LogiCORE IP XPS Serial Peripheral Interface (SPI) (v2.02a)

DS570 July 23, 2010 **Product Specification**

Introduction

The XPS Serial Peripheral Interface (SPI) connects to the PLB V4.6 (Processor Local Bus with Xilinx simplifications) and provides a serial interface to SPI devices such as SPI EEPROMs and SPI serial flash devices. The SPI protocol, as described in the Motorola M68HC11 data sheet, provides a simple method for a master and a selected slave to exchange data.

Features

- Connects as a 32-bit slave on PLB V4.6 buses of 32, 64 or 128 bits
- Supports four signal interface (MOSI, MISO, SCK and $\overline{SS}$)
- Supports slave select ($\overline{SS}$) bit for each slave on the SPI bus
- Supports full-duplex operation
- Supports master and slave SPI modes
- Supports programable clock phase and polarity
- Supports continuous transfer mode for automatic scanning of a peripheral
- Supports back-to-back transactions
- Supports automatic or manual slave select modes
- Supports MSB/LSB first transactions
- Supports transfer length of 8-bits, 16-bits or 32-bits
- Supports local loopback capability for testing
- Supports multiple master and multiple slave environment
- Optional 16 element deep (an element is a byte, a half-word or a word) transmit and receive FIFOs

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Spartan®-3, Spartan-3E, Spartan-3A/3AN, Spartan-6, Spartan-3A DSP, Automotive Spartan-3/3A/3A DSP/ 3E, Virtex®-4, Virtex-4Q, Virtex-4QV, Virtex-5/5FX, Virtex-6
Supported User Interfaces	32-bit PLBv46 Slave
Resources	
Block RAMS	For Virtex-4 FPGA, see Table 16. For Virtex-5 FPGA, see Table 17. For Spartan-3A, see Table 18. For Spartan-6 FPGA, see Table 19. For Virtex-6 FPGA, see Table 20.
LUTs	
Slices	
FFs	
Provided with Core	
Documentation	Product Specification
Design Files	VHDL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Not Provided
Simulation Model	Not Provided
Tested Design Tools	
Design Entry Tools	Xilinx ISE® v12.2 and above
Simulation	Mentor Graphic ModelSim v6.5c and above
Synthesis Tools	XST v12.2 and above
Support	
Provided by Xilinx, Inc.	

Notes:

1. For a complete listing of supported devices, see the release notes for this core.

Functional Description

The top level block diagram for the XPS SPI IP Core is shown in Figure 1..

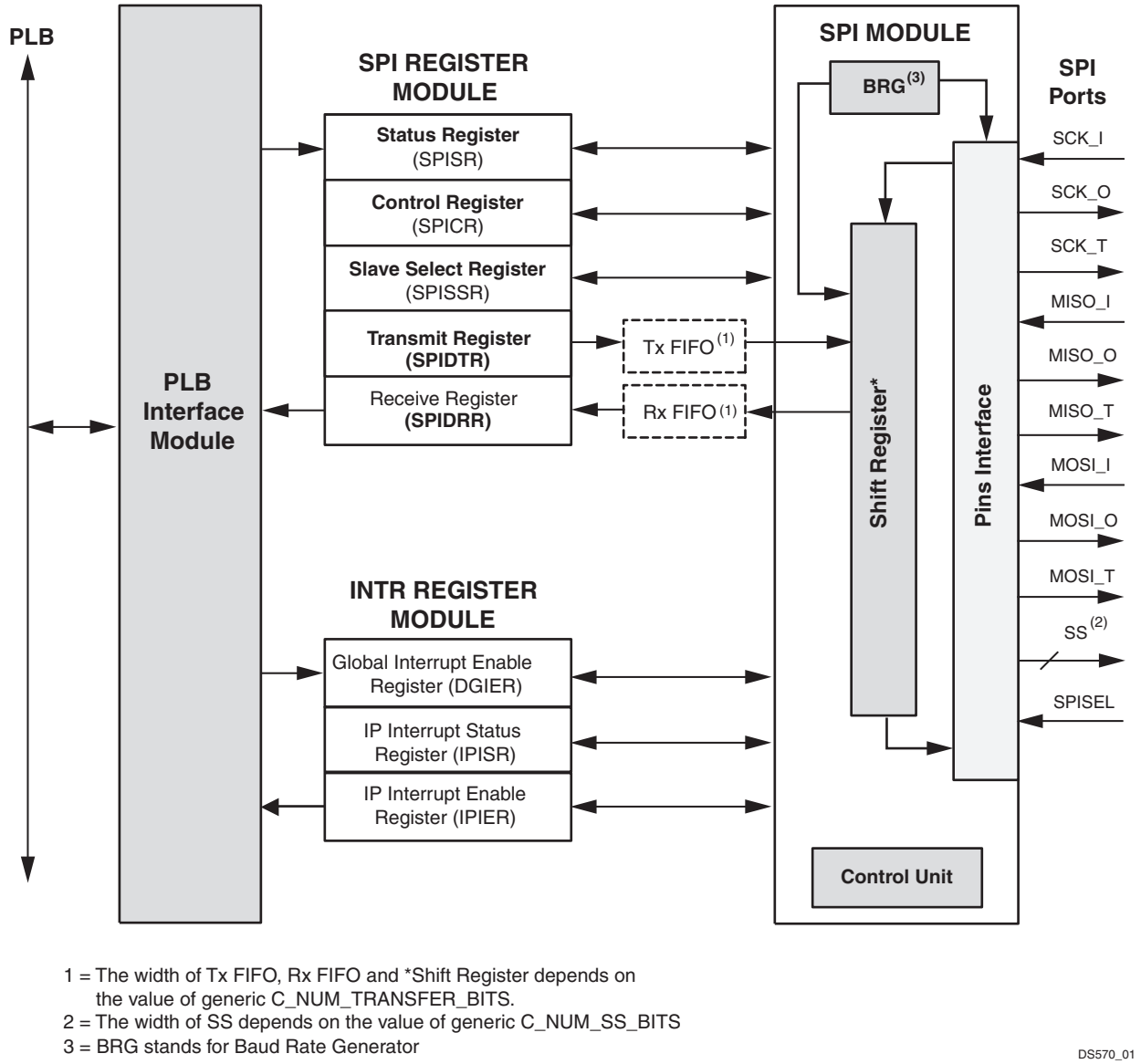

Figure 1: **Top-Level Block Diagram for the XPS SPI IP Core**

The XPS SPI IP Core is a full-duplex synchronous channel that supports four-wire interface (receive, transmit, clock and slave-select) between a master and a selected slave.

The XPS SPI IP Core supports Manual Slave Select Mode as the Default Mode of operation. This mode allows the user to manually control the slave select line by the data written to the slave select register. This allows transfers of an arbitrary number of elements without toggling the slave select line between elements. However, the user must toggle the slave select line before starting a new transfer.

The other mode of operation is Automatic Slave Select Mode. In this mode the slave select line is toggled automatically after each element transfer. This mode is described in more detail in SPI Protocol with Automatic Slave Select Assertion section.

The XPS SPI IP Core supports continuous transfer mode, wherein when configured as master the transfer continues till the data is available in transmit register/FIFO. This capability is provided in both manual and automatic slave select modes.

When XPS SPI IP Core is configured as a slave and if inadvertently its slave select line (SPISEL) goes high (i.e. in-active state) in between the data element transfer, then the current transfer is aborted. Again if the slave select line goes low then the aborted data element is transmitted again.

The XPS SPI IP Core permits additional slaves to be added with automatic generation of the required decoding logic for individual slave select outputs by the master. Additional masters can be added as well. However, means to detect all possible conflicts are not implemented with this interface standard. To eliminate conflicts, software is required to arbitrate bus control.

The XPS SPI IP Core can communicate with both off-chip and on-chip masters and slaves. The number of slaves is limited to 32 by the size of the Slave Select Register. However, the number of slaves and masters will impact the achievable performance in terms of frequency and resource utilization.

All the SPI and INTR registers are 32-bit wide. The XPS SPI IP Core supports only word access to all SPI and INTR register modules.

The XPS SPI IP Core modules are described in the sections below.

PLB Interface Module: The PLB Interface Module provides the interface to the PLB V4.6 slave single. The read and write transactions at the PLB are translated into equivalent IP Interconnect (IPIC) transactions. The register interfaces of the SPI connect to the IPIC. The PLB Interface Module also provides an address decoding service for XPS SPI Core.

SPI Register Module: The SPI Register Module includes all memory mapped registers (as shown in Figure 1). It interfaces to the PLB. It consists of Status Register, Control Register, N-bit Slave Select Register ($N \leq 32$) and a pair of Transmit/Receive Registers.

INTR Register Module: The INTR Register Module consists of interrupt related registers namely device global interrupt enable register (DGIER), IP interrupt enable register (IPIER) and IP interrupt status register (IPISR).

SPI Module: The SPI Module consists of a shift register, a parameterized baud rate generator (BRG) and a control unit. It provides the SPI interface, including the control logic and initialization logic. It is the heart of core.

Optional FIFOs: The Tx FIFO and Rx FIFO are implemented on both transmit and receive paths when enabled by the parameter C_FIFO_EXIST. The width of Tx FIFO and Rx FIFO is same and it depends on generic C_NUM_TRANSFER_BITS. The depth of these FIFO's is 16, which is FIFO design dependent.

Design Parameters

To allow the user to obtain a XPS SPI IP Core that is uniquely tailored for the system, certain features can be parameterized. Parameterization affords a measure of control over the function, resource usage, and performance of the actually implemented XPS SPI IP Core. The features that can be parameterized are as shown in Table 1.

Table 1: **Design Parameters**

Generic	Feature/Description	Parameter Name	Allowable Values	Default Value	VHDL Type
System Parameters					
G1	Target FPGA family	C_FAMILY	spartan3, aspartan3, spartan3an, spartan3a, spartan3e, spartan3adsp, aspartan3e, aspartan3a, aspartan3adsp, virtex4, virtex5,virtex5fx, qvirtex4, qrvirtex4, spartan6, aspartan6, virtex6.virtex6cx	spartan3	string
PLB Parameters					
G2	PLB base address	C_BASEADDR	Valid Address(1)	None(2)	std_logic_vector
G3	PLB high address	C_HIGHADDR	Valid Address(1)	None(2)	std_logic_vector
G4	PLB least significant address bus width	C_SPLB_AWIDTH	32	32	integer
G5	PLB data width	C_SPLB_DWIDTH	32, 64, 128	32	integer
G6	Shared bus topology	C_SPLB_P2P	0 = Shared bus topology(3)	0	integer
G7	PLB master ID bus Width	C_SPLB_MID_WIDTH	$\log_2$(C_SPLB_NUM_MASTERS) with a minimum value of 1	1	integer
G8	Number of PLB masters	C_SPLB_NUM_MASTERS	1 - 16	1	integer
G9	Width of the slave data bus	C_SPLB_NATIVE_DWIDTH	32	32	integer
G10	Burst support	C_SPLB_SUPPORT_BURSTS	0 = No burst support(4)	0	integer
XPS SPI IP Core Parameters					
G11	Include receive and transmit FIFOs	C_FIFO_EXIST	0 = FIFOs not included 1 = FIFOs included	1	integer
G12	SPI clock frequency ratio	C_SCK_RATIO	2(5), 4, Nx16 for N = 1, 2, 3, ...,128	32(6)	integer
G13	Total number of slave select bits	C_NUM_SS_BITS	1 - 32	1	integer
G14	Select number of transfer bits as 8	C_NUM_TRANSFER_BITS	8, 16, 32	8	integer

Table 1: **Design Parameters *(Cont'd)***

Generic	Feature/Description	Parameter Name	Allowable Values	Default Value	VHDL Type

Notes:

1. The range C_BASEADDR to C_HIGHADDR is the address range for the XPS SPI IP Courthouse range is subject to restrictions to accommodate the simple address decoding scheme that is employed: The size, C_HIGHADDR - C_BASEADDR + 1, must be a power of two and must be at least 0x80 to accommodate all XPS SPI IP Core registers. However, a larger power of two may be chosen to reduce decoding logic. C_BASEADDR must be aligned to a multiple of the range size.
2. No default value will be specified to insure that an actual value appropriate to the system is set. User must set the values.
3. Point to point bus topology is not allowed in this version of XPS SPI IP Core.
4. Burst to-from PLB is not supported in this version of XPS SPI IP Core.
5. C_SCK_RATIO = 2 is not supported when XPS SPI IP Core is configured as slave. So, user must take care in not configuring C_SCK_RATIO = 2 while using XPS SPI IP Core as slave.
6. Please read the Precautions to be Taken while Assigning the C_SCK_RATIO Parameter section carefully while using this parameter.

I/O Signals

The I/O signals are listed and described in Table 2.

