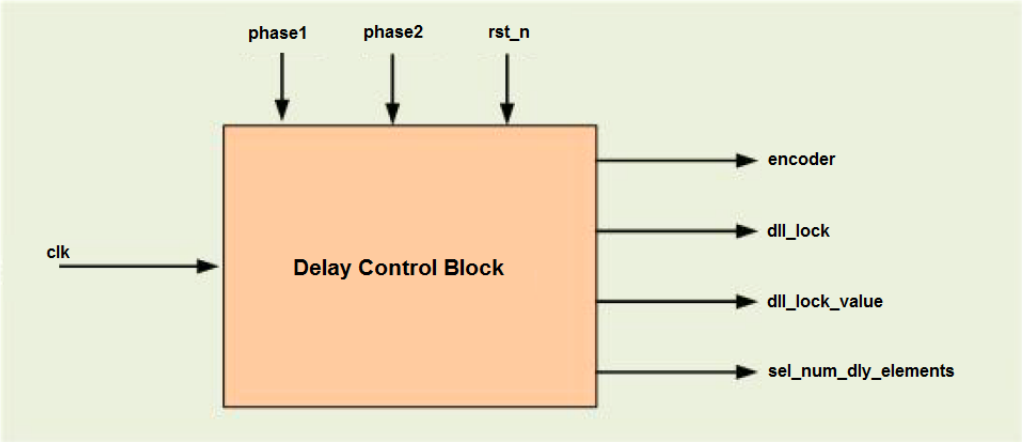


Table 4.4. Delay Control Block Connections

Signal Name	I/O	Description
clk	I	Master clock. **Note** Clock loads in this module should originate from the same clock driver to minimize skew.
rst_n	I	Active-low module reset.
phase1	I	Phase detect signal for early clock.
phase2	I	Phase detect signal for late clock.
dll_lock	O	Indicates if the DLL has achieved lock: • 0 - not locked • 1 - locked
dll_lock_value[6:0]	O	Actual value of encoder sent to the read-only parameter in the memory controller.
encoder[6:0]	O	Encoded delay output to slave delay lines.
sel_num_dly_elements[127:0]	O	One-hot counter select lines for master delay line.

4.1.2.4.3. Delay Line

The Delay Line Block is comprised of multiple delay element blocks. A breakdown of this block is shown in the figure below.

Figure 4.11. Delay Element

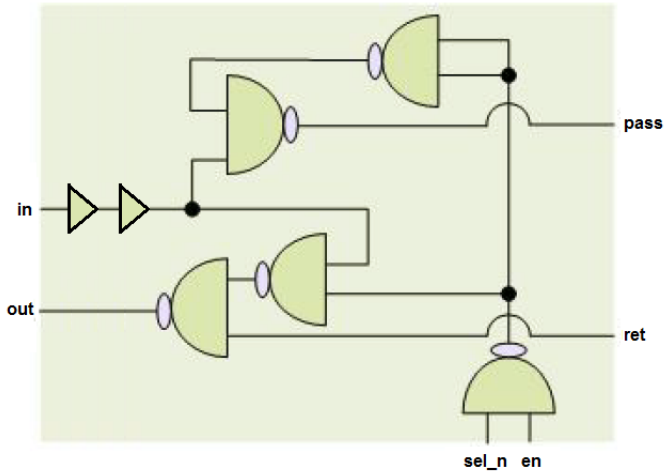


Table 4.5. Delay Element Connections

Signal Name	I/O	Description
en	I	Enable signal. Tied to the *sel_n* input from the previous delay element.
in	I	Input signal. Tied to the *pass* output from previous delay element.
ret	I	Return signal. Tied to the *out* output from *out* signal of the next delay element
sel_n	I	Affects the output signals *out* and *pass*.
out	O	Output signal. Tied to the *ret* signal of the previous delay element.
pass	O	Tied to the *in* signal of the next delay element.

The Delay Line block contains the actual delay lines for the master and slave read and write slices. Each delay line consists of identical delay elements.

