Official Release

AndeShape™ ATCSPI200 Data Sheet

Document Number DS087-16

Date Issued 2016-09-20

ANDES TECHNOLOGY

Copyright Notice

Contact Information

Should you have any problems with the information contained herein, please contact Andes Technology Corporation
by email support@andestech.com
or online website https://es.andestech.com/eservice/
for support giving:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of the problem

General suggestions for improvements are welcome.

Revision History

Rev.	Revision Date	Revised Chapter-Section	Revised Content
1.6	2016-09-20	2, 3.2.2, 3.2.6, 3.2.13	1. Support 4-byte SPI address 2. Support 2MB/4MB EILM memory read port (eilm_addr width is extended to 20-bits and becomes eilm_addr[21:2])
1.5	2015-09-10	1.1, 1.3.1, 3.2.13	1. Describe the multiple address width support 2. Describe the memory read port addressable space limitation 3. Add a table to clarify the latency of memory mapped access
1.4	2015-01-14	3.2.4, 3.2.10, 5.2.1, 5.2.2	1. Fix typos 2. Update interrupt behavior descriptions
1.3	2014-10-21	Ch. 2	1. Modify the bit width of input signal eilm_addr to eilm_addr[19:2]
1.2	2014-09-23	All	1. Add new feature: SPI slave mode 2. Add an independent SPI clock source for the SPI interface. ● SCLK_1T configuration is no longer needed 3. Add an EILM memory read port 4. Add and rename I/O signals: spi_mode3_set is renamed to spi_default_mode3 add scan_enable 5. Add Chapter 6: Integration Guide
1.1	2014-05-14	1.1, 1.2, 1.3, 3.1, 3.2.13, 4.8, 4.9, 4.10	1. Add an AHB memory read port 2. Add an AHB register port 3. Add SPI Memory Access Control Register (0x50) 4. Add configuration macros for the register reset values
1.0	2014-03-20	All	Initial release

Table of Contents

List of Tables

List of Figures

Typographical Convention Index

Document Element	Font	Font Style	Size	Color
Normal text	Georgia	Normal	12	Black
`Command line, source code or file paths`	`Lucida Console`	`Normal`	`11`	`Indigo`
`VARIABLES OR PARAMETERS IN COMMAND LINE, SOURCE CODE OR FILE PATHS`	**`LUCIDA CONSOLE`**	**`BOLD + ALL-CAPS`**	**`11`**	**`INDIGO`**
Note or warning	Georgia	Normal	12	Red
Hyperlink	Georgia	Underlined	12	Blue

1. Introduction

AndeShape™ ATCSPI200 is a Serial Peripheral Interface (SPI) controller which serves as a SPI master or a SPI slave. As a SPI master, the controller connects various SPI devices. As a SPI slave, the controller responds to the master requests for data exchange.

1.1. Features

- AMBA™ AHB/APB 2.0 compliant
- Support for MSB/LSB first transfer
- Support for Direct Memory Access (DMA) data transfer
- Support for programmable SPI SCLK
- Support for memory mapped access (read-only) through AHB bus or EILM bus
- Support for SPI slave mode
- Configurable Dual and Quad I/O SPI interfaces
- Configurable TX/RX FIFO depth (The depth could be 2, 4, 8 or 16)
- Configurable programming port location on AHB/APB/EILM interfaces.

1.2. Block Diagram

Figure 1 shows the block diagram of the ATCSPI200 controller.

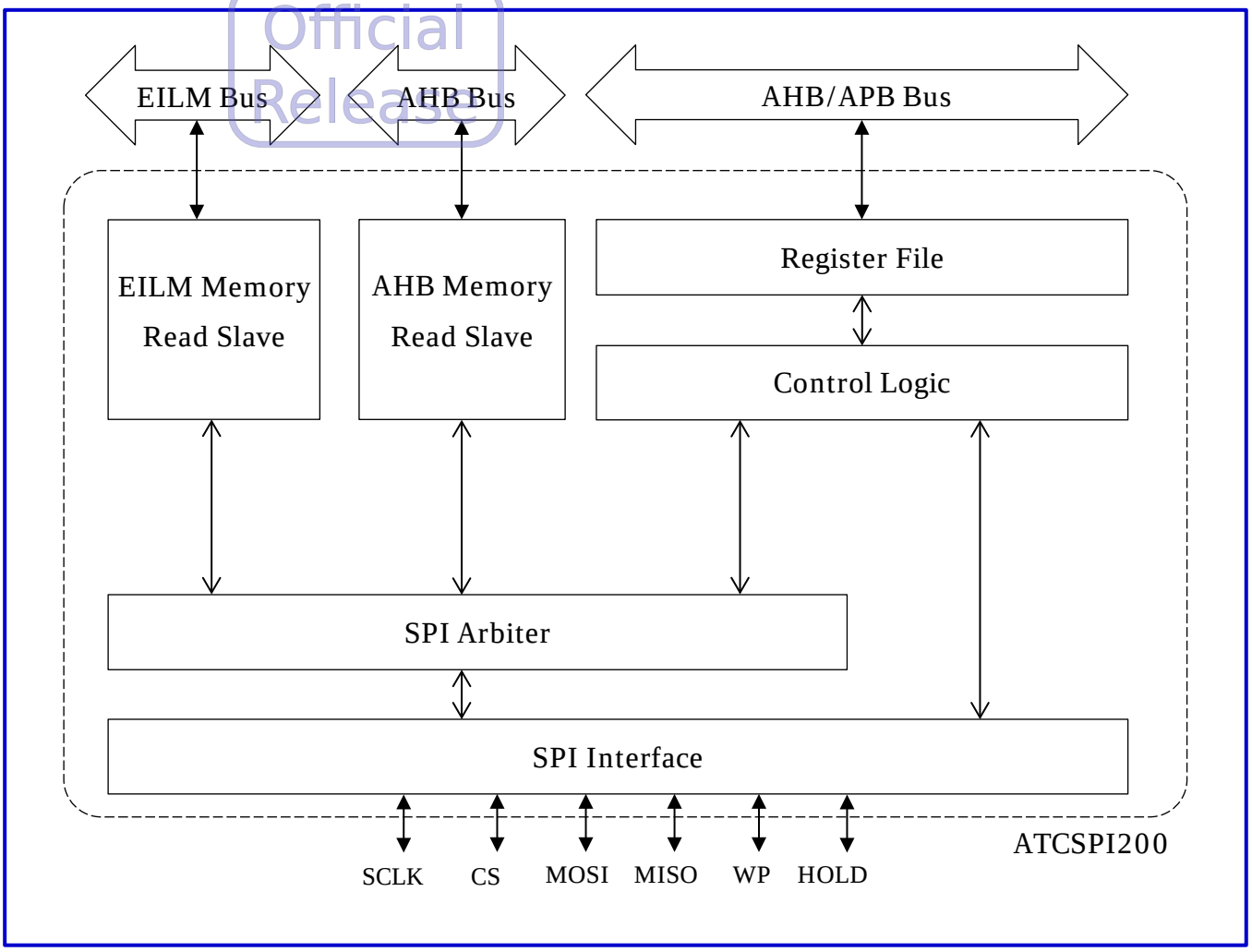

Figure 1. ATCSPI200 Block Diagram

1.3. Function Description

This section describes the SPI controller functions in four sub-sections for master mode, slave mode, dual I/O mode, and quad I/O mode. The master mode and slave mode describe the transfer initiation mechanism and the SPI frame formats. The dual mode and quad mode describe the other two transfer formats in compare to the regular mode.

1.3.1. Master Mode

The ATCSPI200 controller can act as a SPI master initiating SPI transfers on the SPI bus. The SPI transfer format and interface timing are programmable via the AHB/APB programing port. The SPI transfers are initiated through the memory mapped read access on the AHB/EILM bus or through direct register programming.

For SPI transfers initiated through the memory mapped AHB/EILM read transactions, the read transactions are translated to the SPI interface based on the read command format setup in the SPI Memory Access Control Register. The default reset value of the register is hardware configurable to support booting from SPI ROMs. The size of addressable memory mapped space depends on the interface: up-to 16MB on the AHB bus and up-to 1MB on the EILM bus.

For SPI transfers initiated by register programming, Figure 2 shows an example of SPI transfer format which includes command, address, and data phases for TX/RX data transfer. The controller provides dedicated registers to specify the contents of command, address, and data fields. The data register is shared by both TX and RX data transfers. The data transfers can be initiated through Programmed IO (PIO) or Direct Memory Access (DMA).

The ATCSPI200 controller provides TX/RX FIFO threshold interrupts to ease flow control under the PIO programming. The controller also has a programmable bit to issue an interrupt once the transfer completes.

In addition to the supported transfer format, the SPI controller allows direct control of the signals on the SPI interface. This capability enables communication with SPI devices which require special transfer formats. See the SPI Direct IO Control Register (0x14) for more information.

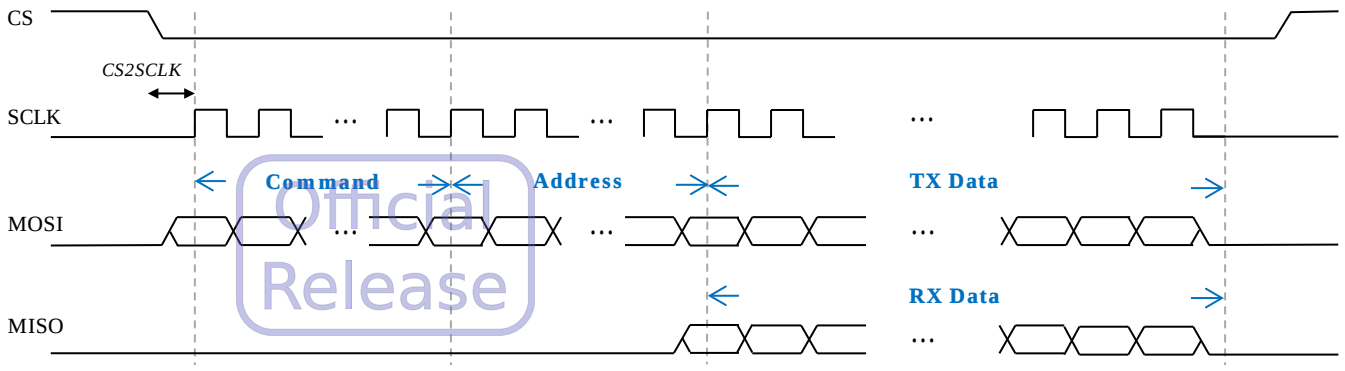

Figure 2. SPI Transfer Format

1.3.2. Slave Mode

The ATCSPI200 controller can also act as SPI slaves and accepts common commands as shown in Table 1. In addition, the controller supports user-defined commands where the slave data field format is defined by the transfer control register.

The ATCSPI200 slave interprets the packet on the SPI I/O interface as three fields: slave command, slave dummy and slave data. The slave command field and slave dummy field are always 8-bit in length. The slave data field format is defined by the received command and the Transfer Control Register.

