AXI Quad SPI v3.2

LogiCORE IP Product Guide

Vivado Design Suite

PG153 August 6, 2021

XILINX

Table of Contents

Introduction

The LogiCORE™ IP AXI Quad Serial Peripheral Interface (SPI) core connects the AXI4 interface to those SPI slave devices that support the Standard, Dual, or Quad SPI protocol instruction set. This core provides a serial interface to SPI slave devices. The Dual/Quad SPI is an enhancement to the standard SPI protocol (described in the Motorola M68HC11 data sheet) and provides a simple method for data exchange between a master and a slave.

Features

- Configurable AXI4 interface; when configured with an AXI4-Lite interface the core is backward compatible with version 1.00 of the core (legacy mode)
- Configurable AXI4 interface for burst mode operation for the Data Receive Register (DRR) and the Data Transmit Register (DTR) FIFO
- Configurable eXecute In Place (XIP) mode of operation
- Connects as a 32-bit slave on either AXI4-Lite or AXI4 interface
- Configurable SPI modes:
  - Standard SPI mode
  - Dual SPI mode
  - Quad SPI mode
- Programmable SPI clock phase and polarity
- Configurable FIFO depth (16 or 256 element deep in Dual/Quad/Standard SPI mode) and fixed FIFO depth of 16 in XIP mode
- Configurable Slave Memories in dual and quad modes are: Mixed, Micron, Winbond, and Spansion (Beta Version)

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	UltraScale+™ UltraScale™ Zynq®-7000 SoC 7 Series FPGAs
Supported User Interfaces	AXI4, AXI4-Lite
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	VHDL
Example Design	VHDL
Test Bench	VHDL
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Not Provided
Supported S/W Driver[2]	Standalone and Linux
Tested Design Flows[3]	
Design Entry	Vivado® Design Suite
Simulation	For a list of supported simulators, see the Xilinx Design Tools: Release Notes Guide
Synthesis	Vivado synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 54408
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. Standalone driver details can be found in the Vitis directory (<install_directory>/Vitis/<release>/data/embeddedsw/doc/xilinx_drivers.htm) on the Xilinx Wiki page.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The top-level block diagram for the AXI Quad SPI core when configured with the AXI4-Lite interface option is shown in Figure 1-1.

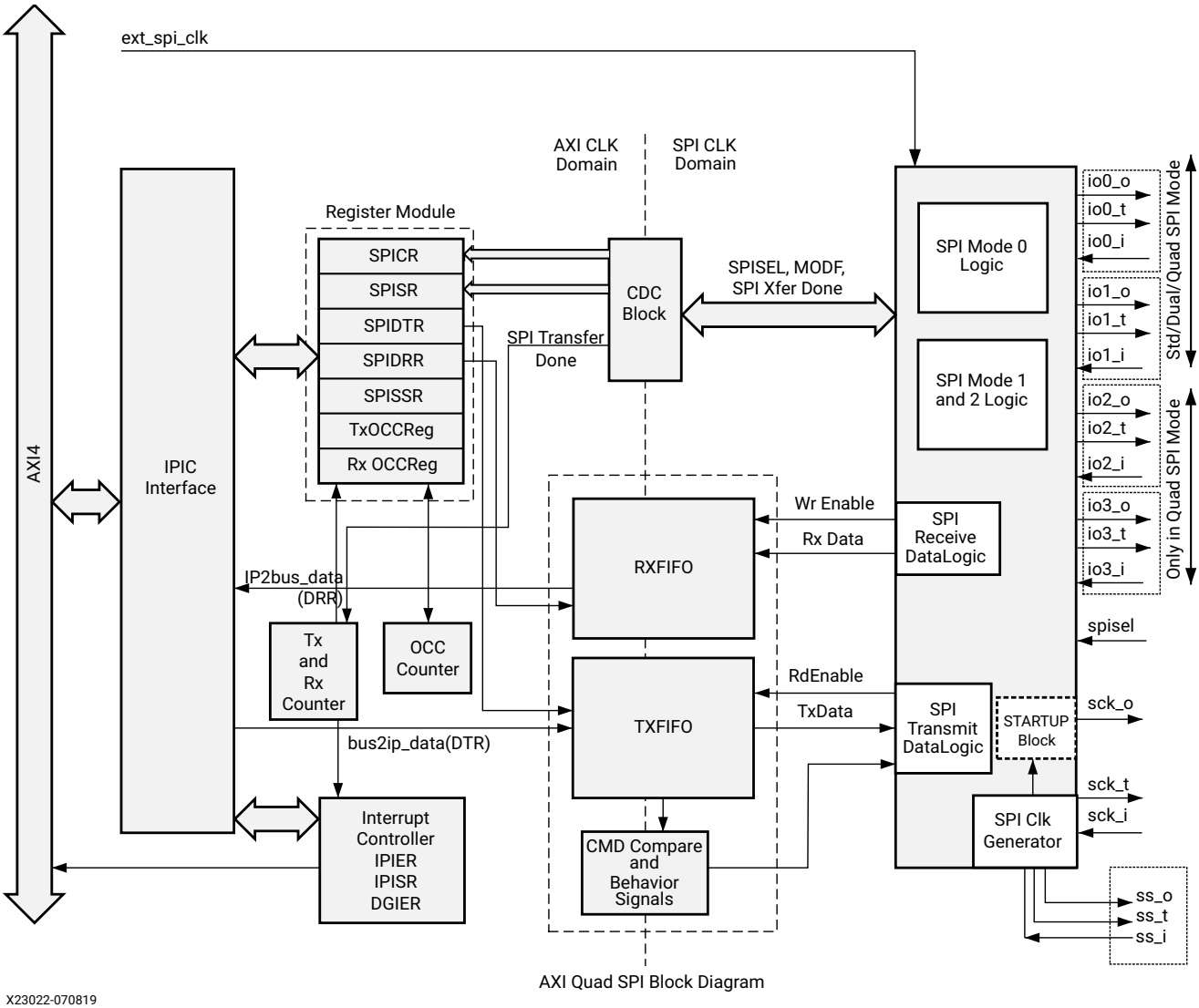

Figure 1-1: **AXI Quad SPI Core Top-Level Block Diagram**

The choice of either AXI4-Lite or AXI4 interface is based on the **Enable Performance Mode** option in the Vivado® Integrated Design Environment (IDE) (see Chapter 4). Performance mode is disabled by default which selects the AXI4-Lite interface. The core always operates as a slave IP when the AXI4 interface is selected.

Legacy Mode

Legacy mode is selected when the **Enable Performance Mode** option in the Vivado Integrated Design Environment (IDE) is disabled. Legacy mode uses the AXI4-Lite interface and is fully backward compatible with all the older versions of the AXI Quad SPI core in terms of functionality, register bit placement, and register access.

The AXI Quad SPI core, when configured in standard SPI mode, is a full-duplex synchronous channel that supports a four-wire interface (receive, transmit, clock, and slave-select) between a master and a selected slave. When configured in Dual/Quad SPI mode, this core supports additional pins for interfacing with external memory. These additional pins are used while transmitting the command, address, and data based on the control register settings and command used.

The core supports the manual slave select mode as the default mode of operation for slave select mode. This mode allows manual control of the slave select line with the data written to the slave select register, thereby allowing transfers of an arbitrary number of elements without toggling the slave select line between elements. However, before starting a new transfer, the slave select line must be toggled.

The other mode related to slave select is automatic slave select mode. In this mode, the slave select line is toggled automatically after each element transfer (when FIFO is disabled). This mode, which is supported only in standard SPI mode, is described in more detail in SPI Protocol Slave Select Assertion Modes in Chapter 3.

The core functionality is divided into standard SPI mode and dual and quad SPI mode. The functionality for each mode differs in the way the slave memory works.

Standard SPI Mode

Standard SPI mode is selected when the **Mode** option in the Vivado IDE is set to **Standard**. The relevant parameters in this mode are:

- Mode
- Enable STARTUPE2 Primitive
- Transaction Width
- No. of Slaves
- Frequency Ratio

- Enable FIFO

The properties of the core in standard SPI mode, including or excluding a FIFO, are described as:

- The choice of inclusion of FIFO is based on the **Enable FIFO** parameter. **FIFO Depth** parameter is linked to **Enable FIFO** parameter. FIFO Depth limits the transmit and receive FIFO depth to **16** or **256** when FIFO is enabled. When FIFO is not enabled, the value of FIFO depth parameter is considered to be 0. A FIFO depth of 256 should be used because this is the most suitable depth in relation to the flash memory page size.
- The valid values for the **FIFO Depth** option in this mode are **16** or **256** when FIFO is enabled through **Enable FIFO** parameter.

When **Enable FIFO** is **0** and no FIFO is included in the core. Data transmission occurs through the single transmit and receive register. When **FIFO Depth** is **16** or **256**, the transmit or receive FIFO is included in the design with a depth of 16 or 256 elements. The width of the transmit and receive FIFO is configured with the **Transaction Width** option.

The AXI Quad SPI core supports continuous transfer mode. When configured as master, the transfer continues until the data is available in the transmit register/FIFO. This capability is provided in both manual and automatic slave select modes. As an example, during the page read command, the command, address, and number of data beats in the DTR must be set equal to the same number of data bytes intended to be read by the SPI memory.

When the core is configured as a slave, if the slave select line (SPISEL) goes High (inactive state) during the data element transfer, the current transfer is aborted. If the slave select line goes Low, the aborted data element is transmitted again. The slave mode of the core is allowed only in the standard SPI mode.

Dual/Quad SPI Mode

Dual SPI mode is selected when the **Mode** option in the Vivado IDE is set to **Dual**. The relevant parameters in this mode are:

- Mode
- Slave Device
- Enable STARTUPE*n* Primitive

  Note: The STARTUPE2 primitive is applicable for 7 series devices. The STARTUPE3 primitive is applicable for UltraScale™ devices.
- Transaction Width
- No. of Slaves
- FIFO Depth

The properties associated with the FIFO are:

- The depth of the FIFO is based on the **FIFO Depth** option which has valid values of **16** or **256**.
- The width of the FIFO is 8-bits because the page size of the SPI slave memories is always 8-bits.

The behavior of the ports in dual mode is:

- For standard SPI mode instructions, the IO0 and IO1 pins are unidirectional [the same as the master out slave in (MOSI) and master in slave out (MISO) pins].
- For dual mode SPI instructions, the IO0 and IO1 pins are bidirectional — depending on the type of command and memory chosen.

The quad SPI mode is selected when the **Mode** option is set to **Quad**. The behavior of the ports in quad SPI mode is:

- For standard mode SPI instructions, the IO0 and IO1 pins are unidirectional and function the same as in standard SPI mode.
- For dual mode SPI instructions, the IO0 and IO1 pins are unidirectional or bidirectional depending on the type of instruction and memory selected by setting the control register bits. The IO2 and IO3-bits are 3-state.
- For quad mode SPI instructions, the IO0, IO1, IO2, and IO3 pins are unidirectional or bidirectional depending on the type of memory used while transmitting the command, address, and data.

When the **Mode** option is **Dual** or **Quad**, the core is forced to operate in dual or quad SPI mode, respectively, while the core continues to support the standard SPI commands and interface. The internal command logic guides the core I/O behavior depending on the command loaded in the DTR FIFO (SPI DTR). The **Mode** option settings also determine the I/O pin availability.

Common Information for Both SPI Modes

The 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 also be added. However, detection of all possible conflicts is not implemented with this interface standard. To eliminate conflicts, the system software is required to arbitrate bus control.

The 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 affects the achievable performance in terms of frequency and resource utilization. All of the SPI core and interrupt registers are 32-bits wide. The core supports only 32-bit access to all SPI and interrupt register modules.

AXI4 Interface

The AXI4 interface is included when the **Enable Performance Mode** option is selected. In this mode, the core can be operated in enhanced mode (**Enable XIP Mode** is not selected) or XIP mode (**Enable XIP Mode** is selected). In performance mode, the AXI4 interface is used for burst transactions at the DTR and DRR locations.

Enhanced Mode

In this mode, the AXI4-Lite interface for the core is replaced with the AXI4 interface. This mode also supports standard, dual and quad modes depending on the **Mode** option setting. The target slave memory can be chosen by setting the **Slave Device** option to **Mixed**, **Winbond**, **Micron,** or **Spansion**. All of the registers are mapped to the same offset as with the AXI4-Lite interface. The AXI4 interface is allowed to do burst transactions at the data transmit register and data receive register only. All other registers should be single access only. This should be noted while designing the application for the core.

The DTR and DRR FIFOs are configurable to 16 or 256 beat depth. The core supports the same functionality as the AXI4-Lite interface. The added advantage for this mode is burst capability at the DTR and DRR locations, reducing the overhead of reading and writing data to and from the core at the AXI4 interface side.

XIP Mode

In XIP mode, the core has an AXI4-Lite as well as an AXI4 interface (see Figure 1-2). The AXI4-Lite interface is chosen for accessing the configuration register and the status register. The AXI4 interface is used only for reading. The AXI4 interface supports only the read channel. No write transactions are allowed. The AXI4-Lite interface can access the configuration register to change the clock polarity (CPOL) or clock phase (CPHA) configuration. In XIP mode, IP support the following two modes:

- High Performance Mode: In this mode, IP supports more than 64 beat transactions, provided read data ready should always high.
- Normal Mode: In this mode, IP supports maximum of 64 beat transactions.

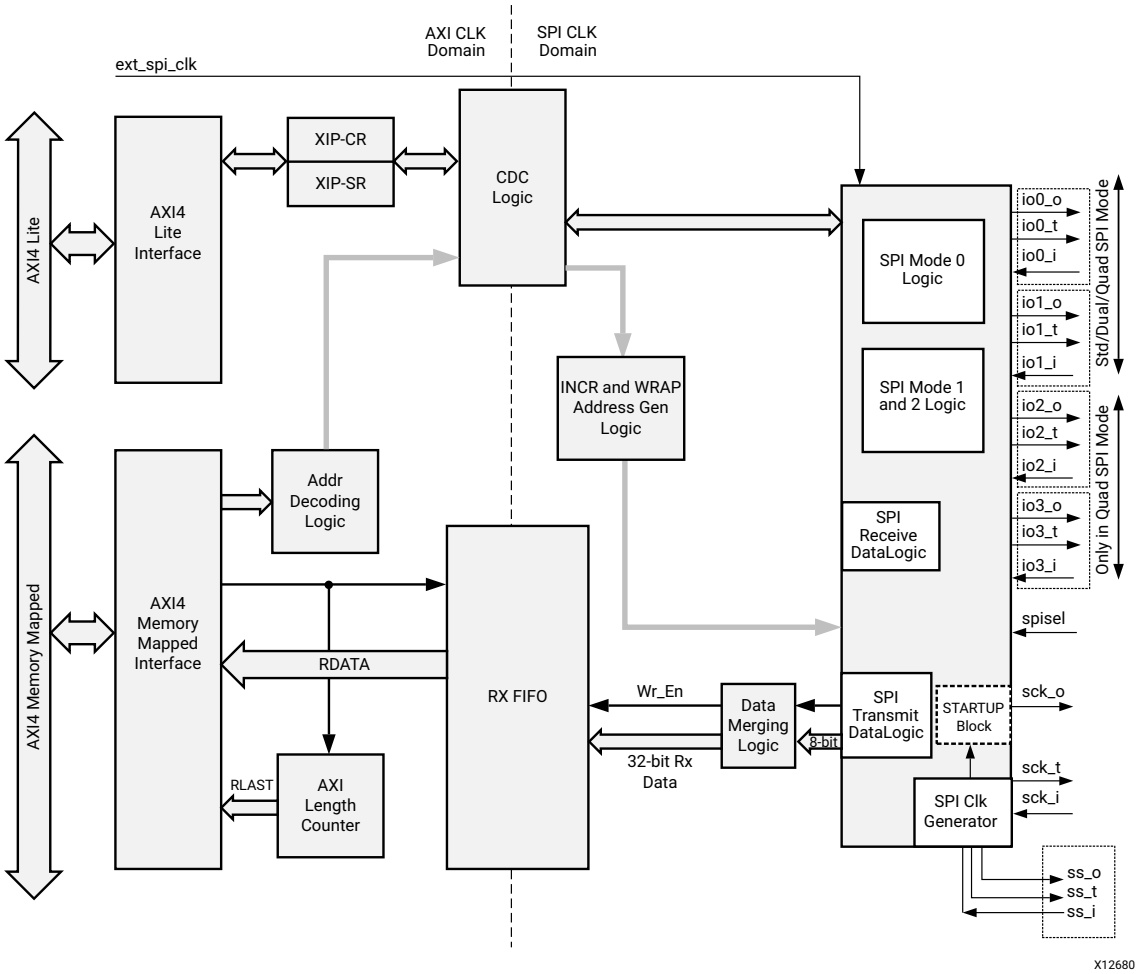

Figure 1-2: **Block Diagram of AXI Quad SPI in XIP Mode**

This mode is suitable for boot operation. In this mode, the core supports INCR and WRAP read transactions only.

In mode, the core considers the SPI as read-only memory. The three read commands are provided with the configuration mode which is used while reading the SPI flash memory. The core functionality is verified by assigning the same frequency to both the AXI4-Lite and AXI4 interface. The three main read commands which are built into the core are fast read (`0x0Bh`), DIOFR (`0xBBh`) and QIOFR (`0xEBh`).

Dual Quad SPI Mode

In Dual Quad SPI mode, the core has two SPI interfaces. This mode can be enabled only in UltraScale devices, when the mode is Quad, STARTUP is enabled and the number of slaves is 2. This feature can be used to access two slaves; slave selection can be done via register configuration.

Core Internal Submodules

The AXI Quad SPI core submodules are described in these sections.

Enable Performance Mode Not Selected

In this mode, the core is backward-compatible with all earlier versions of the AXI Quad SPI core. The core includes these submodules:

AXI4-Lite Interface Module

The AXI4-Lite interface module provides the interface to the AXI4-Lite protocol and IPIC. The read and write transactions at the AXI4-Lite interface are translated into equivalent IP interconnect (IPIC) transactions. This is the default combination for the core.

SPI Register Module

The SPI register module includes all the memory-mapped registers shown in Figure 1-1. This module connects to the AXI4-Lite interface. and consists of status register, control register, N-bit slave select register (N ≤ 32), and a pair of transmit and receive registers.

Interrupt Controller Register Set Module

The interrupt controller register set module consists of the interrupt-related registers:

- Device global interrupt enable register (DGIER)
- IP interrupt enable register (IPIER)
- 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. In standard SPI mode, this module is the center of the SPI operation.

Optional FIFOs

When enabled by the parameter **Enable FIFO**, the transmit FIFO and receive FIFO are implemented on both the transmit and receive paths. The width of the transmit FIFO and receive FIFO are the same and depend on the **Transaction Width** parameter. The FIFOs can be enabled or disabled with depth variable at 16 or 256 when enabled in Standard SPI mode. In dual and quad SPI modes, the FIFO depth is 16 or 256 locations (bytes).

STARTUPEn Module

The STARTUPE2 primitive is applicable for 7 series devices. The STARTUPE3 primitive is applicable for UltraScale devices.