Table 2: **I/O Signal Descriptions**

Port	Signal Name	Interface	I/O	Initial State	Description
System Signals					
P1	SPLB_Clk	System	I	-	PLB clock
P2	SPLB_Rst	System	I	-	PLB reset, active high
P3	IP2INTC_Irpt	System	O	0	Interrupt control signal from SPI
PLB Master Interface Signals					
P4	PLB_ABus[0 : 31]	PLB	I	-	PLB address bus
P5	PLB_PAValid	PLB	I	-	PLB primary address valid
P6	PLB_masterID[0 : C_SPLB_MID_WIDTH - 1]	PLB	I	-	PLB current master identifier
P7	PLB_RNW	PLB	I	-	PLB read not write
P8	PLB_BE[0 : (C_SPLB_DWIDTH/8) - 1]	PLB	I	-	PLB byte enables
P9	PLB_size[0 : 3]	PLB	I	-	PLB size of requested transfer
P10	PLB_type[0 : 2]	PLB	I	-	PLB transfer type
P11	PLB_wrDBus[0 : C_SPLB_DWIDTH - 1]	PLB	I	-	PLB write data bus
Unused PLB Master Interface Signals					
P12	PLB_UABus[0 : 31]	PLB	I	-	PLB upper address bits
P13	PLB_SAValid	PLB	I	-	PLB secondary address valid
P14	PLB_rdPrim	PLB	I	-	PLB secondary to primary read request indicator
P15	PLB_wrPrim	PLB	I	-	PLB secondary to primary write request indicator
P16	PLB_abort	PLB	I	-	PLB abort bus request
P17	PLB_busLock	PLB	I	-	PLB bus lock
P18	PLB_MSize[0 : 1]	PLB	I	-	PLB data bus width indicator

Table 2: **I/O Signal Descriptions *(Cont'd)***

Port	Signal Name	Interface	I/O	Initial State	Description
P19	PLB_lockErr	PLB	I	-	PLB lock error
P20	PLB_wrBurst	PLB	I	-	PLB burst write transfer
P21	PLB_rdBurst	PLB	I	-	PLB burst read transfer
P22	PLB_wrPendReq	PLB	I	-	PLB pending bus write request
P23	PLB_rdPendReq	PLB	I	-	PLB pending bus read request
P24	PLB_wrPendPri[0 : 1]	PLB	I	-	PLB pending write request priority
P25	PLB_rdPendPri[0 : 1]	PLB	I	-	PLB pending read request priority
P26	PLB_reqPri[0 : 1]	PLB	I	-	PLB current request priority
P27	PLB_TAttribute[0 : 15]	PLB	I	-	PLB transfer attribute
PLB Slave Interface Signals					
P28	Sl_addrAck	PLB	O	0	Slave address acknowledge
P29	Sl_SSize[0 : 1]	PLB	O	0	Slave data bus size
P30	Sl_wait	PLB	O	0	Slave wait
P31	Sl_rearbitrate	PLB	O	0	Slave bus rearbitrate
P32	Sl_wrDAck	PLB	O	0	Slave write data acknowledge
P33	Sl_wrComp	PLB	O	0	Slave write transfer complete
P34	Sl_rdDBus[0 : C_SPLB_DWIDTH - 1]	PLB	O	0	Slave read data bus
P35	Sl_rdDAck	PLB	O	0	Slave read data acknowledge
P36	Sl_rdComp	PLB	O	0	Slave read transfer complete
P37	Sl_MBusy[0:C_SPLB_ NUM_MASTERS - 1]	PLB	O	0	Slave busy
P38	Sl_MWrErr[0C_SPLB_ NUM_MASTERS - 1]	PLB	O	0	Slave write error
P39	Sl_MRdErr[0C_SPLB_ NUM_MASTERS - 1]	PLB	O	0	Slave read error
Unused PLB Slave Interface Signals					
P40	Sl_wrBTerm	PLB	O	0	Slave terminate write burst transfer
P41	Sl_rdWdAddr[0 : 3]	PLB	O	0	Slave read word address
P42	Sl_rdBTerm	PLB	O	0	Slave terminate read burst transfer
P43	Sl_MIRQ[0 : C_SPLB_NUM_MASTERS - 1]	PLB	O	0	Master interrupt request
SPI Interface Signals					
P44	SCK_I	SPI	I	-	SPI bus clock input
P45	SCK_O	SPI	O	0	SPI bus clock output
P46	SCK_T	SPI	O	1	Tri-state enable for SPI bus clock. Active low
P47	MOSI_I	SPI	I	-	Master output slave input
P48	MOSI_O	SPI	O	1	Master output slave input

Table 2: **I/O Signal Descriptions *(Cont'd)***

Port	Signal Name	Interface	I/O	Initial State	Description
P49	MOSI_T	SPI	O	1	Tri-state enable master output slave input. Active low
P50	MISO_I	SPI	I	-	Master input slave output
P51	MISO_O	SPI	O	1	Master input slave output
P52	MISO_T	SPI	O	1	Tri-state enable master input slave output. Active low
P53	SPISEL[(1)]	SPI	I	1	Local SPI slave select active low input. Must be set to 1 in idle state
P54	SS_I[0 : C_NUM_SS_BITS - 1]	SPI	I	-	Input one-hot encoded. This signal is a dummy signal and wont be used in the design as chip select input
P55	SS_O[0 : C_NUM_SS_BITS - 1]	SPI	O	1	Output one-hot encoded, active low slave select vector of length n
P56	SS_T	SPI	O	1	Tri-state enable for slave select. Active low

Notes:

1. SPISEL signal is used as a slave select line when XPS SPI is configured as slave.

Parameter - Port Dependencies

The dependencies between the XPS SPI IP Core design parameters and I/O signals are described in Table 3.

Table 3: **Parameter-Port Dependencies**

Generic or Port	Name	Affects	Depends	Relationship Description
Design Parameters				
G5	C_SPLB_DWIDTH	P8, P11, P34	-	Affects the number of bits in data bus
G7	C_SPLB_MID_WIDTH	P6	G8	This value is calculated as: $\log_2$(C_SPLB_NUM_MASTERS) with a minimum value of 1
G8	C_SPLB_NUM_MASTERS	P37, P38, P39, P43	-	Affects the number of PLB masters
G13	C_NUM_SS_BITS	P54, P55	-	Defines the total number of slave select bits
I/O Signals				
P6	PLB_masterID[0 : C_SPLB_MID_WIDTH - 1]	-	G7	Width of the PLB_mastedID varies according to C_SPLB_MID_WIDTH
P8	PLB_BE[0 : (C_SPLB_DWIDTH/8) -1]	-	G5	Width of the PLB_BE varies according to C_SPLB_DWIDTH
P11	PLB_wrDBus[0 : C_SPLB_DWIDTH - 1]	-	G5	Width of the PLB_wrDBus varies according to C_SPLB_DWIDTH
P34	Sl_rdDBus[0 : C_SPLB_DWIDTH - 1]	-	G5	Width of the Sl_rdDBus varies according to C_SPLB_DWIDTH
P37	Sl_MBusy[0 : C_SPLB_NUM_MASTERS - 1]	-	G8	Width of the Sl_MBusy varies according to C_SPLB_NUM_MASTERS

Table 3: **Parameter-Port Dependencies *(Cont'd)***

Generic or Port	Name	Affects	Depends	Relationship Description
P38	SI_MWrErr[0 : C_SPLB_NUM_MASTERS - 1]	-	G8	Width of the SI_MWrErr varies according to C_SPLB_NUM_MASTERS
P39	SI_MRdErr[0 : C_SPLB_NUM_MASTERS - 1]	-	G8	Width of the SI_MRdErr varies according to C_SPLB_NUM_MASTERS
P43	SI_MIRQ[0 : C_SPLB_NUM_MASTERS - 1]	-	G8	Width of the SI_MIRQ varies according to C_SPLB_NUM_MASTERS
P54	SS_I[0 : C_NUM_SS_BITS - 1]	-	G13	The number of SS_I pins are generated based on C_NUM_SS_BITS
P55	SS_O[0 : C_NUM_SS_BITS - 1]	-	G13	The number of SS_O pins are generated based on C_NUM_SS_BITS

Register Descriptions

The Table 4 gives a summary of the XPS SPI IP Core registers. The Transmit FIFO Occupancy Register and the Receive FIFO Occupancy Register exists only when C_FIFO_EXIST = 1.

Table 4: **XPS SPI IP Core Registers**

Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
XPS SPI IP Core Grouping				
C_BASEADDR + 40	SRR	Write	N/A	Software Reset Register
C_BASEADDR + 60	SPICR	R/W	0x180	SPI Control Register
C_BASEADDR + 64	SPISR	Read	0x25	SPI Status Register
C_BASEADDR + 68	SPIDTR	Write	0x0	SPI Data Transmit Register A single register or a FIFO
C_BASEADDR + 6C	SPIDRR	Read	0x0	SPI Data Receive Register A single register or a FIFO
C_BASEADDR + 70	SPISSR	R/W	No slave is selected	SPI Slave Select Register
C_BASEADDR + 74	SPI Transmit FIFO Occupancy Register(1)	Read	0x0	Transmit FIFO Occupancy Register
C_BASEADDR + 78	SPI Receive FIFO Occupancy Register(2)	Read	0x0	Receive FIFO Occupancy Register
Interrupt Controller Grouping				
C_BASEADDR + 1C	DGIER	R/W	0x0	Device Global Interrupt Enable Register
C_BASEADDR + 20	IPISR	R/TOW<RD Red><SP Superscript >(2)	0x0	IP Interrupt Status Register
C_BASEADDR + 28	IPIER	R/W	0x0	IP Interrupt Enable Register

Notes:

1. This register does not exist if C_FIFO_EXIST = 0.
2. TOW = Toggle On Write. Writing a 1 to a bit position within the register causes the corresponding bit position in the register to toggle.

Details of XPS SPI IP Core Registers

Software Reset Register (SRR)

The Software Reset Register permits the programmer to reset the XPS SPI IP Core independent of other cores in the systems. To activate software generated reset, the value of 0x0000_000A must be written to this register. Any other write access generates an error condition with undefined results and result in error generation. The bit assignment in the software reset register is shown in Figure 2 and described in Table 6. The effect of an attempt to read this register will result with undefined data.

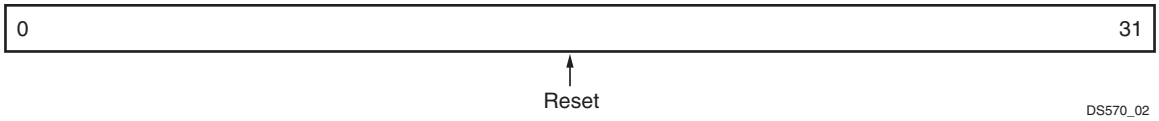

Figure 2: **Software Reset Register**

Table 5: **Software Reset Register (SRR) Description (C_BASEADDR + 0x40)**

Bit(s)	Name	Core Access	Reset Value	Description
0 - 31	Reset	Write only	N/A	The only allowed operation on this register is a write of 0x0000000A, which resets the XPS SPI IP Core.

SPI Control Register (SPICR)

The SPI Control Register (SPICR) gives the programmer control over various aspects of the XPS SPI IP Core. The bit assignment in the SPICR is shown in Figure 3 and described in Table 7.

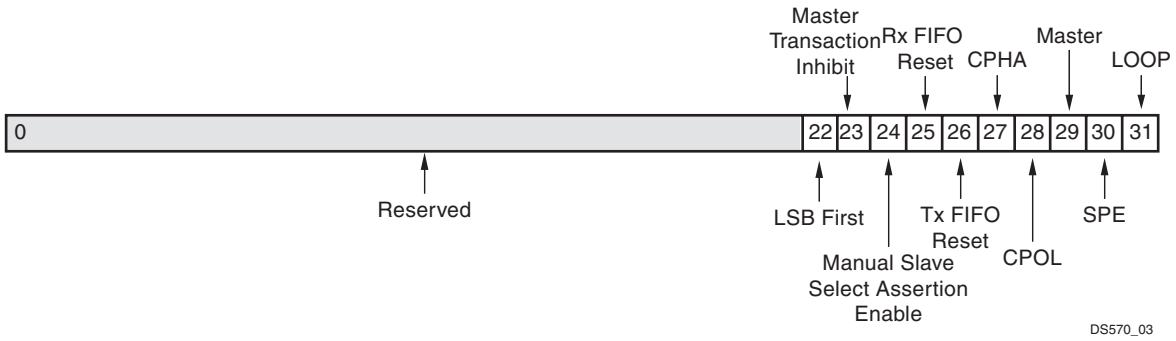

Figure 3: **SPI Control Register (C_BASEADDR + 0x60)**

Table 6: **SPI Control Register (SPICR) Description (C_BASEADDR + 0x60)**

Bit(s)	Name	Core Access	Reset Value	Description
0 - 21	Reserved	N/A	N/A	Reserved
22	LSB First	R/W	'0'	**LSB First.** This bit selects LSB first data transfer format. The default transfer format is MSB first. '0' = MSB first transfer format '1' = LSB first transfer format
23	Master Transaction Inhibit	R/W	'1'	**Master Transaction Inhibit.** This bit inhibits master transactions. This bit has no effect on slave operation. '0' = Master transactions enabled '1' = Master transactions disabled

Table 6: **SPI Control Register (SPICR) Description (C_BASEADDR + 0x60) *(Cont'd)***

Bit(s)	Name	Core Access	Reset Value	Description
24	Manual Slave Select Assertion Enable	R/W	'1'	**Manual Slave Select Assertion Enable.** This bit forces the data in the slave select register to be asserted on the slave select output anytime the device is configured as a master and the device is enabled (SPE asserted). This bit has no effect on slave operation. '0' = Slave select output asserted by master core logic '1' = Slave select output follows data in slave select register
25	Rx FIFO Reset	R/W	'0'	**Receive FIFO Reset.** When written to '1', this bit forces a reset of the Receive FIFO to the empty condition. One PLB clock cycle after reset, this bit is again set to '0'. This bit is unassigned when the XPS SPI IP Core is not configured with FIFOs. '0' = Receive FIFO normal operation '1' = Reset receive FIFO pointer
26	Tx FIFO Reset	R/W	'0'	**Transmit FIFO Reset.** When written to '1', this bit forces a reset of the Transmit FIFO to the empty condition. One PLB clock cycle after reset, this bit is again set to '0'. This bit is unassigned when the XPS SPI IP Core is not configured with FIFOs. '0' = Transmit FIFO normal operation '1' = Reset transmit FIFO pointer
27	CPHA	R/W	'0'	**Clock Phase.** Setting this bit selects one of two fundamentally different transfer formats. See XPS SPI IP Core Design Description
28	CPOL	R/W	'0'	**Clock Polarity.** Setting this bit defines clock polarity. '0' = Active high clock; SCK idles low '1' = Active low clock; SCK idles high
29	Master	R/W	'0'	**Master.** Setting this bit configures the SPI device as a master or a slave. '0' = Slave configuration '1' = Master configuration
30	SPE	R/W	'0'	**SPI System Enable.** Setting this bit to '1' enables the SPI devices as noted below. '0' = SPI system disabled. Both master and slave outputs are in "tri-state" and slave inputs ignored '1' = SPI system enabled. Master outputs active (e.g. MOSI and SCK in idle state) and slave outputs will become active if $\overline{SS}$ becomes asserted. Master will start transfer when transmit data is available
31	LOOP	R/W	'0'	**Local Loopback Mode.** Enables local loopback operation and is functional only in master mode. '0' = Normal operation '1' = Loopback mode. The transmitter output is internally connected to the receiver input. The receiver and transmitter operate normally, except that received data (from remote slave) is ignored Note that the interrupt enable bit which resides at this bit position of the M68HC11 specification resides in the interrupt enable register in this implementation; see Specification Exceptions

SPI Status Register (SPISR)

The SPI Status Register (SPISR) is a read-only register that gives the programmer visibility of the status of some aspects of the XPS SPI IP Core. The bit assignment in the SPISR is shown in Figure 4 and described in Table 7. Writing to the SPISR is not recommended and if it is done by mistake, then no change will be there in register contents.