Figure 4.12. Delay Line

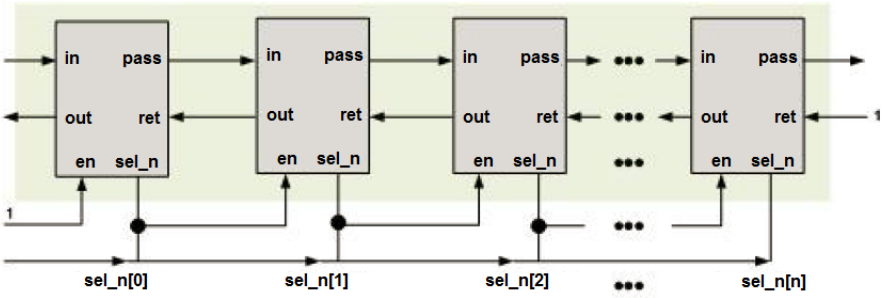


Table 4.6. Delay Line Connections

Signal Name	I/O	Description
en	I	Enable signal from previous delay element. Signal *en* [x] is the *sel_n* [x-1] signal.

Signal Name	I/O	Description
in	I	Input signal to be delayed.
sel_n[127:0]	I	One-hot counter select signal.
out	O	Resulting delayed signal.

4.2. PHY Functional Description

The PHY Module communicates with the QSPI Flash Controller via the aforementioned PHY Interface and handles data transfer on low-level stage of design hierarchy. It ensures a generally transparent integration of PHY for higher-level architecture of the Controller. However, when *ref_clk* is configured to be equal to the SPI clock instead of alternative approach using clock divider, there is just one *ref_clk* cycle (not 4 or more) within single SPI period or half period for DDR Mode (SPI Control Module works on *ref_clk*). Given that *ref_clk* is the input clock for RX FIFO and the output one for TX FIFO, the PHY solution incurs more restrictive requirement for value of system clock (*ahb_clk*) in order to synchronize data without SPI transfer interruption. For example, when the controller operates in DDR 1x QUAD Mode, 1 byte of data (equivalent to one RX FIFO location) is gathered within just single *ref_clk* cycle. The controller cannot predict next AHB access while operating in the Direct Mode (meaning its size or whether it is sequential to the previous one or not). As a result, if the AHB clock is not significantly greater than *ref_clk*, the SPI transfer has to be suspended until the Flash Command Generator forwards new data to TX FIFO.

An optional Pipeline PHY Mode was implemented to avoid the necessity of hardening *ahb_clk* for the Direct Mode when the PHY Mode is enabled and to keep maximum performance while ensuring correct operation of the Controller with the PHY using low frequencies from all its domains (what is a very power-efficient solution). This Mode is a trade-off between large software overhead when operating in the Indirect Mode and above mentioned limitations present in the Direct Mode. More detailed description of the PHY Pipeline Mode is presented in PHY Pipeline Mode section.

To ensure flexibility, it is possible to handle intermediate solution and set divider of 2. When DDR 2x Mode is granted based on configuration – SPI transfer is automatically performed using the PHY Module even if the PHY Mode enable bit is de-asserted. SDR 2x commands are handled with PHY Module paths being bypassed.

4.2.1. PHY Pipeline Mode

This Mode should be used for Direct Read Mode of operation. It is recommended to disable PHY Pipeline Mode if any other operations are intended to execute, and re-enable for subsequent Direct Reads in PHY Mode. Since there is comprehensive software mechanism controlling Read data transfers in Indirect Mode, pipeline of AHB accesses is not effective for this mode. Enable PHY Pipeline feature when more than four AHB 4-byte-sized words are predicted to Read in sequential manner. The Flash Command Generator pipelines and puts them into TX FIFO what causes keeping CS active because low-level SPI controls TX FIFO fill level. In order to correctly triggering Direct Read in Pipeline Mode TX FIFO must be empty therefore polling of IDLE bit - QSPI Configuration Register bit[31] is needed before. The sequential data transfer will be interrupted when *hsel* signal of the AHB interface is asserted to low. This information is also detected by AHB Slave Module which informs the Flash Command Generator that the next access is invalid and dummy TX FIFO locations can be unloaded transparently for the system. Polling of IDLE bit of the controller indicates completion of this operation. No AHB transaction is permitted before getting IDLE bit even if the controller returns *hready* earlier.

This Mode can be enabled when following conditions are met:

- ahb_clk >= ref_clk

  (Comparing the slow AHB clock with the fast reference one makes Pipeline Mode ineffective – Suspend of SPI Transfer would be possible. Consequently, this condition has to be met to operate in this Mode.)