Enable STARTUPE2 Primitive Parameter

STARTUPE2 is a primitive in the Xilinx device. This primitive can be used after the FPGA configuration in the design. For more understanding on the use of this primitive, read the targeted FPGA user guide. This primitive can be included in the design by selecting the **Enable STARTUPE2 Primitive** parameter. The STARTUPE2 primitive is present in 7 series and Zynq®-7000 SoC devices, This primitive has a dedicated clock pin that can be used to provide the SPI clock to the slave memory. The output ports of the STARTUP primitive are taken to the top interface of the core as the STARTUPE2 interface. Spartan-7 7S6 and 7S15 FPGAs do not support the `STARTUPE2.CLK - UserClk` startup clock pin.

Enable STARTUPE3 Primitive Parameter

The STARTUPE3 primitive is present in UltraScale devices. This primitive has a dedicated clock and data pins that can be used to provide the SPI clock and data interface to the slave memory.

For more information refer to the *UltraScale Architecture Libraries Guide* (UG974) [Ref 1].

Quad SPI Control Logic Module

This module is responsible for generation of control signals which are used in dual or quad SPI modes. This module contains logic for a shift register, SPI clock generator, and state machines for various memory configurations.

Feature Summary

- Legacy mode — AXI4-Lite interface based design
  - AXI interface
    - Supports AXI4-Lite interface — legacy mode
    - All registers in the core should be accessed as 32-bit access and only through a single length AXI4-Lite transaction
  - Configurable SPI modes
    - Supports standard, dual and quad SPI modes
    - Standard mode supports:
      - Master and slave SPI mode
      - MSB/LSB first transactions
      - Local loopback capability for testing
      - Multiple master and multiple slave environment
      - Optional 0 or 16 or 256 element deep (an element is a byte, a half-word or a word) transmit and receive FIFOs. Where 0 FIFO Depth refers to as no FIFO
    - Dual/quad SPI mode supports:
      - Master mode only
      - MSB first transfer only
      - SPI transfer length of 8-bits only
      - Multiple master and multiple slave environment
      - Optional 16 or 256 deep transmit and receive FIFO
      - Optional support of 6 pin SPI interface mode (In Quad mode only)
- AXI4 interface mode
  - AXI4 interface
    - Supports AXI4 interface
  - Configurable SPI interface supports:
    - Standard, dual and quad mode of SPI configuration
    - Master mode only
    - 16 or 256 element deep transmit and receive FIFO
    - MSB-only transfer of 8-bit length at SPI flash memory

    - Multiple SPI slaves – configurable up to 32
    - Configurable XIP (execute in place) and non-XIP modes
  - Enhanced mode — Non-XIP mode (burst mode access) — AXI4 design:
    - SPI read and write commands provided by master
    - Only fixed-burst transfer at DTR and DRR FIFO locations.
    - Only one read or one write transaction is acceptable at a time from AXI4 interface
    - All registers in the core should be accessed as 32-bit access and only through single length AXI4 transaction
    - WRAP transactions not supported
  - Read only XIP mode — AXI4-Lite + AXI4 interface based design:
    - AXI4-Lite mode used for setting the configuration register and reading the status/debug register
    - Read commands only at SPI interface
    - INCR and WRAP read transactions at memory mapped address
    - 64 beat deep fixed internal FIFO with 32-bit data width
    - FIXED transactions unsupported

Table 1-1 defines the parameters used to configure the core.

Table 1-1: **Core Operation Mode and Design Parameter Values**

Core Operation	Parameters							
	Enable Performance Mode	Enable XIP Mode	FIFO Depth	Mode	Frequency Ratio	No. of Slaves	Slave Device	Enable STARTUPE*n* Primitive(4)
Legacy mode			0, 16, 256	Standard	2, 4, 8, Nx16 for N = 1, 2, 3,...,128(1)	1-32	(any)	(either)
			16, 256	Dual	2(2)	1-32	(any)	(either)
			16, 256	Quad	2(2)	1-32	(any)	(either)
Enhanced mode	x		0, 16, 256	Standard	2, 4, 8, Nx16 for N = 1, 2, 3,...,128(1)	1-32	(any)	(either)
	x		16, 256	Dual	2(2)	1-32	(any)	(either)
	x		16, 256	Quad	2(2)	1-32	(any)	(either)
XIP mode	x	x	64	Standard	2(2)	1(3)	(any)	(either)
	x	x	64	Dual	2(2)	1(3)	(any)	(either)
	x	x	64	Quad	2(2)	1(3)	(any)	(either)

Notes:

1. In standard mode the `ext_spi_clk` can be the same as the `axi_aclk` or `axi4_aclk`, but it should not be less than the AXI CLK or AXI4 ACLK. This mode is specifically for slow operating devices that work on the SPI protocol. Examples of these devices are EEPROMs and SPI interface-based DACs.
2. In XIP Mode or Dual/Quad variants of Legacy/Enhanced mode,, set `ext_spi_clk` to double the intended SPI clock. The Frequency Ratio parameter divides this clock by 2 to generate the SPI clock. Most of the SPI flash memory commands operate at a higher SPI clock rate with the exception of some commands which operate at a lesser frequency. It is your responsibility to configure the core with the correct `ext_spi_clk` and Frequency Ratio parameter and use the appropriate commands. If slower operating commands are executed on higher SPI clock ratios, the core does not generate an error and passes these commands to the external flash memory, but at the same time, the operation of the flash memory is not guaranteed.
3. In XIP mode, No. of Slaves is always equal to 1 and this parameter cannot be updated.
4. The STARTUPE2 primitive is applicable for 7 series devices. The STARTUPE3 primitive is applicable for UltraScale and later devices.

Unsupported Features

These features relate to the AXI4 interface and are not supported by the core.

- INCR of length more than 1 and WRAP bursts in enhanced mode
- FIXED burst in XIP mode
- Narrow bursts in enhanced mode (only the last 8-bits from the 32 burst bits are valid in enhanced mode)
- Write channel and transactions in XIP mode
- Atomic, locked and cache transactions
- Debug/secure and user signals
- Out-of-order transactions
- Region signals
- Quality of Service (QOS) signals
- Holes in byte strobes
- Barrier transactions
- Write interleaving
- User signals
- AXI TrustZone and low-power state
- Simultaneous read and write transactions in enhanced mode
- Un-aligned address when the core is configured in read-only XIP mode
- Byte access in XIP mode
- Standard Slave mode when the serial clock (SCK) ratio is 2 is not supported by the core.
- In enhance mode, IP waits for `s_axi4_rready` to generate the first `s_axi4_rvalid`.

Licensing and Ordering

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License.

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 IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

The AXI Quad SPI core connects the AXI4 and AXI4-Lite interfaces to SPI slave devices that support the standard, dual or quad SPI protocol instruction set. The core provides a serial interface to SPI slave devices such as SPI serial flash memory from Winbond, Micron, Spansion and Macronix. The dual/quad SPI is an enhancement to the standard SPI protocol (described in the Motorola M68HC11 data sheet). See Specification Exceptions for differences between the core protocol and the Motorola M68HC11-Rev. 4.0 reference manual.

Performance

For full details about performance and resource utilization, visit the Performance and Resource Utilization web page.

The performance characterization of this core was compiled using the margin system methodology. The details of the margin system characterization methodology are described in the *Vivado Design Suite User Guide: Designing With IP* (UG896) [Ref 2].

Note: Performance for Zynq®-7000 SoC and UltraScale™ devices is similar to 7 series devices.

The performance characterization was compiled for these families.

- Virtex®-7
- Kintex®-7
- Artix®-7

The maximum frequencies for the AXI Quad SPI core are shown in Table 2-1.

Table 2-1: **AXI Quad SPI Maximum Frequencies**

Family	Speed Grade	AXI4-LIte Interface Fmax (MHz)	AXI4 Interface Fmax (MHz)	Ext_spi_clk Fmax (MHz)
Virtex-7	-1	180	200	80
	-2	200	240	90
	-3	220	280	100
Kintex-7	-1	180	200	80
	-2	200	240	90
	-3	220	280	100
Artix-7	-1	120	150	60
	-2	140	180	70
	-3	160	200	80

Note: For xip and standard modes `Ext_spi_clk` may be limited to 60 MHz.

Note: UltraScale™ device frequency numbers are expected to be similar to 7 series numbers.

Resource Utilization

For full details about performance and resource utilization, visit the Performance and Resource Utilization web page.

Port Descriptions

Table 2-2: **I/O Signals**

Signal name	Interface	Signal type	Init status	Description
s_axi_aclk	Clock	I	-	AXI Clock. This signal in available only in legacy and XIP modes.
s_axi_aresetn	Reset	I	-	AXI Reset. This signal in available only in legacy and XIP modes.
s_axi4_aclk(1)	Clock	I	-	AXI4 Clock. This signal in available only in enhanced and XIP modes.
s_axi4_aresetn(2)	Reset	I	-	AXI4 Reset. This signal in available only in enhanced and XIP modes.
ext_spi_clk	Clock	I	-	This clock is used for the SPI interface. This clock should be double of the maximum SPI frequency intended at the SPI interface.
ip2intc_irpt	SPI	O	0	Interrupt control signal from SPI.
AXI Interface Signals				
s_axi_*		Input/ Output		See Appendix A of the *Vivado AXI Reference Guide* (UG1037)[Ref 3] for AXI signals. This signal is available only in legacy and XIP modes.
AXI4 Interface Signals				
s_axi4_*		Input/ Output		See Appendix A of the *Vivado AXI Reference Guide* (UG1037)[Ref 3] for AXI signals. This signal is available only in enhanced and XIP modes.
SPI Interface Signals				
sck_i	SPI	I	-	SPI bus clock input. This signal is available only in Standard SPI slave mode.
sck_o	SPI	O	-	SPI bus clock output.
sck_t	SPI	O	1	3-state enable for SPI bus clock. Active-Low.
ss_i[(No. of Slaves – 1):0]	SPI	I	-	This input is not used in the design in any mode.
ss_0[(No. of Slaves – 1):0]	SPI	O	1	Output one-hot encoded, active-Low slave select vector of length n.
ss_t	SPI	O	1	3-state enable for slave select. Active-Low.

Table 2-2: **I/O Signals *(Cont'd)***

Signal name	Interface	Signal type	Init status	Description
io0_i	SPI	I	-	Behaves similar to master output slave input (MOSI) input pin.
io0_o	SPI	O	-	Behaves similar to the master output slave input (MOSI) output pin. This is available only in standard SPI mode. In dual SPI mode, this signal acts as a bidirectional signal based on certain instructions.
io0_t	SPI	O	1	3-state enable master output slave input. Active-Low.
io1_i	SPI	I	-	Behaves similar to the master input slave output (MISO) input. This signal can also be considered as an `IO1_I` port in dual or quad SPI mode.
io1_0	SPI	O	-	Behaves similar to master input slave output (MISO) output. This is available only in standard SPI mode. In dual SPI mode, this signal acts as a bidirectional signal based on certain instructions.
io1_t	SPI	O	1	3-state enable master input slave output. Active-Low.
io2_i	SPI	I	-	IO2 input based on commands used. This signal is available only in quad SPI mode.
io2_o	SPI	O	-	IO2 output based on commands used. This signal is available only in quad SPI mode.
io2_t	SPI	O	1	3-state enable IO2. This signal is available only in quad SPI mode. Active-Low.
io3_i	SPI	I	-	IO3 input based on commands used. This signal is available only in quad SPI mode.
io3_o	SPI	O	-	IO3 output based on commands used. This signal is available only in quad SPI mode.
io3_t	SPI	O	1	3-state enable IO3. This signal is available only in quad SPI mode. Active-Low.

Table 2-2: **I/O Signals *(Cont'd)***

Signal name	Interface	Signal type	Init status	Description
io0_1_i[3]	SPI	I	-	Behaves similar to master output slave input (MOSI) input pin.
io0_1_o[3]	SPI	O	-	Behaves similar to the master output slave input (MOSI) output pin. This signal acts as a bidirectional signal based on certain instructions.
io0__1_t[3]	SPI	O	1	3-state enable master output slave input. Active-Low.
io1_1_i[3]	SPI	I	-	Behaves similar to the master input slave output (MISO) input.
io1_1_0[3]	SPI	O	-	Behaves similar to master input slave output (MISO) output.
io1_1_t[3]	SPI	O	1	3-state enable master input slave output. Active-Low.
io2_1_i[3]	SPI	I	-	IO2 input based on commands used.
io2_1_o[3]	SPI	O	-	IO2 output based on commands used.
io2_1_t[3]	SPI	O	1	3-state enable IO2. Active-Low.
io3_1_i[3]	SPI	I	-	IO3 input based on commands used.
io3_1_o[3]	SPI	O	-	IO3 output based on commands used.
io3_1_t[3]	SPI	O	1	3-state enable IO3. Active-Low.
ss_0_i[3]	SPI	O	-	This input is not used in the design in any mode.
ss_1_i[3]	SPI	O	1	Output one-hot encoded, active-Low slave select vector of length n.
spisel	SPI	I	1	Local SPI slave select active-Low input. This is an input signal when the core is configured in standard SPI slave mode. Must be set to 1 (along with the master bit in the SPICR) in master mode.
STARTUP Interface Signals				

Table 2-2: **I/O Signals *(Cont'd)***

Signal name	Interface	Signal type	Init status	Description
cfgclk	STARTUP_IO	O	-	Clock is only active during configuration and during Master modes with persist set.
cfgmclk	STARTUP_IO	O	-	Free-running clock from on-chip oscillator. Nominally 50 MHz but is not characterized or specified in a data sheet.
eos	STARTUP_IO	O	-	Signal rises upon completion of the STARTUPEn sequence.
preq	STARTUP_IO	O	-	PROGRAM request to FPGA logic

Notes:

1. AXI clock is expected to be faster than `ext_spi_clk`.
2. When `ext_spi_clk` is too slow, it is advised to use FIFO depth 256. (Frequency ratio is in the range of 50 to 100.)
3. These signals are applicable only in dual quad mode.

Register Space (Legacy and Enhanced Non-XIP Mode)

Note: The AXI4-Lite write access register is updated by the 32-bit AXI Write Data (*_wdata) signal, and is not impacted by the AXI Write Data Strobe (*_wstrb) signal. For a Write, both the AXI Write Address Valid (*_awvalid) and AXI Write Data Valid (*_wvalid) signals should be asserted together.

Table 2-3 shows the set of registers applicable whether or not **Enable Performance Mode** is selected and **Enable XIP Mode** is not selected. Some AXI Quad SPI core registers should be accessed individually. These registers are configurable and accessible through either the AXI4-Lite interface or the AXI4 interface (enhanced mode). All registers are accessed as 32-bit. If non-existent registers are accessed, they return an OKAY response. The reading of these registers returns 0, and write does not have any affect.

Table 2-3: **Core Registers in Legacy and Enhanced Mode**

Address Space Offset	Register Name	Access Type	Default Value (hex)	Description
Core Grouping				
40h	SRR	Write	N/A	Software reset register
60h	SPICR	R/W	0x180	SPI control register
64h	SPISR	Read	0x0a5	SPI status register
68h	SPI DTR	Write	0x0	SPI data transmit register. A single register or a FIFO
6Ch	SPI DRR	Read	N/A(1)	SPI data receive register. A single register or a FIFO
70h	SPISSR	R/W	No slave is selected 0xFFFF	SPI Slave select register
74h	SPI Transmit FIFO Occupancy Register(2)	Read	0x0	Transmit FIFO occupancy register
78h	SPI Receive FIFO Occupancy Register(2)	Read	0x0	Receive FIFO occupancy register
Interrupt Control Grouping				
1Ch	DGIER	R/W	0x0	Device global interrupt enable register
20h	IPISR	R/TOW(3)	0x0	IP interrupt status register

Table 2-3: **Core Registers in Legacy and Enhanced Mode *(Cont'd)***

Address Space Offset	Register Name	Access Type	Default Value (hex)	Description
28h	IPIER	R/W	0x0	IP interrupt enable register

Notes:

1. The power-on reset data in the SPI DRR is unknown or all zeros. This register should not be considered for power-on reset conditions.
2. Exists only when FIFO Depth is set to 16 or 256.
3. TOW = Toggle on write. Writing a 1 to a bit position within the register causes the corresponding bit position in the register to toggle.

Register Details

Software Reset Register

The Software Reset Register (SRR) permits resetting the core independently of other cores in the system.

IMPORTANT: *To activate the software-generated reset, the value of `0x0000_000a` must be written to the Software Reset Register.*

Writing `0x0000_000a` to the SRR resets the core register for four AXI clock cycles. Any other write access generates undefined results and results in an error. The bit assignment in the software reset register is shown in Figure 2-1 and described in Table 2-4. Any attempt to read this register returns undefined data.

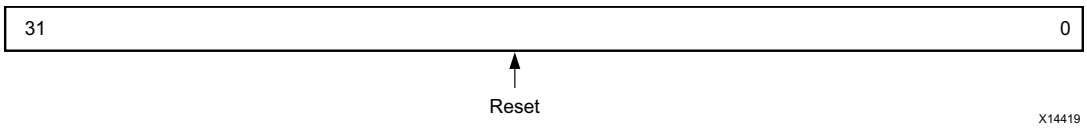

Figure 2-1: **Software Reset Register (Core Base Address + 0x40)**

Table 2-4: **Software Reset Register Description (Core Base Address + 0x40)**

Bits	Name	Core Access	Reset Value	Description
31:0	Reset	Write only	N/A	The only allowed operation on this register is a write of `0x0000000a`, which resets the AXI Quad SPI core.

SPI Control Register

The SPI Control Register (SPICR) allows programmer control over various aspects of the AXI Quad SPI core. The bit assignment in the SPICR is shown in Figure 2-2 and described in Table 2-5.

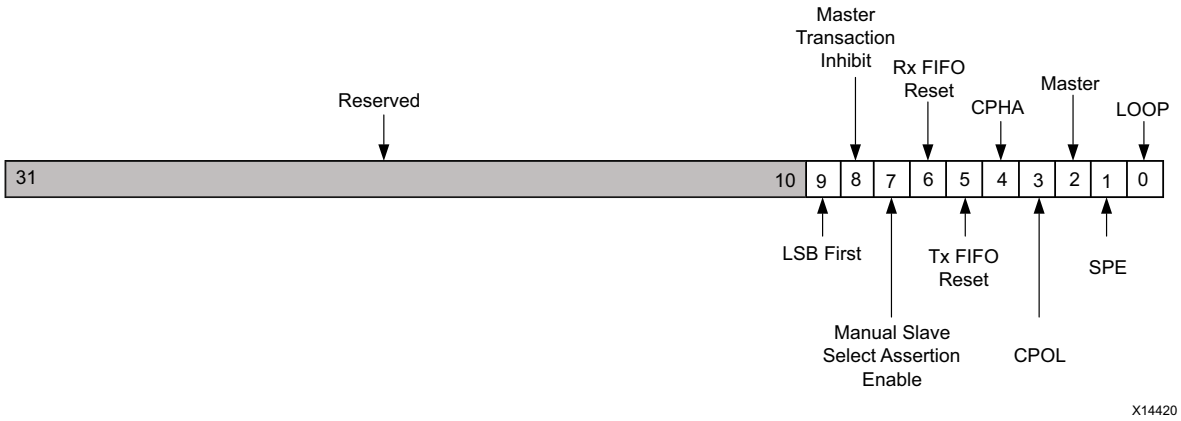

Figure 2-2: **SPI Control Register (Core Base Address + 0x60)**

Table 2-5: **SPI Control Register Description (Core Base Address + 0x60)**

Bits	Name	Core Access	Reset Value	Description
31:10	Reserved	NA	NA	Reserved.
9	LSB First	R/W	0	LSB first:(1) This bit selects LSB first data transfer format. The default transfer format is MSB first. When set to: 0 = MSB first transfer format. 1 = LSB first transfer format. ***Note:*** In Dual/Quad SPI mode, only the MSB first mode of the core is allowed.
8	Master Transaction Inhibit	R/W	1	Master transaction inhibit: This bit inhibits master transactions. This bit has no effect on slave operation. When set to: 0 = Master transactions enabled. 1 = Master transactions disabled. ***Note:*** This bit immediately inhibits the transaction. Setting this bit while transfer is in progress would result in unpredictable outcome.