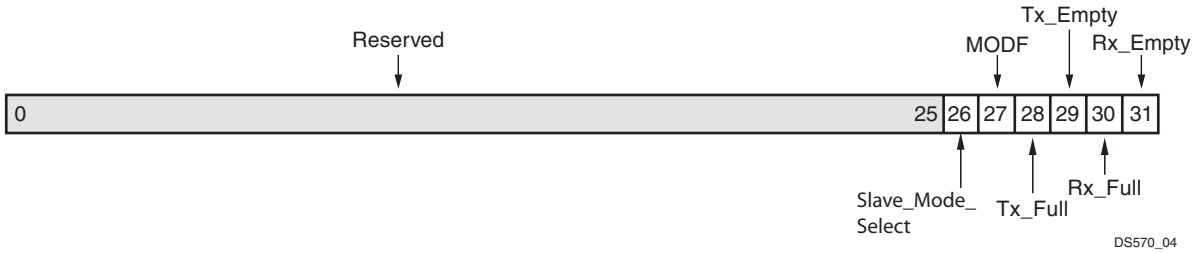

Figure 4: **SPI Status Register (C_BASEADDR + 0x64)**

Table 7: **SPI Status Register (SPISR) Description (C_BASEADDR + 0x64)**

Bit(s)	Name	Core Access	Reset Value	Description
0 - 25	Reserved	N/A	N/A	Reserved
26	Slave_Mode_Select	Read	'1'	**Slave_Mode_Select Flag.** This flag is asserted when the core is configured in slave mode. Slave_Mode_Select will be activated as soon as master SPI core asserts the Chip Select pin for the core. '1' = Default '0' = Asserted when core configured in slave mode and selected by external SPI master
27	MODF	Read	'0'	**Mode-Fault Error Flag.** This flag is set if the $\overline{SS}$ signal goes active while the SPI device is configured as a master. MODF is automatically cleared by reading the SPISR. MODF does generate an interrupt with a single cycle strobe when the MODF bit transitions from a low to high. '0' = No error '1' = Error condition detected
28	Tx_Full	Read	'0'	**Transmit Full.** When a transmit FIFO exists, this bit will be set high when the transmit FIFO is full. When FIFOs don't exist, this bit is set high when an PLB write to the register has been made. This bit is cleared when the SPI transfer is completed.
29	Tx_Empty	Read	'1'	**Transmit Empty.** When a transmit FIFO exists, this bit will be set high when the transmit FIFO is empty. The occupancy of the FIFO is decremented with the completion of each SPI transfer. When FIFOs don't exist, this bit is set with the completion of an SPI transfer. Either with or without FIFOs, this bit is cleared upon a PLB write to the FIFO or transmit register.
30	Rx_Full	Read	'0'	**Receive Full.** When a receive FIFO exists, this bit will be set high when the receive FIFO is full. The occupancy of the FIFO is incremented with the completion of each SPI transaction. When FIFOs don't exist, this bit is set high when an SPI transfer has completed. Rx_Empty and Rx_Full are complements in this case.
31	Rx_Empty	Read	'1'	**Receive Empty.** When a receive FIFO exists, this bit will be set high when the receive FIFO is empty. The occupancy of the FIFO is decremented with each FIFO read operation. When FIFOs don't exist, this bit is set high when the receive register has been read. This bit is cleared at the end of a successful SPI transfer.

SPI Data Transmit Register (SPIDTR)

This register is written with a data to be transmitted on the SPI bus. Once the SPE bit is set to '1' in master mode or SPISEL is active in the slave mode, the data is transferred from the SPIDTR to the shift register.

If a transfer is in progress, the data in the SPIDTR is loaded in the shift register as soon as the data in the shift register is transferred to the SPIDRR and a new transfer starts. The data is held in the SPIDTR until a subsequent write overwrites the data. The SPIDTR is shown in Figure 5, while Table 8 shows specifics of the data format.

When a transmit FIFO exists, data is written directly in the FIFO and the first location in the FIFO is treated as the SPIDTR. The pointer is decremented after completion of each SPI transfer.

This register may not be read and may only be written when it is known that space for the data is available. If an attempt to write is made on a full register or FIFO, then the PLB write transaction completes with an error condition. Reading to the SPIDTR is not allowed and the read transaction will result in undefined data.

Figure 5: **SPI Data Transmit Register (C_BASEADDR + 0x68)**

Table 8: **SPI Data Transmit Register (SPIDTR) Description (C_BASEADDR + 0x68)**

Bit(s)	Name	Core Access	Reset Value	Description
0 - [31-N]	Reserved	N/A	N/A	Reserved
[31-N+1] - 31	Tx Data<RD Red><SP Superscript>(1) (D_0 - D_{N-1})	Write only	0	N-bit SPI transmit data. N can be 8, 16 or 32. The bit postion 31 represents N-1 data bit. N = 8 when C_NUM_TRANSFER_BITS = 8 N = 16 when C_NUM_TRANSFER_BITS = 16 N = 32 when C_NUM_TRANSFER_BITS = 32

Notes:

1. The D_{N-1} bit will always represent the MSB bit irrespective of "LSB first" or "MSB first" transfer selection.

SPI Data Receive Register (SPIDRR)

This register is used to read data that is received from the SPI bus. This is a double buffered register. The received data is placed in this register after each complete transfer. The SPI architecture does not provide any means for a slave to throttle traffic on the bus; consequently, the SPIDRR is updated following each completed transaction only if the SPIDRR was read prior to the last SPI transfer. If the SPIDRR was not read (i.e. is full), then the most recently transferred data will be lost and a receive over-run interrupt will occur. The same condition can occur with a master SPI device as well.

For both master and slave SPI devices with a receive FIFO, the data is buffered in the FIFO. The receive FIFO is a read only buffer. If an attempt to read an empty receive register or FIFO is made, then the PLB read transaction completes with an error condition. The effect is undefined if an attempt is made to write the SPIDRR. The write transaction is not recommended and if it does so then will not affect the register contents. The SPIDRR is shown in Figure 6, while the specifics of the data format is described in Table 9.

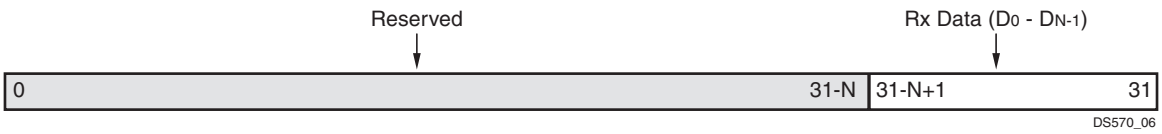

Figure 6: **SPI Data Receive Register (C_BASEADDR + 0x6C)**

Table 9: **SPI Data Receive Register (SPIDRR) Description (C_BASEADDR + 0x6C)**

Bit(s)	Name	Core Access	Reset Value	Description
0 - [31-N]	Reserved	N/A	N/A	Reserved
[31-N+1] - 31	Rx Data<RD Red><SP Superscript>(1) (D_0 - D_{N-1})	Read only	0	N-bit SPI receive data. N can be 8, 16 or 32. The bit postion 31 represents N-1 data bit. N = 8 when C_NUM_TRANSFER_BITS = 8 N = 16 when C_NUM_TRANSFER_BITS = 16 N = 32 when C_NUM_TRANSFER_BITS = 32

Notes:

1. The D_{N-1} bit will always represent the MSB bit irrespective of "LSB first" or "MSB first" transfer selection.

SPI Slave Select Register (SPISSR)

This register contains an active-low, one-hot encoded slave select vector $\overline{SS}$ of length N, where N is the number of slaves set by parameter C_NUM_SS_BITS. The bits of $\overline{SS}$ occupy the right-most bits of the register. At most one bit may be asserted low. This bit denotes the slave with whom the local master will communicate.

The bit assignment in the SPISSR is shown in Figure 7 and described in Table 10.

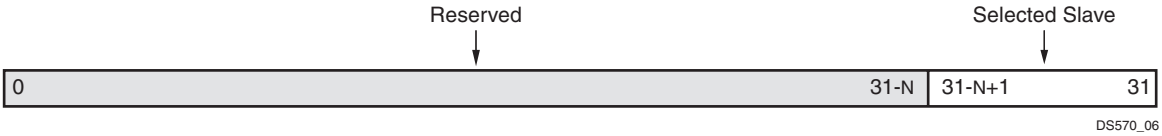

Figure 7: **SPI Slave Select Register (C_BASEADDR + 0x70)**

Table 10: **SPI Slave Select Register (SPISSR) Description (C_BASEADDR + 0x70)**

Bit(s)	Name	Core Access	Reset Value	Description
0 - [31-N]	Reserved	N/A	N/A	Reserved
[31-N+1] - 31	Selected Slave	R/W	1	Active-low, one-hot encoded slave select vector of length N-bits. N must be less than or equal to the databus width (32-bit). The slaves are numbered right to left starting at zero with the LSB. The slave numbers correspond to the indexes of signal $\overline{SS}$.

SPI Transmit FIFO Occupancy Register (Tx_FIFO_OCY)

The SPI Transmit FIFO Occupancy Register is present if and only if XPS SPI IP Core is configured with FIFOs (C_FIFO_EXIST = 1). If it is present and if the Transmit FIFO is not empty, the register contains a four-bit, right-justified value that is one less than the number of elements in the FIFO (occupancy minus one).

This register is a read only. The effect of a write to it (or of a read when the FIFO is empty) will not affect the register contents. The write operation is not recommended on this register. The only reliable way to determine that the FIFO

is empty is by reading the Tx_Empty status bit in the SPI Status Register or the DTR Empty bit in the Interrupt Status Register.

The Transmit FIFO Occupancy register is shown in Figure 8, while the specifics of the data format is described in Table 11.

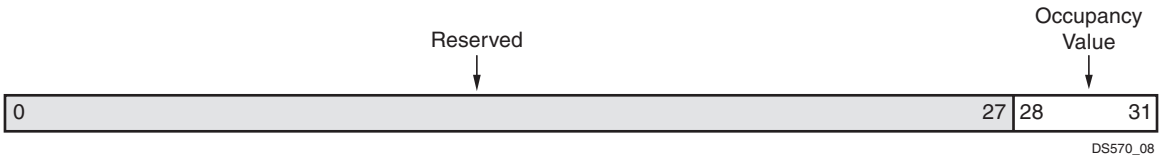

Figure 8: **SPI Transmit FIFO Occupancy Register (C_BASEADDR + 0x74)**

Table 11: **SPI Transmit FIFO Occupancy Register Description (C_BASEADDR + 0x74)**

Bit(s)	Name	Core Access	Reset Value (hex)	Description
0 - 27	Reserved	N/A	N/A	Reserved
28 - 31	Occupancy Value	Read	0	Bit 28 is the MSB. The binary value plus 1 yields the occupancy.

SPI Receive FIFO Occupancy Register (Rx_FIFO_OCY)

The SPI Receive FIFO Occupancy Register is present if and only if XPS SPI IP Core is configured with FIFOs (C_FIFO_EXIST = 1). If it is present and if the Receive FIFO is not empty, the register contains a four-bit, right-justified value that is one less than the number of elements in the FIFO (occupancy minus one).

This register is a read only. The effect of a write to it (or of a read when the FIFO is empty) will not affect the register contents. The write operation is not recommended on this register. The only reliable way to determine that the FIFO is empty is by reading the Rx_Empty status bit in the SPI Status Register.

The Receive FIFO Occupancy register is shown in Figure 9, while the specifics of the data format is described in Table 12.

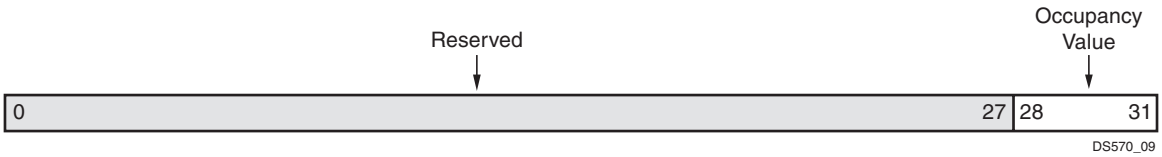

Figure 9: **SPI Receive FIFO Occupancy Register (C_BASEADDR + 0x78)**

Table 12: **SPI Receive FIFO Occupancy Register Description (C_BASEADDR + 0x78)**

Bit(s)	Name	Core Access	Reset Value (hex)	Description
0 - 27	Reserved	N/A	N/A	Reserved
28 - 31	Occupancy Value	Read	0	Bit 28 is the MSB. The binary value plus 1 yields the occupancy.