- Only 4-byte sized AHB Bursts are permitted

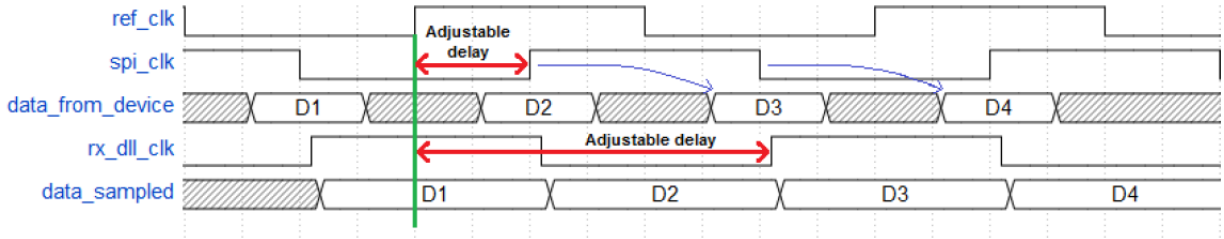
  (to ensure more AHB clock cycles for synchronization of FIFOs between consecutive *hready* pulses)

- Only sequential transfers are permitted. To execute non-sequential ones polling of IDLE bit is required between triggering them.
- Do not use Pipeline Mode along with Continuous Mode (XIP). Benefit of XIP is limited for bulk data transfers intended to execute in Pipeline Mode.

4.2.2. Read Data Capturing by the PHY Module

From the functional standpoint this mechanism is very similar to the one that uses external DLL. Read Data Capturing by the PHY Module is beneficial, as the User is no longer responsible for the design dedicated DLL being compatible with the QSPI Flash Controller. Another benefit is an option to adjust both SPI clock and sampling clock in a very wide range to fit them into individual requirements of any system. If loopback clock (Read Data Capture Register bit[0]) and PHY Mode (QSPI Configuration Register bit[3]) are both enabled, the loopback clock is driven into RX DLL instead of gated *ref_clk*. Because of the architecture of DLL, loopback clock needs to be provided in SPI Mode 0. Sampling data using PHY Module is depicted below. DDR Mode was given as an example.

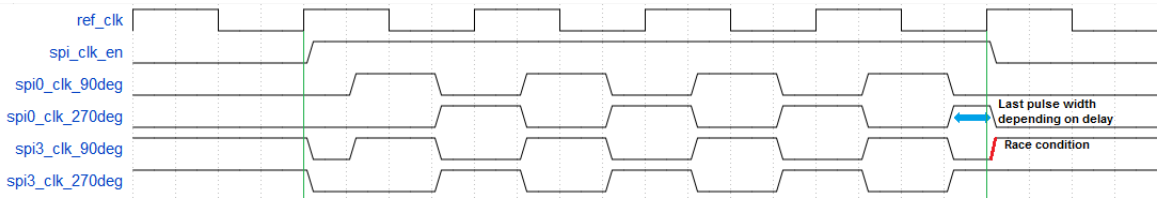
Figure 4.13. Sampling Mechanism using the PHY Module (DDR Mode)



4.2.3. SPI Mode for PHY datapath

When data path with clock divider is selected (PHY Module is bypassed), SPI clock is generated synchronously with *ref_clk*. It allows the low level SPI Control FSM generating SPI clock output directly fixed into configured SPI Mode. Conversely, when the PHY Mode is enabled and consequently SPI clock is delayed and gated variant of *ref_clk* – both are becoming asynchronous for each other. This causes some side effects which should be noticed. Let us consider four quotable scenarios as for picture below:

Figure 4.14. Various SPI clock delay scenarios depending on SPI Mode



The first one covers the case with the clock delay smaller than half cycle of *ref_clk* in SPI Mode 0 (spi0_clk_90deg). There are no limitations with this provided that clock edges hit at valid data window for actual system requirements.

The second one covers the case with the clock delay greater than half cycle of *ref_clk* and smaller than full cycle in SPI Mode 0 (spi0_clk_270deg). If the delay is getting closer full cycle of *ref_clk* the last pulse width shrinks what may drive potential timing problems on SPI clock pin. Given that reference delay of SPI clock is 90 degrees for DDR and 180 degrees for SDR, the last pulse width removal timing is not violated even if deviations caused by operating conditions would be taken into account. Delay greater than half cycle of *ref_clk* should not be set for DDR since there are two valid data locations (issued on both edges of *ref_clk*) before first edge of spi0_clk_270deg what would cause overlooking the first one.