Table 2-5: **SPI Control Register Description (Core Base Address + 0x60) *(Cont'd)***

Bits	Name	Core Access	Reset Value	Description
7	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. When set to: 0 = Slave select output asserted by master core logic. 1 = Slave select output follows data in slave select register. ***Note:*** The manual slave assertion mode is supported in standard SPI mode only. For more information see SPI Protocol Slave Select Assertion Modes.
6	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 AXI clock cycle after reset, this bit is again set to 0. When set to: 0 = Receive FIFO normal operation. 1 = Reset receive FIFO pointer.
5	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 AXI clock cycle after reset, this bit is again set to 0. When set to: 0 = Transmit FIFO normal operation. 1 = Reset transmit FIFO pointer.
4	CPHA	R/W	0	Clock phase:(2) Setting this bit selects one of two fundamentally different transfer formats. See Clocking (SPI Clock Phase and Polarity Control) in Chapter 3.
3	CPOL	R/W	0	Clock polarity:(2) Setting this bit defines clock polarity. When set to: 0 = Active-High clock; SCK idles Low. 1 = Active-Low clock; SCK idles High.
2	Master	R/W	0	Master (SPI master mode):(3) Setting this bit configures the SPI device as a master or a slave. When set to: 0 = Slave configuration. 1 = Master configuration. ***Note:*** In dual/quad SPI mode only the master mode of the core is allowed. ***Note:*** Standard Slave mode is not supported for SCK ratio = 2.

Table 2-5: **SPI Control Register Description (Core Base Address + 0x60) *(Cont'd)***

Bits	Name	Core Access	Reset Value	Description
1	SPE	R/W	0	SPI system enable: Setting this bit to 1 enables the SPI devices as noted here. When set to: • 0 = SPI system disabled. Both master and slave outputs are in 3-state and slave inputs are ignored. • 1 = SPI system enabled. Master outputs active (for example, IO0 (MOSI) and SCK in idle state) and slave outputs become active if $\overline{SS}$ becomes asserted. The master starts transferring when transmit data is available.
0	LOOP	R/W	0	Local loopback mode:(4) Enables local loopback operation and is functional only in standard SPI master mode. When set to: • 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.

Notes:

1. Setting of this bit (LSB First) is allowed only in Standard SPI mode. Dual/quad SPI modes support MSB first mode only. In dual/quad SPI mode if this bit is set, the corresponding error bit is set in SPISR and an interrupt is generated.
2. In dual and quad SPI mode, values for CPHA-CPOL of either 00 or 11 are allowed. Setting of other configurations causes a malfunction while communicating with memory. If other values are set, the corresponding error bit is set in the SPISR and an interrupt is generated if the corresponding bit is enabled in the SPI IPIER register.
3. The slave mode support is available only in standard SPI mode. In dual or quad SPI mode, only the master mode of the core is supported. If other values are set, the corresponding error bit is set in SPISR and an interrupt is generated if the corresponding bit is enabled in the SPI IPIER register.
4. Loopback is allowed in standard SPI mode.

SPI Status Register

The SPI Status Register (SPISR) is a read-only register that provides the status of some aspects of the AXI Quad SPI core to the programmer. The bit assignment in the SPISR is shown in Figure 2-3 and described in Table 2-6. Writing to the SPISR does not modify the register contents.

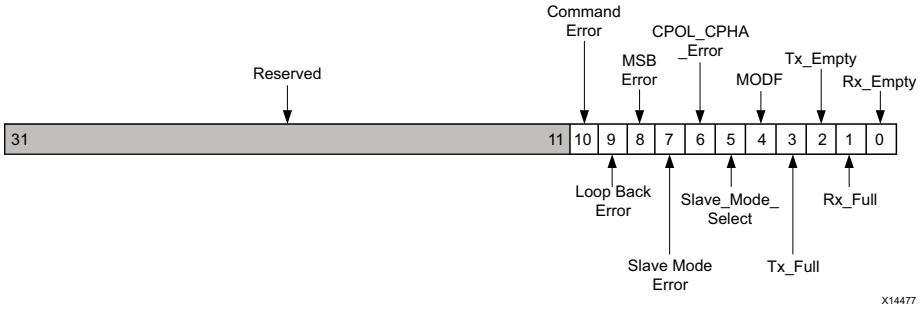

Figure 2-3: **SPI Status Register (Core Base Address + 0x64)**

Table 2-6: **SPI Status Register Description (Core Base Address + 0x64)**

Bits	Name	Core Access	Reset Value	Description
31:11	Reserved	N/A	N/A	Reserved
10	Command Error	Read	0	Command error flag. When set to: • 0 = Default. • 1 = When the core is configured in dual/quad SPI mode and the first entry in the SPI DTR FIFO (after reset) do not match with the supported command list for the particular memory, this bit is set. ***Note:*** Command error is only applicable when the core is configured either in dual or quad mode in legacy or enhanced mode AXI4 interface.
9	Loopback Error	Read	0	Loopback error flag. When set to: • 0= Default. The loopback bit in the control register is at default state. • 1 = When the SPI command, address, and data bits are set to be transferred in other than standard SPI protocol mode and this bit is set in control register (SPICR). ***Note:*** Loopback is only allowed when the core is configured in standard mode. Other modes setting of the bit causes an error and the interrupt bit is set in legacy or enhanced mode AXI4 interface.
8	MSB Error	Read	0	MSB error flag. When set to: • 0= Default. • 1 = This bit is set when the core is configured to transfer the SPI transactions in either dual or quad SPI mode and LSB first bit is set in the control register (SPICR). ***Note:*** In dual/quad SPI mode, only the MSB first mode of the core is allowed. MSB error flag is only applicable when the core is configured either in dual or quad mode in legacy or enhanced mode AXI4 interface.

Table 2-6: **SPI Status Register Description (Core Base Address + 0x64) *(Cont'd)***

Bits	Name	Core Access	Reset Value	Description
7	Slave Mode Error	Read	1	Slave mode error flag. When set to: • 1 = This bit is set when the core is configured with dual or quad SPI mode and the master is set to 0 in the control register (SPICR). • 0 = Master mode is set in the control register (SPICR). ***Note:*** Quad SPI mode, only the master mode of the core is allowed. Slave mode error flag is only applicable when the core is configured either in dual or quad mode in legacy or enhanced AXI4 mode interface.
6	CPOL_CPHA_ Error	Read	0	CPOL_CPHA_Error flag. When set to: • 0 = Default. • 1 = The CPOL and CPHA are set to 01 or 10. When the SPI memory is chosen as either Winbond, Micron or Spansion or Macronix and CPOL and CPHA are configured as 01 or 10, this bit is set. These memories support CPOL=CPHA mode in 00 or in 11 mode. CPOL_CPHA_Error flag is only applicable when the core is configured either in dual or quad mode in legacy or enhanced mode AXI4 interface.
5	Slave_Mode_ Select	Read	1	Slave_Mode_Select flag. This flag is asserted when the core is configured in slave mode. Slave_Mode_Select is activated as soon as the master SPI core asserts the chip select pin for the core. • 1 = Default in standard mode. • 0 = Asserted when core configured in slave mode and selected by external SPI master.
4	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. A Low-to-High MODF transition generates a single-cycle strobe interrupt. 0 = No error. 1 = Error condition detected.
3	Tx_Full	Read	0	Transmit full. When a transmit FIFO exists, this bit is set High when the transmit FIFO is full. ***Note:*** When FIFOs do not exist, this bit is set High when an AXI write to the transmit register has been made (this option is available only in standard SPI mode). This bit is cleared when the SPI transfer is completed.

Table 2-6: **SPI Status Register Description (Core Base Address + 0x64) *(Cont'd)***

Bits	Name	Core Access	Reset Value	Description
2	Tx_Empty	Read	1	Transmit empty. When a transmit FIFO exists, this bit is set to High when the transmit FIFO is empty. This bit goes High as soon as the TX FIFO becomes empty. While this bit is High, the last byte of the data that is to be transmitted would still be in the pipeline. The occupancy of the FIFO is decremented with the completion of each SPI transfer. ***Note:*** When FIFOs do not exist, this bit is set with the completion of an SPI transfer (this option is available only in standard SPI mode). Either with or without FIFOs, this bit is cleared on an AXI write to the FIFO or transmit register. For Dual/Quad SPI mode, the FIFO is always present in the core.
1	Rx_Full	Read	0	Receive full. When a receive FIFO exists, this bit is set High when the receive FIFO is full. The occupancy of the FIFO is incremented with the completion of each SPI transaction. ***Note:*** When FIFOs do not exist, this bit is set High when an SPI transfer has completed (this option is available only in standard SPI mode). Rx_Empty and Rx_Full are complements in this case.
0	Rx_Empty	Read	1	Receive Empty. When a receive FIFO exists, this bit is set High when the receive FIFO is empty. The occupancy of the FIFO is decremented with each FIFO read operation. ***Note:*** When FIFOs do not exist, this bit is set High when the receive register has been read (this option is available only in standard SPI mode). This bit is cleared at the end of a successful SPI transfer. For dual/quad SPI mode, the FIFO is always present in the core.

SPI Data Transmit Register

The SPI Data Transmit Register (SPI DTR) is written with the data to be transmitted on the SPI bus. After the SPE bit is set to 1 in master mode or `spisel` is active in the slave mode, the data is transferred from the SPI DTR to the shift register.

The SPI DTR should be in reset state before filling so that the DTR FIFO write pointer is pointing to the 0th location.

Dual/Quad Mode

The first write must always be a SPI command from AXI transactions, followed by the address (either 24-bit or 32-bit), then filled with the data to be transmitted. When reading the status register of the memory, then as per the command requirements, this register should be filled with dummy bytes along with a command and address (optional).

In one of these modes, the dummy bytes are required to fill in the DTR which is used for the internal count of the data reception from memory.

In commands such as dual mode read, these bytes are not transferred on the data lines, but are used by the internal logic to keep track of the number of data bytes to be read from the SPI slave memory.

If a transfer is in progress, the data in the SPI DTR is loaded into the shift register as soon as the data in the shift register is transferred to the SPI DRR and a new transfer starts. The data is held in the SPI DTR until replaced by a subsequent write. The SPI DTR is shown in Figure 2-4 and Table 2-7 shows the data format.

When a transmit FIFO exists, data is written directly into the FIFO and the first location in the FIFO is treated as the SPI DTR. The pointer is decremented after completion of each SPI transfer. The choice of inclusion or exclusion of the FIFO in the design is available only when the core is configured in Standard SPI mode. If the core is configured in dual or quad SPI mode, the FIFO always exists. In this mode, the FIFO depth is defined with the parameter **FIFO Depth** (allowed values are 16 or 256).

This register cannot be read and can 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, the AXI write transaction completes with an error condition. Reading the SPI DTR is not allowed and the read transaction results in undefined data.

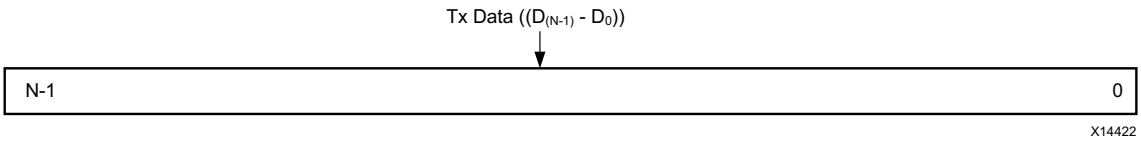

Figure 2-4: **SPI Data Transmit Register (Core Base Address + 0x68)**

Table 2-7: **SPI Data Transmit Register Description (Core Base Address + 0x68)**

Bits	Name	Core Access	Reset Value	Description
[N–1]:0	TX Data[1] (D_{N-1} – D_0)	Write only	0	N-bit SPI transmit data. N can be 8, 16 or 32.[2] N = 8 when the Transfer Width parameter is 8. N = 16 when the Transfer Width parameter is 16. N = 32 when the Transfer Width parameter is 32.

Notes:

1. The D_{N-1} bit always represents the MSB bit irrespective of LSB first or MSB first transfer selection. When the Transfer Width parameter is 8 or 16, the unused upper bits ((AXI data width – 1) to N) are reserved.
2. In standard SPI mode, the width of this register can be 8 or 16 or 32 based on the core configuration. In dual or quad SPI mode, this register is 8-bits wide.

SPI Data Receive Register

The SPI Data Receive Register (SPI DRR) 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 SPI DRR is updated following each completed transaction only if the SPI DRR was read prior to the last SPI transfer.

If the SPI DRR was not read and is full, the most recently transferred data is lost and a receive overrun interrupt occurs. The same condition can also occur with a master SPI device.

The choice of inclusion (**FIFO Depth** = 16 or 256) or exclusion (**FIFO Depth** = 0) of the FIFO in the design is available only when the core is configured in standard SPI mode. When the core is configured in dual or quad SPI mode, the FIFO always exists. In this mode, the FIFO depth is defined with the parameter **FIFO Depth** (allowed values are 16 or 256). For both master and slave SPI configuration mode of the core (in Standard SPI mode only) with a receive FIFO, the data is buffered in the FIFO. The receive FIFO is a read-only buffer. If an attempt is made to read an empty receive register or FIFO, it gives out an error in the Status register. Writes to the SPI DRR do not modify the register contents and return with a successful OK response.

The power-on reset values for the SPI DRR are unknown. When known data has been written into the receive FIFO during core transactions, the data in this register can be considered for reading. The SPI DRR is shown in Figure 2-5, while the specifics of the data format is described Table 2-8.

IMPORTANT: *Based on the command that is issued to the SPI device, a certain unwanted number of bytes are written to the Receive FIFO. These bytes have to be discarded.*

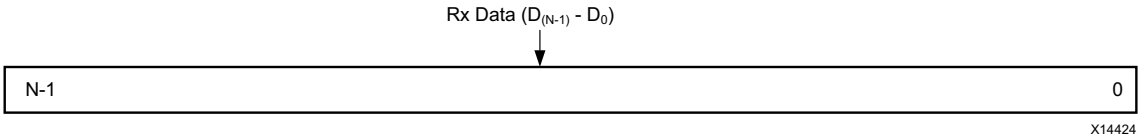

Figure 2-5: **SPI Data Receive Register (Core Base Address + 0x6C)**

Table 2-8: **SPI Data Receive Register Description (Core Base Address + 0x6C)**

Bits	Name	Core Access	Reset Value	Description
[N–1]:0	RX Data[1] (D_{N-1} - D_0)	Read only	N/A	N-bit SPI receive data. N can be 8, 16 or 32.[2] N = 8 when the Transfer Width parameter is 8. N = 16 when the Transfer Width parameter is 16. N = 32 when the Transfer Width parameter is 32.

Notes:

1. The D_{N-1} bit always represents the MSB bit irrespective of LSB first or MSB first transfer selection. When the Transfer Width parameter is 8 or 16, the unused upper bits ((AXI data width – 1) to N) are reserved.
2. In standard SPI mode, the width of this register can be 8 or 16 or 32 based on the core configuration. In dual or quad SPI mode, this register is 8-bit wide.
3. In standard mode, reading an empty DRR FIFO returns a slave error.

SPI Slave Select Register

The SPI Slave Select Register (SPISSR) contains an active-Low, one-hot encoded slave select vector $\overline{SS}$ of length N, where N is the number of slaves set by the **No. of Slaves** parameter. The $\overline{SS}$ vector occupies the right-most bits of the register. At most, one bit can be asserted Low. This bit denotes the slave with which the local master communicates. The bit assignment in the SPISSR is shown in Figure 2-6 and described in Table 2-9.

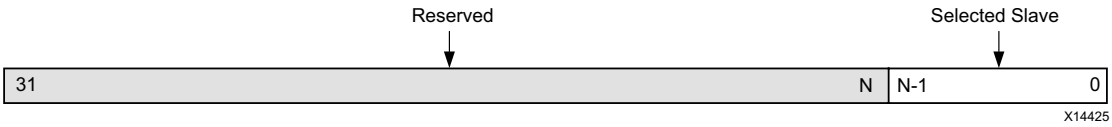

Figure 2-6: **SPI Slave Select Register (Core Base Address + 0x70)**

Table 2-9: **SPI Slave Select Register Description (Core Base Address + 0x70)**

Bits	Name	Core Access	Reset Value	Description
31:N	Reserved	N/A	N/A	Reserved
[N–1]:0	Selected Slave	R/W	1	Active-Low, one-hot encoded slave select vector of length N-bits. N must be ≤ the data bus 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

The SPI Transmit FIFO Occupancy Register (TX_FIFO_OCY) is present only if the AXI Quad SPI core is configured with FIFOs (**FIFO Depth** = 16 or 256). 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 read-only. When written, or read when the FIFO is empty, the register contents are not affected. The only reliable way to determine that the transmit FIFO is empty/full is by reading the Tx_Empty/Tx_Full 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 2-7, while the specifics of the data format are described in Table 2-10.

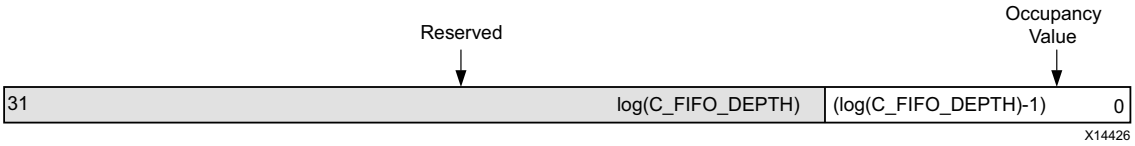

Figure 2-7: **SPI Transmit FIFO Occupancy Register (Core Base Address + 0x74)**

Table 2-10: **SPI Transmit FIFO Occupancy Register Description (Core Base Address + 0x74)**

Bits	Name	Core Access	Reset Value (hex)	Description
31 – log(FIFO Depth)	Reserved	N/A	N/A	Reserved
(log(FIFO Depth)–1):0	Occupancy Value	Read	0	The binary value plus 1 yields the occupancy.

SPI Receive FIFO Occupancy Register

The SPI Receive FIFO Occupancy Register (RX_FIFO_OCY) is present only if the AXI Quad SPI core is configured with FIFOs (**FIFO Depth** = 16 or 256). If the register 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 read-only. A write to it (or of a read when the FIFO is empty) does not affect the register contents. The only reliable way to determine that the receive FIFO is empty/full is by reading the Rx_Empty/Rx_Full status bit in the SPI status register.

The receive FIFO occupancy register is shown in Figure 2-8, while the specifics of the data format are described in Table 2-11.