XPS SPI IP Core Interrupt Register Description

The XPS SPI IP Core has number of distinct interrupts that are sent to the interrupt controller module which is one of the sub-modules of XPS SPI IP Core. The Interrupt controller module allows each interrupt to be enabled independently (via the IP interrupt enable register (IPIER)).

The interrupt registers are in the interrupt module. The XPS SPI IP Core permits multiple conditions for an interrupt, or an interrupt strobe which occurs only after the completion of a transfer.

Setting the parameter C_FIFO_EXIST = 1 makes available almost all the interrupts shown in Table 14 when the core is configured in the master mode.

Setting the parameter C_FIFO_EXIST=0 will disable the interrupt for Tx FIFO Half Empty and DRR Not Empty,(bit(25) and bit (23) respectively) of IPISR and IPIER. Writing to these bits will not have any effect and reading these bits will return zero.

Device Global Interrupt Enable Register (DGIER)

The Device Global Interrupt Enable Register is used to globally enable the final interrupt output from the Interrupt controller as shown in Figure 10 and described in Table 13. This bit is a read/write bit and is cleared upon reset.

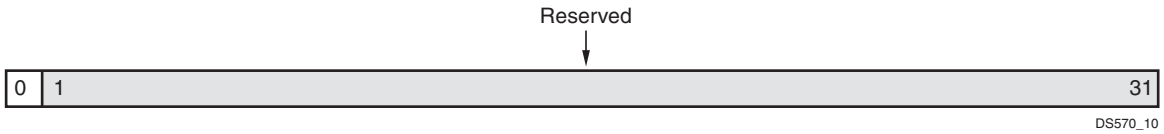

Figure 10: **Device Global Interrupt Enable Register (DGIER) (C_BASEADDR + 0x1C)**

Table 13: **Device Global Interrupt Enable Register(DGIER) Description (C_BASEADDR + 0x1C)**

Bit(s)	Name	Access	Reset Value	Description
0	GIE	R/W	'0'	**Global Interrupt Enable.** It enables all individually enabled interrupts to be passed to the interrupt controller. '0' = Disabled '1' = Enabled
1 - 31	Reserved	N/A	N/A	Reserved

IP Interrupt Status Register (IPISR)

Up to nine unique interrupt conditions are possible depending upon whether the system is configured with FIFOs or not as well as if configured in master mode or slave mode. A system without FIFOs has seven interrupts.

The interrupt controller has 32-bit Interrupt Status Register. This register collects all the interrupts events based upon the activity on the individual bits (applicable bits). These bits assignment in the Interrupt register for a 32-bit data bus is shown in Figure 11 and described in Table 14. Setting of the bits of this register is depend only upon the activities on the corresponding bit. The interrupt register is a read/toggle on write register and by writing a '1' to a bit position within the register causes the corresponding bit position in the register to 'toggle'. All register bits are cleared upon reset.

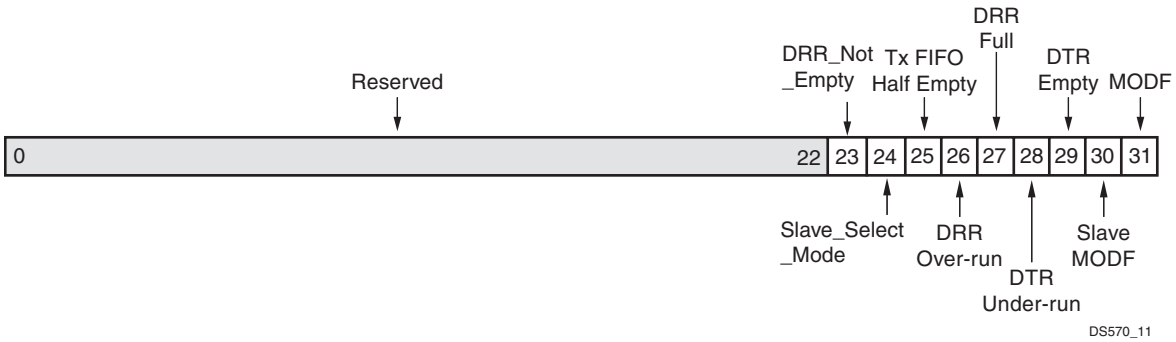

Figure 11: **Interrupt Status Register (IPISR) (C_BASEADDR + 0x20)**

Table 14: **IP Interrupt Status Register (IPISR) Description (C_BASEADDR + 0x20)**

Bit(s)	Name	Access	Reset Value	Description
0 - 22	Reserved	N/A	N/A	Reserved
23	DRR_Not_ Empty	R/TOW(1)	'0'	DRR Not Empty. IPISR bit(23) is the DRR Not Empty bit. The assertion of this bit is applicable only in case where C_FIFO_EXIST = 1 and the core is configured in slave mode. This bit is set when the DRR FIFO receives the first data during the SPI transaction. This bit is set by one-clock period strobe to the interrupt register when the core receives first data beat. Please note that assertion of this bit is applicable only when the C_FIFO_EXIST = 1 and core is configured in slave mode. In C_FIFO_EXIST = 0 this bit will always return '0'. So it is recommended to use this bit only in C_FIFO_EXIST = 1 condition when the core is configured in slave mode. In master mode, this bit always returns '0'.
24	Slave_ Select_ Mode	R/TOW(1)	'0'	Slave Select Mode. IPISR bit(24) is the Slave Select Mode bit. The assertion of this bit is applicable only when the core is configured in slave mode. This bit is set when the other SPI master core selects the core by asserting the Slave Select line. This bit is set by one-clock period strobe to the interrupt register. Please note that this bit is applicable only when the core is configured in the slave mode. In master mode, this bit always returns '0'.
25	Tx FIFO Half Empty	R/TOW(1)	'0'	**Transmit FIFO Half Empty.** IPISR bit(25) is the transmit FIFO half empty interrupt. This bit is set by a one-clock period strobe to the interrupt register when the occupancy value is decremented from "1000" to "0111". Note that "0111" means there are 8 elements in the FIFO to be transmitted. This interrupt exists only if the XPS SPI IP Core is configured with FIFOs.
26	DRR Over-run	R/TOW(1)	'0'	**Data Receive Register/FIFO Over-run.** IPISR bit(26) is the data receive FIFO over-run interrupt. This bit is set by a one-clock period strobe to the interrupt register when an attempt to write data to a full receive register or FIFO is made by the SPI core logic in order to complete an SPI transfer. This can occur when the SPI device is in either master or slave mode.
27	DRR Full	R/TOW(1)	'0'	**Data Receive Register/FIFO Full.** IPISR bit(27) is the data receive register full interrupt. Without FIFOs, this bit is set at the end of an SPI element (An element can be a byte, half-word or word depending on the value of C_NUM_TRANSFER_BITS generic) transfer by a one-clock period strobe to the interrupt register. With FIFOs, this bit is set at the end of the SPI element transfer when the receive FIFO has been filled by a one-clock period strobe to the interrupt register.

Table 14: **IP Interrupt Status Register (IPISR) Description (C_BASEADDR + 0x20) *(Cont'd)***

Bit(s)	Name	Access	Reset Value	Description
28	DTR Under-run	R/TOW(1)	'0'	**Data Transmit Register/FIFO Under-run.** IPISR bit(28) is the data transmit register/FIFO under-run interrupt. This bit is set at the end of an SPI element transfer by a one-clock period strobe to the interrupt register when data is requested from an "empty" transmit register/FIFO by the SPI core logic in order to perform an SPI transfer. This can occur only when the SPI device is configured as a slave and is enabled, i.e. SPE bit set. All zeros are loaded in the shift register and transmitted by the slave in an under-run condition.
29	DTR Empty	R/TOW(1)	'0'	**Data Transmit Register/FIFO Empty.** IPISR bit(29) is the data transmit register/FIFO empty interrupt. Without FIFOs, this bit is set at the end of an SPI element transfer by a one-clock period strobe to the interrupt register. With FIFOs, this bit is set at the end of the SPI element transfer when the transmit FIFO is emptied by a one-clock period strobe to the interrupt register. See section <RD Red>Transfer Ending Period. In the context of the M68HC11 reference manual, when configured without FIFOs, this interrupt is equivalent in information content to the complement of SPI transfer complete flag ($\overline{SPIF}$) interrupt bit. In master mode if this bit is set to '1' no more SPI transfers are permitted.
30	Slave MODF	R/TOW(1)	'0'	**Slave Mode-Fault Error.** IPISR bit(30) is the slave mode-fault error flag. This interrupt is generated if the $\overline{SS}$ signal goes active while the SPI device is configured as a slave but is not enabled. This bit is set immediately upon $\overline{SS}$ going active and continually set if $\overline{SS}$ is active and the device is not enabled.
31	MODF	R/TOW(1)	'0'	**Mode-Fault Error.** IPISR bit(31) is the mode-fault error flag. This interrupt is generated if the $\overline{SS}$ signal goes active while the SPI device is configured as a master. This bit is set immediately upon $\overline{SS}$ going active.

Notes:

1. TOW = Toggle On Write. Writing a '1' to a bit position within the register causes the corresponding bit position in the register to toggle.

IP Interrupt Enable Register (IPIER)

The Interrupt controller has a register IPIER that can cause a system level interrupt. This interrupt is generated if the enabled bit in IPIER detects any activity on corresponding IPISR bit. The IPIER has an enable bit for each defined bit of the IPISR as shown in Figure 12 and described in Table 15. All bits are cleared upon reset.

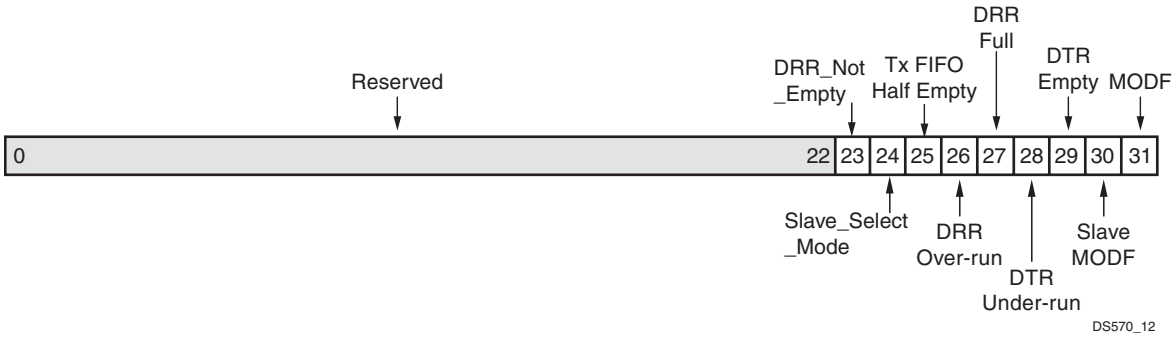

Figure 12: **IP Interrupt Enable Register (IPIER) (C_BASEADDR + 0x28)**

Table 15: **IP Interrupt Enable Register (IPIER) Description (C_BASEADDR + 0x28)**

Bit(s)	Name	Access	Reset Value	Description
0 - 22	Reserved	N/A	N/A	Reserved
23	DRR_Not_Empty	R/W	'0'	DRR_Not_Empty. '0' = Disabled '1' = Enabled Please note that setting of this bit is applicable only when the C_FIFO_EXIST = 1 and the core is configured in slave mode. If C_FIFO_EXIST = 0, setting of this bit wont affect. It means this bit won't be set in IPIER. So it is recommended to use this bit only in C_FIFO_EXIST = 1 condition when the core is configured in slave mode.
24	Slave_Select_Mode	R/W	'0'	Slave_Select_Mode. '0' = Disabled '1' = Enabled Please note that this bit is applicable only when the core is configured in the slave mode. If in the master mode, this bit is set, it won't have effect on any logic. It is recommended to use this bit only when the core is configured in slave mode.
25	Tx FIFO Half Empty	R/W	'0'	**Transmit FIFO Half Empty.** '0' = Disabled '1' = Enabled Please note that setting of this bit is applicable only when the C_FIFO_EXIST = 1. If C_FIFO_EXIST = 0, setting of this bit wont affect any logic. It means this bit wont be set in IPIER. So it is recommended to use this bit only in C_FIFO_EXIST = 1 condition.
26	DRR Over-run	R/W	'0'	**Receive FIFO Over-run.** '0' = Disabled '1' = Enabled
27	DRR Full	R/W	'0'	**Data Receive Register/FIFO Full.** '0' = Disabled '1' = Enabled
28	DTR Under-run	R/W	'0'	**Data Transmit FIFO Under-run.** '0' = Disabled '1' = Enabled
29	DTR Empty	R/W	'0'	**Data Transmit Register/FIFO Empty.** '0' = Disabled '1' = Enabled
30	Slave MODF	R/W	'0'	**Slave Mode-Fault Error Flag.** '0' = Disabled '1' = Enabled
31	MODF	R/W	'0'	**Mode-Fault Error Flag.** '0' = Disabled '1' = Enabled

XPS SPI IP Core Design Description

SPI Device Features

In addition to the features listed in the Features section, the SPI device also includes the following standard features:

- Three signal in/out (in, out, tri-state) for implementing tri-state SPI device in/outs to support multi-master configuration within the FPGA.
- Works with N times 8-bit data characters in default configuration. The default mode implements manual control of the $\overline{SS}$ output via data written to the SPISSR. This appears directly on the $\overline{SS}$ output when the master is enabled. This mode can be used only with external slave devices. In addition, an optional operation where the $\overline{SS}$ output is toggled automatically with each 8-bit character transfer by the master device. This can be selected via a bit in the SPICR for SPI master devices.
- Multi-master environment supported (implemented with tri-state drivers and requires software arbitration for possible conflict). Please refer <RD Red>SPI in Multi-Master Configuration section.
- Multi-slave environment supported (automatic generation of additional slave select output signals for the master).
- Supports maximum SPI clock rates up to one-half of the PLB clock rate in master mode and one-fourth of the PLB clock rate in slave modes, i.e. C_SCK_RATIO = 2 is not supported in Slave Mode (This is due to the synchronization issue between PLB and SPI clock). It is required to take care of PLB and external clock signal alignment when this core is configured in slave mode.
- Parameterizable baud rate generator.
- The WCOL flag is not supported as a write collision error as described in the M68HC11 reference manual. The user must take care of not writing into the transmit register when SPI data transfer is in progress.
- Back to Back transactions are supported, which means there can be multiple byte/half-word/word transfers taking place without interruption provided the transmit FIFO never gets empty and receive FIFO never gets full.
- All SPI transfers are full-duplex where an 8-bit data character is transferred from the master to the slave and an independent 8-bit data character is transferred from the slave to the master. This can be viewed as a circular 16-bit shift register; an 8-bit shift register in the SPI master device and another 8-bit shift register in a SPI slave device that are connected.