Table 1. Supported Commands under the Slave Mode

Slave Command Name	OP Code	Slave Data
Read Status Single IO	0x05	32-bit Status
Read Status Dual IO	0x15	32-bit Status
Read Status Quad IO	0x25	32-bit Status
Read Data Single IO	0x0B	Reply data from the Data Register in the FIFO manner
Read Data Dual IO	0x0C	Reply data from the Data Register in the FIFO manner
Read Data Quad IO	0x0E	Reply data from the Data Register in the FIFO manner
Write Data Single IO	0x51	Data saved to the Data Register in the FIFO manner
Write Data Dual IO	0x52	Data saved to the Data Register in the FIFO manner

Slave Command Name	OP Code	Slave Data
Write Data Quad IO	0x54	Data saved to the Data Register in the FIFO manner
User-defined	Any 8-bit numbers other than the listed OP Codes	Depending on the Transfer Controller Register

For the status-reading commands, the slave returns the value of the Slave Status Register. The protocol format is illustrated in Figure 3.

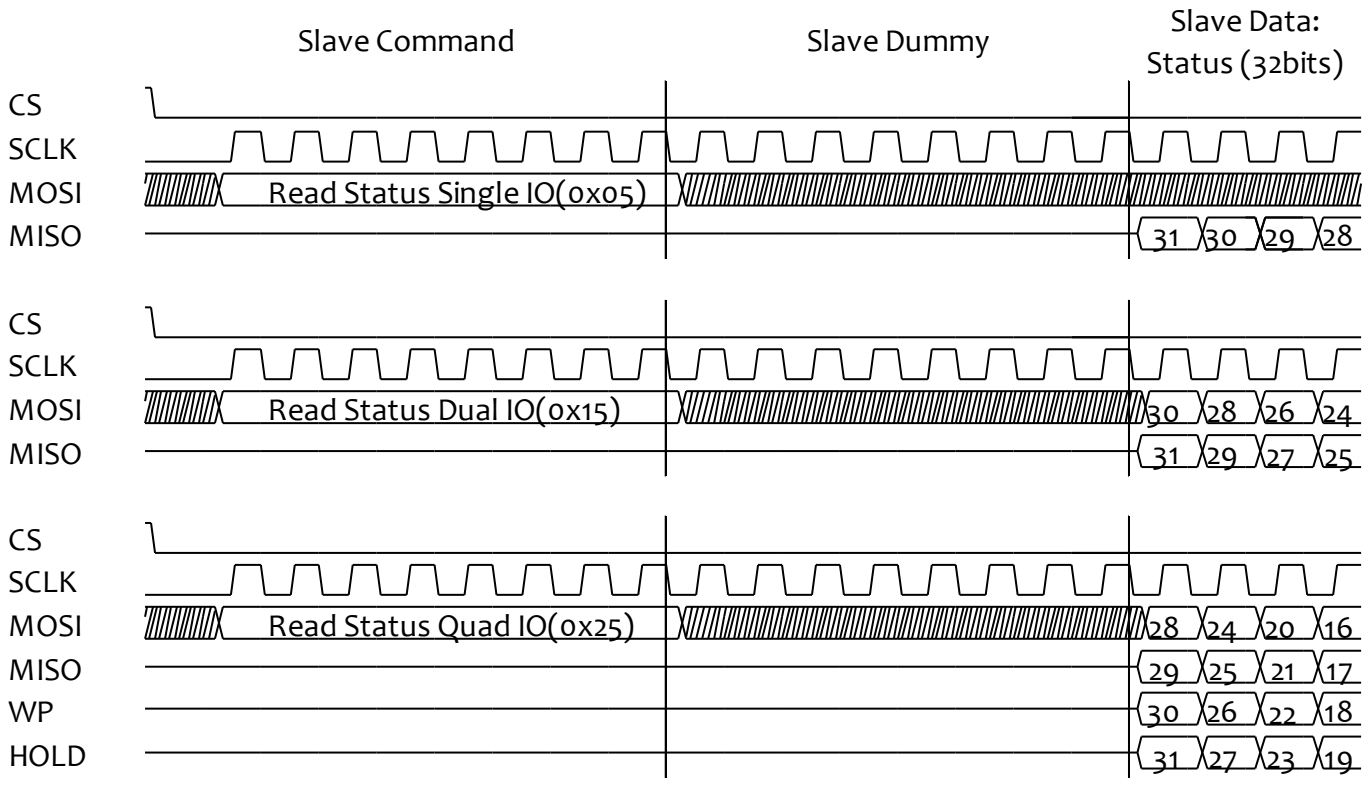

Figure 3. Timing Diagram of Status-Reading Commands (MSB First, DataMerge=0)

For the data-reading commands, the protocol formats are illustrated in Figure 4, Figure 5, and Figure 6.

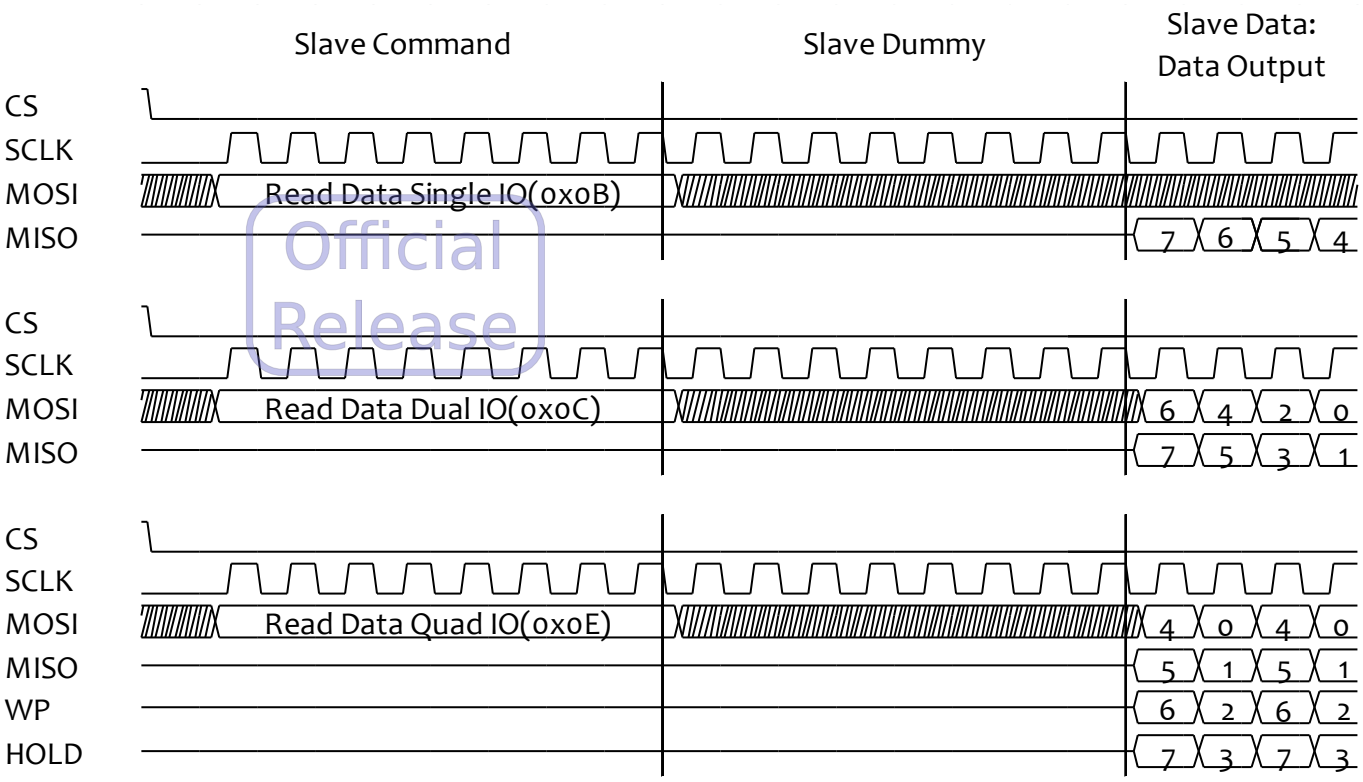

Figure 4. Timing Diagram of Data-Reading Commands (MSB First, Merge Mode)

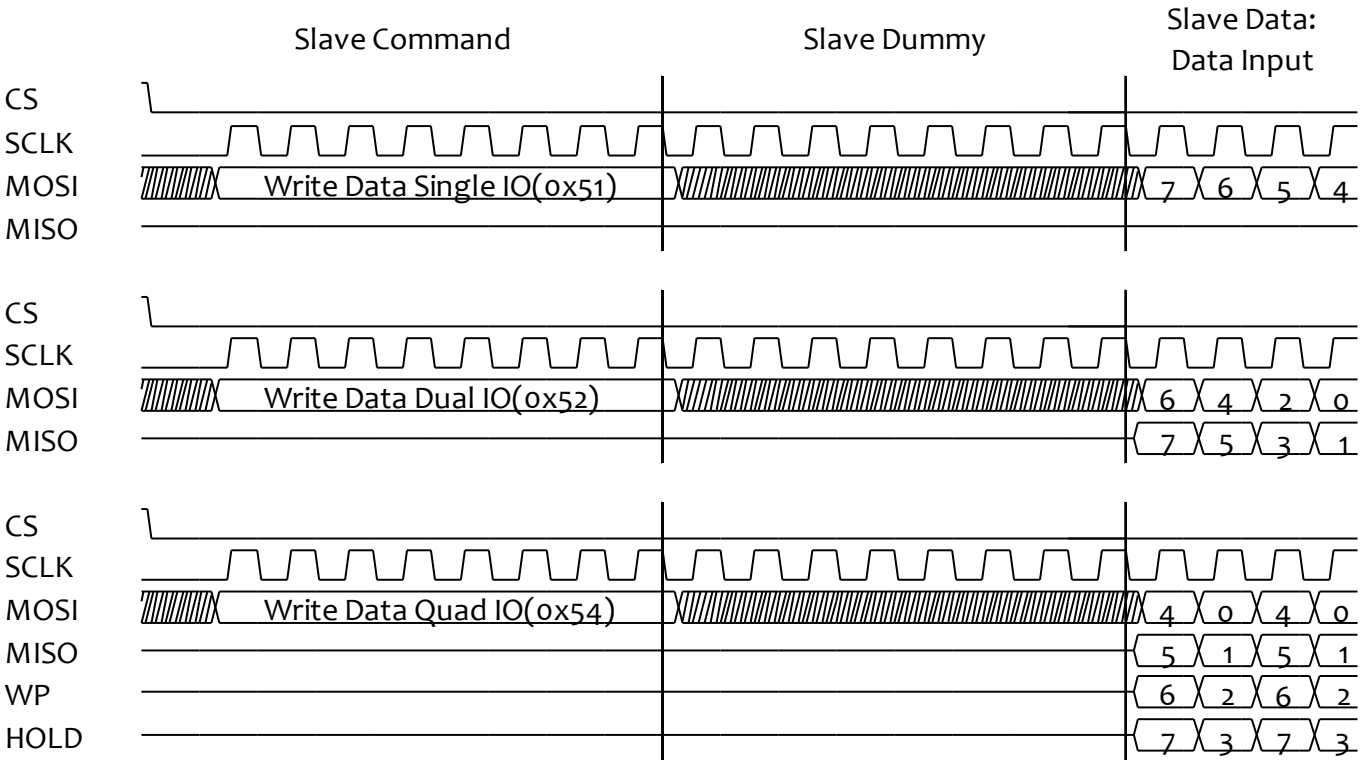

Figure 5. Timing Diagram of Data-Writing Commands (MSB First, Merge Mode)