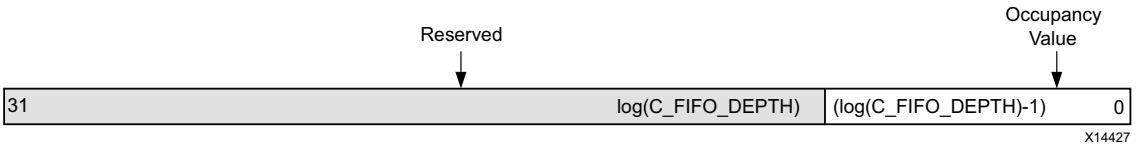

Figure 2-8: **SPI Receive FIFO Occupancy Register (Core Base Address + 0x78)**

Table 2-11: **SPI Receive FIFO Occupancy Register Description (Core Base Address + 0x78)**

Bits	Name	Core Access	Reset Value (hex)	Description
31–log(FIFO Depth)	Reserved	N/A	N/A	Reserved
(log(FIFO Depth)–1):0	Occupancy Value	Read	0	The binary value plus 1 yields the occupancy.

Interrupt Register Set Description

The AXI Quad SPI core has many distinct interrupts that are sent to the interrupt controller. The core interrupt controller allows each interrupt to be enabled independently (using the IP interrupt enable register (IPIER)). The interrupt registers are contained within the interrupt controller. An interrupt strobe can be generated under multiple conditions or only after a transfer completion. In standard, dual or quad SPI mode, and master mode, when the parameter **FIFO Depth** is set to 16 or 256, almost all of the interrupts shown in Table 2-13 are available.

In Standard SPI mode, when **FIFO Depth** is set to 0, all of the interrupts are available except:

- Bit[6]: Transmit FIFO half empty
- Bit[8]: DRR not empty (not present in this mode)

Device Global Interrupt Enable Register

The Device Global Interrupt Enable Register (DGIER) is used to globally enable the final interrupt output from the interrupt controller as shown in Figure 2-9 and described in Table 2-12. This is a read/write bit and is cleared on reset.

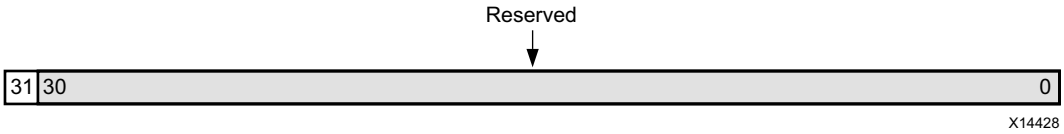

Figure 2-9: **Device Global Interrupt Enable Register (Core Base Address + 0x1C)**

Table 2-12: **Device Global Interrupt Enable Register Description (Core Base Address + 0x1C)**

Bits	Name	Core Access	Reset Value	Description
31	GIE	R/W	0	Global Interrupt Enable. Allows passing all individually enabled interrupts to the interrupt controller. When set to: 0 = Disabled. 1 = Enabled.
30:0	Reserved	N/A	N/A	Reserved

IP Interrupt Status Register (IPISR)

Up to fourteen unique interrupt conditions are possible depending on whether the system is configured with FIFOs or not, as well as if it is configured in master mode or slave mode. A system without FIFOs has seven interrupts. The 32-bit interrupt status register within the interrupt controller can enable each interrupt independently. The IP Interrupt Status Register (IPISR) collects all of the interrupt events. Bit assignments are shown in Figure 2-10 and described in Table 2-13. The interrupt register is a read/toggle-on-write register. Writing a 1 to a bit position within the register causes the corresponding bit to *toggle*. All register bits are cleared on reset.

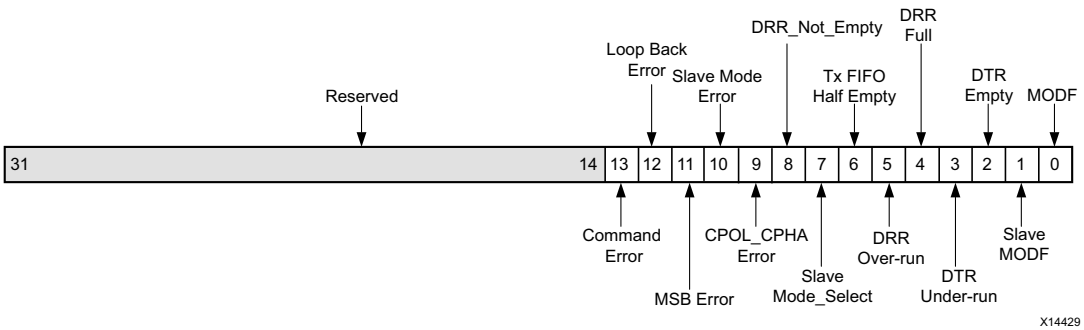

Figure 2-10: **IP Interrupt Status Register (Core Base Address + 0x20)**

Table 2-13: **IP Interrupt Status Register Description (Core Base Address + 0x20)**

Bits	Name	Core Access	Reset Value	Description
31:14	Reserved	N/A	N/A	Reserved
13	Command Error	R/TOW[(1)]	0	Command error. IPISR Bit[13] is the command error. When set to: 1 = This flag is asserted when: • The core is configured in dual/quad SPI mode and • The first entry in the SPI DTR FIFO (after reset) does not match with the supported command list for particular memory. When the SPI command in DTR FIFO does not match with the internal supported command list, the core completes the SPI transactions in standard SPI format. This bit is set to show this behavior of the core. In standard SPI mode this bit is always in default state.
12	Loopback Error	R/TOW[(1)]	0	Loopback error. IPISR Bit[12] is the loopback error. When set to: 1 = This flag is asserted when: • The core is configured in dual or quad SPI transfer mode and • The LOOP bit is set in control register (SPICR(0)). In standard SPI mode, this bit is always in default state.
11	MSB Error	R/TOW[(1)]	0	MSB error. IPISR Bit[11] is the MSB error. When set to: 1 = This flag is asserted when: • The core is configured in either dual or quad SPI mode and • The LSB First bit in the control register (SPICR) is set to 1. In standard SPI mode, this bit is always in default state.
10	Slave Mode Error	R/TOW[(1)]	1	I/O mode instruction error. IPISR Bit[10] is the slave mode error. This flag is asserted when: • The core is configured in either dual or quad SPI mode and • The core is configured in master = 0 in control register (SPICR(2)). In standard SPI mode, this bit is always in default state.

Table 2-13: **IP Interrupt Status Register Description (Core Base Address + 0x20) *(Cont'd)***

Bits	Name	Core Access	Reset Value	Description
9	CPOL_CPHA Error	R/TOW[1]	0	CPOL_CPHA error. IPISR Bit[9] is the CPOL_CPHA error. This flag is asserted when: • The core is configured in either dual or quad SPI mode and • The CPOL - CPHA control register bits are set to `01` or `10`. In standard SPI mode, this bit is always in default state.
8	DRR_Not_Empty	R/TOW[1]	0	DRR not empty. IPISR Bit[8] is the DRR not empty bit. The assertion of this bit is applicable only in the case where **FIFO Depth** is 16 or 256 and the core is configured in slave mode and standard SPI mode. This bit is set when the DRR FIFO receives the first data value during the SPI transaction. This bit is set by a one-clock period strobe to the interrupt register when the core receives the first data beat. ***Note:*** The assertion of this bit is applicable only when the **FIFO Depth** parameter is 16 or 256 and the core is configured in slave mode in standard SPI mode. When **FIFO Depth** is set to 0, this bit always returns 0. This bit has no significance in dual/quad mode.
7	Slave_Select_Mode	R/TOW[1]	0	Slave select mode. IPISR Bit[7] is the slave select mode bit. The assertion of this bit is applicable only when the core is configured in slave mode in standard SPI configuration. This bit is set when the other SPI master core selects the core by asserting the slave select line. This bit is set by a one-clock period strobe to the interrupt register. ***Note:*** This bit is applicable only in standard SPI slave mode.
6	TX FIFO Half Empty	R/TOW[1]	0	Transmit FIFO half empty. In standard SPI configuration, IPISR Bit[6] is the transmit FIFO half-empty interrupt. For example, when FIFO depth = 16, 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. In this mode, the FIFO depth is fixed to 16 only. The same logic applies when the FIFO depth is 256 where `10000000` changes to `01111111`. In dual or quad SPI configuration, based on the FIFO depth, this bit is set at half-empty condition. ***Note:*** This interrupt exists only if the AXI Quad SPI core is configured with FIFOs (In standard, dual or quad SPI mode).

Table 2-13: **IP Interrupt Status Register Description (Core Base Address + 0x20) *(Cont'd)***

Bits	Name	Core Access	Reset Value	Description
5	DRR Overrun	R/TOW[1]	0	Data receive register/FIFO overrun. IPISR Bit[5] is the data receive FIFO overrun 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 to complete a SPI transfer. This can occur when the SPI device is in either master or slave mode (in standard SPI mode) or if the IP is configured in SPI master mode (dual or quad SPI mode).
4	DRR Full	R/TOW[1]	0	Data receive register/FIFO full. IPISR Bit[4] is the data receive register full interrupt. Without FIFOs, this bit is set at the end of a SPI element transfer by a one-clock period strobe to the interrupt register (An element can be a byte, half-word, or word depending on the value of **Transfer Width**). With FIFOs, this bit is set at the end of the SPI element transfer, when the receive FIFO has been completely filled by a one-clock period strobe to the interrupt register.
3	DTR Underrun	R/TOW[1]	0	Data transmit register/FIFO underrun. IPISR Bit[3] is the data transmit register/FIFO under-run interrupt. This bit is set at the end of a 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 to perform a SPI transfer. This can occur only when the SPI device is configured as a slave in standard SPI configuration and is enabled by the SPE bit as set. All zeros are loaded in the shift register and transmitted by the slave in an under-run condition.
2	DTR Empty[2]	R/TOW[1]	0	Data transmit register/FIFO empty. IPISR Bit[2] is the data transmit register/FIFO empty interrupt. It is set when the last byte of data has been transferred out to the external flash memory. See Transfer End Period in Chapter 3. In the context of the M68HC11 reference manual, when configured without FIFOs, this interrupt is equivalent in information content to the complement of the SPI transfer complete flag ($\overline{\text{SPIF}}$) interrupt bit. In master mode if this bit is set to 1, no more SPI transfers are permitted.
1	Slave MODF	R/TOW[1]	0	Slave mode-fault error. IPISR Bit[1] is the slave mode-fault error flag. This interrupt is generated if the $\overline{\text{SS}}$ signal goes active while the SPI device is configured as a slave, but is not enabled. This bit is set immediately on $\overline{\text{SS}}$ going active and continually set if $\overline{\text{SS}}$ is active and the device is not enabled.

Table 2-13: **IP Interrupt Status Register Description (Core Base Address + 0x20) *(Cont'd)***

Bits	Name	Core Access	Reset Value	Description
0	MODF	R/TOW(1)	0	Mode-fault error. IPISR Bit[0] 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 on $\overline{SS}$ 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.
2. Look for the DTR Empty bit in IPISR to make sure that the transaction from the IP to flash memory is complete.

IP Interrupt Enable Register (IPIER)

The Interrupt Enable Register (IPIER) register allows the system interrupt output to be active. This interrupt is generated if an active bit in the IPISR register corresponds to an enabled bit in the IPIER register. The IPIER register has an enable bit for each defined bit of the IPISR as shown in Figure 2-11 and described in Table 2-14. All bits are cleared on reset.

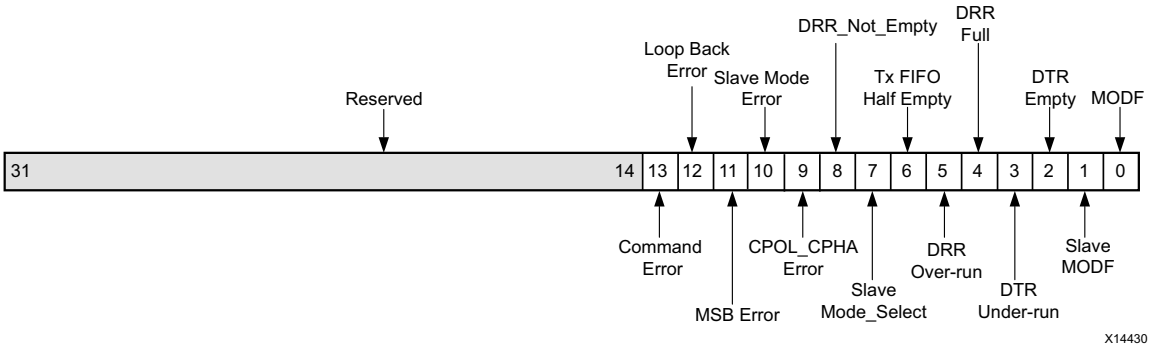

Figure 2-11: **IP Interrupt Enable Register (Core Base Address + 0x28)**

Table 2-14: **IP Interrupt Enable Register Description (Core Base Address + 0x28)**

Bits	Name	Core Access	Reset Value	Description
31:14	Reserved	N/A	N/A	Reserved
13	Command Error	R/W	0	Command error. 0 = Disabled. 1 = Enabled. This bit is applicable only when the core is configured in dual or quad SPI mode.

Table 2-14: **IP Interrupt Enable Register Description (Core Base Address + 0x28) *(Cont'd)***

Bits	Name	Core Access	Reset Value	Description
12	Loopback Error	R/W	0	Loopback error. 0 = Disabled. 1 = Enabled. This bit is applicable only when the core is configured in dual or quad SPI mode.
11	MSB Error	R/W	0	MSB error. 0 = Disabled. 1 = Enabled. This bit is applicable only when the core is configured in dual or quad SPI mode.
10	Slave Mode Error	R/W	0	I/O mode instruction error. 0 = Disabled. 1 = Enabled. This bit is applicable only when the core is configured in dual or quad SPI mode.
9	CPOL_CPHA Error	R/W	0	CPOL_CPHA error. 0 = Disabled. 1 = Enabled. This bit is applicable only when the core is configured in dual or quad SPI mode.
8	DRR_Not_ Empty	R/W	0	DRR_Not_Empty. 0 = Disabled. 1 = Enabled. ***Note:*** The setting of this bit is applicable only when **FIFO Depth** is set to 1 and the core is configured in slave mode of standard SPI mode. If **FIFO Depth** is set to 0, the setting of this bit has no effect. This is allowed only in standard SPI configuration. It means this bit is not set in the IPIER register. Therefore, this bit should only be used when **FIFO Depth** is set to 1 and when the core is configured in slave mode. This bit has no significance in dual or quad mode.
7	Slave_Select_ Mode	R/W	0	Slave_Select_Mode. 0 = Disabled. 1 = Enabled. This bit is applicable only when the core is configured in slave mode by selecting the active-Low status on `spisel`. In master mode, setting this bit has no effect.
6	TX FIFO Half Empty	R/W	0	Transmit FIFO half empty. 0 = Disabled. 1 = Enabled. ***Note:*** This bit is meaningful only if the AXI Quad SPI core is configured with FIFOs.

Table 2-14: **IP Interrupt Enable Register Description (Core Base Address + 0x28) *(Cont'd)***

Bits	Name	Core Access	Reset Value	Description
5	DRR Overrun	R/W	0	Receive FIFO overrun. 0 = Disabled. 1 = Enabled.
4	DRR Full	R/W	0	Data receive register/FIFO full. 0 = Disabled. 1 = Enabled.
3	DTR Underrun	R/W	0	Data transmit FIFO underrun. 0 = Disabled. 1 = Enabled.
2	DTR Empty	R/W	0	Data transmit register/FIFO empty. 0 = Disabled. 1 = Enabled.
1	Slave MODF	R/W	0	Slave mode-fault error flag. 0 = Disabled. 1 = Enabled.
0	MODF	R/W	0	Mode-fault error flag. 0 = Disabled. 1 = Enabled.

XIP Mode

When the AXI Quad SPI core is configured in XIP mode, only the following registers are available through the AXI4-Lite interface:

- XIP Configuration Register
- XIP Status Register

These 32-bit registers are configurable and accessible individually through the AXI4-Lite interface. Table 2-15 provides a summary of the AXI Quad SPI core registers in XIP mode.

Table 2-15: **Core Registers in Enhanced Mode XIP Mode**

Base Address Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
Core Grouping				
60	XIP Config_Reg (XIP-CR)	R/W	0x0	XIP configuration register
64	XIP Status_Reg (XIP-SR)	Read	0x0	XIP status register

IMPORTANT: *In XIP mode, the AXI QUAD SPI does not generate any interrupt. The interrupt pin can be left unconnected.*

XIP Mode Commands

The registers in XIP mode are accessed through the AXI4-Lite interface. Based on the core configuration in this mode, the core supports three read commands:

- Fast Read (`0x0Bh`)
- Fast Read Dual I/O (`0xBBh`)
- Fast Read Quad I/O (`0xEBh`)

Mode determines which one of these commands is supported by the core. These commands are built in and no other commands need to be configured. When **Mode** is set, the same command operates during the whole transaction.

XIP Configuration Register

The bit assignments for the XIP Configuration Register (XIP-CR) are shown in Figure 2-12. This register is used to configure the XIP (read-only) mode. This is a read/write register used to configure the CPOL and CPHA modes. In XIP mode, either CPOL–CPHA = 00 or CPOL–CPHA = 11 is supported. For any other combination the error flag is set in the status register and the transaction on the AXI4 interface is not accepted by the core. Before the start of any new AXI4 transaction, the core checks the CPOL and CPHA settings.

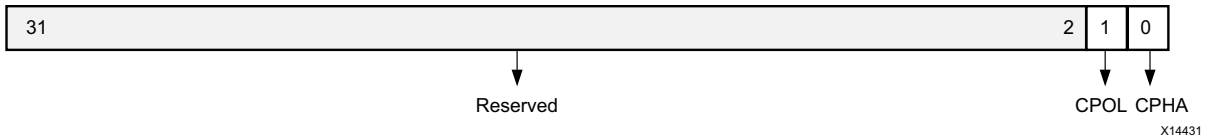

Figure 2-12: **XIP Configuration Register (Core Base Address+0x60)**

XIP Status Register

The bit assignments for the XIP Status Register (XIP-SR) are shown in Figure 2-13 and described in Table 2-16. This register is used to check the status of commands and other processes performed by the core. In the case where the core receives write commands or write transactions, an error is generated and this is indicated in the status register. This is a read-only register. Any attempt to write to this register completes with a standard acknowledge and the register contents are not updated. The contents of this register are reset as soon as it is read. As there is no timeout counter involved in the core, if the AXI4 transaction is not accepted, check the status registers for any errors. If there are errors, the core does not accept the AXI4 transaction.