The third one covers the case with clock delay smaller than half cycle of *ref_clk* in SPI Mode 3 (spi3_clk_90deg). SPI Mode 3 is applicable for SDR only since there is no SPI clock negedge at the end of the transfer needed to capture the last data of the transition. This scenario is not recommended to use at all. Let us consider the scenario that CSEOT delay has its default value of 0. In this case CS signal is de-asserted together with the last posedge of spi3_clk_90deg what is marked in red in the picture. Both CS and SPI clock are feeding the flash device what creates the race condition between them and possibly timing violation on memory interface. Even if CSEOT is prolonged (CS is de-asserted later), there is a redundant positive clock edge which causes writing or reading unwanted data depending on actual transfer direction.

The fourth one covers the case with clock delay greater than half cycle of *ref_clk* and smaller than full cycle in SPI Mode 3 (spi3_clk_270deg). It is ensured that with this delay range high clock value will be captured when spi_clk_en goes to low what prevents generation of redundant posedge on spi3_clk_270deg.

Generating the delay greater than full ref_clk cycle always results in error.

4.3. PHY Programmer's Guide

There is the description in Section 2.2.2, "*Configuring the QSPI Controller for optimal use*" of the way to configure the IP for optimal use after reset. Current section covers exemplary software algorithm of adapting the Controller with the PHY Module incorporated to work in QUAD 1x clock DDR Protocol and assumes that all necessary configuration steps described in Section 2.2.2, "*Configuring the QSPI Controller for optimal use*" have been completed.

1. Set PHY mode enable (QSPI Configuration Register bit[3]) and DDR protocol (QSPI Configuration Register bit[24]). It is assumed that Device is configured to work in DDR Protocol (do not confuse with DDR commands – difference is described in this document).
2. Before setting the DLL parameters, software calibration could be needed:
    a. DLL Bypass Mode (follow only if operating in this Mode)
        i. Depending on frequency of *ref_clk*, calculate how many delay elements should be used to shift this clock by 25% of its period (Best case for DDR transfers from setup/hold timings standpoint). Note that delay could be slightly different in a real design. TX Delay is configured in PHY Configuration Register bits[22:16].
        ii. Re-synchronize DLLs by asserting PHY Configuration Register bit[31] (If this bit is already set by previous re-synchronization, toggle sequence from "0" to "1" must be generated in order to trigger re-synchronization DLL logic.) and set PHY Bypass Mode Enable bit in PHY DLL Master Control Register bit[23].
    b. DLL Master Mode (follow only if operating in this Mode)
        i. Drive DLL reset bit - PHY Configuration Register bit[30] into low.
        ii. Calculate initial delay value for the Master DLL according to the PHY DLL Master Control Register bits[6:0].
        iii. Depending on frequency of *ref_clk*, calculate how many delay elements should be used to shift this clock by 25% of its period (Best case for DDR transfers from setup/hold timings standpoint). Note that delay could be slightly different in a real design. TX Delay is configured in PHY Configuration Register bits[22:16].
        iv. Re-synchronize DLLs by asserting PHY Configuration Register bit[31] (If this bit is already set by previous re-synchronization, toggle sequence from "0" to "1" must be generated in order to trigger re-synchronization DLL logic.) and set DLL reset bit back to high (Since both bits are within the same register, it is acceptable to set both bits simultaneously).
        v. Poll DLL Observable Register Lower bit[15]. When set – lock is done.
        vi. Re-synchronize Slave DLLs by asserting PHY Configuration Register bit[31] (If this bit is already set by previous re-synchronization, toggle sequence from "0" to "1" must be generated in order to trigger re-synchronization DLL logic.) and set TX DLL Delay (PHY Configuration Register bits[22:16]) and RX DLL

Delay (PHY Configuration Register bits[6:0]) fields which are equivalent to percentage clock offsets now. It is recommended to waiting for the new configuration being propagated by 20 *ref_clk* cycles before triggering the next SPI transfer.

c. Consider Read Data from location where its value is predictable. This step can be performed in different ways, depending on the device. Parameter Page, ID, Status, Data from OTP region or Data from location of Flash Array the value of which is known can act as the pattern.

d. Trigger Read request chosen from above options

e. Check correctness of data and store that information:

i. Increment value of RX clock delay – it is configurable in the PHY Configuration Register bits[6:0]

ii. Re-synchronize DLLs

iii. Trigger valid Read request

iv. Check correctness of data and store information

v. If range boundary of RX clock delay is achieved, go to step 3. Otherwise go back to point i.