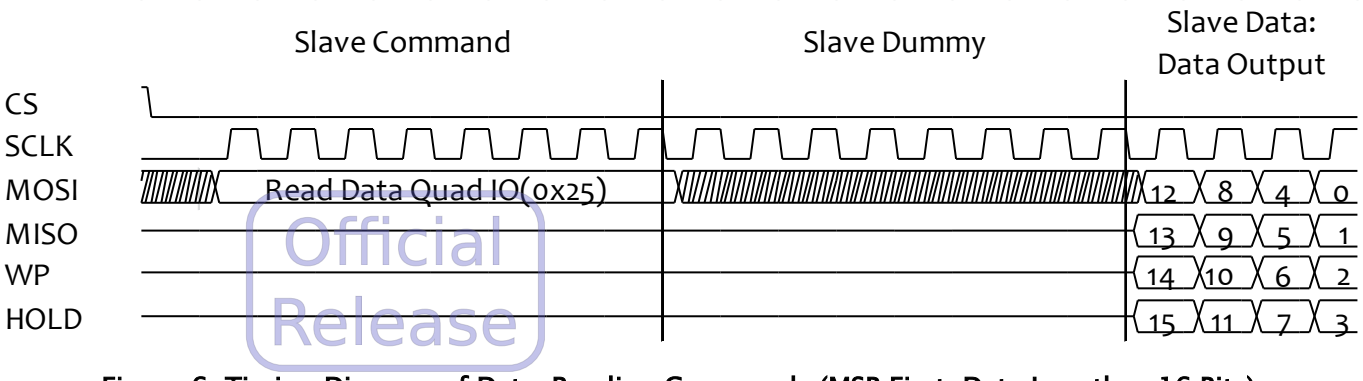

Figure 6. Timing Diagram of Data-Reading Commands (MSB First, Data Length = 16 Bits)

For the user-defined command, the slave data field format is defined by the transfer control register (0x20). For example, if the transfer mode is {Dummy, Write}, only the write field will be logged into the data register and the dummy field is dropped.

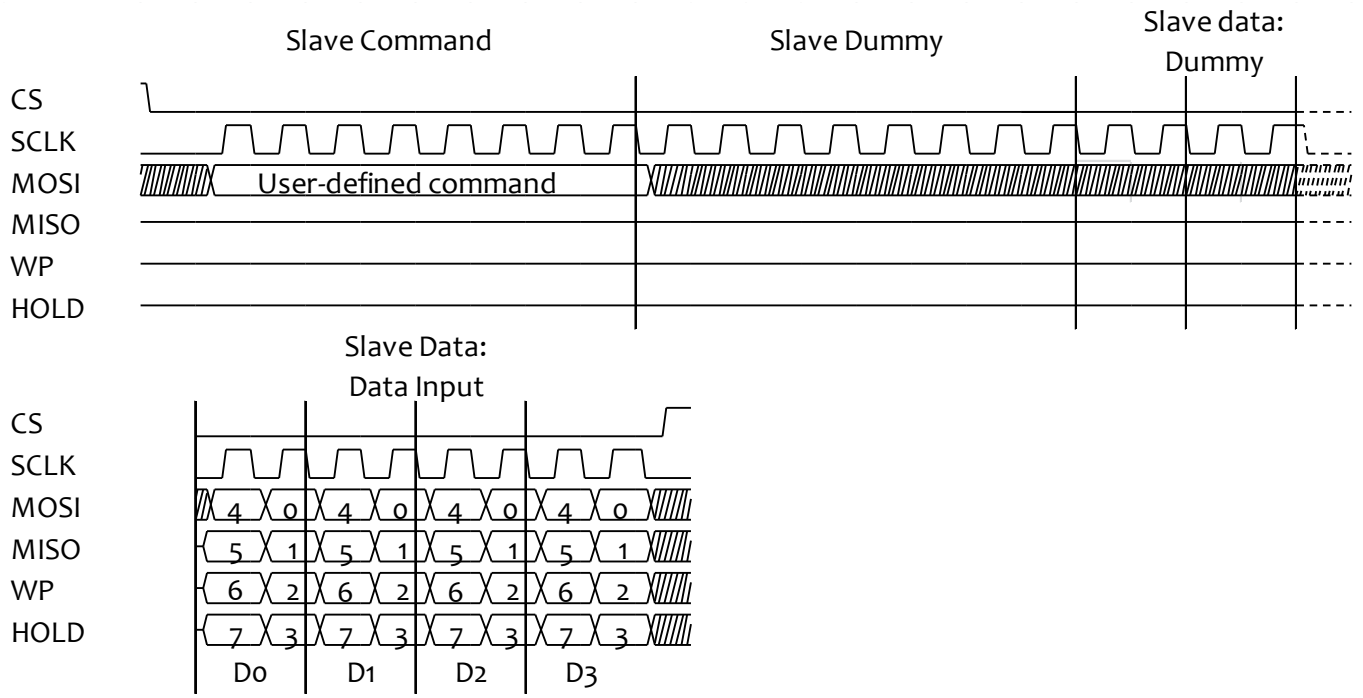

Figure 7. Timing Diagram of Slave User-Defined Command (MSB First, Merge Mode, TransMode = {Dummy, Write}, DualQuad = Quad, DummyCnt = 1, WrTranCnt = 3, Data length = 8 Bits)

1.3.3. Dual I/O Mode

The dual I/O mode doubles the SPI bandwidth by treating the master-input/slave-output (MISO) and master-output/slave-input (MOSI) signals as bidirectional wires. The SPI controller provides two transfer formats at the dual I/O mode. In one format, both the address phase and the data phase make use of the two wires (MISO and MOSI). In the other format, only the data phase makes use of the two wires. See AddrFmt and DualQuad bits in the SPI Transfer Format Register (section 3.2.4) for more information. Figure 8 shows an example of dual I/O transfer.

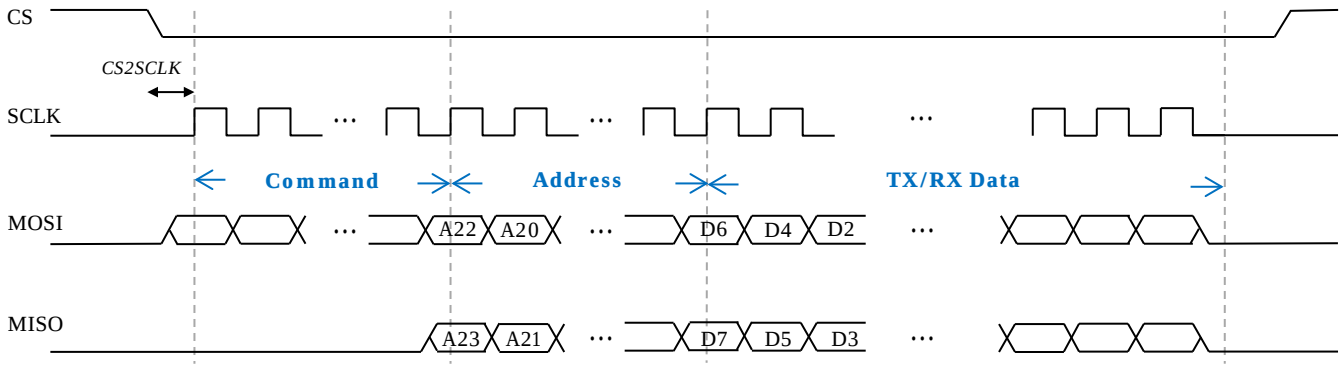

Figure 8. SPI Dual I/O Transfer (3-byte address)

1.3.4. Quad I/O Mode

The quad I/O mode quadruples the SPI bandwidth by treating the master-input/slave-output (MISO), master-output/slave-input (MOSI), write protect (WP), and HOLD signals as bidirectional wires. The SPI controller provides two transfer formats at the quad I/O mode. In one format, both the address phase and data phase make use of the four wires (MISO, MOSI, WP and HOLD). In the other format, only the data phase makes use of the four wires. See AddrFmt and DualQuad bits in the SPI Transfer Format Register (section 3.2.4) for more information. Figure 9 shows an example of quad I/O transfer.

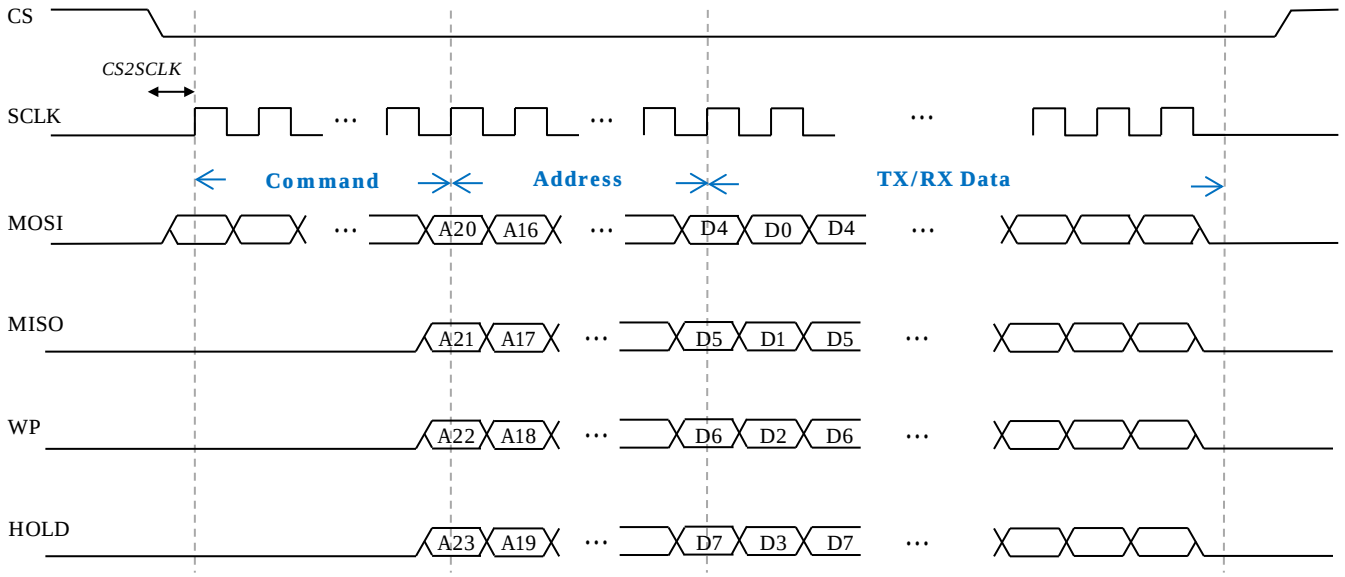

Figure 9. SPI Quad I/O Mode Transfer (3-byte address)

2. Signal Description

Table 2 gives the detailed descriptions of ATCSPI200 I/O signals.