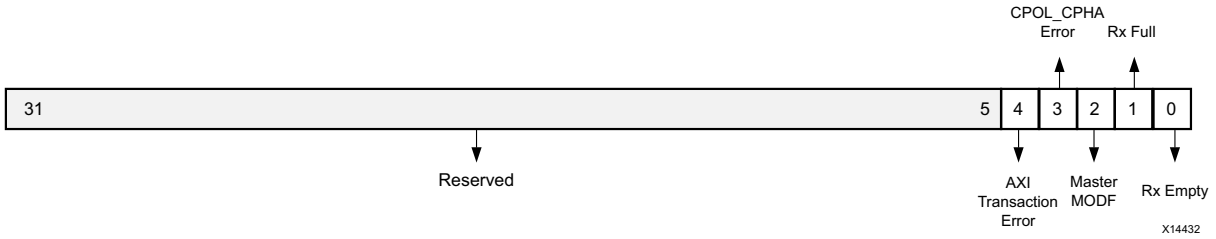

Figure 2-13: **XIP Status Register (Core Base Address+0x64)**

Table 2-16: **XIP Status Register Description (Core Base Address+0x64)**

Bits	Name	Core Access	Reset Value	Description
31:5	Reserved	N/A	0	Reserved
4	AXI Transaction Error	R	0	AXI transaction error
3	CPOL_CPHA Error	R	0	CPOL_CPHA error
2	Master MODF	R	0	Master mode fault. This bit is set to 1 if the `spisel` line is deasserted.
1	RX Full	R	0	Receiver full
0	RX Empty	R	1	Receiver empty

Specification Exceptions

Exceptions from the Motorola M68HC11-Rev. 4.0 Reference Manual

- A slave mode-fault error is added to provide an interrupt if a SPI device is configured as a slave and is selected when not enabled.
- In this design, the SPI DTR and SPI DRR registers have independent addresses. This is an exception to the M68HC11 specification that calls for two registers to have the same address.
- 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.
- Manual control of the $\overline{SS}$ signals is provided by setting Bit[7] in the SPICR register. When the device is configured as a master and is enabled while Bit[7] 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.

- A control bit is provided to inhibit master transfers. This bit is effective in any master mode, but its primary intent is manual control of the $\overline{SS}$ signals.
- In the M68HC11 implementation, the transmit register is transparent to the shift register which necessitates the write collision error (WCOL) detection hardware. This feature is not implemented in this design.
- 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 place of the SPIE bit, there is a bit to select local master loopback mode for testing.
- An option is implemented in this design to implement FIFOs on both transmit and receive (Full Duplex only) mode.
- M68HC11 implementation supports only byte transfer. In this design either a byte, half-word, or word transfer can be configured using the Transaction Width parameter.
- The baud rate generator is specified by Motorola to be programmable using bits in the control register; however, in this FPGA design the baud rate generator is programmable using parameters in the VHDL implementation. Thus, run-time configuration of the baud rate is not possible. Beyond the ratios of 2, 4, 16 and 32, all integer multiples of 16 up to 2048 are allowed.
- Most of the SPI slave devices support the SPI modes 0 and 3. For some of these devices, the data valid time of 8 ns from the falling edge of `SCK` is applicable. While operating with these devices at a higher speed of 50 MHz (most of the instructions support this speed), the core should be configured with **Frequency Ratio** set to 2 (where the AXI 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 is registered in the middle of each SPI rising and next consecutive falling edge inside the core. This adds 5 ns of time to the core (while operating at 100 MHz on the `ext_spi_clk` port) for registering the data. As per the M68HC11 document, the master should register data on each rising edge of `SCK` in SPI modes 1 and 3. Note that the data registering mechanism when **Frequency Ratio** is 2 follows a different pattern than specified in the standard although this is applicable to the data registering mechanism in the 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.
- When the AXI Quad SPI core is configured in slave mode (**Mode** set to **Standard**), the data in the core is registered on the SCK rising edge + one AXI clock cycle. Internally, this data is registered on the next rising edge of the AXI clock cycle. The core changes the data on the `SCK` falling edge + one AXI clock cycle.
- In Standard SPI mode of operation, when the SCK RATIO = 16 is used, the core provides approximately seven cycles of `ext_spi_clk` setup time for the downstream device to register the data on the next SPI rising edge.

Other Exceptions

- The AXI Quad SPI core supports one memory selection at a time. This means, in multi-slave systems, the core should be configured to select and perform operations only on one slave at a time.
- The core is based on the specific memory parts from Winbond (W25Q80), Micron (N25Q256), Macronix (MX66U1G45G), and Spansion (S70FL01GS). To test the core with other memory parts, ensure that the internal command decoding logic and its bit positions are understood for correct operation of the core.
  Also, check the common command set between the Winbond or Micron or Spansion or Macronix memory part data sheet and other memory parts to confirm that the common commands between these documents are executed. In quad mode, the design supports the Micron memory parts with HOLD functionality only. The memory parts with RESET functionality are not supported in the design.
- See Following are unsupported Commands for Dual/Quad SPI Mode and Winbond or Micron or Spansion in Chapter 3.
- Dedicated memory should be used as AXI Quad SPI slaves because the core supports a limited set of commands which are common in Winbond, Micron, and Spansion memories in terms of command, address, data requirement and their Macronix behavior. If single dedicated memories are used with the core, a wider range of commands is supported for the respective memory and performance is optimized.
- In XIP mode, there are several clock domain crossing signals present between the AXI4 and AXI4-Lite interfaces and the SPI domain. When reading the XIP SR, note that the core takes five clock cycles to update the status bits.
- XIP configuration mode does not support byte access mode.
- The core does not support queued commands. The core design is based on commands supported by standard SPI devices such as Winbond, Micron, Macronix, and Spansion memory only.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

General Design Guidelines

Functionality Based on AXI Interfaces

AXI4-Lite Interface

If the AXI4-Lite interface is chosen, all the registers including the SPI DTR and SPI DRR are 32-bit, single-access registers. The SPI DTR and SPI DRR registers have only 8 valid bits out of 32.

AXI4 Interface (Enhanced Mode)

If the AXI4 interface is chosen, it is mandatory that all the registers are single-length access only. If burst is attempted (except for the SPI DTR or SPI DRR), the core behavior is not guaranteed. The SPI DTR and SPI DRR registers can be accessed as 32-bits. The FIXED burst is allowed only for the SPI DTR and SPI DRR registers. Out of 32 bits of FIXED burst, only 8 bits are valid. Read the occupancy registers before initiating the recursive FIXED burst. The occupancy registers provide the length of the FIXED burst to be performed. Only the last eight bits should be considered as actual data.

While carrying out the write burst operation in the DTR register, it is illegal to have holes in the write strobes. This is not allowed by the core and core behavior in this instance is not guaranteed.

When starting any new transaction at the SPI flash memory, the DTR FIFO should be always filled with a command followed by address, dummy bytes, then data bytes. This sequence should be strictly adhered to by the application. The read or write data occupancy registers indicate the number of locations left in the receive or transmit FIFO which help to determine the burst length of the next transaction. This mode is most suitable for CDMA-based applications.

AXI4 Read-Only Interface (XIP Mode)

In this interface, only the read transactions are allowed from the AXI4 interface. The write transactions are not allowed and are considered an error by the core. The XIP registers are accessible through the AXI4-Lite interface. This mode is most suitable for using flash devices in ROM-based applications.

Standard SPI Device Features with Only AXI4-Lite Interface

The SPI device includes these standard features in Standard SPI configuration plus those listed in the Features section of IP Facts:

- Supports multi-master configuration within the FPGA with separated _I, _O, _T representation of 3-state ports.
- Works with N times 8-bit data characters in default configuration. The default mode implements manual control of the $\overline{SS}$ output using data written to the SPISSR. This appears directly on the $\overline{SS}$ output when the master is enabled. This mode can only be used with external slave devices. An optional operation can be selected where the $\overline{SS}$ output is toggled automatically (when FIFO is disabled) with each 8-bit character transfer by the master device using a bit in the SPICR for SPI master devices.
- Multi-master environment supported (implemented with 3-state drivers and requires software arbitration for possible conflict). See AXI4-Lite Interface Functionality in Standard SPI Multi-Master Configuration.
- Multi-slave environment supported (automatic generation of additional slave select output signals for the master).
- Supports maximum SPI clock rates up to one-half the AXI clock rate in master mode and one-fourth the AXI clock rate in slave modes (C_SCK_RATIO = 2 is not supported in slave mode due to the synchronization method used between the AXI and SPI clocks). It is required that the AXI and external clock signals be aligned when configured in slave mode.
- Parameterizable baud rate generator for the SPI clock signal.
- The WCOL flag is not supported as a write collision error as described in the M68HC11 reference manual. Do not write to the transmit register when a SPI data transfer is in progress.
- Back-to-back transactions are supported — multiple, uninterrupted byte/half-word/word transfers can occur provided that the transmit FIFO never gets empty and the 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 in that an 8-bit shift register in the SPI master device is connected to another 8-bit shift register in a SPI slave device.

AXI4-Lite Interface Functionality in Standard SPI Multi-Master Configuration

The SPI bus to a given slave device (Nth device) consists of four wires:

- Serial clock (SCK)
- IO0 (Master out, slave in (MOSI))
- IO1 (Master in, slave out (MISO))
- Slave select (SS(N))

The signals SCK, IO0(MOSI), and IO1(MISO) are shared for all slaves and masters. See Figure 3-1, where any one of the SPI devices can be configured as a master and the others can be configured as slaves (via register(60h) configuration). In such cases the master will drive the SPISEL pin of the slaves from the ss pins.

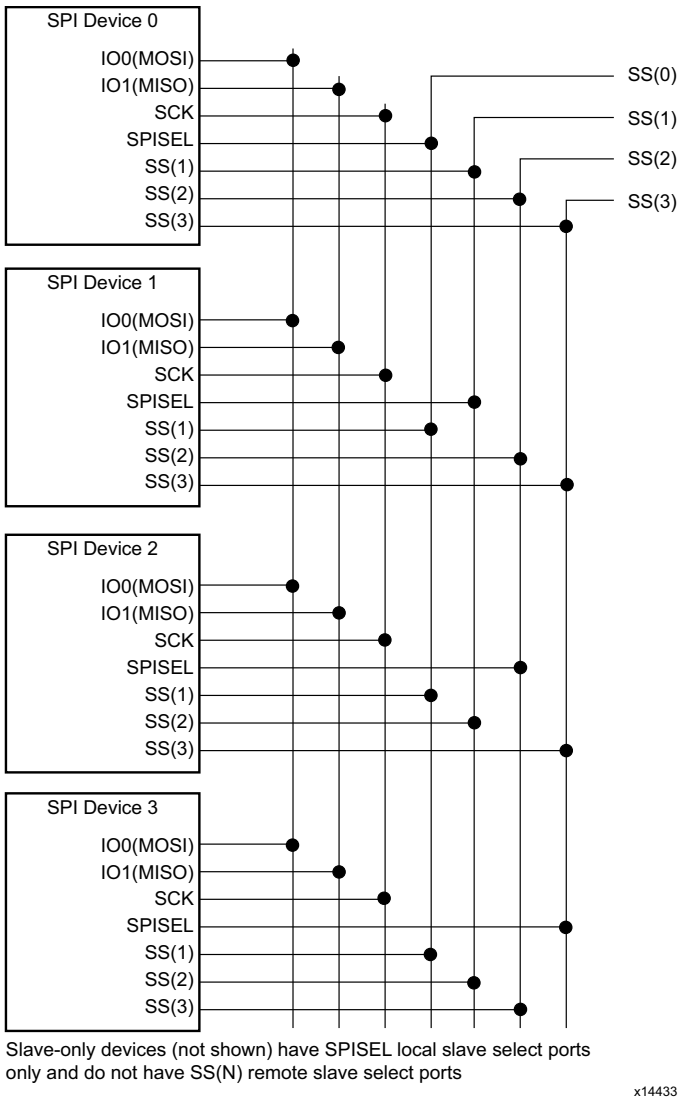

Figure 3-1: **Multi-Master Configuration Block Diagram for Standard SPI Mode**

Note: When the core is generated in master mode but via register configuration (60h) if it is configured as a slave, SPISEL should be driven to the core from the SPI master.

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 both SPI master/slave devices to be internal to the FPGA and SPI slave devices to be external to the FPGA. SPI pins are automatically generated through the Vivado® Design Suite when interfacing to an external SPI slave device. Multiple SPI master/slave devices are shown in Figure 3-1. The same configuration diagram is applicable for dual mode.

AXI4-Lite Interface Standard SPI Mode — Optional FIFOs in Legacy Mode

When the core is configured in standard SPI mode you have the flexibility of including optional FIFOs of either 16 or 256 deep in the design. Because the AXI Quad SPI is full-duplex, both transmit and receive FIFOs are instantiated as a pair and can be included in the core as shown in Figure 1-1, page 5.

When FIFOs are implemented, the slave select address is required to be the same for all data buffered in the FIFOs. This is necessary because there is no FIFO for the slave select address. Both transmit and receive FIFOs are 16 (or 256) elements deep and are accessed using single AXI transactions because burst mode is not supported.

The transmit FIFO is write-only. When data is written into the FIFO, the occupancy number is incremented and when a SPI transfer is completed, the number is decremented. As a consequence of this operation, aborted SPI transfers still have the data available for the transmission retry. The transfers can only be aborted in the master mode by setting the master transaction inhibit bit, bit 8 of the SPICR, to 1 during a transfer. Setting this bit in the slave mode has no effect on the operation of the slave. These aborted transfers are on the SPI interface. The occupancy number is contained in a read-only register.

If a write is attempted when the FIFO is full, an acknowledgment is given along with a generated error signal. Interrupts associated with the transmit FIFO include:

- Data transmit FIFO empty
- Transmit FIFO half empty
- Transmit FIFO under-run

See Interrupt Register Set Description in Chapter 2 for details.

The receive FIFO is read-only. When data is read from the FIFO, the occupancy number is decremented and when a SPI transfer is completed, the number is incremented. If a read is attempted when the FIFO is empty, acknowledgment is given along with a generated error signal. 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 a SPI transfer. If the receive FIFO is full and more data is received, a receive FIFO overflow interrupt is issued. When this occurs, all attempts to write data to the full receive FIFO by the SPI module are lost.

When the AXI Quad SPI core is configured with FIFOs, SPI transfers 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, the SPI transfer begins immediately after the write to the master transmit FIFO. If the FIFO is emptied using SPI transfers before additional elements are written to the transmit FIFO, an interrupt is asserted. When the AXI to SPI SCK frequency ratio is sufficiently small, this scenario is highly probable.

Alternatively, the FIFO can be loaded with up to 16 or 256 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.

AXI4-Lite Interface Dual/Quad SPI Mode — Optional FIFO Depth

When the core is configured in dual or quad SPI mode, the FIFO is always present and there is the option to choose its depth. The depth of this FIFO can be either 16 or 256. The width of the FIFO in this mode is always 8 bits.

AXI4-Lite Interface SPI Master Loopback Operation

SPI master loopback operation, although not included in the M68HC11 reference manual, has been implemented to expedite testing. This operation is selected by setting the loopback bit in the SPICR (SPICR(0)) enabling an internal connection from the transmitter output to the receiver input. The receiver and transmitter operate normally, except that received data (from a remote slave) is ignored. This operation is relevant only when the SPI device is configured as a master and operated in standard SPI transaction mode. When the core is configured in dual or quad SPI mode, loopback mode is not supported.

AXI4-Lite Interface Hardware Error Detection

The SPI architecture relies on software-controlled bus arbitration for multi-master configurations to avoid conflicts and errors. However, limited error detection is implemented in the SPI hardware. The first error detection mechanism to be discussed is contention error detection. This mechanism detects when a SPI device, configured as a master, is selected (that is, its SS bit is asserted) by another SPI device which is 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, the process might require clearing the FIFOs.

A similar error detection mechanism has been implemented for SPI slave devices. The detected error occurs when a SPI device configured as a slave, but not enabled, is selected (that is, its SS bit is asserted) by another SPI device. When this condition is detected, IPISR bit 1 is set by a strobe to the IPISR register.

Underrun and overrun condition error detection is also provided. Underrun conditions can happen only when operated in slave mode. This condition occurs when a master commands a transfer, but the slave does not have data in the transmit register or FIFO to transfer. In this case, the slave underrun interrupt is asserted and the slave shift register is loaded with all zeros for transmission. An overrun condition can occur to both master and slave devices

where a transfer occurs when the receive register or FIFO is full. During an overrun condition, the data received in that transfer is not registered (it is lost) and the IPISR overrun interrupt bit 5 is asserted.

Setting the Frequency Ratio Parameter

The AXI Quad SPI core is tested in hardware with SPI slave devices such as Micron, Winbond, Macronix, and Spansion Flash memories in standard SPI mode. Standard, dual and quad modes are tested with Winbond, Micron, Spansion, and Macronix SPI flash memories. Read the data sheet of the targeted SPI slave flash memory or EEPROMs for the maximum speed of operation. Ensure that the correct values are used when deciding on the AXI clock and selecting the Frequency Ratio parameter. The AXI clock and the Frequency Ratio parameter determine the clock frequency at the SCK pin of the core. While using different external SPI slave devices, the Frequency Ratio parameter should be set carefully, and the maximum clock frequencies supported by all the external SPI slave devices should be taken into account.

AXI4-Lite Interface SPI Slave Mode — Standard SPI Configuration in Legacy Mode Only

The AXI Quad SPI core can be configured in slave mode by connecting the slave select line of the external master to SPISEL and by setting bit 2 of the SPI control register (SPICR) to 0. Slave mode is allowed only in standard SPI mode. In dual or quad SPI mode, the core only supports master mode.

All incoming signals are synchronized to the AXI clock when the Frequency Ratio parameter is greater than 4. Because of the tight timing requirements when the Frequency Ratio parameter is set to 4, the incoming SCK clock signal and its synchronized signals are used directly in the internal logic. Therefore, the external clock must be synchronized with the AXI clock when the Frequency Ratio parameter is 4. For other Frequency Ratio parameter values, it is preferred, but might not be necessary to have such synchronization.

RECOMMENDED: *In slave mode operation, use the FIFO by setting **FIFO Depth** to 16 or 256 in standard SPI mode.*

In slave mode, two interrupts are available in addition to the other interrupts in the IPISR:

- DRR_Not_Empty (bit 8)
- Slave_Mode_Select (bit 7)

Before the other SPI master starts communication, it is mandatory to fill the slave core transmit FIFO with the required data beats. When the master starts communication, with the core configured in slave mode, the core transfers data till the data is available in its transmit FIFO. At the end of last data beat transmitted from the slave FIFO, the core (in slave mode)

generates the DTR empty signal to notify that new data beats need to be filled in its transmit FIFO before further communication can start.

Using the Enable STARTUPE*n* Primitive Parameter

The Enable STARTUPE*n* Primitive parameter is applicable in master SPI mode. When this parameter is enabled, the STARTUPE2 primitive (for 7 series) and STARTUPE3 primitive (for UltraScale™ devices) are included in the design and become part of the core after configuration of the FPGA.

See the answer records or the device user guide to understand more about the functionality and use of the STARTUPE*n* primitive.

Enable STARTUPEn Primitive is Selected

Core Behavior and Ports

- The `SCK_O` port from the core is interfaced with the STARTUPEn primitive. The STARTUPEn can also be used in the pre-configuration process of the FPGA where the external SPI slave memory is configured prior to the FPGA.
- After configuration of the FPGA, the `SCK_O` port (output from the core) drives the `USRCCLK0` port of the primitive. This signal is not available as the external port of the core.
- Instantiating STARTUPE*n* affects the maximum frequency of `ext_spi_clk`. For maximum supported frequencies refer to Table 2-1. Also see Required Constraints in Chapter 4.

Using the Dual Quad Mode

The **Enable Dual Quad Mode** parameter is applicable only (UltraScale™ and future devices) in following configuration of the IP core.

- Master mode
- STARTUP enabled
- SPI mode is QUAD
- Number of slaves is 2

When this parameter is enabled, the IP core has two SPI interfaces. IP is connected to two flashes/SPI slave devices as shown in Figure 3-2.