3. Set RX clock delay value for one from the middle of valid range based on information in storage
4. Re-synchronize DLLs
5. Set Device Read Instruction Register Device Read Instruction Register for Quad Read DDR Configuration (each transfer phase should be configured to work in Quad Mode, Number of Dummy cycles should be set as specified in the documentation of the device or more when because of additional read paths delays of actual systems data is predicted to be flopped by PHY Module with delay excesses actual cycle of SPI clock generated by the controller).
6. Enable Pipeline Mode in the QSPI Configuration Register bit[25]
7. Perform Sequential Read of Data consistent with conditions indicated within PHY Pipeline Mode section.
8. After de-asserting *hsel* by software – poll IDLE bit - QSPI Configuration Register bit[31]
9. When it is asserted to high – next transfer request can be triggered.

4.4. PHY Programming Interface

4.4.1. PHY Configuration Register

Description: This register defines the configuration of PHY Module and controls the internal DLL. This register should be setup while the controller is idle.

Table 4.7. PHY Configuration Register

Offset	Bit	R/W	Description	Reset
0xB4	31	WO	Re-synchronisation DLL bit This bit is used for re-synchronisation delay lines to update them with values from TX DLL Delay and RX DLL Delay fields.	n/a
0xB4	30	WO	DLL Reset bit This bit is used for reset of Delay Lines by software. The reset is active when de-asserted.	n/a

Offset	Bit	R/W	Description	Reset
0xB4	29	R/W	RX DLL Bypass bit This bit is only valid when loopback feature (Read Data Capture Register bit[0]) and PHY Mode (QSPI Configuration Register bit[3]) are both enabled. If the external delay of loopback clock is sufficient to capture valid input data, no additional delay by RX DLL may be needed. Setting this bit bypasses the RX DLL for loopback clock.	1'h0
0xB4	28:23	RO	Unused bits. Read as zero.	6'h00
0xB4	22:16	R/W	TX DLL Delay This field determines the number of delay elements to insert on data path between reference clock and spi clock.	7'h00
0xB4	15:7	RO	Unused bits. Read as zero.	9'h00
0xB4	6:0	R/W	RX DLL Delay This field determines the number of delay elements to insert on data path between reference clock and sampling clock.	7'h00

4.4.2. PHY DLL Master Control Register

Description: This register defines the configuration and control logic of DLL intended to work in DLL Master Mode.

Table 4.8. PHY DLL Master Control Register

Offset	Bit	R/W	Description	Reset
0xB8	31:25	RO	Unused bits. Read as zero.	7'h00
0xB8	24	R/W	Determines if the master delay line locks on a full cycle or half cycle of delay. This bit need not be written by software. If DLL does not lock in full cycle, it will automatically try to lock in half cycle mode. Optionally, HOST can choose to direct the DLL to try for lock in half clock mode from beginning by setting this bit: 0 = Full cycle of delay 1 = Half cycle of delay This is used with low-frequency operation to reduce the delay line. In low-frequency operation, the data valid window is wider and the resolution from locking on half a cycle is sufficient.	1'h0
0xB8	23	R/W	DLL bypass mode control Controls the bypass mode of the master and slave DLLs. If this bit is set, The *dll_bypass_mode* is intended to be used only for debug. 0 = Master operational mode DLL works in normal mode of operation where the slave delay line settings are used as fractional delay of the master delay line encoder reading of the number of delays in one cycle.	1'h1

Offset	Bit	R/W	Description	Reset
			1 = Bypass mode Master DLL is disabled with only 1 delay element in its delay line. The slave delay lines decode delays in absolute delay elements rather than as fractional delays. Delays are defined in PHY Configuration Register bits[22:16] (TX delay) and PHY Configuration Register bits[6:0] (RX delay).	
0xB8	22:20	R/W	DLL Phase Detect Selector for sampling clock generation to handle the clock domain crossing between the reference clock and sampling clock. Selects the number of delay elements to be inserted between the phase detect flip-flops: 3'b000 = One delay element 3'b001 = Two delay element 3'b010 = Three delay element 3'b011 = Four delay element 3'b100 = Five delay element 3'b101 = Six delay element 3'b110 = Seven delay element 3'b111 = Eight delay element	3'h0
0xB8	19	RO	Unused bit. Read as zero.	1'h0
0xB8	18:16	R/W	Holds the number of consecutive increment or decrement indications that will trigger an unlock condition and increment the *dll_unlock_cnt* field - DLL Observable Register Lower bits[7:3] and either the *lock_dec_dbg* (DLL Observable Register Lower bits[23:16]) or *lock_inc_dbg* (DLL Observable Register Lower bits[31:24]).	3'h0
0xB8	15:7	RO	Unused bits. Read as zero.	9'h000
0xB8	6:0	R/W	This value is the initial delay value for the Master DLL. This value is also used as the increment value if the initial value is less than a half-clock cycle. This field should be set in such a way that it is not greater than 7/8 of a clock period given the worst case element delay. For example, if the frequency is 200MHz (5ns cycle time) with a worst case element 80ps delay, this this field should be set to = 5 * (7/8) / 0.080 = 54 elements. This calculation helps to determine the start point which achieves the fastest lock. However, a small value such as 0x04 may be used instead to ensure that the DLL does not lock on a harmonic. Note that with a small value like this, the initial lock time will be longer.	7'h00