Table 2. ATCSPI200 Signal Definition

Signal Name	I/O Type	Description
SPI Clock signals		
spi_clock	I	The clock source for the SPI interface. The SCLK of the SPI interface is generated by dividing this clock source by a programmable value.
spi_rstn	I	The reset signal for the spi_clock domain. Active Low
AHB Global Signals		
hclk	I	AHB clock
hresetn	I	AHB reset signal. Active Low
apb2ahb_clken	I	APB to AHB clock enable is an AHB domain signal indicating the valid AHB clock cycles to sample and update APB domain signals when the AHB frequency is a multiple of the APB frequency.
AHB Memory Read Port		
hsel_mem	I	AHB slave select
htrans_mem[1:0]	I	AHB transfer type
haddr_mem[N:0]	I	AHB address bus N is 23 if AHB address width is 24-bit. It is 31 if the AHB address width is 32-bit. Unused most-significant-bits must padded with zero, otherwise undetermined behavior may occur
hreadyin_mem	I	AHB bus ready
hwrite_mem	I	AHB transfer direction
hwdata_mem[31:0]	I	AHB write data bus

Signal Name	I/O Type	Description
hreadyout_mem	O	AHB slave transfer done
hresp_mem	O	AHB transfer response
hrdata_mem[31:0]	O	AHB read data bus
EILM Memory Read Port		
eilm_clk	I	EILM bus clock
ahb2eilm_clken	I	AHB to EILM clock enable is an EILM domain signal indicating the valid EILM clock cycles to sample and update AHB domain signals when the EILM frequency is a multiple of the AHB frequency.
apb2eilm_clken	I	APB to EILM clock enable is an EILM domain signal indicating the valid EILM clock cycles to sample and update APB domain signals when the EILM frequency is a multiple of the APB frequency.
eilm_resetn	I	EILM bus reset signal
eilm_req	I	EILM request
eilm_web[3:0]	I	EILM write enable
eilm_wait	O	EILM early wait state 1 = the interface should be in the wait state for the next cycle 0 = the data will be ready at the next cycle
eilm_wait_cnt[1:0]	I	Represents the static wait cycles for the CPU
eilm_wdata[31:0]	I	EILM write data
eilm_addr[21:2]	I	EILM address bus
eilm_rdata[31:0]	O	EILM read data
AHB Register Programming Port		
hsel_reg	I	AHB slave select
htrans_reg[1:0]	I	AHB transfer type
haddr_reg[N:0]	I	AHB address bus N is 23 if AHB address width is 24-bit. It is 31 if the AHB

Signal Name	I/O Type	Description
		address width is 32-bit.
hreadyin_reg	I	AHB bus ready
hwrite_reg	I	AHB transfer direction
hwdata_reg[31:0]	I	AHB write data bus
hreadyout_reg	O	AHB slave transfer done
hresp_reg	O	AHB transfer response
hrdata_reg[31:0]	O	AHB read data bus
APB Register Programming Port		
pclk	I	APB clock (must be synchronous with the AHB/EILM clock)
presetn	I	APB reset signal. Active Low
paddr[31:0]	I	APB address bus
penable	I	APB enable signal
pwrite	I	APB write signal
psel	I	APB select signal
pwdata[31:0]	I	APB write data bus
prdata[31:0]	O	APB read data
SPI Interface		
spi_sclk_in	I	Status of the SPI SCLK signal
spi_cs_n_in	I	Status of the SPI CS (chip select) signal
spi_miso_in	I	Status of the SPI master-input/slave-output (MISO) signal
spi_mosi_in	I	Status of the SPI master-output/slave-input (MOSI) signal
spi_wp_n_in	I	Status of the SPI Flash write protect signal
spi_hold_n_in	I	Status of the SPI Flash hold signal
spi_sclk_out	O	Output for the SPI SCLK signal
spi_cs_n_out	O	Output for the SPI CS signal
spi_miso_out	O	Output for the SPI MISO signal
spi_mosi_out	O	Output for the SPI MOSI signal

Signal Name	I/O Type	Description
spi_wp_n_out	O	Output for the SPI Flash write protect signal
spi_hold_n_out	O	Output for the SPI Flash hold signal
spi_sclk_oe	O	Output enable for the SPI SCLK signal
spi_cs_n_oe	O	Output enable for the SPI CS signal
spi_miso_oe	O	Output enable for the SPI MISO signal
spi_mosi_oe	O	Output enable for the SPI MOSI signal
spi_wp_n_oe	O	Output enable for the SPI Flash write protect signal
spi_hold_n_oe	O	Output enable for the SPI Flash hold signal
Miscellaneous		
spi_default_mode3	I	Select the default SPI mode 0x0: Both default values of CPOL and CPHA are 0x0 0x1: Both default values of CPOL and CPHA are 0x1
spi_default_as_slave	I	Set the ATCSPI200 controller as a slave after reset 0x0: Default as a master 0x1: Default as a slave
spi_tx_dma_ack	I	TX FIFO DMA acknowledge
spi_rx_dma_ack	I	RX FIFO DMA acknowledge
spi_tx_dma_req	O	TX FIFO DMA request
spi_rx_dma_req	O	RX FIFO DMA request
spi_boot_intr	O	The SPI controller interrupt
scan_enable	I	Scan enable (active high during ATPG scan/shift phase)
scan_test	I	Scan test mode (active high during ATPG test)

3. Programming Model

3.1. Summary of Registers

Table 3 shows a summary of the ATCSPI200 registers.

Table 3. ATCSPI200 Register Summary

Offset	Name	Description
0x00	IdRev	ID and Revision Register
0x04~0x0C	-	Reserved
0x10	TransFmt	SPI Transfer Format Register
0x14	DirectIO	SPI Direct IO Control Register
0x18~0x1C		Reserved
0x20	TransCtrl	SPI Transfer Control Register
0x24	Cmd	SPI Command Register
0x28	Addr	SPI Address Register
0x2C	Data	SPI Data Register
0x30	Ctrl	SPI Control Register
0x34	Status	SPI Status Register
0x38	IntrEn	SPI Interrupt Enable Register
0x3C	IntrSt	SPI Interrupt Status Register
0x40	Timing	SPI Interface Timing Register
0x44 ~ 0x4C		Reserved
0x50	MemCtrl	SPI Memory Access Control Register
0x54 ~ 0x5c		Reserved
0x60	SlvSt	SPI Slave Status Register (Exists only when ATCSPI200_SLAVE_SUPPORT is defined)
0x64	SlvDataCnt	SPI Slave Data Count Register (Exists only when ATCSPI200_SLAVE_SUPPORT is defined)
0x68~0x78		Reserved
0x7C	Config	Configuration Register

3.2. Register Description

The abbreviations for the Type column are summarized below.

RO: read only

RW: readable and writable

W1C: readable and write 1 clear

3.2.1. ID and Revision Register (0x00)

This register holds the ID number and the revision number. The reset values of the two revision fields are revision dependent

Table 4. ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:12	RO	ID number for ATCSPI200	0x02002
RevMajor	11:4	RO	Major revision number	Revision dependent
RevMinor	3:0	RO	Minor revision number	Revision dependent

3.2.2. SPI Transfer Format Register (0x10)

This register defines the SPI transfer format.

Table 5. SPI Transfer Format Register

Name	Bit	Type	Description	Reset
Reserved	31:18	-	-	-
AddrLen	17:16	RW	Address length in bytes 0x0: 1 bytes 0x1: 2 bytes 0x2: 3 bytes 0x3: 4 bytes	0x2
Reserved	15:13	-	-	-
DataLen	12:8	RW	The length of each data unit in bits. The actual bit number of a data unit is (DataLen + 1)	0x07
DataMerge	7	RW	Enable Data Merge mode, which does automatic data split on write and data coaleasing on read. This bit only takes effect when DataLen = 0x7.Under Data Merge mode, each write to the Data Register will transmit all fourbytes of the write data; each read from the Data Register will retrieve four bytes of received data as a single word data. When Data Merge mode is disabled, only the least (DataLen+1) significient bits of the Data Register are valid for read/write operations; no automatic data split/coaleasing will be performed.	0x1
Reserved	6:5	-	-	-
MOSIBiDir	4	RW	Bi-directional MOSI in regular (single) mode. 0x0: MOSI is uni-directional signal in regular mode. 0x1: MOSI is bi-directional signal in regular mode. This bi-directional signal replaces the two	0x0

Name	Bit	Type	Description	Reset
			uni-directional data signals, MOSI and MISO.	
LSB	3	RW	Transfer data with least significant bit first. 0x0: Most significant bit first 0x1: Least significant bit first	0x0
SlvMode	2	RW	SPI Master/Slave mode selection 0x0: Master mode 0x1: Slave mode (Exist only when configuration ATCSPI200_SLAVE_SUPPORT is defined)	Depends on the input pin spi_default_as_slave
CPOL	1	RW	SPI Clock Polarity 0x0: SCLK is LOW in the idle states 0x1: SCLK is HIGH in the idle states	Depends on the input pin spi_default_mode3
CPHA	0	RW	SPI Clock Phase 0x0: Sampling data at odd SCLK edges 0x1: Sampling data at even SCLK edges	Depends on the input pin spi_default_mod3

3.2.3. SPI Direct IO Control Register (0x14)

This register enables the direct control of the SPI interface signals. The register is valid only when the configuration ATCSPI200_DIRECT_IO_SUPPORT is defined.

Table 6. SPI Direct IO Control Register

Name	Bit	Type	Description	Reset
Reserved	31:25	-	-	-
DirectIOEn	24	RW	Enable Direct IO 0x0: Disable 0x1: Enable	0x0
Reserved	23:22	-	-	-
HOLD_OE	21	RW	Output enable for the SPI Flash hold signal	0x0
WP_OE	20	RW	Output enable for the SPI Flash write protect signal	0x0
MISO_OE	19	RW	Output enable fo the SPI MISO signal	0x0
MOSI_OE	18	RW	Output enable for the SPI MOSI signal	0x0
SCLK_OE	17	RW	Output enable for the SPI SCLK signal	0x0
CS_OE	16	RW	Output enable for SPI CS (chip select) signal	0x0
Reserved	15:14	-	-	-
HOLD_O	13	RW	Output value for the SPI Flash hold signal	0x1
WP_O	12	RW	Output value for the SPI Flash write protect signal	0x1
MISO_O	11	RW	Output value for the SPI MISO signal	0x0
MOSI_O	10	RW	Output value for the SPI MOSI signal	0x0
SCLK_O	9	RW	Output value for the SPI SCLK signal	0x0
CS_O	8	RW	Output value for the SPI CS (chip select) signal	0x1
Reserved	7:6	-	-	-
HOLD_I	5	RO	Status of the SPI Flash hold signal	Depends on the status of the spi_hold_n_in pin
WP_I	4	RO	Status of the SPI Flash write protect signal	Depends on the

Name	Bit	Type	Description	Reset
				status of the spi_wp_n_in pin
MISO_I	3	RO	Status of the SPI MISO signal	Depends on the status of the spi_miso_in pin
MOSI_I	2	RO	Status of the SPI MOSI signal	Depends on the status of the spi_mosi_in pin
SCLK_I	1	RO	Status of the SPI SCLK signal	Depends on the status of the spi_clk_in pin
CS_I	0	RO	Status of the SPI CS (chip select) signal	Depends on the status of the spi_cs_n_in pin

3.2.4. SPI Transfer Control Register (0x20)

This register controls aspects of SPI transfers. Please see section 0 for starting a SPI transfer.