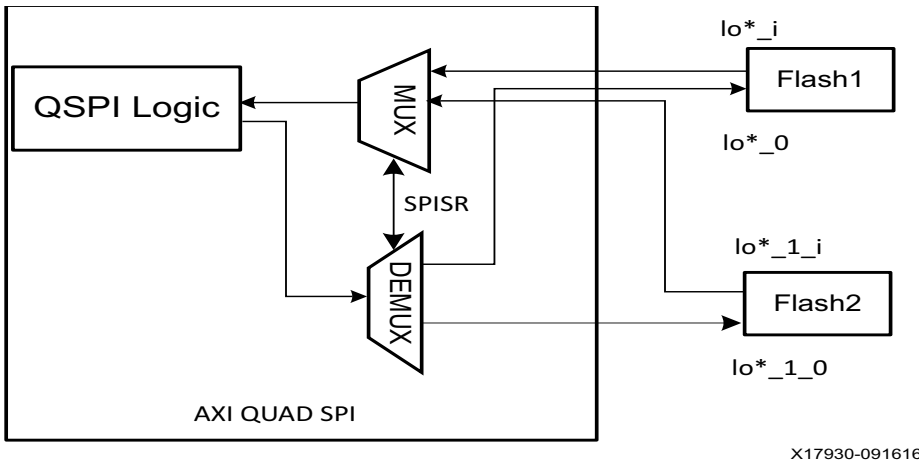

Figure 3-2: **Dual Quad Mode**

Traffic to SPI slaves can be controlled using the slave select register (SPISSR).

Enable STARTUPEn Primitive is Not Selected

Core Behavior and Ports

- As the `SCK_O` and `IO1_I` ports are part of the core and no primitive in instantiated in the core, these ports are available as external ports of the core and they are placed in an IOB at a user-configured location.

Core Behavior in Legacy and Enhanced Non-XIP Mode

This mode is set whether or not **Enable Performance Mode** is selected and when it is, **Enable XIP Mode** is not selected. The AXI Quad SPI core supports Winbond, Micron, Spansion and Macronix memories. Check the commands if different memories must be tested with the core. If the commands, address, and data behavior are the same for a different memory, that device can be chosen as the base memory to test the core.

The core understands the commands and its expected behavior for the targeted memory through internal logic. The commands which are not supported by the Winbond, Micron, Macronix, and Spansion memory device mentioned in the data sheet are marked with a command error. After the command error is set, the core does not execute the SPI transaction for that command and a command error interrupt is generated. After the command phase, if there is an address phase included, the next DTR contents are transferred on a SPI transaction in the modes defined by the address mode bits. If the data phase is present for the particular command, the data phase is executed based on read or write, with the modes set by the data mode bits.

The dummy bytes, which are required in some of the instructions for the selected memory, should be part of the SPI DTR along with the number of bytes intended to read from memory. For more information on the number of dummy bytes needed for a particular instruction, consult the data sheet for the targeted memory.

For read commands, after the transmission of the address bits, the core immediately reverts to input mode and starts storing data in the DRR. Therefore, be aware of how many dummy bytes are to be ignored in the DRR. For example, for the fast read dual output command in Winbond memory, the DTR should be filled with one command byte plus three address bytes plus two dummy bytes for the dummy cycles plus the number of dummy bytes to be read from memory. The command and address are transferred in standard SPI mode, after which the core reverts to the input mode and starts storing the data. The data is transferred on the `IO0_I` and `IO_1` lines and stored in the SPI DRR, which includes the two dummy cycles plus valid data. Therefore, while reading the SPI DRR, ignore the first 6 bytes of the SPI DRR. The valid data available in the FIFO begins with the seventh byte. This also applies to other dual or quad read commands.

For each fresh transaction, the SPI DTR FIFO must be cleared. The first entry in the SPI DTR is always considered the command entry, which is cross-checked against built-in logic for the respective memory in the selected SPI mode.

Core Behavior in XIP Mode

When **Enable Performance Mode** and **Enable XIP Mode** are both selected, the core supports standard, dual and quad modes:

- Standard mode is set with **Mode** as **Standard** and **Slave Device** as **Winbond**, **Micron**, **Macronix**, or **Spansion**.
- Dual mode is set with **Mode** is **Dual** and **Slave Device** is **Winbond**, **Micron**, **Macronix,** or **Spansion**.
- Quad mode is set with **Mode** is **Quad** and **Slave Device** is **Winbond**, **Micron**, **Macronix,** or **Spansion**.

Assumptions for this mode are:

Winbond Memory

Before executing the DIOFR (`0xBBh`) or QIOFR (`0xEBh`), the core executes the high performance mode command on each power-on reset state. This ensures that the Winbond memory is configured in high-performance mode.

IMPORTANT: *When quad mode is set, the Winbond memory must be pre-configured by writing to the status register to set the QE bit to 1.*

It is your responsibility to pre-configure the memory. The core does not write anything to the status register. The core assumes that this exercise is completed previously in XIP mode.

When the core is configured in dual or quad mode prior to executing the DIOFR or QIOFR commands, the core performs the high performance command write to the memory on POR before accepting the transaction on the AXI4 interface. The HPM command requires one command and three dummy SPI cycles. This writing of the HPM command in memory is performed only at the power-on condition. The memory is now placed in high performance mode (HPM — `0xA3h`) and allows DIOFR or QIOFR to operate in their respective modes.

Micron Memory

In Micron memory, the volatile as well as nonvolatile configuration registers are left with the default configuration of dummy cycles. That is, VCR[7:4] and NVCR[15:12] are set to 1111. The core behavior is based on this assumption only and if these dummy cycle register values are changed, the core behavior is not guaranteed.

RECOMMENDED: *Do not change the default Volatile Configuration Register (VCR) and Non Volatile Configuration Register (NVCR) configuration.*

At the start of each new transaction, the core sends the respective command, address and required dummy cycles and then receives data.

Spansion Memory

It is your responsibility to pre-configure the Spansion memory. The core does not write anything to the status register. The core assumes that this exercise is completed previously in XIP mode.

IMPORTANT: *When quad mode is set, the Spansion memory must be pre-configured by writing to the configuration register to set the QUAD bit to 1.*

Macronix Memory

In Macronix memory, the configuration register handles the dummy cycle information. Configuration of dummy cycle numbers is different depending on the bit6 & bit7 (DC0 & DC1) setting in the configuration register. By Default, the values of bit6 and bit 7 are 00. Macronix data sheet provides information on dummy cycle for varying the DC0 and DC1.

Note: For more details, refer "Dummy Cycle and Frequency Table (MHz)" Table in Macronix data sheet MX66U1G45G.

RECOMMENDED: *Do not change the default Configuration Register (CR) configuration.*

Commonly Supported Commands for Dual SPI and Mixed Memory Mode

For the configuration with **Mode** set to **Dual** and **Slave Device** set to **Mixed**, the core supports the commands listed in Table 3-1, which are identical (in terms of command, address and data behavior) in Winbond, Micron and Spansion memory devices. See the memory data sheet for command details. The commands that are not supported in this mode include fast read, dual I/O fast read, and dual output fast read. These commands are not supported by the core in mixed mode because the dummy bytes or dummy cycles are different between the Winbond, Micron and Spansion devices.

Also, in mixed mode, the volatile configuration register of the Micron device is not supported as this command is not present in the Winbond device. See Following are unsupported Commands for Dual/Quad SPI Mode and Winbond or Micron or Spansion for a list of unsupported commands.

Table 3-1: **Supported Commands in Dual SPI and Mixed Memory Mode**

Opcode (Hex)	Command Description
01	Write status register
02	Page program
03	Read data bytes
04	Write disable
05	Read status register
06	Write enable
4B	Read One Time Programmable (OTP) (Read of OTP area)
75	Program/erase suspend
7A	Program/erase resume
9F	Read identification ID
C7	Bulk erase
D8	Sector erase

Commonly Supported Commands for Quad SPI and Mixed Memory Mode

For the configuration with **Mode** set to **Quad** and **Slave Device** set to **Mixed**, the core supports the commands listed in Table 3-2, which are identical (in terms of command, address and data behavior) in Winbond, Micron, and Spansion memory devices. See the memory data sheet for command details. The commands which are not supported in this mode include fast read, dual output fast read, quad output fast read, dual i/o fast read, and quad i/o fast read. These commands are not supported by the core in mixed mode because the dummy bytes or dummy cycles are different in Winbond and Micron devices. Also in mixed mode, the volatile configuration register of Micron devices is not supported as this

command is not present in the Winbond device. See Following are unsupported Commands for Dual/Quad SPI Mode and Winbond or Micron or Spansion for a list of unsupported commands.

Table 3-2: **Supported Commands in Quad Mode and Mixed Mode Memory**

Opcode (Hex)	Command Description
01	Write status register
02	Page program
03	Read data bytes
04	Write disable
05	Read status register
06	Write enable
32	Quad IP page program
C7	Bulk erase
75	Erase suspend
7A	Erase resume
4B	Read unique ID number
9F	Read JEDEC® ID number
D8	Sector erase

XIP Mode Commands

In XIP mode, the core supports three read commands:

- In standard mode: fast read – `0x0Bh`
- In dual mode: fast read dual I/O – `0xBBh`
- Quad mode: fast read quad I/O – `0xEBh`

Table 3-3: **Supported Commands in AXI Quad SPI Core**

Command Opcode	Command Description	Winbond	Spansion	Numonyx /Micron	Macronix
9E/9F	JEDEC ID(electronic identity of each flash memory)/SPAN_RDID in SPANSION / READ ID in Numonyx in Macronix	Yes	Yes	Yes	Yes
0B	Fast read(read operating at highest possible frequency of flash)	Yes	Yes	Yes	Yes
3B	Fast read dual output(2 lines fast read)	Yes	Yes	Yes	Yes

Table 3-3: **Supported Commands in AXI Quad SPI Core**

Command Opcode	Command Description	Winbond	Spansion	Numonyx /Micron	Macronix
BB	Fast read dual I/O(similar to (3Bh) ,but with the capability to input the Address bits (A23-0) two bits per clock)	Yes	Yes	Yes	Yes
6B	Fast read quad output(fast read 4 lines)	Yes	Yes	Yes	Yes
EB	Fast read quad I/O(similar to (3Bh) ,but with the capability to input the Address bits (A23-0) 4 bits per clock.,)	Yes	Yes	Yes	Yes
06	WEN(write enable sets the Write Enable Latch (WEL) bit in the Status Register to a 1. The WEL bit must be set prior to every Page Program, Sector Erase, Block Erase, Chip Erase and Write Status Register instruction.) WRITE ENABLE in Numonyx / in Macronix	Yes	Yes	Yes	Yes
04	Write disable(sets WEL to 0)	Yes	Yes	Yes	Yes
02	Page program(The Page Program instruction allows from one byte to 256 bytes (a page) of data to be programmed at previously erased (FFh) memory locations)	Yes	Yes	Yes	Yes
32	Quad page program(page program with 4 pins)	Yes	Yes	Yes	No
D8	Block erase(The Block Erase instruction sets all memory within a specified block (64K-bytes) to the erased state of all 1s (FFh) / SECTOR ERASE in Numonyx	Yes	Yes	Yes	Yes
01	Write status register(Only Non-volatile Status Register bits SRP0, SEC, TB, BP2, BP1, BP0 (bits 7, 5, 4, 3, 2 of Status Register-1) and QE, SRP1(bits 9 and 8 of Status Register-2) can be written to.)	Yes	Yes	Yes	Yes
03	Read data(The Read Data instruction allows one more data bytes to be sequentially read from the memory.)	Yes	Yes	Yes	Yes
05	Read SR-1(The Read Status Register instructions allow the 8-bit Status Registers to be read.)	Yes	Yes	Yes	Yes

Table 3-3: **Supported Commands in AXI Quad SPI Core**

Command Opcode	Command Description	Winbond	Spansion	Numonyx /Micron	Macronix
7A	Erase resume,(The Erase Resume instruction "7Ah" must be written to resume the Sector or Block Erase operation after an Erase Suspend.)	Yes	Yes	Yes	No
75	Erase suspend,(The Erase Suspend instruction "75h", allows the system to interrupt a Sector or Block Erase operation and then read from or program data to, any other sectors or blocks.)	Yes	Yes	Yes	No
C7	Chip erase, (The Chip Erase instruction sets all memory within the device to the erased state of all 1s (FFh) (C4 in case of numonyx)	Yes	Yes	Yes	Yes
4B	Read unique ID.(The Read Unique ID Number instruction accesses a factory-set read-only 64-bit number that is unique to each flash device.) OTP Read for spansion and Numonyx	Yes	Yes	Yes	No
13	Read (4 byte address)-(The instruction code for the ReadData Bytes using 4 Bytes Address (READ4BYTE) instruction is followed by a 4-byte address (A31-A0), each bit being latched-in during the rising edge of Serial Clock (C).)	No	Yes	Yes	Yes
0C	Read fast(4 byte address)	No	Yes	Yes	Yes
5A	5A--->Read serial flash discoverable parameters(allows reading the SerialFlash Discovery Parameter area (SFDP), composed of 2048 read-only bytes containing operating characteristics and vendor specific information. The SFDP area is factory programmed./READ SERIAL FLASH DISCOVERY PARAMETER in Numonyx	No	Yes	Yes	Yes
3C	Fast Read Dual out(4-byte address)	No	Yes	Yes	Yes
BC	Dual I/O read(4 byte address)	No	Yes	Yes	Yes
6C	Read quad out(4 byte address)	No	Yes	Yes	Yes
EC	Quad I/O read(4 byte address),High freq,4 lines, addr on 2 lines	No	Yes	Yes	Yes
12	Page program (4 byte address),Page program with 4byte addressing.	No	Yes	Yes	Yes

Table 3-3: **Supported Commands in AXI Quad SPI Core**

Command Opcode	Command Description	Winbond	Spansion	Numonyx /Micron	Macronix
42	Program OTP .The Program OTP instruction (POTP) is used to program at most 64 bytes to the OTP memory area (by changing bits from 1 to 0, only).These 64 bytes can be permanently locked by a particular Program OTP (POTP) sequence.	No	Yes	Yes	Yes
E8	Read lock register.(There are two software protected modes, SPM1 and SPM2, that can be combined to protectthe memory array as required. The SPM2 can be locked by hardware with the help of the input pin. SPMThe first software protected mode (SPM1) is managed by specific Lock Registers assigned to each 64 Kbyte sector./READ VOLATILE LOCK BITS in Numonyx	No	Yes	Yes	Yes
85	Read volatile config register. (The Non Volatile Configuration Register (NVCR) bits affects the default memory configuration after power-on. It can be used to make the memory start in the configuration to fit the application requirements. Program Suspend in spansion	No	Yes	Yes	Yes
E9	Exit 4 byte address mode. (The Exit 4-byte address mode (EX4BYTEADDR) instruction disables 4-byte address mode.) Command is password unlock in spansion	No	Yes	Yes	Yes
52	Block Erase /32KB SUBSECTOR ERASE in Numonyx	Yes	No	Yes	Yes
A2	Dual input fast program. Highest freq, The dual input fast program (DIFP) instruction makes it possible to program up to 256 bytesusing two input pins at the same time (by changing bits from '1' to '0')	No	No	Yes	No
D2	Dual input extended fast program. The Dual Input Extended Fast Program (DIEFP) instruction is an enhanced version of the Dual Input Fast Program instruction, allowing to transmit address across two data lines.	No	No	Yes	No

Table 3-3: **Supported Commands in AXI Quad SPI Core**

Command Opcode	Command Description	Winbond	Spansion	Numonyx /Micron	Macronix
E5	Write to lock register/Write Volatile Lock Bits in Numonyx	No	No	Yes	No
70	Read flag status register	No	No	Yes	No
50	Clear flag status register	No	No	Yes	No
B5	Read NV config register	No	No	Yes	No
B1	Write NV config register /Enter Secured OTP in Macronix	No	No	Yes	Yes
81	Write volatile config register	No	No	Yes	No
65	Read volatile enhanced config register	No	No	Yes	No
61	Write volatile enhanced config register. The Volatile Enhanced Configuration Register (VECR) affects the memory configuration after every execution of Write Volatile Enhanced Configuration Register (WRVECR) instruction. This instruction overwrites the memory configuration set during the POR sequence by the Non Volatile Configuration Register (NVCR). Its purpose is enabling of QIO-SPI protocol and DIO-SPI protocol.	No	No	Yes	No
B7	Enter 4 byte address mode	No	No	Yes	Yes
C5	Write extended addr reg.	No	No	Yes	Yes
C8	Read extended addr register	No	No	Yes	Yes
66	Reset enable	No	No	Yes	Yes
99	Reset memory. The Reset Enable and Reset Memory operation is used as a system software reset that puts the device in the power-on reset condition.	No	No	Yes	Yes
20	Subsector erase. The Subsector Erase (SSE) instruction sets to '1' (FFh) all bits inside the chosen subsector. Before it can be accepted, a Write Enable (WREN) instruction must have been executed previously/SECTOR ERASE in Macronix	Yes	No	Yes	Yes
35	Read SR-2/ ENTER Quad IO Mode in Numonyx	Yes	Yes	Yes	Yes

Table 3-3: **Supported Commands in AXI Quad SPI Core**

Command Opcode	Command Description	Winbond	Spansion	Numonyx /Micron	Macronix
B9	Power down. Although the standby current during Normal operation is relatively low, standby current can be further reduced with the Power-down instruction. The lower power consumption makes the Power-down instruction especially useful for battery powered applications. Bank Register Access in spansion enter deep power down in Numonyx in Macronix	Yes	Yes	Yes	Yes
A3	HPM. The High Performance Mode (HPM) instruction must be executed prior to Dual or Quad I/O instructions when operating at high frequencies (see FR and FR1 in AC Electrical Characteristics). This instruction allows pre-charging of internal charge pumps so the voltages required for accessing the Flash memory array are readily available.	Yes	No	No	No
FF	Continuous read mode reset. For Fast Read Dual/Quad I/O operations, "Continuous Read Mode" Bits (M7-0) are implemented to further reduce instruction overhead. Mode Bit Reset in spansion	No	Yes	No	No
AB	Release Power down/HPM. The Release from Power-down or High performance Mode/Device ID instruction is a multi-purpose instruction. It can be used to release the device from the power-down state or High Performance Mode, or obtain the devices electronic identification (ID) number. Command is Read Electronic Signature in spansion release from deep power down in Numonyx in Macronix Read Electronic Id	Yes	Yes	Yes	Yes
90	Device ID. The Read Manufacturer/ Device ID instruction is an alternative to the Release from Power-down/ Device ID instruction that provides both the JEDEC assigned manufacturer ID and the specific device ID.	Yes	Yes	No	Yes