4.4.3. DLL Observable Register Lower

Description: This register allows the user to observe and debug DLL status. This register can toggling dynamically. Knowing that it is synchronized into APB domain by the QSPI Flash Controller, handshaking approach was implemented.

Table 4.9. DLL Observable Register Lower

Offset	Bit	R/W	Description	Reset
0xBC	31:24	RO	Holds the state of the cumulative lock incremental steps when the *dll_unlock_cnt* field - DLL Observable Register Lower bits[7:3] of this parameter was triggered to increment or was last saturated at a value of 0x1F	8'h00
0xBC	23:16	RO	Holds the state of the cumulative lock decremental steps when the *dll_unlock_cnt* field - DLL Observable Register Lower bits[7:3] of this parameter was triggered to increment or was last saturated at a value of 0x1F	8'h00
0xBC	15	RO	This bit indicates that lock of loopback is done	1'h0
0xBC	14:8	RO	DLL Lock Value Reports the DLL encoder value from the master DLL to the slave DLLs. The slaves use this value to set up their delays for TX and RX lines. Indicates status of DLL.	7'h00
0xBC	7:3	RO	DLL Unlock Counter Reports the number of increments or decrements required for the master DLL to complete the locking process.	5'h00
0xBC	2:1	RO	DLL Locked Mode Indicates status of DLL. Defines the mode in which the DLL has achieved the lock: 2'b00 - Full clock mode. The master delay line was long enough to lock on one full clock cycle of delay. 2'b01 - Reserved. 2'b10 - Half clock mode. The master delay line was not long enough to lock one full cycle of delay but could lock on a half-cycle of delay. 2'b11 - Saturation mode. The master delay line was not long enough to lock on a full or a half clock cycle. In this mode, the encoder value is fixed at the maximum delay line setting and the master DLL will be disabled. The slave delay lines continue to use the fractional delays based upon the fixed saturation value of the delay line.	2'h0
0xBC	0	RO	DLL Lock Indicates status of DLL 0 = DLL has not locked	1'h0

Offset	Bit	R/W	Description	Reset
			1 = DLL is locked	

4.4.4. DLL Observable Register Upper

Description: This register allows the user to observe and debug DLL status. This register can toggling dynamically. Knowing that it is synchronized into APB domain by the QSPI Flash Controller, handshaking approach was implemented.

Table 4.10. DLL Observable Register Upper

Offset	Bit	R/W	Description	Reset
0xC0	31:23	RO	Unused bits. Read as zero.	9'h000
0xC0	22:16	RO	TX DLL decoder output Holds the encoded value for the TX delay line for this slice.	7'h00
0xC0	15:7	RO	Unused bits. Read as zero.	9'h000
0xC0	6:0	RO	RX DLL decoder output Holds the encoded value for the RX delay line for this slice.	7'h00

Part II. Appendixes

Table of Contents

Appendix A. Document Revison History

Table A.1. Document Revison History

Revsion	Modification	Data	Author
0.01	First Draft Release	12 Aug 2015	maciejw@cadence.com
0.02	• The Description of the Virtual Cadence Flash Memory Model was added • The Description of the STIG Memory Bank feature was added	12 Oct 2015	maciejw@cadence.com
0.03	• Some grammar/material correctnesses were introduced after internal review	6 Nov 2015	maciejw@cadence.com
0.04	• Configuration management description was refined	9 Feb 2016	maciejw@cadence.com
1.00	• Phisical Estimates were updated	18 Feb 2016	maciejw@cadence.com
1.01	• The description how to configure STIG Memory Bank depth was added	18 Mar 2016	maciejw@cadence.com

Appendix B. Cadence Virtual Memory Model

Click on the hyper link to open the appendix: cdns_virtual_flash_memory_model.pdf