Table 7. SPI Transfer Control Register

Name	Bit	Type	Description	Reset
CmdEn	30	RW	The SPI command phase enable 0x0: Disable the command phase 0x1: Enable the command phase (Master mode only)	0x0
AddrEn	29	RW	The SPI address phase enable 0x0: Disable the address phase 0x1: Enable the address phase (Master mode only)	0x0
AddrFmt	28	RW	The SPI address phase format 0x0: Address phase is the regular (single) mode 0x1: The format of the address phase is the same as the data phase (DualQuad). (Master mode only)	0x0
TransMode	27:24	RW	The transfer mode The transfer sequence could be 0x0: Write and read at the same time 0x1: Write only 0x2: Read only 0x3: Write, Read 0x4: Read, Write 0x5: Write, Dummy, Read 0x6: Read, Dummy, Write 0x7: None Data (must enable CmdEn or AddrEn in Master mode) 0x8: Dummy, Write 0x9: Dummy, Read 0xa~0xf: Reserved	0x0

Name	Bit	Type	Description	Reset
DualQuad	23:22	RW	The SPI data phase format 0x0: Regular (Single) mode 0x1: Dual I/O mode 0x2: Quad I/O mode 0x3: Reserved	0x0
TokenEn	21	RW	Append an one-byte special token following the address phase for SPI read transfers. The value of the special token should be selected in the TokenValue. 0x0: Disable the one-byte special token 0x1: Enable the one-byte special token (Master mode only)	0x0
WrTranCnt	20:12	RW	Transfer count for write data WrTranCnt indicates the number of units of data to be transmitted to the SPI bus from the Data Register. The actual transfer count is (WrTranCnt+1). WrTranCnt only takes effect when TransMode is 0, 1, 3, 4, 5, 6 or 8. The size (bit-width) of a data unit is defined by the DataLen field of the Transfer Format Register. For TransMode 0, WrTranCnt must be equal to RdTranCnt.	0x0
TokenValue	11	RW	The value of the one-byte special token following the address phase for SPI read transfers. 0x0: token value = 0x00 0x1: token value = 0x69 (Master mode only)	0x0

Name	Bit	Type	Description	Reset
DummyCnt	10:9	RW	Dummy data count. The actual dummy count is (DummyCnt +1). The number of dummy cycles on the SPI interface will be (DummyCnt+1)* ((DataLen+1)/SPI IO width) The Data pins are put into the high impedance during the dummy data phase. DummyCnt is only used for TransMode 5, 6, 8 and 9, which has dummy data phases. Following table shows dummy cycle settings under some common transfer formats:	0x0

DummyCnt +1	DataLen+1	DualQuad	#Dummy Cycles on the SPI Interface
1	8	Regular/ Single	8
1	8	Dual	4
1	8	Quad	2
2	8	Quad	4
3	8	Quad	6
1	32	Quad	8

Name	Bit	Type	Description	Reset
RdTranCnt	8:0	RW	Transfer count for read data RdTranCnt indicates the number of units of data to be received from SPI bus and stored to the Date Register. The actual received count is (RdTranCnt+1). RdTransCnt only takes effect when TransMode is 0, 2, 3, 4, 5, 6 or 9. The size (bit-width) of a data unit is defined by the DataLen field of the Transfer Format Register. For TransMode 0, WrTranCnt must be equal to RdTranCnt.	0x0

3.2.5. SPI Command Register (0x24)

Write operations on this register trigger SPI transfers. This register must be written with a dummy value to start a SPI transfer even when the command phase is not enabled. When the ATCSPI200 controller is programmed to the slave mode, the command field of the last received SPI transaction is stored in this SPI Command Register.

Table 8. SPI Command Register

Name	Bit	Type	Description	Reset
Reserved	31:8	-	-	-
CMD	7:0	RW	SPI Command	0x0

3.2.6. SPI Address Register (0x28)

Table 9. SPI Address Register

Name	Bit	Type	Description	Reset
ADDR	31:0	RW	SPI Address (Master mode only)	0x0

3.2.7. SPI Data Register (0x2C)

When the controller is in the data merge mode, the byte endian of the SPI Data Register is little endian.

Table 10. SPI Data Register

Name	Bit	Type	Description	Reset
DATA	31:0	RW	Data to transmit or the received data For writes, data is enqueued to the TX FIFO. The least significant byte is always transmitted first. For reads, data is read and dequeued from the RX FIFO. The least significant byte is the first received byte. The FIFOs decouple the speed of the SPI transfers and the software's generation/consumption of data. When the TX FIFO is empty, SPI transfers will hold until more data is written to the TX FIFO; when the RX FIFO is full, SPI transfers will hold until there is more room in the RX FIFO. If more data is written to the TX FIFO than the write transfer count (WrTranCnt), the remaining data will stay in the TX FIFO for the next transfer or until the TX FIFO is reset.	0x0

3.2.8. SPI Control Register (0x30)

Table 11. SPI Control Register

Name	Bit	Type	Description	Reset
Reserved	31:21	-	-	-
TXTHRES	20:16	RW	Transmit (TX) FIFO Threshold The TXFIFOInt interrupt or DMA request would be issued to replenish the TX FIFO when the TX data count is less than or equal to the TX FIFO threshold.	0x0
Reserved	15:13	-	-	-
RXTHRES	12:8	RW	Receive (RX) FIFO Threshold The RXFIFOInt interrupt or DMA request would be issued for consuming the RX FIFO when the RX data count is more than or equal to the RX FIFO threshold.	0x0
Reserved	7:5	-	-	-
TXDMAEN	4	RW	TX DMA enable	0x0
RXDMAEN	3	RW	RX DMA enable	0x0
TXFIFORST	2	RW	Transmit FIFO reset Write 1 to reset. It is cleared to 0 after the reset operation completes.	0x0
RXFIFORST	1	RW	Receive FIFO reset Write 1 to reset. It is cleared to 0 after the reset operation completes.	0x0
SPIRST	0	RW	SPI reset Write 1 to reset. It is cleared to 0 after the reset operation completes.	0x0

3.2.9. SPI Status Register (0x34)

Table 12. SPI Status Register

Name	Bit	Type	Description	Reset
Reserved	31:24	-	-	-
TXFULL	23	RO	Transmit FIFO Full flag	0x0
TXEMPTY	22	RO	Transmit FIFO Empty flag	0x1
Reserved	21	-	-	-
TXNUM	20:16	RO	Number of valid entries in the Transmit FIFO	0x0
RXFULL	15	RO	Receive FIFO Full flag	0x0
RXEMPTY	14	RO	Receive FIFO Empty flag	0x1
Reserved	13	-	-	-
RXNUM	12:8	RO	Number of valid entries in the Receive FIFO	0x0
Reserved	7:1	-	-	-
SPIActive	0	RO	SPI direct register programming is in progress	0x0

3.2.10. SPI Interrupt Enable Register (0x38)

Table 13. SPI Interrupt Enable Register

Name	Bit	Type	Description	Reset
Reserved	31:6	-	-	-
SlvCmdEn	5	RW	Enable the Slave Command Interrupt. Control whether interrupts are triggered whenever slave commands are received in the Slave mode. (Slave mode only)	0x0
EndIntEn	4	RW	Enable the End of SPI Transfer interrupt. Control whether interrupts are triggered when SPI transfers end. (In slave mode, end of read status transaction doesn't trigger this interrupt)	0x0
TXFIFOIntEn	3	RW	Enable the SPI Transmit FIFO Threshold interrupt. Control whether interrupts are triggered when the valid entries are less than or equal to the TX FIFO threshold.	0x0
RXFIFOIntEn	2	RW	Enable the SPI Receive FIFO Threshold interrupt. Control whether interrupts are triggered when the valid entries are greater than or equal to the RX FIFO threshold.	0x0
TXFIFOURIntEn	1	RW	Enable the SPI Transmit FIFO Underrun interrupt. Control whether interrupts are triggered when the Transmit FIFO run out of data. (Slave mode only)	0x0
RXFIFOORIntEn	0	RW	Enable the SPI Receive FIFO Overrun interrupt. Control whether interrupts are triggered when the Receive FIFO overflows. (Slave mode only)	0x0

3.2.11. SPI Interrupt Status Register (0x3C)

Table 14. SPI Interrupt Status Register

Name	Bit	Type	Description	Reset
Reserved	31:6	-	-	-
SlvCmdInt	5	W1C	Slave Command Interrupt. This bit is set when Slave Command interrupts occur. (Slave mode only)	0x0
EndInt	4	W1C	End of SPI Transfer interrupt. This bit is set when End of SPI Transfer interrupts occur.	0x0
TXFIFOInt	3	W1C	TX FIFO Threshold interrupt. This bit is set when TX FIFO Threshold interrupts occur.	0x0
RXFIFOInt	2	W1C	RX FIFO Threshold interrupt. This bit is set when RX FIFO Threshold interrupts occur.	0x0
TXFIFOURInt	1	W1C	TX FIFO Underrun interrupt. This bit is set when TX FIFO Underrun interrupts occur. (Slave mode only)	0x0
RXFIFOORInt	0	W1C	RX FIFO Overrun interrupt. This bit is set when RX FIFO Overrun interrupts occur. (Slave mode only)	0x0

3.2.12. SPI Interface Timing Register (0x40)

This register controls the SPI interface timing for satisfying the SPI Slave interface timing requirements. Only the master needs program this register.

Table 15. SPI Interface Timing Register

Name	Bit	Type	Description	Reset
Reserved	31:14	-	-	-
CS2SCLK	13:12	RW	The minimum time between the edges of SPI CS and the edges of SCLK. The actual duration is $\frac{\text{SCLK period}}{2} \times (\text{CS2SCLK} + 1)$	Configuration dependent
CSHT	11:8	RW	The minimum time that SPI CS should stay HIGH. The actual duration is $\frac{\text{SCLK period}}{2} \times (\text{CSHT} + 1)$	Configuration dependent
SCLK_DIV	7:0	RW	The clock frequency ratio between the clock source and SPI interface SCLK. SCLK period = $\left((\text{SCLK\_DIV}+1) \times 2\right) \times (\text{Period of the SPI clock s}$ The SCLK_DIV value 0xff is a special value which indicates that the SCLK frequency should be the same as the SPI source clock frequency.	Configuration dependent

3.2.13. SPI Memory Access Control Register (0x50)

This register defines the SPI command format for issuing the memory mapped AHB/EILM read access. The memory mapped AHB/EILM read accesses should be stopped prior to programming this register or the SPI Interface Timing Register (0x40). The AHB/EILM accesses could be resumed when MemCtrlChg bit is cleared.