Table 3-3: **Supported Commands in AXI Quad SPI Core**

Command Opcode	Command Description	Winbond	Spansion	Numonyx /Micron	Macronix
30	Clear SR-1/PGM/ERS Resume in Macronix	No	Yes	No	Yes
15	Autoboot reg write/Read config register in Macronix	No	No	No	Yes
16	Bank register read. Bank address register may need to be changed during a suspend to reach a sector for read or program. / Read Fast Boot Register in Macronix	No	Yes	No	Yes
17	Bank register write / Write Fast Boot Register in Macronix	No	Yes	No	Yes
38	EXTENDED QUAD INPUT FAST PROGRAM(enhanced version of the Quad Input Fast Program instruction, allowing parallel input on the 4 input pins, including the address being sent to the device.)	No	Yes	Yes	Yes
34	Quad page program (4 byte address)	No	Yes	Yes	No
60	Bulk erase/CHIP ERASE in Macronix	Yes	Yes	No	Yes
E0	DYBRD. It may be necessary to remove and restore dynamic protection during erase suspend to allow programming during erase suspend / 4-BYTE READ VOLATILE LOCK in Numonyx/ Write DPB Register in Macronix BITS	No	Yes	Yes	Yes
E1	DYBWR / 4-BYTE WRITE VOLATILE LOCK BITS in Numonyx / Write DPB Register in Macronix	No	Yes	Yes	Yes
E2	PPBRD. Allowed for checking persistent protection before attempting a program command during erase suspend. / READ NonVolatile Lock BITS in Numonyx / Read SPB Status in Macronix	No	Yes	Yes	Yes

Table 3-3: **Supported Commands in AXI Quad SPI Core**

Command Opcode	Command Description	Winbond	Spansion	Numonyx /Micron	Macronix
E3	PPB program octal word read quad I/O (for windbond). The Octal Word Read Quad I/O (E3h) instruction is similar to the Fast Read Quad I/O (EBh) instruction except that the lower four address bits (A0, A1, A2, A3) must equal 0. As a result, the four dummy clocks are not required, which further reduces the instruction overhead allowing even faster random access for code execution (XIP). / Write NonVolatile Lock Bits in Numonyx	Yes	Yes	Yes	Yes
E4	PPB erase / Erase NonVolatile Lock Bits in Numonyx	No	Yes	Yes	Yes
2B	ASP Read /READ Security Register in Macronix	No	Yes	No	Yes
2F	ASP program/WRITE Security Register in Macronix	No	Yes	No	Yes
A7	PPB lock bit read / READ GLOBAL FREEZE BIT in Numonyx	No	Yes	Yes	No
A6	PPB lock bit write /WRITE GLOBAL FREEZE BIT in Numonyx	No	Yes	Yes	No
E7	Quad IO word read /Password Read in Spansion	No	Yes	No	No
F0	Reset	No	Yes	No	No
8A	Program resume	No	Yes	No	No
07	Read SR-2 SP	No	Yes	No	No
DC	Erase 256KB (4 byte addr)/4-BYTE SECTOR ERASE in Numonyx	No	Yes	Yes	Yes
27	Read 4 byte password	No	No	Yes	Yes
28	Write password	No	No	Yes	Yes
29	Unlock password	No	No	Yes	Yes
3E	4xIO page program/ 4-BYTE QUAD INPUT EXTENDED FAST PROGRAM in Numonyx	No	No	Yes	Yes
21	Sector erase 4 byte address/ 4-BYTE 4KB SUBSECTOR ERASE in Numonyx	No	No	Yes	Yes
5C	Block erase 32kb/4-BYTE 32KB SUBSECTOR ERASE in Numonyx	No	No	Yes	Yes
C0	Set burst read length	No	No	No	Yes

Table 3-3: **Supported Commands in AXI Quad SPI Core**

Command Opcode	Command Description	Winbond	Spansion	Numonyx /Micron	Macronix
18	Erase fast boot register	No	No	No	Yes
F5	Reset qpi mode / Reset Quad IO Mode in Numonyx	No	No	Yes	Yes
2D	Read lock register/Read Sector Protection in Numonyx	No	No	Yes	Yes
2C	Write lock register/ PROGRAM SECTOR PROTECTION in Numonyx	No	No	Yes	Yes
AF	QPI ID/MULTIPLE I/O READ ID in Numonyx	No	No	Yes	Yes
B0	PGM/ERS Suspend (suspends Program/Erase)	No	No	No	Yes
C4	DIE ERASE	No	No	Yes	No
68	Write Protection Selection WPSEL	No	No	No	Yes
C1	Exit secured OTP	No	No	No	Yes
7E	Gang Block Lock	No	No	No	Yes
98	Gang Block UnLock	No	No	No	Yes
00	NOP	No	No	No	Yes

Following are unsupported Commands for Dual/Quad SPI Mode and Winbond or Micron or Spansion

Winbond Memory (Ex: W25Q64VSFIG)

The core supports 24-bit addressing mode only.

Unsupported commands/features for this memory are:

- Fast read dual I/O continuous read mode.
- Fast read quad I/O continuous read mode.
- The command `ABh` is supported only for releasing the flash from power down or high performance mode. This command should not be used for reading the device ID. The command FFh (Mode bit Reset) is not supported, as this is recommended when using Dual/Quad mode feature.

Exceptional behavior for certain commands:

- Release power down/high performance mode (`ABh`): This command supports release power down/high performance mode or reading the device ID with a different combination of dummy bytes. The core only supports release power down/high

performance-mode where only one command byte is required to be placed in the DTR. The other mode of the command is not supported, as there is another command (`90h`) to read the device ID.

Micron Memory (Ex: N25Q256)

The core supports 24 and 32-bit addressing modes.

Unsupported commands/features for this memory are:

- XIP mode or continuous read mode in all the memories is not supported in Legacy or enhanced mode.
- All commands in dual or quad mode are supported in extended SPI mode. Dual In Out (DIO) and Quad In Out (QIO) modes are not supported. Following are the commands that are not supported by core:
  - 96h (Read General Purpose Read Register)
  - 9Bh (Interface Activation)
  - E7h (Quad IO Word Read)
  - E1h (4-Byte Write volatile lock bits)

    Note: No Support In Dual mode
- In quad mode, the design supports the Micron memory parts with HOLD functionality only. The parts with RESET functionality are not supported in the design.

Note: See *Migrating from Micron's N25Q to Micron's MT25 technical note* [Ref 21], for compatibility of Micron MT25Q devices with validated N25Q devices.

Spansion Memory (Ex: S70FL01GS)

The core supports 24-bit and 32-bit addressing modes. Unsupported commands/features for this memory are:

- All the double data rate (DDR) commands are not supported in any mode of the core.
- Spansion Flash supports 32-bit addressing mode with 24-bit address commands provided the ExtAddr bit is set to 1. This feature is not supported by the core.
- Autoboot commands of Spansion Flash are not supported by the core in any mode.

Macronix Memory (Ex: MX66U1G45G)

The core supports 24 and 32-bit addressing modes. Unsupported commands/features for this memory are:

- All commands in dual or quad mode are supported in extended SPI mode.
- Dual In Out (DIO) and Quad In Out (QIO) modes are not supported.

Clocking (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 (clock idle state is Low) or active-Low clock (clock idle state is High). Determination of whether the edge of interest is the rising or falling edge depends on the idle state of the clock (that is, the CPOL setting).

The clock phase (CPHA) bit can be set to select one of two different transfer formats. If CPHA is 0, data is valid on the first SCK edge (rising or falling) after $\overline{SS}$(N) has been asserted. If CPHA is 1, data is valid on the second SCK edge (rising or falling) after $\overline{SS}$(N) has asserted.

IMPORTANT: *For successful transfers, the clock phase and polarity must be identical to those of the master SPI device and the selected slave device.*

The first SCK cycle begins with a transition of the SCK signal from its idle state, which denotes the start of the data transfer. Because the clock transition from idle denotes the start of a transfer, the M68HC11 specification notes that the $\overline{SS}$(N) line can 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 on a write to the transmit register. Consequently, it is your responsibility to ensure that the data is properly loaded into 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.

Resets

The core supports active-Low reset input from the AXI interface in all configuration modes. External reset must be synchronous to the AXI clock. The core also supports the internal reset register which generates a local reset signal to the core causing all registers to obtain default settings on each bit.

Protocol Description

For the SPI protocol description, see the *Motorola M68HC11-Rev. 4.0 Reference Manual*.

For the AXI protocol description, see the *AMBA® AXI4-Stream Protocol Specification* [Ref 4].

AXI Quad SPI Core Behavior in Legacy and Enhanced Mode

When the Micron SPI memory is targeted as a slave, be aware of the different types of instruction sets supported. The Micron memory part N25Q_256_3 data sheet supports only extended SPI instructions. Read the data sheet to verify the expected behavior of the core in different modes (set when **Mode** is **Dual** or **Quad** and **Slave Device** is set to **Micron**).

Mode = Dual

The core is configured to support dual and standard mode SPI commands. The configuration modes and their behavior are described in these sections:

Slave Device = Mixed

In this mode, it is assumed that there is more than one SPI slave device. As the core supports only Winbond and Micron memories, the slave must be one of these two devices.

This is considered to be a mixed memories mode where Winbond memories are taken as the base for defining the behavior of the core. The instructions which are common to Winbond, Micron and Spansion memories (from the extended SPI protocol instructions set) are supported. Table 3-4 shows the core behavior.

Table 3-4: **SPI Command Core Behavior for Dual Mode and Mixed Mode Memories**

Command Type	Winbond	Micron/ Spansion	Command Error	Core Behavior
Standard SPI	Supported	Supported	No	Standard format
Standard SPI	Not supported	Supported	Yes	No SPI transaction
Standard SPI	Supported	Not supported	No	Standard format
Standard SPI	Not supported	Not supported	Yes	No SPI transaction
Dual mode	Supported	Supported	No	Dual mode instruction format
Dual mode	Not supported	Supported	Yes	No SPI transaction
Dual mode	Supported	Not supported	No	Dual mode instruction format
Dual mode	Not supported	Not supported	Yes	No SPI transaction
Quad mode	Supported	Supported	Yes	No SPI transaction
Quad mode	Not supported	Supported	Yes	No SPI transaction
Quad mode	Supported	Not supported	Yes	No SPI transaction

Table 3-4: **SPI Command Core Behavior for Dual Mode and Mixed Mode Memories *(Cont'd)***

Command Type	Winbond	Micron/ Spansion	Command Error	Core Behavior
Quad mode	Not supported	Not supported	Yes	No SPI transaction

Notes:

1. **Slave Device** = **Mixed** is mixed memory mode. For **Mode** = **Dual**, the dual as well as standard SPI commands are supported. For each command, the Winbond memory base behavior is used as the default operating mode. For the commands supported only by Micron or Spansion. the command error flag is set and the command is not executed. In this mode, the command set in common with Winbond and Micron memories is supported.

Slave Device = Winbond

This is a dedicated mode and supports only Winbond memories as SPI slave devices. Most of the Winbond memory SPI commands are supported. Table 3-5 shows the core behavior.

Table 3-5: **SPI Command Core Behavior for Dual Mode and Winbond Memory**

Command Type	Winbond Memory	Command Error	Core Behavior
Standard SPI	Supported	No	Standard format
Standard SPI	Not supported	Yes	No SPI transaction
Dual mode	Supported	No	Dual mode instruction format as given in the data sheet
Dual mode	Not supported	Yes	No SPI transaction
Quad mode	Supported	Yes	No SPI transaction
Quad mode	Not supported	Yes	No SPI transaction

Notes:

1. The core is designed to support Winbond W25Q64BV memory. See the device data sheet for the command, address, dummy bytes and data bytes required for each command and for the command, address and data bits operating details.

Slave Device = Micron

This is a dedicated mode and supports only Micron memories as SPI slave devices. The core supports most of the dual and standard Micron memory SPI commands. Table 3-6 shows the core behavior.

Table 3-6: **SPI Command Core Behavior for Dual Mode and Micron Memory**

Command Type	Micron Memory	Command Error	Core Behavior
Standard SPI	Supported	No	Standard format
Standard SPI	Not supported	Yes	No SPI transaction
Dual mode	Supported	No	Dual mode instruction format as given in the data sheet
Dual mode	Not supported	Yes	No SPI transaction
Quad mode	Supported	Yes	No SPI transaction
Quad mode	Not supported	Yes	No SPI transaction

Notes:

1. The core is designed to support the Micron N25Q256-3V memory in this mode for all dual and standard mode commands. See the device data sheet for the command, address, dummy bytes and data bytes required for each command and for the command, address and data bits operating details.

Slave Device = Spansion

This is a dedicated mode and supports only Spansion memories as SPI slave devices. The core supports most of the dual and standard Spansion memory SPI commands. Table 3-7 shows the core behavior.

Table 3-7: **SPI Command Core Behavior for Dual Mode and Spansion Memory**

Command Type	Spansion Memory	Command Error	Core Behavior
Standard SPI	Supported	No	Standard format
Standard SPI	Not Supported	Yes	No SPI transaction
Dual mode	Supported	No	Dual mode instruction format as given in the data sheet.
Dual mode	Not Supported	Yes	No SPI transaction
Quad mode	Supported	Yes	No SPI transaction
Quad mode	Not Supported	Yes	No SPI transaction

Notes:

1. The core is designed to support the Spansion S70FL01GS memory in this mode for all dual and standard mode commands. See the device data sheet for the command, address, dummy bytes and data bytes required for each command and for the command, address and data bits operating details.

Mode = Quad

The core is configured to support quad, dual and standard mode SPI commands based on the type of memory used as a SPI slave. The configuration modes and their behavior are described in these sections:

Slave Device = Mixed

In this mode, it is assumed that there is more than one SPI slave device. As the core supports only the Winbond, Micron, and Spansion memories, the slave must be one of these three devices.

This is a mixed memories mode where Winbond memories are taken as the base for defining the behavior of the core. Most of the instructions which are common to Winbond, Micron, and Spansion memories (extended SPI commands) are supported. Table 3-8 shows the core behavior.

Table 3-8: **SPI Command Core Behavior for Quad Mode and Mixed Mode Memories**

Command Type	Winbond	Micron/ Spansion	Command Error	Core Behavior
Standard SPI	Supported	Supported	No	Standard format
Standard SPI	Not supported	Supported	Yes	No SPI transaction
Standard SPI	Supported	Not supported	No	Standard Format
Standard SPI	Not supported	Not supported	Yes	No SPI transaction
Dual mode	Supported	Supported	No	Dual mode instruction format
Dual mode	Not supported	Supported	Yes	No SPI transaction
Dual mode	Supported	Not supported	No	Dual mode instruction format
Dual mode	Not supported	Not supported	Yes	No SPI transaction
Quad mode	Supported	Supported	Yes	Quad mode instruction format
Quad mode	Not supported	Supported	Yes	No SPI transaction
Quad mode	Supported	Not supported	Yes	Quad mode instruction format
Quad mode	Not supported	Not supported	Yes	No SPI transaction

Notes:

1. **Slave Device** = **Mixed** is mixed memory mode. In **Mode** = **Quad**, the quad, dual and standard SPI commands are supported. For each command, the Winbond memory base behavior is taken as the default operating mode. For commands supported only by Micron or Spansion, the command error flag is set and the command is not executed. In this mode, the command set in common with Winbond and Micron memories is supported.

Slave Device = Winbond

This is a dedicated mode and supports only Winbond memories as SPI slave devices. Most of the Winbond W25Q64BV memories SPI commands are supported. Table 3-9 shows the generalized core behavior.

Table 3-9: **SPI Command Core Behavior for Dual Mode and Winbond Memory**

Command Type	Winbond	Command Error	Core Behavior
Standard SPI	Supported	No	Standard format
Standard SPI	Not supported	Yes	No SPI transaction
Dual mode	Supported	No	Dual mode instruction format as given in the data sheet
Dual mode	Not supported	Yes	No SPI transaction
Quad mode	Supported	No	Quad mode instruction format as given in the data sheet
Quad mode	Not supported	Yes	No SPI transaction

Notes:

1. The core is designed to support the Winbond W25Q64BV memory. See the device data sheet for the command, address, dummy bytes and data bytes required for each command and for the command, address and data bits operating details.

Slave Device = Micron

This is a dedicated mode and supports only Micron memories as SPI slave devices. The core supports most of the Micron N25Q256-3V memory quad, dual, and standard SPI commands. Table 3-10 shows the core behavior.

Table 3-10: **SPI Command Core Behavior for Dual Mode and Micron Memory**

Command Type	Micron	Command Error	Core Behavior
Standard SPI	Supported	No	Standard SPI format
Standard SPI	Not supported	Yes	No SPI transaction
Dual mode	Supported	No	Dual mode instruction format as given in the data sheet
Dual mode	Not supported	Yes	No SPI transaction
Quad mode	Supported	No	Quad mode instruction format as given in the data sheet
Quad mode	Not supported	Yes	No SPI transaction

Notes:

1. The core is designed to support the Micron N25Q256-3V memory device in this mode for all quad, dual and standard commands. See the device data sheet for the command, address, dummy bytes and data bytes requirements for each command and for the command, address and data bits operating details.

Slave Device = Spansion

This is a dedicated mode and supports only Spansion memories as SPI slave devices. The core supports most of the Spansion S70FL01GS memory quad, dual, and standard SPI commands.

Table 3-11: **SPI Command Core Behavior for Dual Mode and Spansion Memory**

Command Type	Spansion Memory	Command Error	Core behavior
Standard SPI	Supported	No	Standard format
Standard SPI	Not Supported	Yes	No SPI transaction
Dual mode	Supported	No	Dual mode instruction format as given in data sheet
Dual mode	Not Supported	Yes	No SPI transaction
Quad mode	Supported	Yes	Quad mode instruction format as given in the data sheet
Quad mode	Not Supported	Yes	No SPI transaction

Notes:

1. The core is designed to support the Spansion S70FL01GS memory device in this mode for all quad, dual and standard commands. See the device data sheet for the command, address, dummy bytes and data bytes requirements for each command and for the command, address and data bits operating details.

Core Behavior in XIP Mode

This mode is set when **Enable XIP Mode** is selected. This mode is especially useful when using the flash in ROM operations where the executable file is stored and accessed by the processor or any master.

In XIP mode, the core supports 24-bit addressing mode as the common mode across Winbond, Micron, and Spansion devices, while for Micron and Spansion memories the core also supports 32-bit address mode. In 32-bit addressing mode of IP, flash must be configured in 32 bit mode. For the supported command set corresponding flash data sheet can be referred. The 24-bit addressing is applicable to both Winbond and Micron memories with the core in Standard, Dual, and Quad modes.

In this mode, the AXI4-Lite interface is first used to configure the core with the proper CPOL and CPHA modes. The valid modes are 00 and 11. Any other combination causes the core to not accept the AXI4 transaction. The AXI4-Lite interface is only used to set the configuration register and read the status register. The AXI4 interface is used to read the data from memory with the address provided by the AXI4 interface. The read channel of the AXI4 interface should provide the starting address which is converted by the core to the SPI transactions at the SPI interface. The operating mode of the core is set using **Mode** while the targeted memory is selected based on the **Slave Device** setting. In this case, a single memory is targeted and multiple memories are not supported by the core. The target memory can be any from Winbond, Micron, or Spansion or any other memory which supports the default three read commands. The maximum burst length for read transaction

on AXI4 interface in this mode is 64 as the FIFO depth is 64. The core behavior is not guaranteed otherwise.

The default commands are fast read (`0x0Bh`), fast read dual I/O (`0xBBh`) and fast read quad I/O (`0xEBh`). Based on the setting of **Mode**, the same command is used throughout the operation. The command cannot be changed as it is generated internally by the core.

The core has internal reference logic, which pads the dummy bytes required for the particular read command.

The XIP mode of core operation is based on the Winbond, Micron, and Spansion memory data sheet specification for read command behavior.

Parameters used for this configuration:

- Enable Performance Mode
- Enable XIP Mode
- Mode
- Slave Device
- Enable STARTUPE*n* Primitive

  Note: The STARTUPE2 primitive is applicable for 7 series devices. The STARTUPE3 primitive is applicable for UltraScale devices.