SPI in Multi-Master Configuration

The SPI bus to a given slave device (N-th device) consists of four wires, Serial Clock (SCK), Master Out Slave In (MOSI), Master In Slave Out (MISO) and Slave Select ($\overline{SS}$(N)). The signals SCK, MOSI and MISO are shared for all slaves and masters.

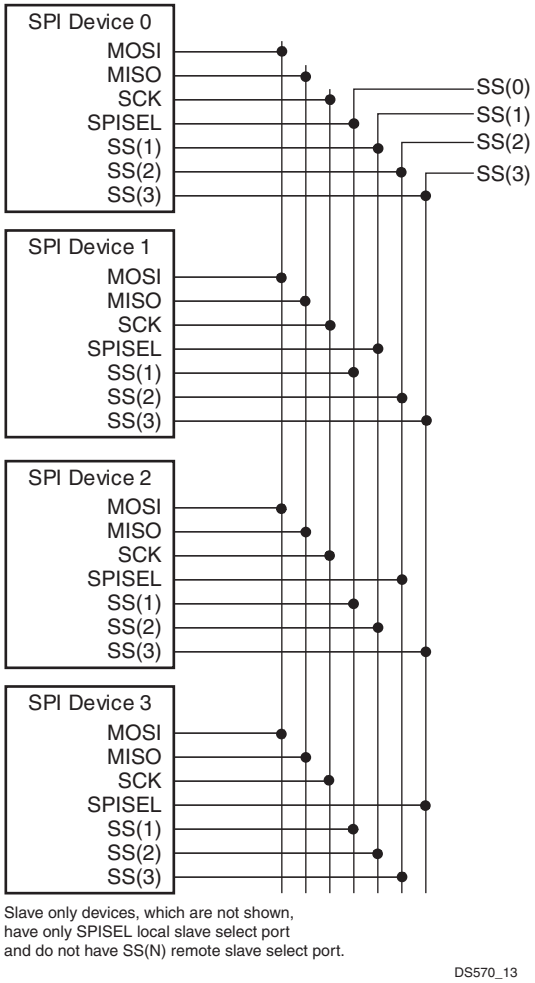

Figure 13: **Multi-Master Configuration Block Diagram**

Each master SPI device has the functionality to generate an active-low, one-hot encoded $\overline{SS}$(N) vector where each bit is assigned an $\overline{SS}$ signal for each slave SPI device. It is possible for SPI master/slave devices to be both internal to the FPGA and SPI slave devices to be external to the FPGA. SPI pins will be automatically generated through Xilinx Platform Generator when interfacing to an external SPI slave device. Multiple SPI master/slave devices are shown in Figure 13.

Optional FIFOs

The user has the option to include FIFOs in the XPS SPI IP Core as shown in Figure 1. Since SPI is full-duplex, both transmit and receive FIFOs are instantiated as a pair.

When FIFOs are implemented, the slave select address is required to be the same for all data buffered in the FIFOs. This is required because a FIFO for the slave select address is not implemented. Both transmit and receive FIFOs are 16 elements deep and are accessed via single PLB transactions since burst mode is not supported.

The transmit FIFO is write-only. When data is written in the FIFO, the occupancy number is incremented and when an SPI transfer is completed, the number is decremented. As a consequence of this operation, aborted SPI transfers still has the data available for the transmission retry. The transfers can only be aborted in the master mode by setting Master Transaction Inhibit bit, bit(23) of SPICR to '1' during a transfer. Setting this bit in the slave mode has no affect on the operation of the slave. These aborted transfers are on the SPI interface. The occupancy number is a read-only register.

If a write is attempted when the FIFO is full, then acknowledgement is given along with an error signal generation. Interrupts associated with the transmit FIFO include data transmit FIFO empty, transmit FIFO half empty and transmit FIFO under-run. See the section on <RD Red>XPS SPI IP Core Interrupt Register Description for details.

The receive FIFO is read-only. When data is read from the FIFO, the occupancy number is decremented and when an SPI transfer is completed, the number is incremented. If a read is attempted when the FIFO is empty, then acknowledgement is given along with an error signal generation. When the receive FIFO becomes full, the receive FIFO full interrupt is generated.

Data is automatically written to the FIFO from the SPI module shift register after the completion of an SPI transfer. If the receive FIFO is full and more data is received, then a receive FIFO overflow interrupt is issued. When this happens, all data attempted to be written to the full receive FIFO by the SPI module is lost.

The SPI transfers, when the XPS SPI IP Core is configured with FIFOs, can be started in two different ways depending on when the enable bit in the SPICR is set. If the enable bit is set prior to the first data being loaded in the FIFO, then the SPI transfer begins immediately after the write to the master transmit FIFO. If the FIFO is emptied via SPI transfers before additional elements are written to the transmit FIFO, an interrupt will be asserted. When the PLB to SPI SCK frequency ratio is sufficiently small, this scenario is highly probable.

Alternatively, the FIFO can be loaded up to 16 elements and then the enable bit can be set which starts the SPI transfer. In this case, an interrupt is issued after all elements are transferred. In all cases, more data can be written to the transmit FIFOs to increase the number of elements transferred before emptying the FIFOs.

Local Master Loopback Operation

Local master loopback operation, although not included in the M68HC11 reference manual, has been implemented to expedite testing. This operation is selected via setting the loop bit in the SPICR, the transmitter output is internally connected to the receiver input. The receiver and transmitter operate normally, except that received data (from remote slave) is ignored. This operation is relevant only when the SPI device is configured as a master.

Hardware Error Detection

The SPI architecture relies on software controlled bus arbitration for multi-master configurations to avoid conflicts and errors but limited error detection is implemented in the SPI hardware.

The first error detection mechanism to be discussed is contention error detection. This detects when an SPI device configured as a master is selected (i.e. its $\overline{SS}$ bit is asserted) by another SPI device simultaneously configured as master.

In this scenario, the master being selected as a slave immediately drives its outputs as necessary to avoid hardware damage due to simultaneous drive contention. The master also sets the mode-fault error (MODF) bit in the SPISR. This bit is automatically cleared by reading the SPISR. Following a MODF error, the master must be disabled and re-enabled with correct data. When configured with FIFOs this may require clearing the FIFOs.

A similar error detection mechanism has been implemented for SPI slave devices. The error detected is when a SPI device configured as a slave but is not enabled and is selected (i.e. its $\overline{SS}$ bit is asserted) by another SPI device. When this condition is detected, IPISR bit(30) is set by a strobe to the IPISR register.

Under-run and over-run conditions error detection is provided as well. Under-run conditions can happen only in slave mode operation. This happens when a master commands a transfer but the slave does not have data in the transmit register or FIFO for transfer. In this case, the slave under-run interrupt is asserted and the slave shift register is loaded with all zeros for transmission. Over-run can happen to both master and slave devices where a transfer occurs when the receive register or FIFO is full. During an over-run condition, the data received in that transfer is not registered (i.e. it is lost) and the IPISR over-run interrupt bit(26) is asserted.

Precautions to be Taken while Assigning the C_SCK_RATIO Parameter

XPS SPI IP Core is tested in hardware with the SPI slave devices like serial EEPROM's, ATMEL, STMicro-Electronics and Intel flash memories. Please read the data sheet of targeted SPI slave flash memory or EEPROM's for maximum speed of operation. It is user's responsibility to mention the correct values while deciding the PLB clock and selecting the C_SCK_RATIO parameter of the core. The PLB clock and the C_SCK_RATIO will decide the clock at SCK pin of XPS SPI IP Core. While using different external SPI slave devices , the C_SCK_RATIO should be set carefully and maximum of the clock supported by all the external SPI slave devices should be taken into account.

XPS SPI Core Configured in Slave Mode

The XPS SPI core can be configured in the slave mode by connecting the external master's slave select line to SPISEL and by setting bit 29 of SPI Control Register (SPICR) to '0'. All the incoming signals are synchronized to the PLB when C_SCK_RATIO > 4. Due to the tight timing requirements when C_SCK_RATIO = 4 the incoming SCK clock signal and its synchronized signals are used directly in the internal logic. Therefore it is required that the external clock be synchronized with the PLB clock when C_SCK_RATIO = 4. For other C_SCK_RATIO values, it is preferred but may not be necessary to have such synchronization.

During the slave mode operation it is strongly recommended to use the FIFO by setting C_FIFO_EXIST = 1. In the slave mode, two new interrupts are available in IPISR DRR_Not_Empty - bit 23 and Slave_Mode_Select - bit 24 along with the available interrupts. Before other SPI master starts communication, it is mandatory to fill the slave core transmit FIFO with the required data beats. Once the master starts communication, with the core configured in slave mode, the core will transfer data till the data exists in its transmit FIFO. At the end of last data beat transmitted from slave FIFO, core (in slave mode) will generate DTR Empty signal to notify that new data beats needed to be filled in its transmit FIFO before further communication started.

When the core is intended to be used in the slave mode, then the user should take care of pre-filling the core's DTR FIFO at least for 2 locations. This should be done immediately after POR to avoid any under run interrupt from the core. The external master clock should be set at higher SPI clock division ratios like 1/64 or more, which will be useful for the processor to fill the DTR FIFO of core, after receiving the slave mode select interrupt.

SPI IP Core Transfer Formats

SPI Clock Phase and Polarity Control

Software can select any of four combinations of serial clock (SCK) phase and polarity with programmable bits in the SPICR. The clock polarity (CPOL) bit selects an active high (i.e. the clock's idle state = low) or active low clock (i.e.

the clock's idle state = high). Determination of whether the edge of interest is rising or falling edge depends on the idle state of the clock (i.e. CPOL setting).

The clock phase (CPHA) bit can be set to select one of two different transfer formats. If CPHA = '0', data is valid on the first SCK edge (rising or falling) after $\overline{SS}$(N) has been asserted. If CPHA = '1', data is valid on the second SCK edge (rising or falling) after $\overline{SS}$(N) has asserted. For successful transfers the clock phase and polarity must be identical for the master SPI device and the selected slave device.

The first SCK cycle begins with a transition of SCK signal from its idle state and this denotes the start of the data transfer. Because the clock transition from idle denotes the start of a transfer, the M68HC11 specification notes that $\overline{SS}$(N) line may remain active low between successive transfers. The specification states that this format is useful in systems with a single master and single slave. In the context of the M68HC11 specification, transmit data is placed directly in the shift register upon a write to the transmit register. Consequently, it is the user's responsibility to insure that the data is properly loaded in the SPISSR register prior to the first SCK edge

The $\overline{SS}$ signal is toggled for all CPHA configurations and there is no support for SPISEL being held low. It is required that all $\overline{SS}$ signals be routed between SPI devices internally to the FPGA. Toggling the $\overline{SS}$ signal reduces FPGA resources.

The different transfer format are described in following sections.

CPHA Equals Zero Transfer Format

Figure 14 shows the timing diagram for an SPI data write cycle and Figure 15 shows the timing diagram for an SPI data read cycle when CPHA = '0'. The waveforms are shown for CPOL = '0', LSB First = '0', and the value of generic C_SCK_RATIO = 4. All PLB and SPI signals will have same relation with respect to SPLB_Clk and SCK respectively.

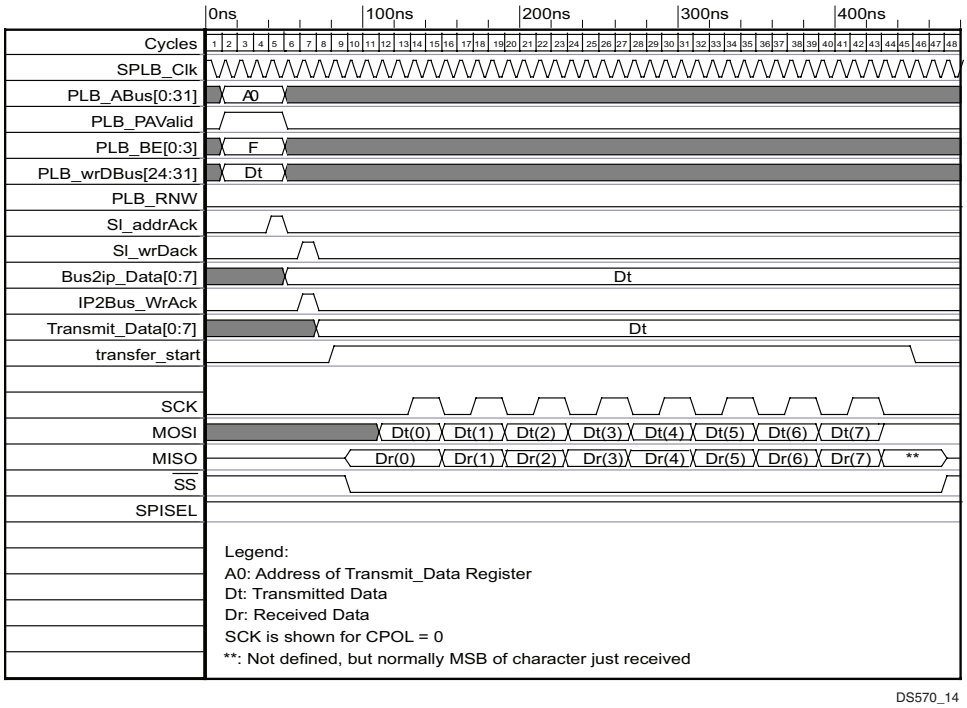

Figure 14: **Data Write Cycle on SPI Bus with CPHA = 0 and SPICR(24) = 0 for 8-bit data**

Signal SCK remains in the idle state until one-half period following the assertion of the slave select line which denotes the start of a transaction. Since assertion of the $\overline{SS}$(N) line denotes the start of a transfer, it must be de-asserted and re-asserted for sequential element transfers to the same slave device.