Table 16. SPI Memory Access Control Register

Name	Bit	Type	Description	Reset
Reserved	31:9	-	-	-
MemCtrlChg	8	RO	This bit is set when this register (0x50) or the SPI Interface Timing Register (0x40) is written; it is automatically cleared when the new programming takes effect.	0
Reserved	7:4	-	-	-
MemRdCmd	3:0	RW	Selects the SPI command format when serving the memory mapped reads on the AHB/EILM bus The command encoding table is listed in Table 17. The latency of each command is listed in Table 18.	Configuration dependent

Table 17. SPI Read Command Format for Supporting Memory Mapped AHB/EILM Read

MemRdCmd	Command	Address	Dummy	Data
0	0x03	3 bytes in Regular mode	NA	Regular mode
1	0x0B	3 bytes in Regular mode	1 byte in Regular mode	Regular mode
2	0x3B	3 bytes in Regular mode	1 byte in Regular mode	Dual mode
3	0x6B	3 bytes in Regular mode	1 byte in Regular mode	Quad mode
4	0xBB	(3-byte address + 1-byte 0) in Dual mode	NA	Dual mode
5	0xEB	(3-byte address + 1-byte 0) in Quad mode	2 bytes in Quad mode	Quad mode

MemRdCmd	Command	Address	Dummy	Data
6–7	Reserved	-	-	-
8	0x13	4 bytes in Regular mode	NA	Regular mode
9	0x0C	4 bytes in Regular mode	1 byte in Regular mode	Regular mode
10	0x3C	4 bytes in Regular mode	1 byte in Regular mode	Dual mode
11	0x6C	4 bytes in Regular mode	1 byte in Regular mode	Quad mode
12	0xBC	(4-byte address + 1-byte 0) in Dual mode	NA	Dual mode
13	0xEC	(4-byte address + 1-byte 0) in Quad mode	2 bytes in Quad mode	Quad mode
14-15	Reserved			

Table 18. Latency of a 4 Bytes Data Transfer through the AHB/EILM Memory Read Port

Command	Non-sequential	Sequential	Sequential (prefetched*)
0x03	8 BUS_CLK + 10 SPI_CLK + 64 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x0B	8 BUS_CLK + 10 SPI_CLK + 72 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x3B	8 BUS_CLK + 10 SPI_CLK + 56 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0x6B	8 BUS_CLK + 10 SPI_CLK + 48 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK
0xBB	8 BUS_CLK + 10 SPI_CLK + 40 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0xEB	8 BUS_CLK + 10 SPI_CLK + 28 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK
0x13	8 BUS_CLK + 10 SPI_CLK + 72 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x0C	8 BUS_CLK + 10 SPI_CLK + 80 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x3C	8 BUS_CLK + 10 SPI_CLK + 64 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0x6C	8 BUS_CLK + 10 SPI_CLK + 56 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK
0xBC	8 BUS_CLK + 10 SPI_CLK + 44 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0xEC	8 BUS_CLK + 10 SPI_CLK + 30 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK

BUS_CLK: bus (AHB or EILM) clock cycle; SPI_CLK: spi_clock clock cycle; SCLK: SCLK clock cycle

* The memory mapped accesses are prefetched to speedup sequential access.

3.2.14. SPI Slave Status Register (0x60)

The Slave Status Register keeps slave statuses. An SPI master can get these statuses by issuing status-reading commands.

Table 19. SPI Slave Status Register

Name	Bit	Type	Description	Reset
Reserved	31:19	-	-	-
UnderRun	18	W1C	Data underrun occurs in the last transaction	0
OverRun	17	W1C	Data overrun occurs in the last transaction	0
Ready	16	RW	Set this bit to indicate that the ATCSPI200 is ready for data transaction. When an SPI transaction other than slave status-reading command ends, this bit will be cleared to 0.	0
USR_Status	15:0	RW	User defined status flags	0

3.2.15. SPI Slave Data Count Register (0x64)

This register shows the data count of the read/write transactions in the slave mode. You should access the data register based on the data count information.

Table 20. SPI Slave Data Count Register

Name	Bit	Type	Description	Reset
Reserved	31:25	-	-	
WCnt	24:16	RO	Slave transmitted data count	0
Reserved	15:9	-	-	
RCnt	8:0	RO	Slave received data count	0

3.2.16. Configuration Register (0x7C)

Table 21. Configuration Register

Name	Bit	Type	Description	Reset
Reserved	31:15	-	-	-
Slave	14	RO	Support for SPI Slave mode	Configuration dependent
EILMMem	13	RO	Support for memory mapped access (read-only) through EILM bus	Configuration dependent
AHBMem	12	RO	Support for memory mapped access (read-only) through AHB bus	Configuration dependent
DirectIO	11	RO	Support for Direct SPI IO	Configuration dependent
Reserved	10	-	-	-
QuadSPI	9	RO	Support for Quad I/O SPI	Configuration dependent
DualSPI	8	RO	Support for Dual I/O SPI	Configuration dependent

Name	Bit	Type	Description	Reset
Reserved	7:6	-	-	-
TxFIFOSize	5:4	RO	Depth of TX FIFO 0x0: 2 words 0x1: 4 words 0x2: 8 words 0x3: 16 words	Configuration dependent
Reserved	3:2	-	-	-
RxFIFOSize	1:0	RO	Depth of RX FIFO 0x0: 2 words 0x1: 4 words 0x2: 8 words 0x3: 16 words	Configuration dependent

4. Hardware Configuration Options

The ATCSPI200 controller provides the following configuration options to select the hardware features. The configuration result can be read from the Configuration Register (0x7C).

4.1. Address Width

Define the following macro to set the AHB address width to 24-bit. Default address width is 32-bit.

```
`define ATCSPI200_ADDR_WIDTH_24
```

4.2. Dual I/O Mode

Define ATCSPI200_DUALSPI_SUPPORT to support the dual I/O mode.

```
`define ATCSPI200_DUALSPI_SUPPORT
```

4.3. Quad I/O Mode

Define ATCSPI200_QUADSPI_SUPPORT to support both the dual I/O mode and the quad I/O mode.

```
`define ATCSPI200_QUADSPI_SUPPORT
```

4.4. TX FIFO Depth

Define ATCSPI200_TXFIFO_DEPTH_***n***W to identify depth of the TX FIFO depth, where ***n*** could be 2, 4, 8, and, 16. The default depth is 2 words.
For example, define the TX FIFO depth to 4 words by:

```
`define ATCSPI200_TXFIFO_DEPTH_4W
```

4.5. RX FIFO Depth

Define ATCSPI200_RXFIFO_DEPTH_***n***W to identify depth of the RX FIFO depth, where ***n*** could be 2, 4, 8, and, 16. The default depth is 2 words.

For example, define the RX FIFO depth to 4 words by:


```
`define ATCSPI200_RXFIFO_DEPTH_4W
```

4.6. Direct IO Control

Define ATCSPI200_DIRECT_IO_SUPPORT to enable Direct IO control.

```
`define ATCSPI200_DIRECT_IO_SUPPORT
```

4.7. Memory Mapped Access Support

The ATCSPI200 provides one memory mapped access interface. The interface can be either AHB bus or EILM bus.

Define ATCSPI200_AHB_MEM_SUPPORT to allow memory mapped read accesses on the AHB bus.

```
`define ATCSPI200_AHB_MEM_SUPPORT
```

Define ATCSPI200_EILM_MEM_SUPPORT to allow memory mapped read accesses on the EILM bus.

```
`define ATCSPI200_EILM_MEM_SUPPORT
```

4.8. Slave Mode

Define ATCSPI200_SLAVE_SUPPORT to support the SPI Slave mode.

```
`define ATCSPI200_SLAVE_SUPPORT
```

4.9. AHB Register Port

Define ATCSPI200_REG_AHB to change the register file programming port to AHB. The default programming port is APB.


```
`define ATCSPI200_REG_AHB
```

4.10. Memory Mapped AHB/EILM Read

Define ATCSPI200_MEM_RDCMD_DEFAULT to specify the reset value of the MemRdCmd field in the SPI Memory Access Control Register (0x50). For example, define the reset value to 1 by:

```
`define ATCSPI200_MEM_RDCMD_DEFAULT 4'd1
```

4.11. SPI Interface Timing Parameters

Define following macros to specify the reset values of the CS2SCLK, CSHT, and SCLK_DIV fields in the SPI Interface Timing Register (0x3C). For example:

```
`define ATCSPI200_CS2CLK_DEFAULT    3'h0
`define ATCSPI200_CSHT_DEFAULT      3'h2
`define ATCSPI200_SCLKDIV_DEFAULT   8'h10
```

5. Programming Sequence

This chapter describes the programming sequence to initiate SPI transfers via the register programming. The register programming should not be performed while the memory mapped reads on the AHB/EILM bus is active. Similarly, the memory mapped read operation cannot be performed when the register programming is in progress. The active state of the register programming port is indicated by the SPI Active bit of the SPI Status Register (0x34).

5.1. Master Mode

5.1.1. SPI Write with DMA

5.1.1.1 Scenario

The following sample programming sequence sets up the controller for

- Transmitting two-byte address and 8-bit data width,
- Total transmission count of 16,
- Merging data from four bytes to one word,
- DMA data transfer with hardware handshaking,
- Triggering interrupts at the end of the SPI transfer,
- SPI SCLK frequency being half of the SPI clock source frequency,
- Issuing the 'Page-Program' command (0x02) to the ROM.

5.1.1.2 Program Sequence

SPI transfer format setup

1. Read TX/RX FIFO depth in the Configuration Register (0x70).
2. Wait for the previous SPI transfer to finish by waiting for the SPIActive bit of the SPI Status Register (0x34) to become zero.
3. Set the SPI Transfer Format Register (0x10) as follows:
   i. AddrLen = 1 (address length – 1)
   ii. DataLen = 7 (data length – 1)
   iii. DataMerge = 1

iv. Reset values for other fields

4. Set the SPI Transfer Control Register (0x20) as follows:
   i. CmdEn = 1
   ii. AddrEn = 1
   iii. TransMode = 1 (write only)
   iv. WrTranCnt = 15 (total transmission count – 1)
   v. Reset values for other fields
5. Set the SPI Control Register (0x30) to enable DMA, reset the TX FIFO, and specify the TX FIFO threshold according to the TX FIFO configuration, e.g., half of the TX FIFO depth.
6. Set the SPI Interrupt Enable Register (0x38) to enable the EndIntEn interrupt.
7. Set the SPI Interface Timing Register (0x40) to set SCLK_DIV to 0.

SPI transfer execution

1. Set DMA controller to move data from memory to the SPI Data Register (0x2C).
2. Set the SPI Address Register (0x28).
3. Write the 'Page-Program' command (0x02) to the SPI Command Register (0x24) to trigger the SPI transfer. The encoding of the command may be different depending on SPI devices.
4. Wait the EndInt interrupt by checking the EndInt bit of the SPI Interrupt Status Register (0x3C).
5. Write one to clear the EndInt bit of the SPI Interrupt Status Register (0x3C).

5.1.2. SPI Read with DMA

5.1.2.1 Scenario

The following sample programming sequence sets up the controller for

- Receiving 16 bytes from a two-byte address,
- Merging data from four bytes to one word,
- DMA data transfer with hardware handshaking,
- Triggering interrupts at the end of SPI transfer,
- SPI SCLK frequency being half of the SPI clock source frequency,
- Issuing the 'Read-Data' command (0x03) to a ROM.

5.1.2.2 Program Sequence

SPI transfer format setup

1. Check TX/RX FIFO depth in the Configuration Register (0x70).
2. Wait for the previous SPI transfer to finish by waiting for the SPIActive bit of the SPI Status Register (0x34) to become zero.
3. Set the SPI Transfer Format Register (0x10) as follows:
   i. AddrLen = 1 (address length – 1)
   ii. DataLen = 7 (data length – 1)
   iii. DataMerge = 1
   iv. Reset values for other fields
4. Set the SPI Transfer Control Register (0x20) as follows:
   i. CmdEn = 1
   ii. AddrEn = 1
   iii. TransMode = 2 (read only)
   iv. RdTranCnt = 15 (total transmission count – 1)
   v. Reset values for other fields
5. Set the SPI Control Register (0x30) to enable DMA and specify the RX FIFO threshold.
6. Set the SPI Interrupt Enable Register (0x38) to enable the EndIntEn interrupt.
7. Set the SPI Interface Timing Register (0x40) to set SCLK_DIV to 0.

SPI transfer execution

1. Set DMA controller to move data from the SPI Data Register (0x2C) to memory.
2. Set the SPI Address Register (0x28).
3. Write the 'Read-Data' command (0x03) to the SPI Command Register (0x24) to trigger the SPI transfer. The encoding of this command may be different depending on SPI devices.
4. Wait for the EndInt interrupt by checking the EndInt bit of the SPI Interrupt Status Register (0x3C).
5. Write one to clear the EndInt bit of the SPI Interrupt Status Register (0x3C).

5.2. Slave Mode

5.2.1. Receiving Data from SPI Masters

5.2.1.1 Scenario

Assuming that the SPI transfer format is:

- 8-bit data width,
- quad write command (0x54),
- 20 bytes are transferred.

Since the data merging is enabled by default, this will correspond to 5 data accesses to the Data Register.

5.2.1.2 Program Sequence

1. Reset the RX FIFO by writing one to RXFIFORST bit and wait the RXFIFORST bit to be cleared to 0.
2. Set the RX FIFO threshold (RXTHRES) in the SPI Control Register (0x30) according to the RX FIFO configuration, e.g., half of the RX FIFO depth.
3. Set the SPI Interrupt Enable Register (0x38) to enable the slave command interrupt (SlvCmdEn), receive FIFO threshold interrupt (RXFIFOIntEn), and transfer end interrupt (EndIntEn).
4. In the interrupt service routine:
   A. Wait for the Slave Command interrupt by checking the SlvCmdInt bit of the SPI Interrupt Status Register (0x3C).
      i. Prepare to handle the received SPI request as recorded in the SPI Command Register (0x24).
      ii. Write one to clear the SlvCmdInt bit of the SPI Interrupt Status Register (0x3C).
   B. Wait for the RX FIFO interrupt by checking the RXFIFOInt bit of the SPI Interrupt Status Register (0x3C).
      i. Pop words from the SPI Data Register (0x2C) according to the RX FIFO threshold (RXTHRES) setup.
      ii. Write one to clear the RXFIFOInt bit of the SPI Interrupt Status Register (0x3C).

C. Wait for transfer end interrupt by checking the EndInt bit of the SPI Interrupt Status Register (0x3C).
   i. Check the RX FIFO entries (RXNUM) of the SPI Status Register (0x34).
   ii. Pop all words from the SPI Data Register (0x2C).
   iii. Write one to clear the EndInt bit of the SPI Interrupt Status Register (0x3C).

5.2.2. Transmitting Data to SPI Masters

5.2.2.1 Scenario

The SPI master should arrange the slave to make data ready for transmission before submitting read commands. Assuming this is done through a user-defined command to indicate the address offset and data count of the next read transfer.

The user-defined command is followed by the read status command (0x05) to check for the progress of the slave in the processing of the user-defined commands (whether or not the data that the user-defined command asks for is ready).

Finally, the SPI master issues a data read command to initiate the data transfer. Assuming the data transmission format is:

- 8-bit data width,
- quad read command (0x54),
- the user-defined command ask for 32 bytes of data to be transferred.

Since the data merging is enabled by default, this will correspond to 8 data accesses to the Data Register.

5.2.2.2 Program Sequence

1. When receiving the user-defined command which asks for 32-bytes of data,
   A. Reset the TX FIFO by writing one to TXFIFORST bit and wait the TXFIFORST cleared to 0.
   B. Set the TX FIFO threshold (TXTHRES) in the SPI Control Register (0x30) according to the TX FIFO configuration, e.g., half of the TX FIFO depth.
   C. Prepare the requested data (8 words = 32 bytes) and fill a number of words larger than the TXTHRES setup into the SPI Data Register (0x2C).
   D. Set the SPI Interrupt Enable Register (0x38) to enable the slave command interrupt (SlvCmdEn), transmit FIFO threshold interrupt (TXFIFOIntEn), transfer end interrupt (EndIntEn).
   E. Set the Ready bit in the SPI Slave Status Register (0x60).
2. In the interrupt service routine:
   A. Wait for the Slave Command interrupt by checking the SlvCmdInt bit of the SPI Interrupt Status Register (0x3C).
      i. Verify that the received command is a read command by checking the SPI Command Register (0x24).
      ii. Write one to clear the SlvCmdInt bit of the SPI Interrupt Status Register (0x3C).
   B. Wait for the TX FIFO interrupt by checking the TXFIFOInt bit of the SPI Interrupt Status Register (0x3C).
      i. Fill a number of words larger than the TXTHRES setup into the SPI Data Register (0x2C).
      ii. If the data count of the read transfer has been completely filled into the Data Register, clear the transmit FIFO threshold interrupt (TXFIFOIntEn) in the SPI Interrupt Enable Register (0x38) to avoid a redundant interrupt.
      iii. Write one to clear the TXFIFOInt bit of the SPI Interrupt Status Register (0x3C).
   C. Wait for transfer end interrupt by checking the EndInt bit of the SPI Interrupt Status Register (0x3C).
      i. Write one to clear the EndInt bit of the SPI Interrupt Status Register (0x3C).

6. Integration Guideline

This chapter describes the integration guideline for the ATCSPI200 controller. General integration guidelines are listed in the bullets below and special requirements for the slave mode, clock enable signals, clock gating cell, and DFT considerations are described in the subsections.

General integration guidelines for the ATCSPI200 controller are:

- eilm_clk, hclk, and pclk must be synchronous;
- eilm_clk frequency ≥ hclk frequency ≥ pclk frequency;
- spi_clk_in, spi_clk_out and spi_clk_oe must be connected to the I/O pad of SCLK as shown in Figure 10;
- Since the spi_clk_in is loop-back from the spi_clk_out, the spi_clk_in should set as a generated clock which is derived from the spi_clk_out with PAD latency;
- As shown in Figure 11, the spi_in_r to shift register path should be set as two-cycle path;
- (DataLen + 1)/(*SPI IO width*) must be greater than 1, where the *SPI IO width* is 2 for the dual mode and 4 for the quad mode;
- The controller cannot be accessed concurrently by register programming and memory mapped accesses.

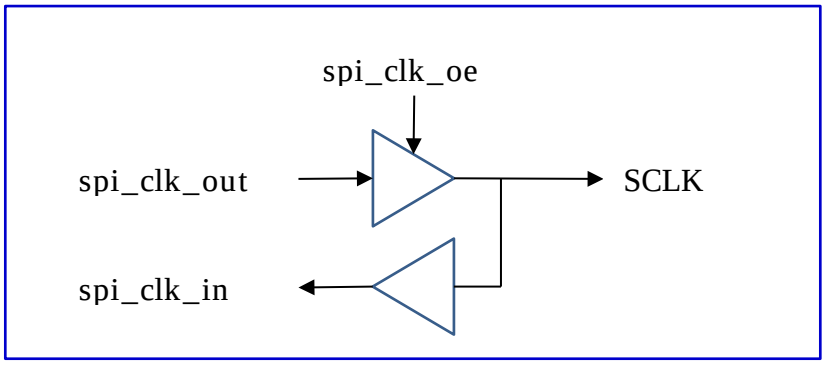

Figure 10. SCLK I/O Pad

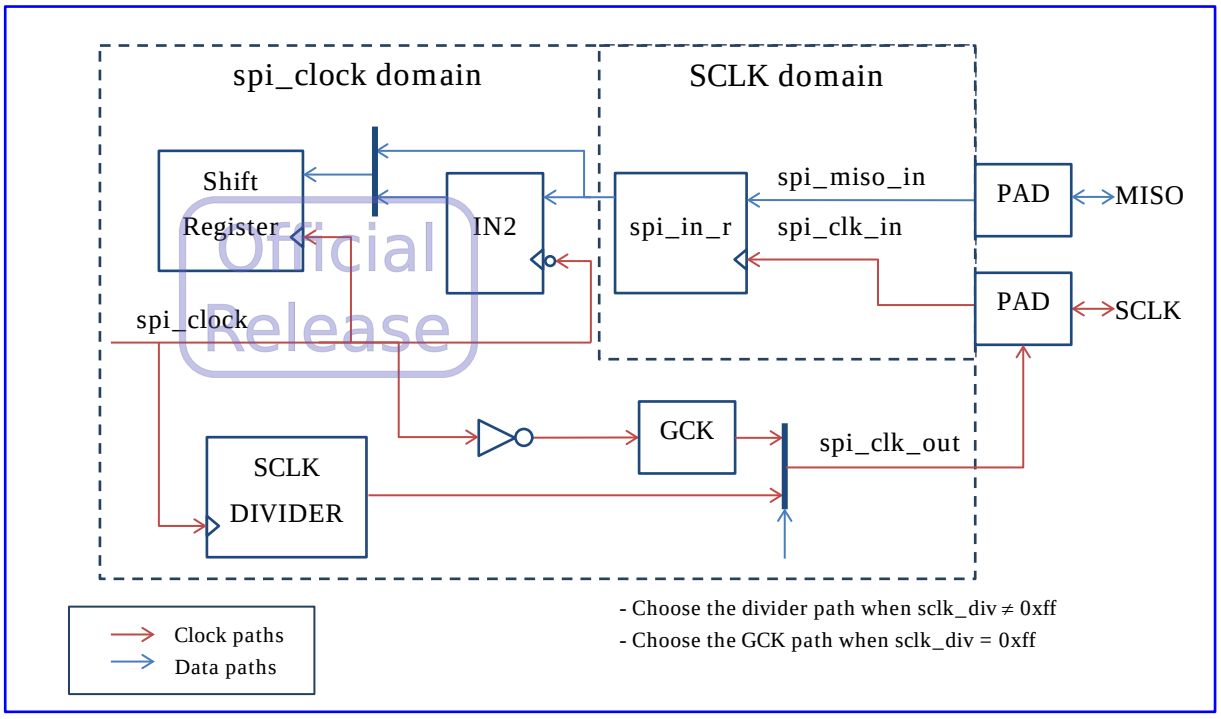

Figure 11. Relation Between SCLK Domain and spi_clock Domain

6.1. Slave Mode

6.1.1. SCLK Frequency

The frequency of SPI SCLK should be smaller than or equal to 1/4 of the SPI clock source frequency. As a result, the SPI master should drive the SCLK slower than 1/4 of the SPI clock source frequency:

$$\text{SCLK frequency} \leq \frac{1}{4} \text{ (Slave SPI clock source frequency)}$$

6.1.2. Time between the Edges of SPI CS to the First Edge of SCLK

The minimum time between the edges of SPI CS and the first edge of SCLK should be large than 2 * period of SPI clock source.

6.2. Clock Enable Signal

The ATCSPI200 design has several clock enable input signals. A Foo to Bar clock enable signal indicates valid Bar-domain clock cycles to sample and update the Foo clock domain signals when the Bar clock frequency is a multiple of the Foo clock frequency. The clock enable signal must assert for one Bar clock cycle for each Foo clock cycle. An example is illustrated in Figure 12 for the case that the AHB frequency is 4 times of the APB frequency. For the special case that AHB clock and APB clock are identical, the clock enable signal must be a constant HIGH.