One bit of data is transferred per SCK clock period. Data is shifted on one edge of SCK and is sampled on the opposite edge when the data is stable. Consistent with the M68HC11 SPI specification, selection of clock polarity and a choice of two different clocking protocols on an 8-bit/16-bit/32-bit oriented data transfer is possible via bits in the SPICR.

The MOSI and MISO ports behave differently depending on whether the SPI device is configured as a master or a slave. When configured as a master, the MOSI port is a serial data output port and the MISO is a serial data input port. The opposite is true when the device is configured as a slave; the MISO port is a slave serial data output port and the MOSI is a serial data input port. There may be only one master and one slave transmitting data at any given time. The bus architecture provides limited contention error detection (i.e. multiple devices driving the shared MISO and MOSI signals) and requires the software to provide arbitration to prevent possible contention errors.

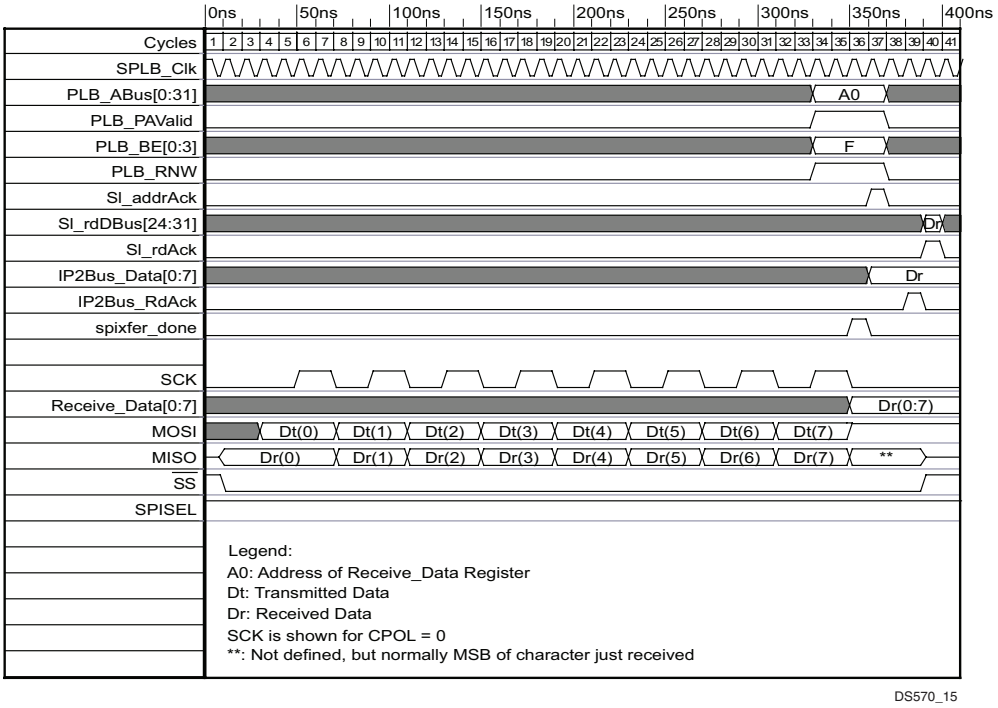

Figure 15: **Data Read Cycle on SPI Bus with CPHA = 0 and SPICR(24) = 0 for 8-bit data**

All SCK, MOSI, and MISO pins of all devices are respectively hardwired together. For all transactions, a single SPI device is configured as a master and all other SPI devices on the SPI bus are configured as slaves.

The single master drives the SCK and MOSI pins to the SCK and MOSI pins of the slaves. The uniquely selected slave device drives data out from its MISO pin to the MISO master pin, thus realizing full-duplex communication.

The Nth bit of the $\overline{SS}$(N) signal selects the Nth SPI slave with an active-low signal. All other slave devices ignore both SCK and MOSI signals. In addition, the non-selected slaves (i.e. $\overline{SS}$ pin high) drive their MISO pin to tri-state so as not to interfere with SPI bus activities.

When external slave SPI devices are implemented, SCK, MOSI and MISO, as well as the needed $\overline{SS}$(N) signals, are brought out to pins. All signals are true tri-state bus signals and erroneous external bus activity can corrupt internal transfers when both internal and external devices are present.

The user must ensure that external pull-up or pull-down of external SPI tri-state signals are consistent with the sink/source capability of the FPGA I/O drivers. Recall that the I/O drivers can be configured for different drive strengths as well as internal pull-ups.

The tri-state signals for multiple external slaves can be implemented as per system design requirements, but the external bus must follow the SPI M68HC11 specifications.

CPHA Equals One Transfer Format

With CPHA = '1', the first SCK cycle begins with an edge on the SCK line from its inactive level to active level (rising or falling depending on CPOL) as shown in Figure 16. The timing diagram for an SPI data write cycle is shown in Figure 16.

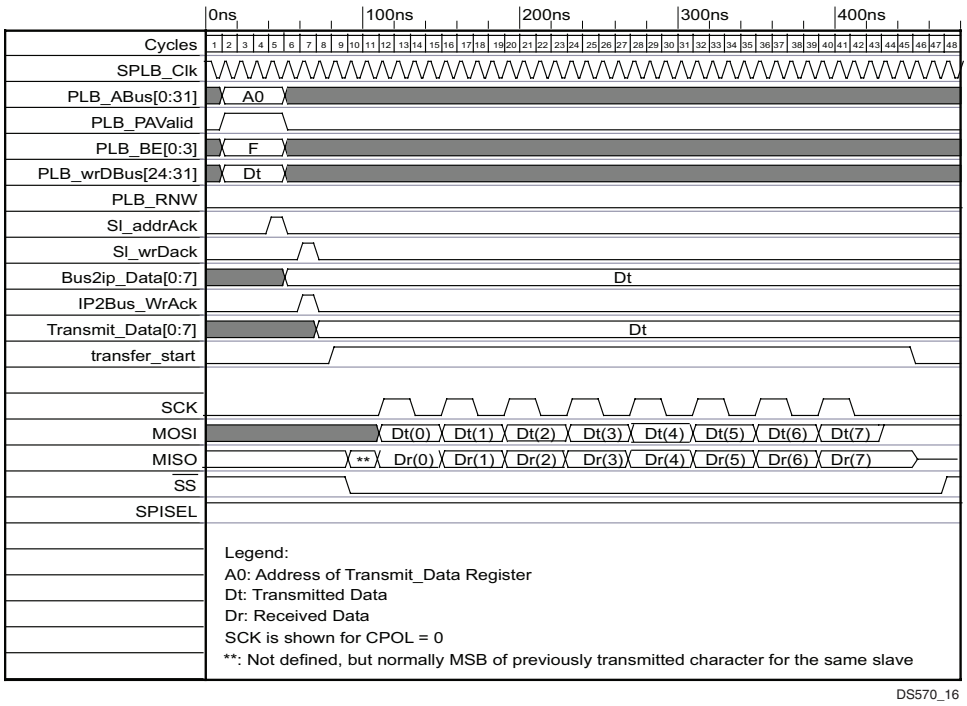

Figure 16: **Data Write Cycle on SPI Bus with CPHA = 1 and SPICR(24) = 0 for 8-bit data**

The timing diagram for an SPI data read cycle when CPHA = '1' is shown in Figure 17. The waveforms are shown for CPOL = '0', LSB First = '0', and the value of generic C_SCK_RATIO = 4. All PLB and SPI signals will have same relation with respect to SPLB_Clk and SCK respectively.

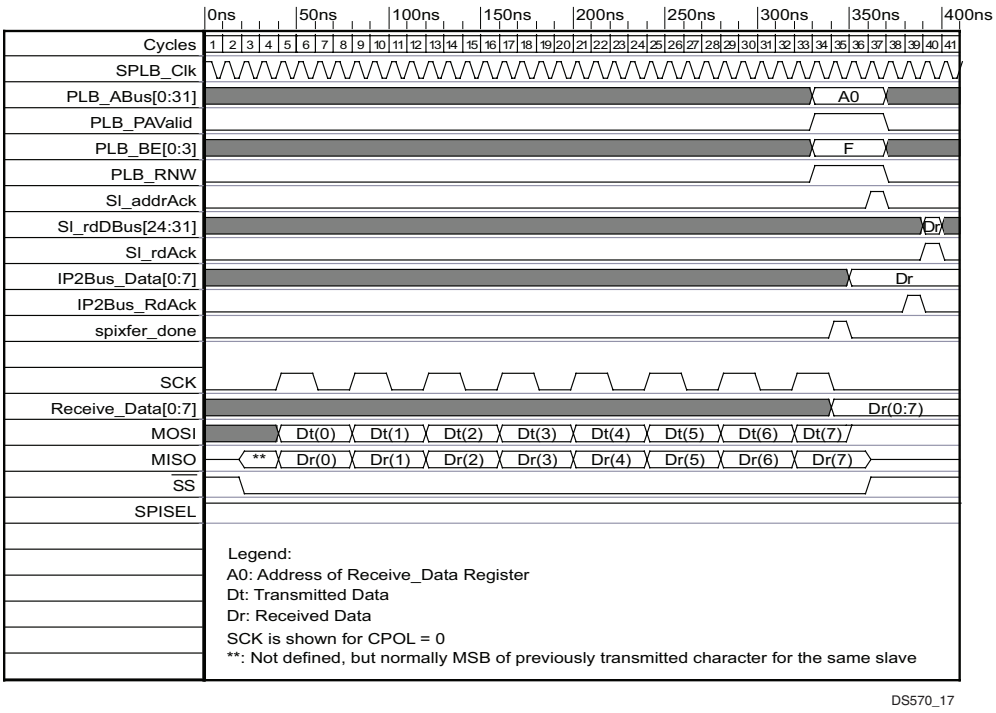

Figure 17: **Data Read Cycle on SPI Bus with CPHA = 1 and SPICR(24) = 0 for 8-bit data**

SPI Protocol Slave Select Assertion Modes

The SPI protocol is designed to have automatic slaves select assertion and manual slave select assertion which are described in the following sections. All the SPI transfer formats described in SPI Clock Phase and Polarity Control section are valid for both Automatic and Manual slave select assertion mode.

SPI Protocol with Automatic Slave Select Assertion

This section describes the SPI protocol where slave select ($\overline{SS}$(N)) is asserted automatically by the SPI master device (i.e SPICR bit(24) = '0').

This is the configuration mode provided to permit transfer of data with automatic toggling of slave select ($\overline{SS}$) signal until all the elements are transferred. In this mode the data in the SPISSR register appears on the $\overline{SS}$(N) output when the new transfer starts. After every byte (or element) transfer the $\overline{SS}$(N) output goes to '1'. The data in SPISSR register again appears on $\overline{SS}$(N) output at the beginning of new transfer. The user does not need to manually control slave select signal.

SPI Protocol with Manual Slave Select Assertion

This section briefly describes the SPI protocol where slave select ($\overline{SS}$(N)) is manually asserted by the user (i.e. SPICR bit(24) = 1).

This is the configuration mode provided to permit transfers of an arbitrary number of elements without toggling slave select until all the elements are transferred. In this mode, the data in the SPISSR register appears directly on the $\overline{SS}$(N) output.

As described earlier, SCK must be stable before the assertion of slave select. Therefore, when manual slave select mode is utilized, the SPI master must be enabled first (SPICR bit(24) = 1) to assert SCK to the idle state prior to asserting slave select.

Note that the master transfer inhibit (SPICR bit(23)) can be utilized to inhibit master transactions until the slave select is asserted manually and all data registers of FIFOs are initialized as desired. This can be utilized before the first transaction and after any transaction that is allowed to complete.

When the above rules are followed, the timing is the same as presented for the automatic slave select assertion mode with the exception that assertion of slave select signal and the number of elements transferred is controlled by the user.

Beginning and Ending SPI Transfers

The details of the beginning and ending periods depends on the CPHA format selected and whether the SPI is configured as a master or a slave. Following section describes the beginning and ending period for SPI transfers.

Transfer Beginning Period

The definition of the transfer beginning period for the XPS SPI IP Core is consistent with the M68HC11 reference manual. This manual can be referenced for more details. All SPI transfers are started and controlled by a master SPI device.