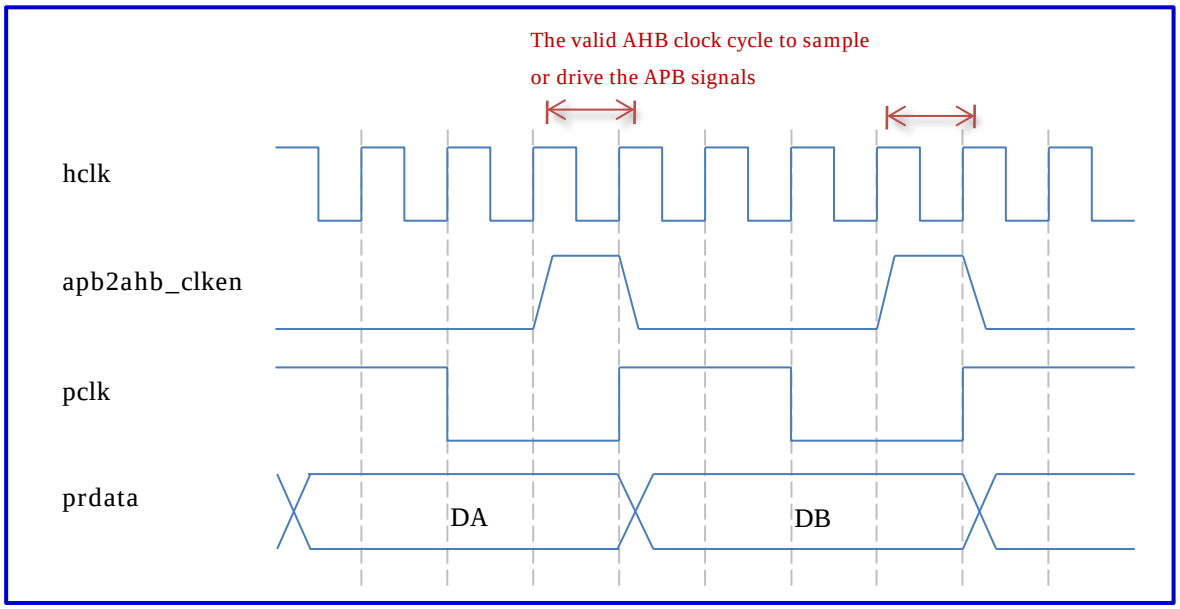

Figure 12.ATCSPI200 Clock Enable Signal for 4:1 AHB-to-APB Clock Ratio

6.3. Clock Gating Cell

There is one clock gating cell model in the ATCSPI200 design. The model is located at $NDS_HOME/andes_ip/macro/hdl/gck.v. It is a latch based clock gating cell. The RTL modeling of gck.v is synthesizable. However, integrated clock gating cells from the standard library is preferred to better control clock skews.

The synthesis script automatically scans the standard cell library for an integrated clock gating cell that matches gck.v's behavior. A gck_autogen.v file will be created under the script directory if such a cell is found in the standard library. Please double check the correctness of the cell. If the heuristic fails, it will fall back to let the synthesizer synthesize the gck.v. One could still modify gck.v to instantiate a proper integrated clock gating cell if the cell in question cannot be found by the synthesis script.

The synthesis script selects clock gating cells according to the following rule: it scans the library to find all cells that have attribute 'clock_gating_integrated_cell' and the value of the attribute is "latch_posedge_precontrol." For example,

```
# .lib file
cell (GCK_VENDOR_CELL) {
    …
    clock_gating_integrated_cell : "latch_posedge_precontrol";
    …
}
```

When there are multiple cells found in the library, the one with the smallest leakage power is selected. The script then connects the pins through these attributes in the library: clock_gate_clock_pin, clock_gate_enable_pin, clock_gate_test_pin and clock_gate_out_pin.

Figure 13 shows how the gck module should look like in case the synthesis script fails or you want to generate gck.v manually.

```
module gck (clk_out, clk_en, clk_in, test_en);
input clk_in;         // clock input
input clk_en;         // gated clock signal
input test_en;        // enable shifting scan data
output clk_out;       // clock output

`ifdef SYNTHESIS
   GCK_VENDOR_CELL gck(
           .Q(clk_out),
           .E(clk_en),
           .TE(test_en),
           .CK(clk_in)
           );
`else
   // gated clock behavior code:
   // FPGA note: this code is synthesizable under
   // FPGA with FixGatedClock support.

       reg latch_out;

       always @(clk_in or clk_en or test_en)
           if (~clk_in)
               latch_out <= clk_en | test_en;

       // clock gating occurs at the negedge of clk_in.
       assign clk_out = clk_in & latch_out;
`endif //SYNTHESIS
endmodule
```


Figure 13. Clock Gating Logic for Simulation and Synthesis

Figure 14 further depicts the requirement for the clock gating cell. The internal latch of the clock gating cell should be in the transparent mode when clock is low and in the latch mode when clock is high.

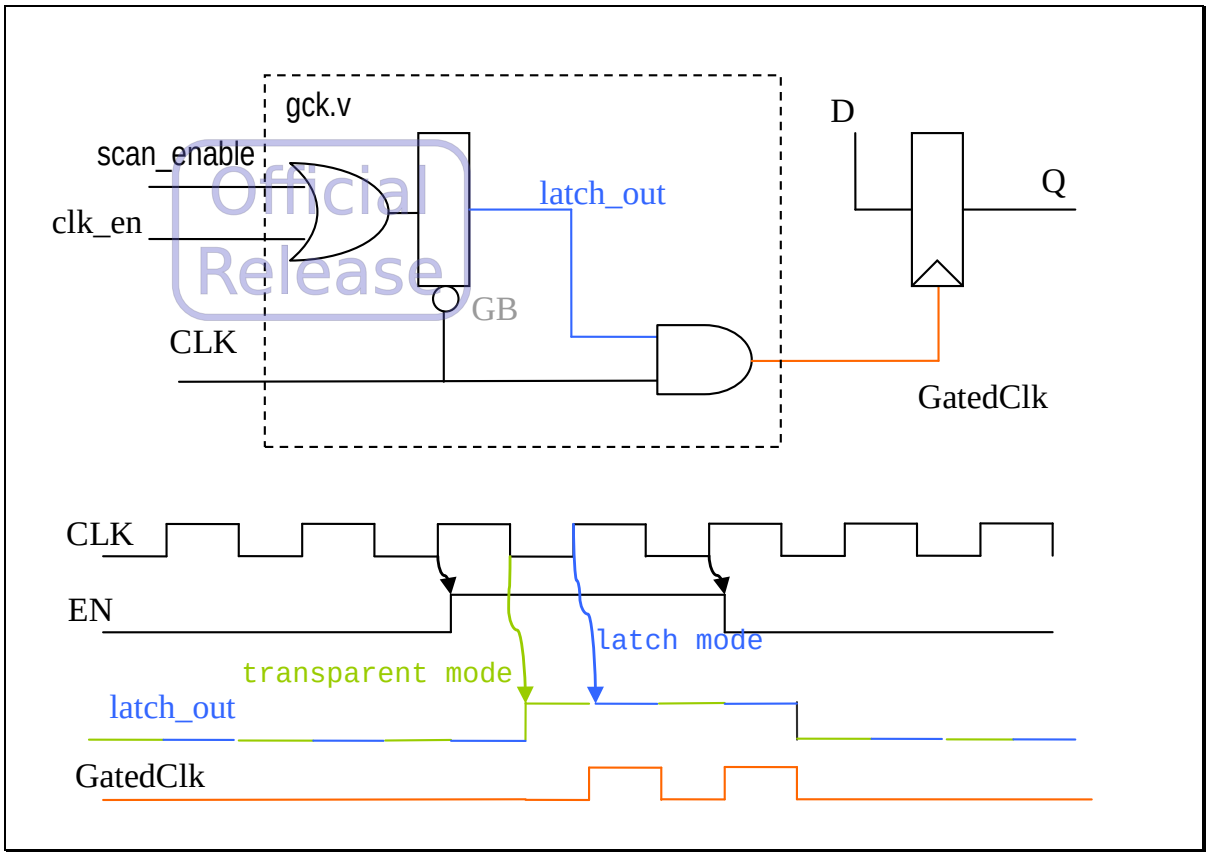

Figure 14. Clock Gating Cell Diagram with Waveform

It is recommend that after the manual instantiation of the gck cell, a formal equivalence checking is performed to make sure that the selected cell matches the behavior code. However, unless gck cells with latch_posedge_precontrol is used, the equivalence checking will most likely not match.

6.4. DFT Considerations

The ATCSPI200 design provides scan_test and scan_enable input pins to control its behavior under ATPG test.

The scan_test pin should be asserted (active high) during the entire session of the ATPG test mode. There is a loop-back clock path in the ATCSPI200 design as shown in Figure 15. The loop-back clock should be controllable under the ATPG test mode to enhance the ATPG coverage. With the scan_test pin being HIGH for the entire duration of the ATPG test mode, the flops in the loop-back clock domain will be clocked by spi_clock and be immune to the changes of spi_clk_oe and spi_clk_out pins.

The scan_enable pin should only be active high during the ATPG scan/shift phase, such that all clocks are turned on and ATPG test patterns and results can be shifted through the scan chains. The clock gating logic is disabled (clock running freely) when scan_enable is active. The scan_enable pin is off during ATPG capture phase to allow observabilities for clock gating enable pins.

The DFT tools need to be informed about the usage of these two signals using the sample commands listed below.

```
set_dft_signal -view existing_dft -port scan_enable -type ScanEnable
set_dft_signal -view existing_dft -port scan_test -type TestMode
```

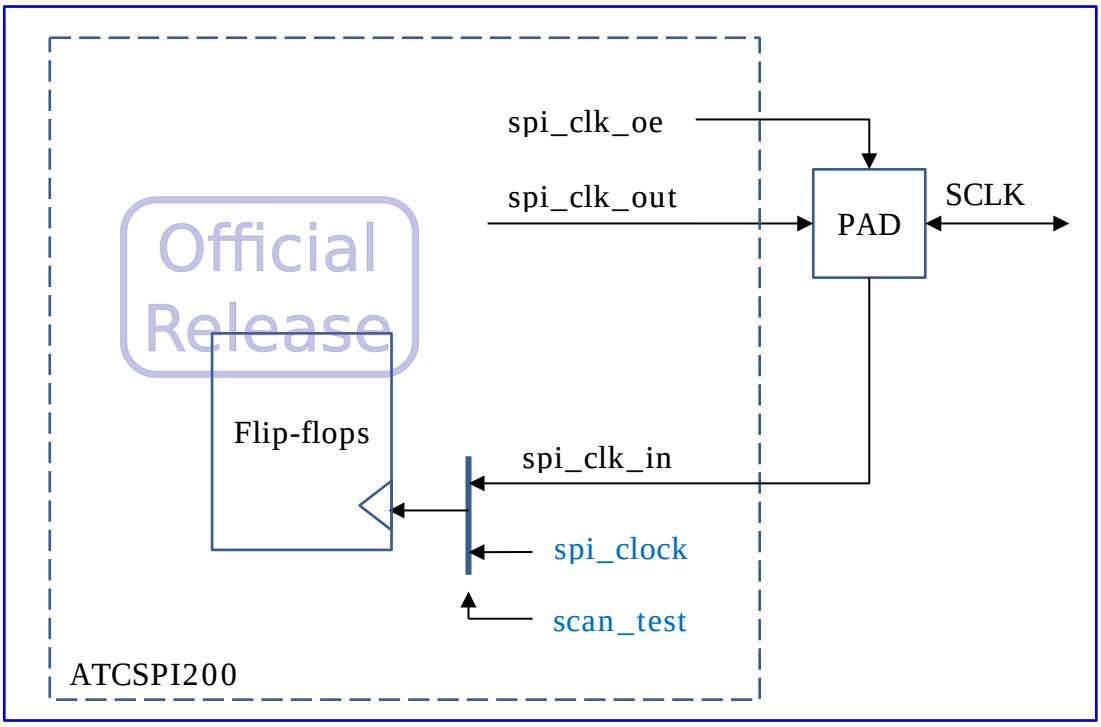

Figure 15. ATCSPI200 Design for ATPG Test