As a slave, the processor considers a transfer to begin with the first SCK edge or the falling edge of $\overline{SS}$, depending on the CPHA format selected. When CPHA equals zero, the falling edge of $\overline{SS}$ indicates the beginning of a transfer. When CPHA equals one, the first edge on the SCK indicates the start of the transfer. In either CPHA format, a transfer can be aborted by de-asserting the $\overline{SS}$(N) signal. This causes the SPI slave logic and bit counters to be reset. In this implementation, the software driver can deselect all slaves (i.e. $\overline{SS}$(N) is driven high) to abort a transaction. Although the hardware is capable of changing slaves during the middle of a single or burst transfer, It is recommended that the software be designed to prevent this.

In slave configuration, the data is transmitted from the SPIDTR register on the first PLB rising clock edge following $\overline{SS}$ signal being asserted. The data should be available in the register or FIFO. If data is not available, then the under-run interrupt is asserted.

Transfer Ending Period

The definition of the transfer ending period for the XPS SPI IP Core is consistent with the M68HC11 reference manual. The SPI transfer is signaled complete when the SPIF flag is set. However, depending on the configuration of the SPI system, there may be additional tasks to be performed before the system can consider the transfer complete.

When configured without FIFOs, the Rx_Full bit, bit(30) in the SPISR is set to denote the end of transfer. When data is available in the SPIDRR register, bit(27) of the IPISR is asserted as well. The data in the SPIDRR is sampled on the same clock edge as the assertion of the SPIDRR register Full interrupt.

When the SPI device is configured as a master without FIFOs, Rx_Empty bit, bit(31) and Tx_Full bit, bit(28) in the SPISR are cleared, Tx_Empty bit, bit(29) and Rx_Full bit, bit(30) in SPISR are set, and DRR Full bit, bit(27) and Slave MODF bit, bit(30) in the IPISR are set on the first rising PLB clock edge after the end of the last SCK cycle.

Note that the end of the last SCK cycle is a transition on SCK for CPHA = '0', but is not denoted by a transition on SCK for CPHA = '1'. See Figure 14 and Figure 16. However, the internal master clock provides this SCK edge which prompts the setting/clearing of the bits noted.

In this design, a counter was implemented which permits the simultaneous setting of SPISR and IPISR bits for both master and slave SPI devices. Note that external SPI slave devices may use an internal clock that is asynchronous to the SCK clock. This can cause status bits in the SPISR and IPISR to be inconsistent with each other. Therefore, the XPS SPI IP Core cannot be used with external slave devices that do not use the PLB clock.

When the XPS SPI IP Core is configured with FIFOs and a series of consecutive SPI 8-bit/16-bit/32-bit element transfers are performed, SPISR bits and IPISR do indicate completion of the first and the last SPI transfers with no indication of intermediate transfers. The only way to monitor when intermediate transfers are completed is to monitor the receive FIFO occupancy number. There is also an interrupt when the transmit FIFO is half empty, bit(25) of IPISR. There is another interrupt added (DRR_Not_Empty) to indicate that the receive FIFO has at least one beat of data.

When the SPI device is configured as a slave, the setting/clearing of the bits discussed above for a master coincides with the setting/clearing of the master bits for both cases of CPHA = '0' and CPHA = '1'. Recall that for CPHA = '1' (i.e. no SCK edge denoting the end of the last clock period) the slave has no way of knowing when the end of the last SCK period occurs unless an PLB clock period counter was included in the SPI slave device. In the slave mode, when the master SPI selects the core by asserting Slave Select (SS) line, the Slave_Mode_Select interrupt will be generated. This bit is added in SPISR, IPISR and IPIER registers.

SPI Registers Flow Description

This section provides information on setting the SPI registers to initiate and complete bus transactions.

SPI master device with or without FIFOs where the slave select vector is asserted manually via SPICR bit(24) assertion.

This flow permits the transfer of N number of byte/half-word/word by toggling of the slave select vector just once. This is the default mode of operation. Follow these steps to successfully complete an SPI transaction:

1. Start from proper state including SPI bus arbitration.
2. Configure DGIER and IPIER registers as desired.
3. Configure target slave SPI device as required.
4. Write initial data to master SPIDTR register/FIFO. This assumes that the SPI master is disabled.
5. Insure SPISSR register has all ones.
6. Write configuration data to master SPI device SPICR as desired including setting bit(24) for manual asserting of $\overline{SS}$ vector and setting both enable bit and master transfer inhibit bit. This initializes SCK and MOSI but inhibits transfer.
7. Write to SPISSR to manually assert $\overline{SS}$ vector.
8. Write the above configuration data to master SPI device SPICR, but clear inhibit bit which starts transfer.
9. Wait for interrupt (typically IPISR bit(27)-DRR_Full) or poll status for completion. Wait time depends on SPI clock ratio.
10. Set master transaction inhibit bit to service interrupt request. Write new data to master register/FIFOs and slave device then clear master transaction inhibit bit to continue N 8-bit element transfer. Note that an overrun of the SPIDRR register/FIFO can occur if the SPIDRR register/FIFOs are not read properly. Also note that SCK will have "stretched" idle levels between element transfers (or groups of element transfers if utilizing FIFOs) and that MOSI can transition at end of a element transfer (or group of transfers) but will be stable at least one-half SCK period prior to sampling edge of SCK.
11. Repeat previous two steps until all data is transferred.
12. Write all ones to SPISSR or exit manual slave select assert mode to deassert $\overline{SS}$ vector while SCK and MOSI are in the idle state.
13. Disable devices as desired.

SPI master and slave devices without FIFOs performing one 8-bit/16-bit/32-bit transfer (optional mode).

Follow these steps to successfully complete an SPI transaction:

1. Start from proper state including SPI bus arbitration.
2. Configure master DGIER and IPIER. Also configure slave DGIER and IPIER registers as desired.
3. Write configuration data to master SPI device SPICR as required.
4. Write configuration data to slave SPI device SPICR as required.
5. Write the active-low, one-hot encoded slave select address to the master SPISSR.
6. Write data to slave SPIDTR register as required.
7. Write data to master SPIDTR register to start transfer.
8. Wait for interrupt (typically IPISR bit(30)) or poll status for completion.
9. Read IPISR of both master and slave SPI devices as required.
10. Perform interrupt requests as required.
11. Read SPISR of both master and slave SPI devices as required.
12. Perform actions as required or dictated by SPISR data.

SPI master and slave devices where registers/FIFOs are filled before SPI transfer is started and multiple discrete 8-bit transfers are performed (optional mode)

Follow these steps to successfully complete an SPI transaction:

1. Start from proper state including SPI bus arbitration.
2. Configure master DGIER and IPIER. Also configure slave DGIER and IPIER registers as desired.
3. Write configuration data to master SPI device SPICR as required, don't enable the transaction.
4. Write configuration data to slave SPI device SPICR as required in slave mode.
5. Write the active-low, one-hot encoded slave select address to the master SPISSR.
6. Write all data to slave SPIDTR Register/FIFO as required.
7. Write all data to master SPIDTR Register/FIFO.
8. Write enable bit to master SPICR which starts transfer.
9. Wait for interrupt (typically IPISR bit(27)) or poll status for completion.
10. Read IPISR of both master and slave SPI devices as required.
11. Perform interrupt requests as required.
12. Read SPISR of both master and slave SPI devices as required.
13. Perform actions as required or dictated by SPISR data.

SPI master and slave devices with FIFOs where some initial data is written to FIFOs, the SPI transfer is started, data is written to the FIFOs as fast or faster than the SPI transfer and multiple discrete 8-bit transfers are performed (optional mode).

Follow these steps to successfully complete an SPI transaction:

1. Start from proper state including SPI bus arbitration.
2. Configure master DGIER and IPIER. Also configure slave DGIER and IPIER registers as desired.
3. Write configuration data to master SPI device SPICR as required.
4. Write configuration data to slave SPI device SPICR as required.
5. Write the active-low, one-hot encoded slave select address to the master SPISSR.

6. Write initial data to slave transmit FIFO as required.
7. Write initial data to master transmit FIFO.
8. Write enable bit to master SPICR which starts transfer.
9. Continue writing data to both master and slave FIFOs.
10. Wait for interrupt (typically IPISR bit(27)) or poll status for completion.
11. Read IPISR of both master and slave SPI devices as required.
12. Perform interrupt requests as required.
13. Read SPISR of both master and slave SPI devices as required.
14. Perform actions as required or dictated by SPISR data.

Design Constraints

Note: An example UCF for this core is available and must be modified for use in the system. Please refer to the *EDK Getting Started Guide* for the location of this file.

Timing Constraints

For complete timing coverage, it is recommended that Flip-Flop edge to edge constraint is specified along with SPLB_Clk input constraint. An example is shown in Figure 18.

```
NET "SPLB_Clk" TNM_NET = "splb_clk;
TIMESPEC "TS_splb_clk" = PERIOD "splb_clk" 10 ns HIGH 50%;
```

DS570_18

Figure 18: **Timing Constraints**

Design Implementation

Target Technology

The target technology is an FPGA listed in the Supported Device Family field in the LogiCORE IP Facts Table.

Device Utilization and Performance Benchmarks

Core Performance

Since the XPS SPI IP Core will be used with other design modules in the FPGA, the utilization and timing numbers reported in this section are estimates only. When the XPS SPI IP Core is combined with other designs in the system, the utilization of FPGA resources and timing of the XPS SPI IP Core design will vary from the results reported here.

The XPS SPI IP Core resource utilization for various parameter combinations measured with the Virtex-4 FPGA as the target device are detailed in Table 16.

Table 16: **Performance and Resource Utilization Benchmarks on the Virtex-4 FPGA (xc4vfx12-ff668-10)**

Parameter Values (Other parameters at default values)				Device Resources			Performance
C_FIFO_EXIST	C_SCK_RATIO	C_NUM_SS_BITS	C_NUM_TRANSFER_BITS	Slices	Slice Flip-Flops	LUTs	Fmax (MHz)
0	2	2	8	221	174	304	149
1	2	2	8	290	188	349	149
0	4	2	8	290	188	349	148
1	4	2	8	309	186	293	136
0	16	2	8	305	185	292	145
1	16	2	8	309	186	293	136
0	32	2	8	289	189	350	148
1	32	2	8	228	170	277	178

The XPS SPI IP Core resource utilization for various parameter combinations measured with the Virtex-5 FPGA as the target device are detailed in Table 17.

Table 17: **Performance and Resource Utilization Benchmarks on the Virtex-5 FPGA (xc5vlx50-ff676-1)**

Parameter Values (Other parameters at default values)				Device Resources		Performance
C_FIFO_EXIST	C_SCK_RATIO	C_NUM_SS_BITS	C_NUM_TRANSFER_BITS	Slice Flip-Flops	LUTs	Fmax (MHz)
0	2	2	8	170	173	174
1	2	2	8	189	256	154
0	4	2	8	174	233	167
1	4	2	8	185	254	166
0	16	2	8	185	227	189
1	16	2	8	189	257	172

Table 17: **Performance and Resource Utilization Benchmarks on the Virtex-5 FPGA (xc5vlx50-ff676-1)**

Parameter Values (Other parameters at default values)				Device Resources		Performance
C_FIFO_EXIST	C_SCK_RATIO	C_NUM_SS_BITS	C_NUM_TRANSFER_BITS	Slice Flip-Flops	LUTs	Fmax (MHz)
0	32	2	8	184	226	180
1	32	2	8	188	257	171

The XPS SPI IP Core resource utilization for various parameter combinations measured with the Spartan-3A FPGA as the target device are detailed in Table 18.

Table 18: **Performance and Resource Utilization Benchmarks on the Spartan-3A FPGA (xc3s700a-fg484-4)**

Parameter Values (other parameters at default values)				Device Resources			Performance
C_FIFO_EXIST	C_SCK_RATIO	C_NUM_SS_BITS	C_NUM_TRANSFER_BITS	Slices	Slice Flip-Flops	LUTs	Fmax (MHz)
0	2	2	8	235	170	235	103
1	2	2	8	264	188	306	107
0	4	2	8	273	174	292	104
1	4	2	8	236	182	257	105
0	16	2	8	281	185	303	104
1	16	2	8	248	185	260	104
0	32	2	8	266	186	261	103
1	32	2	8	279	189	307	108

The XPS SPI IP Core resource utilization for various parameter combinations measured with the Spartan-6 FPGA as the target device are detailed in Table 19.

Table 19: **Performance and Resource Utilization Benchmarks on the Spartan-6 FPGA (xc6slx45tfgg484-2)**

Parameter Values (other parameters at default values)				Device Resources			Performance
C_FIFO_EXIST	C_SCK_RATIO	C_NUM_SS_BITS	C_NUM_TRANSFER_BITS	Slices	Slice Flip-Flops	LUTs	Fmax (MHz)
0	2	2	8	98	171	180	121
1	2	2	8	114	176	222	115
0	4	2	8	104	171	182	124
1	4	2	8	114	176	222	124
0	16	2	8	106	176	221	117
1	16	2	8	131	189	261	122
0	32	2	8	113	182	220	117
1	32	2	8	136	190	254	118

The XPS SPI IP Core resource utilization for various parameter combinations measured with the Virtex-6 FPGA as the target device are detailed in Table 20.

Table 20: **Performance and Resource Utilization Benchmarks on the Virtex-6 FPGA (xc6vlx130tff1156-1)**

Parameter Values (other parameters at default values)				Device Resources			Performance
C_FIFO_EXIST	C_SCK_RATIO	C_NUM_SS_BITS	C_NUM_TRANSFER_BITS	Slices	Slice Flip-Flops	LUTs	Fmax (MHz)
0	2	2	8	98	171	189	177
1	2	2	8	131	175	243	177
0	4	2	8	114	175	243	175
1	4	2	8	115	186	212	173
0	16	2	8	137	190	292	188
1	16	2	8	124	185	217	184

Table 20: **Performance and Resource Utilization Benchmarks on the Virtex-6 FPGA (xc6vlx130tff1156-1)**

Parameter Values (other parameters at default values)				Device Resources			Performance
C_FIFO_EXIST	C_SCK_RATIO	C_NUM_SS_BITS	C_NUM_TRANSFER_BITS	Slices	Slice Flip-Flops	LUTs	Fmax (MHz)
0	32	2	8	126	186	291	182
1	32	2	8	137	190	292	188

System Performance

To measure the system performance (Fmax) of this XPS SPI IP core, it was added as the Device Under Test (DUT)to a Virtex-4 FPGA system as shown in Figure 19, a Virtex-5 FPGA system as shown in Figure 20, a Spartan-3A FPGA system as shown in Figure 21, a Spartan-6 FPGA system as shown in Figure 22, and a Virtex-6 FPGA system as shown in Figure 23.

Because the XPS SPI IP Core will be used with other design modules in the FPGA, the utilization and timing numbers reported in this section are estimates only. When this core is combined with other designs in the system, the utilization of FPGA resources and timing of the core design will vary from the results reported here.

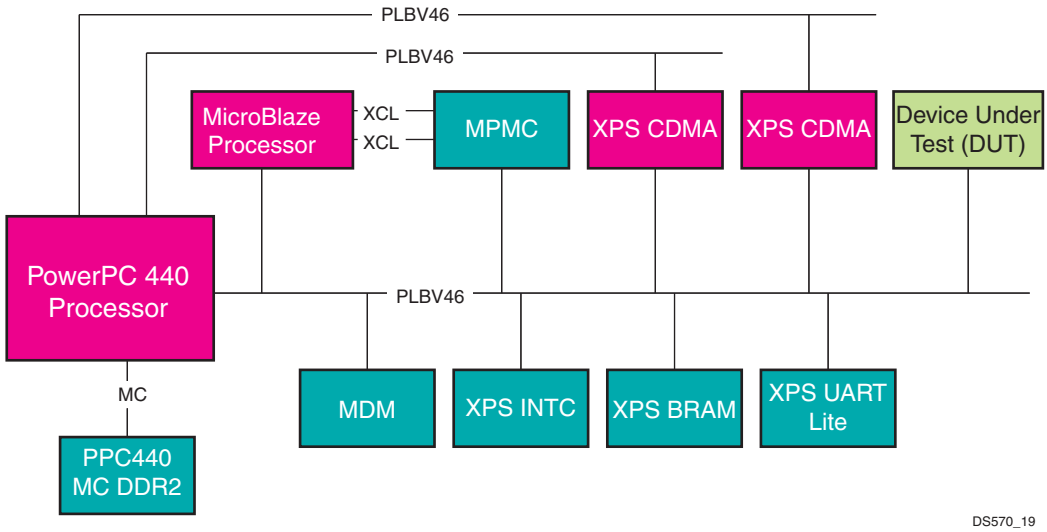

Figure 19: **Virtex-4 FX FPGA System with the XPS SPI Core as the DUT**

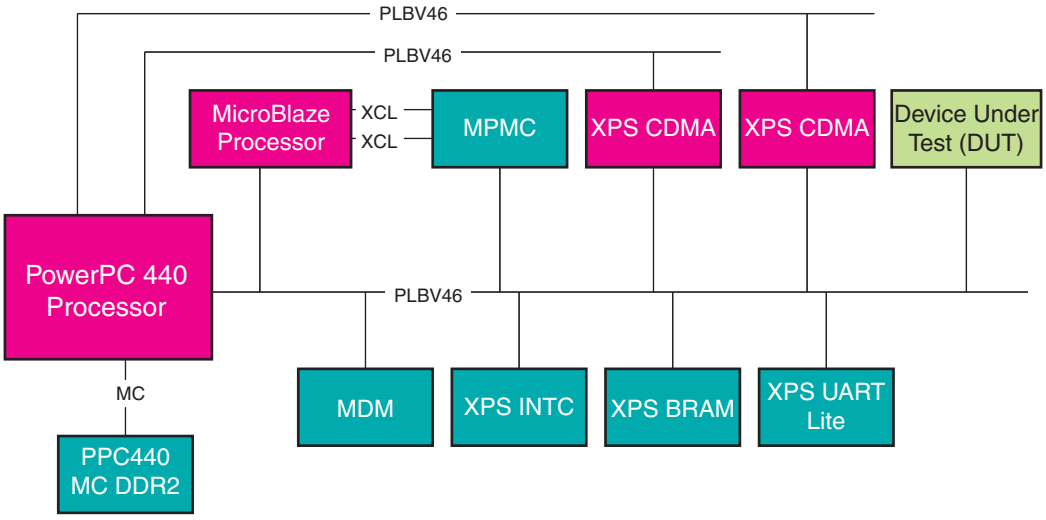

Figure 20: **Virtex-5 FX FPGA System with the XPS SPI Core as the DUT**

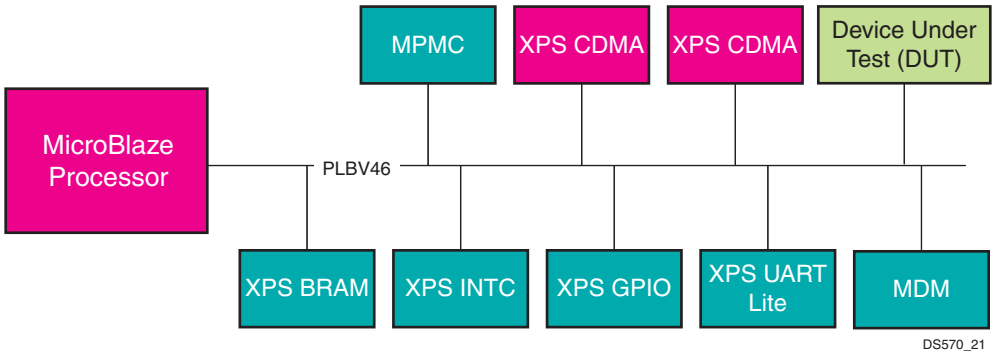

Figure 21: **Spartan-3A System with the XPS SPI Core as the DUT**

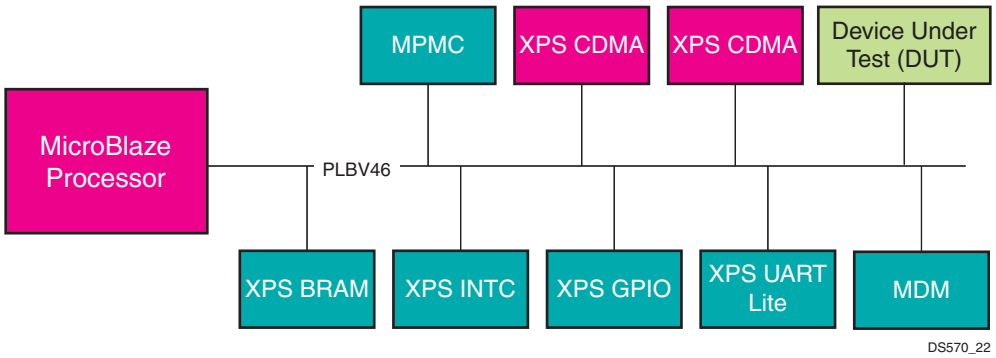

Figure 22: **Spartan-6 System with the XPS SPI Core as the DUT**

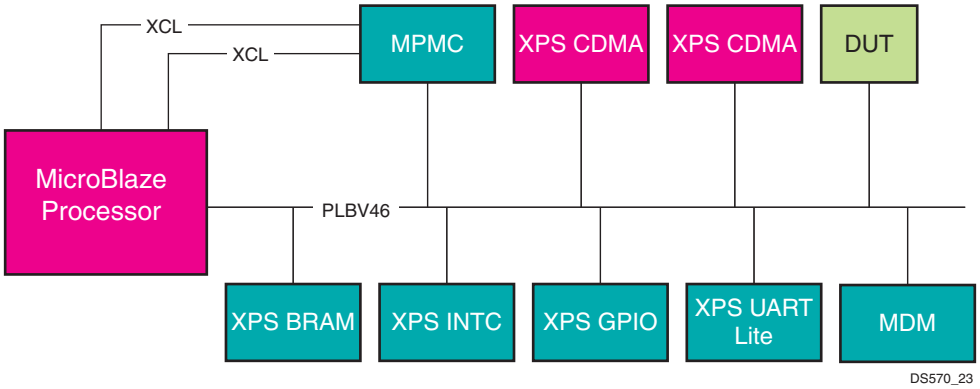

Figure 23: **Virtex-6 System with the XPS SPI Core as the DUT**

The target FPGA was then filled with logic to drive the LUT and BRAM utilization to approximately 70% and the I/O utilization to approximately 80%. Using the default tool options and the slowest speed grade for the target FPGA, the resulting target F_{MAX} numbers are shown in Table 21.

Table 21: XPS SPI IP Core System Performance

Target FPGA	Target F_{MAX} (MHz)
S3A700 -4	90
V4FX60 -10	100
V5LXT50 -1	120
V6LXT130-1	150
S6LXT45-2	100

The target F_{MAX} is influenced by the exact system and is provided for guidance. It is not a guaranteed value across all systems.

Specification Exceptions

Exceptions from the Motorola M68HC11-Rev. 4.0 Reference Manual

1. A slave mode-fault -error interrupt is added to provide an interrupt if a SPI device is configured as a slave and is selected when not enabled. A Slave_Mode_Select interrupt is added to indicate that the core is selected in the slave mode when its SPISEL is asserted by master SPI.
2. In this design, the SPIDTR and SPIDRR registers have independent addresses. This is an exception to the M68HC11 specification which calls for two registers to have the same address.
3. All $\overline{SS}$ signals are required to be routed between SPI devices internally to the FPGA. This is because toggling of the $\overline{SS}$ signal is utilized in slaves to minimize FPGA resources.
4. Manual control of the $\overline{SS}$ signals is provided by setting bit(24) in the SPICR register. When the device is configured as a master and is enabled and bit(24) of the SPICR register is set, the vector in the SPISSR register is asserted. When this mode is enabled, multiple elements can be transferred without toggling the $\overline{SS}$ vector.
5. A control bit is provided to inhibit master transfers. This bit is effective in any master mode, but has main utility in manual control of the $\overline{SS}$ signals.
6. In the M68HC11 implementation, the transmit register is transparent to the shift register which necessitates the write collision error (WCOL) detection hardware. This is not implemented in this design.

7. The interrupt enable bit (SPIE) defined by the M68HC11 specifications which resides in the M68HC11 control register has been moved to the IPIER register. In the position of the SPIE bit, there is a bit to select local master loopback mode for testing.
8. An option is implemented in this FPGA design to implement FIFOs on both transmit and receive (Full Duplex only) mode.
9. M68HC11 implementation supports only byte transfer. In this design either a byte, half-word or word transfer can be configured via a generic C_NUM_TRANSFER_BITS.
10. The baud rate generator is specified by Motorola to be programmable via bits in the control register; however, in this FPGA design the baud rate generator is programmable via parameters in the VHDL implementation. Thus, in this implementation run time configuration of baud rate is not possible. Furthermore, in addition to the ratios of 2, 4, 16 and 32, all integer multiples of 16 up to 2048 are allowed.
11. The XPS SPI IP Core is tested with Atmel AT45DB161D and ST Microelectronics M25P16 serial SPI slave devices. These devices support the SPI mode 0 and mode 3. These devices have data valid time of 8 ns from the falling edge of SCK. While operating with these devices at higher speed of 50 MHz (most of the instructions supports this speed), the core should be configured in C_SCK_RATIO = 2 mode (where the PLB is configured to operate at 100 MHz). Due to limited time availability in the design as well as real SPI slave behavior for data change, the data in the SPI core will be registered in the middle of each falling edge and next consecutive rising edge. As per the M68HC11 document, the master should register data on each rising edge of SCK in SPI mode 1 and 3. Please note that the data registering mechanism in case of C_SCK_RATIO =2, follows different pattern than specified in the standard. Although this is applicable to the data registering mechanism in IP core only. The SPI core when configured in master mode, changes data on each falling edge and this behavior is as per the M68HC11 standard.
12. When XPS SPI IP core is configured in the slave mode, the data in the core will be registered on the SCK rising edge + 1 PLB clock signal. Internally, this data is registered on the next rising edge of PLB. The core changes the data on the SCK falling edge + PLB clock cycle.

Reference Documents

The following documents contain reference information important to understanding the XPS SPI IP Core design:

1. Motorola M68HC11-Rev. 4.0 Reference Manual
2. *Motorola MPC8260 PowerQUICC II™ Users Manual* 4/1999 Rev. 0
3. IBM CoreConnect™ *128-Bit Processor Local Bus, Architectural Specification* (v4.6)
4. DS561 PLBV46_Slave_Single
5. Spartan-3AN FPGA In-System Flash User Guide,UG333

Support

Xilinx provides technical support for this LogiCORE product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY*.

Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx ISE Design Suite Embedded Edition software under the terms of the Xilinx End User License. The core is generated using the Xilinx ISE Embedded Edition software (EDK).

Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE modules and software, please contact your local Xilinx sales representative.

Revision History

Date	Version	Revision
03/03/08	2.0	Updated and redesigned version.
05/12/08	2.1	Revision is changed. Updated with 16-bit, 32-bit support.
7/24/08	2.1	Added QPro Virtex-4 Hi-Rel and QPro Virtex-4 Rad Tolerant support.
11/18/08	2.2	Updated for new core version.
4/24/09	2.3	Replaced references to supported device families and tool name(s) with hyperlink to PDF file.
7/17/09	2.4	Updated minor version
3/30/10	2.5	Updated version to fix CR: 543500. Added two interrupt bits and other functionality in the core.
7/23/10	2.6	Updated to 12.2; converted to new DS template; updated images.

Notice of Disclaimer