The core uses the `ext_spi_clk` as the reference clock for the SPI logic. This clock is separate from the AXI4 interface clock and it should be double the desired SPI frequency at the SPI interface. The core uses a **Frequency Ratio** setting of 2, which is fixed for this mode, for generating the SPI clock at the SPI interface with reference to this clock. There is no soft reset register or interrupt register associated with XIP mode. The only way to reset the core is to reset the interconnect.

In this mode, the core supports WRAP as well as INCR type AXI4 transactions only. The FIXED transaction results in an error and the core does not accept the transaction. Instead, the transaction error flag is set in the XIP status register. In this case, the targeted memory is Winbond or Spansion and if the core is configured in quad mode, ensure that the QE bit of the status register of the Winbond memory or the QE bit of the configuration register of the Spansion memory is set prior to the booting the core from SPI flash memory. This should be performed using external programming tools.

Standard SPI Mode Transactions

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

SPI Master Device with/without FIFOs and Slave Select Vector Asserted Manually Using SPICR Bit 7

This flow permits the transfer of N number of byte/half-word/word elements with a single toggling of the slave select vector. This is the default mode of operation. Use these steps to successfully complete the SPI transaction:

1. Start from a known state, including SPI bus arbitration.
2. Configure DGIER and IPIER registers as required.
3. Configure the target slave SPI device as required. This includes configuration of the DTR and control register of the slave SPI core as well as enabling it.
4. Write initial data to the master SPI DTR register/FIFO. This assumes that the SPI master is disabled.
   a. In Legacy mode, the AXI4-Lite transactions are written to the DTR one at a time.
   b. In Enhanced mode, the AXI4 interface must generate a FIXED burst transaction only. An INCR transaction with length 0 is acceptable but if the INCR burst is targeted at the FIFO locations (DTR or DRR), the core behavior is not guaranteed. The INCR transactions are treated as FIXED transactions. To avoid FIFO overflow or underflow errors, The transmit or receive occupancy register should be read before initiating a burst of any length. The maximum burst length for read command to DRR supported in non-XIP mode is 16 or the occupancy value of the Rx FIFO, whichever is lower. The core behavior is not guaranteed otherwise. In case of writing to DTR register, the core behavior guaranteed only if a write length is less than or equal to FIFO_Depth - DTR occupancy value.
5. Ensure the SPISSR register contains all ones.
6. Write configuration data to the master SPI device SPICR as required, including setting bit 7 for manual assertion of the $\overline{SS}$ vector and setting both enable and master transfer inhibit bits. This initializes SCK and IO0 but inhibits transfer.
7. Write to the SPISSR register to manually assert the $\overline{SS}$ vector.
8. Write the preceding configuration data to the master SPI device SPICR register, but clear the inhibit bit which starts the transfer.
9. Wait for an interrupt (typically IPISR bit 4) or poll status for completion. The wait time depends on the SPI clock ratio.
10. Set the master transaction inhibit bit to service the interrupt request.
11. Write new data to the master register/FIFOs and slave devices.
12. Clear the master transaction inhibit bit to continue the N 8-bit element transfer.

Note: An overrun of the SPI DRR register/FIFO can occur if the SPI DRR register/FIFOs are not read properly. Also, SCK has stretched the idle levels between element transfers (or groups of element transfers if using FIFOs) and that IO0 can transition at the end of an element transfer (or group of transfers), but it is stable for divide-by-2 clocking modes. IO0 is valid at the falling edge

of SCK, and for all other modes, it is two ext_spi_clk cycles after the falling edge of SCK. Also there are no idle cycles between each new SPI transaction.

13. Repeat step 10 through step 12 until all data is transferred.
14. Write all ones to the SPISSR register or exit the manual slave select assert mode to deassert the $\overline{SS}$ vector while SCK and IO0 are in the idle state.
15. Disable devices as required.

SPI Master and Slave Devices without FIFOs Performing One 8-bit/16-bit/ 32-bit Transfer (Optional Mode)

Use these steps to successfully complete a SPI transaction:

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

SPI Master and Slave Devices where Registers/FIFOs are Filled Before the SPI Transfer Begins and Multiple Discrete 8-bit Transfers are Performed (Optional Mode)

The slave operation of the core supports a FIXED burst at the transmit or receive FIFO only. The length of this burst transaction should be based on the **FIFO Depth** parameter as well as the transmit or receive occupancy register. Take note of this to avoid any overrun or underrun errors of the DTR or DRR FIFO.

Use these steps to successfully complete a SPI transaction:

1. Start from proper state including SPI bus arbitration.

2. Configure the master DGIER and IPIER registers. Also configure the slave DGIER and IPIER registers as required.
3. Write configuration data to the master SPI device SPICR register as required.
4. Write configuration data to the slave SPI device SPICR register as required.
5. Write the active-Low, one-hot encoded slave select address to the master SPISSR register.
6. Write all data to the slave SPI DTR register/FIFO as required.
7. Write all data to the master SPI DTR register/FIFO.
8. Write the enable bit to the master SPICR register which starts the transfer.
9. Wait for interrupt (typically IPISR bit 4) or poll status for completion.
10. Read the IPISR register of both master and slave SPI devices as required.
11. Perform interrupt requests as required.
12. Read the SPISR register of both master and slave SPI devices as required.
13. Perform actions as required or dictated by SPISR register 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).

Use these steps to successfully complete a SPI transaction:

1. Start from the proper state including SPI bus arbitration.
2. Configure the master DGIER and IPIER registers. Also configure the slave DGIER and IPIER registers as required.
3. Write configuration data to the master SPI device SPICR register as required.
4. Write configuration data to the slave SPI device SPICR register as required.
5. Write the active-Low, one-hot encoded slave select address to the master SPISSR register.
6. Write initial data to the slave transmit FIFO as required.
7. Write initial data to the master transmit FIFO.
8. Write the enable bit to the master SPICR register which starts the transfer.
9. Continue writing data to both the master and slave FIFOs.
10. Wait for interrupt (typically IPISR bit 4) or poll status for completion.
11. Read the IPISR register of both master and slave SPI devices as required.

12. Perform interrupt requests as required.
13. Read the SPISR register of both master and slave SPI devices as required.
14. Perform actions as required or dictated by SPISR register data.

Dual/Quad SPI Mode Transactions

In this mode, the core must be configured in master mode only. Otherwise, an error interrupt is generated.

The sequence flow to operate the core in dual mode from the core point of view is shown. Check the inter-dependency of the commands before filling the SPI DTR register and enabling the SPI core to start the transaction.

1. Ensure that **Mode** is **Dual** and that **Slave Device** is set for the correct SPI slave memory.
2. Ensure that the instructions driven by the required SPI clock and set by **Frequency Ratio**, are listed.
3. Set the **FIFO Depth** parameter. This parameter can either be 16 or 256.
4. Write to the soft reset register to reset the core. This reset is active for 16 AXI cycles, during which each FIFO register is in the reset state.
5. Write to the SPICR register to put the core in master mode; set the CPOL, CPHA values, and make sure that the master transaction inhibit bit is set.
6. Write to the IPIER and IPISR registers to enable the required interrupts.
7. Write to the SPI DTR with the command, address, dummy, and data bytes to be transmitted in the same sequence provided in the data sheet of the target device.
8. Write to the SPI DTR with the number of data bytes intended to be read or written to memory along with command, address, and dummy bytes.
9. Write to the SPISSR register to assert the chip select signal from the core.
10. Write to the SPICR register to enable the master transaction inhibit bit, so that the core starts the SPI clock.
11. For a write, wait until the DTR empty interrupt is generated.
12. When the DTR empty interrupt is generated, data can still be written into the SPI DTR for further transactions if the slave select register remains unchanged (such as if the slave is selected).
13. If further data is written, the SPISSR register continues to be asserted; only the SPI clock is stopped. When the DTR FIFO is not empty, the SPI clock is enabled again and data is transmitted.
14. When reading, step 11, step 12, and step 13 also apply. Fill the DTR FIFO with any random data beats while reading the SPI slave memory.

15. Disabling the selected slave by setting all SPISSR register bits to 1 (or carrying out the master transactions inhibit bit set to 1 in the SPICR register), the SPI clock is stopped.
16. After disabling the selected slave by setting all SPISSR register bits to 1, If the SPI DTR FIFO is filled again, the core considers this a fresh transaction and compares the first entry in the DTR FIFO with the supported commands.
17. Between new transactions, make sure that the SPI DTR and DRR FIFOs are reset by writing into the SPICR register bits while the slave is deselected. This allows these two FIFOs to be reset and the first entry of the DTR FIFO to be compared with the supported commands.

Note: Core will support both the approaches of writing all at a time to the DTR (After Master inhibit set, Write all (Command,Address, Data) to the DTR then, Resetting the Master inhibit as mentioned in (step 7)) and writing one at a time to the DTR (After Master inhibit set, Write command to the SPI DTR , and then reset the inhibit, and similarly repeat the above procedure for Address,Data).

Dual/Quad Mode SPI Configuration

In dual or quad mode SPI configuration, based on the type of memory (either Winbond, Micron, or Spansion), the SPI DTR FIFO must be filled prior to the transmission of SPI data beats. Reset both the SPI DTR and DRR FIFOs before filling the new transaction. The SPI DTR FIFO should be filled in command, address, dummy bytes and data order format. The first entry in the SPI DTR FIFO is always compared with the internal command map table and, based on the supported command, the core behavior is determined. If the first entry in the SPI DTR FIFO does not match any commands in the supported command list, the core treats this as an error and SPI transactions do not proceed. An interrupt is also generated for this error.

Always check the supported and unsupported command list for the Winbond, Micron, and Spansion memories (unsupported commands for dual/quad SPI Mode and Winbond, Micron, or Spansion memory). If unsupported commands are executed, the core behavior is not guaranteed. An interrupt is set to indicate a command error, which means that the command does not match any of the supported commands in the core. The core in Dual/Quad mode always sends the command phase of the SPI transaction on a single line i.e., IO0_* ports. 2-2-2 and 4-4-4 (Command, address, and data) SPI modes are not supported in Dual/Quad SPI Configuration of the core.

Transfer Formats

In Standard SPI mode CPHA - CPOL, any one of the 00, 01, 10, or 11 modes are allowed.

In dual or quad SPI mode CPHA - CPOL, only 00 or 11 modes are allowed. If any other modes are set in the SPICR, an interrupt is set indicating the error and the core does not perform as expected.

CPHA Equals Zero Transfer Format

Figure 3-3 shows the timing diagram for a standard SPI mode data write-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 AXI and SPI signals have the same relation regarding S_AXI_ACIk and SCK respectively.

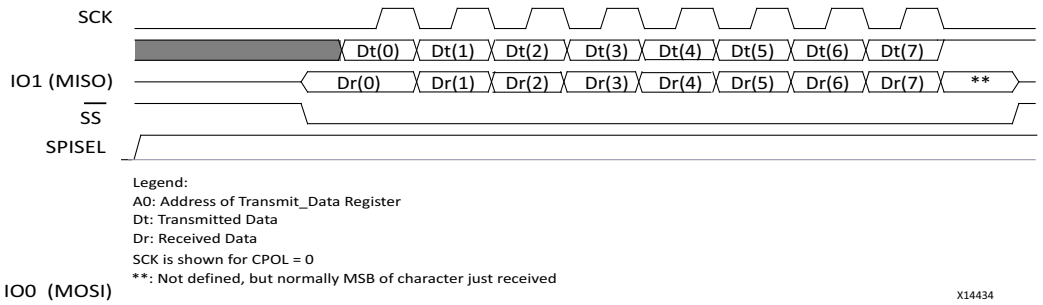

Figure 3-3: **Data Write-Read Cycle on SPI Bus with CPHA = 0 and SPICR(7) = 0 for 8-bit Data**

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. Because assertion of the $\overline{SS}$(N) line denotes the start of a transfer, it must be deasserted 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 8/16/32-bit oriented data transfer is possible using bits in the SPICR.

The IO0 pin is equivalent to IO0 (MOSI) in standard SPI mode. The IO1 pin is equivalent to IO1 (MISO) in standard SPI mode.

The IO0 and IO1 ports behave differently depending on whether the SPI device is configured as a master or a slave. When configured as a master, the IO0 port is a serial data output port, while the IO1 is a serial data input port. The opposite is true when the device is configured as a slave; the IO1 port is a slave serial data output port and the IO0 is a serial data input port. There can be only one master and one slave transmitting data at any given time. The bus architecture provides limited contention error detection (that is, multiple devices driving the shared IO1 and IO0 signals) and requires the software to provide arbitration to prevent possible contention errors.

All SCK, IO0, and IO1 pins of all devices are hard-wired 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 IO0 pins to the SCK and IO0 pins of the slaves. The uniquely-selected slave device drives data from its IO1 pin to the IO1 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 IO0 signals. In addition, the non-selected slaves (that is, $\overline{SS}$ pin High) drive their IO1 pin to 3-state so as not to interfere with SPI bus activities. When external slave SPI devices are implemented, the SCK, IO0 and IO1, as well as the needed $\overline{SS}(N)$ signals, are brought out to pins. All signals are true 3-state bus signals and erroneous external bus activity can corrupt internal transfers when both internal and external devices are present.

Ensure that the external pull-up or pull-down of external SPI 3-state signals are consistent with the sink/source capability of the FPGA I/O drivers. The I/O drivers can be configured for different drive strengths, as well as internal pull-ups. The 3-state signals for multiple external slaves can be implemented per the 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 3-4. The waveforms are shown for CPOL = 0, LSB First = 0, and the value of generic C_SCK_RATIO = 4. All AXI and SPI signals have the same relation regarding S_AXI_Clk and SCK respectively.

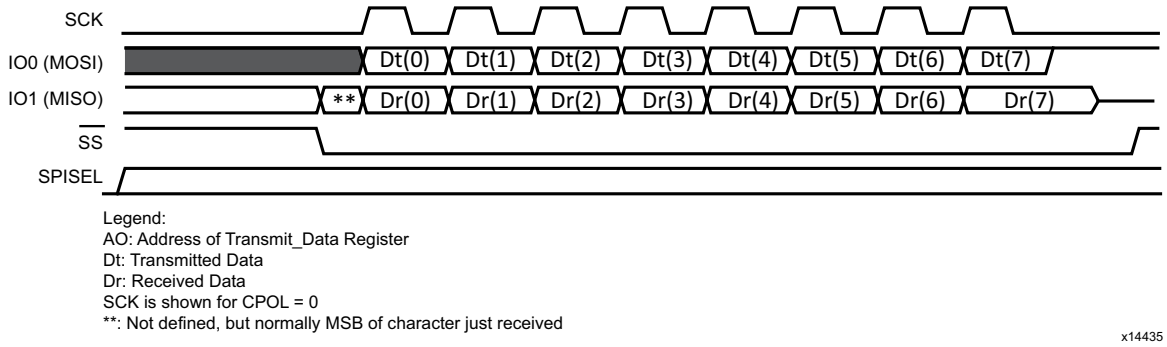

Figure 3-4: **Data Write-Read Cycle on SPI Bus with CPHA = 1 and SPICR(7) = 0 for 8-bit Data**

SPI Protocol Slave Select Assertion Modes

The standard mode SPI protocol is designed to have automatic slave select assertion and manual slave select assertion which are described in these sections. All the SPI transfer formats described in the Clocking (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 (when FIFO is disabled) by the SPI master device (SPICR bit 7 = 0).

This configuration mode is provided to permit transfer of data with automatic toggling of the slave select ($\overline{SS}$) signal until all 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 beat of the SPI transaction (configured through **Transaction Width** in the Vivado Integrated Design Environment (IDE)), the $\overline{SS}(N)$ output goes to 1. The data in the SPISSR register again appears on the $\overline{SS}(N)$ output at the beginning of a new transfer. The slave select signal does not need to be manually controlled. This mode is not supported in dual or quad SPI modes in cores using the AXI4-Lite and AXI4 interfaces.

SPI Protocol with Manual Slave Select Assertion

This section briefly describes the SPI protocol where the slave select, $\overline{SS}(N)$, is user asserted (that is, SPICR bit 7 = 1). This configuration mode is 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 used, the SPI master must be enabled first (SPICR bit 7 = 1) to put SCK in the idle state prior to asserting slave select.

The master transfer inhibit (SPICR bit 8) can be used to inhibit master transactions until the slave select is asserted manually and all FIFO data registers are initialized as required. This can be used before the first transaction and after any transaction that is allowed to complete.

When the preceding rules are followed, the timing is the same as presented for the automatic slave select assertion mode, with the exception that you control the assertion of the slave select signal and the number of elements transferred. While performing complete memory read or page read operations, the manual slave select mode should be used.

Beginning and Ending SPI Transfers

The details of the beginning and ending periods depend on the CPHA format selected and whether the SPI is configured as a master or a slave. These sections describe the beginning and ending period for SPI transfers.

Transfer Begin Period

The definition of the transfer beginning period for the AXI Quad SPI 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, which causes the SPI slave logic and bit counters to be reset. In this implementation, the software driver can deselect all slaves (that is, $\overline{SS}(N)$ is driven High) to abort a transaction. Although the hardware is capable of changing slaves during the middle of a single or burst transfer, the software should be designed to prevent this.

In slave configuration, the data is transmitted from the SPI DTR register on the first AXI 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 underrun interrupt is asserted.

Transfer End Period

The definition of the transfer end period for the AXI Quad SPI 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 might be additional tasks to be performed before the system can consider the transfer complete.

When configured without FIFOs, the Rx_Full bit (1) in the SPISR is set to denote the end of a transfer. When data is available in the SPI DRR register, bit 4 of the IPISR is asserted as well. The data in the SPI DRR is sampled on the same clock edge as the assertion of the SPI DRR register full interrupt.

When the SPI device is configured as a master without FIFOs, these steps occur:

- Rx_Empty bit (0), Tx_Full bit, and bit 3 in the SPISR are cleared.
- Tx_Empty bit (2), Rx_Full bit, and bit 1 in SPISR are set.
- DRR Full bit (4), Slave MODF bit, and bit 1 in the IPISR are set on the first rising AXI clock edge after the end of the last SCK cycle.

  Note: 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 3-3 and Figure 3-4). However, the internal master clock provides this SCK edge which prompts the setting and clearing of the bits noted.

In this design, a counter is implemented that permits the simultaneous setting of SPISR and IPISR bits for both master and slave SPI devices. External SPI slave devices can use an internal AXI 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 AXI Quad SPI core cannot be used in a system with external SPI slave devices that do not use the AXI clock.

When the AXI Quad SPI core is configured with FIFOs and a series of consecutive SPI 8-bit/16-bit/32-bit element transfers are performed (based on parameter settings), the 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 6 of IPISR.

When the SPI device is configured as a slave, the setting/clearing of the bits discussed previously for a master coincides with the setting or clearing of the master bits for both cases of CPHA = 0 and CPHA = 1. Keep in mind that for CPHA = 1 (that is, 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 AXI clock period counter was included in the SPI slave device.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1]
- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 5]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado® Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 5] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the `validate_bd_design` command in the Tcl Console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the Vivado IP catalog.
2. Double-click the selected IP, or select the **Customize IP** command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6].

Note: Figure in this chapter is an illustration of the Vivado IDE. This layout might vary from the current version.

Figure 4-1 shows the AXI Quad SPI Vivado Integrated Design Environment (IDE) screen.

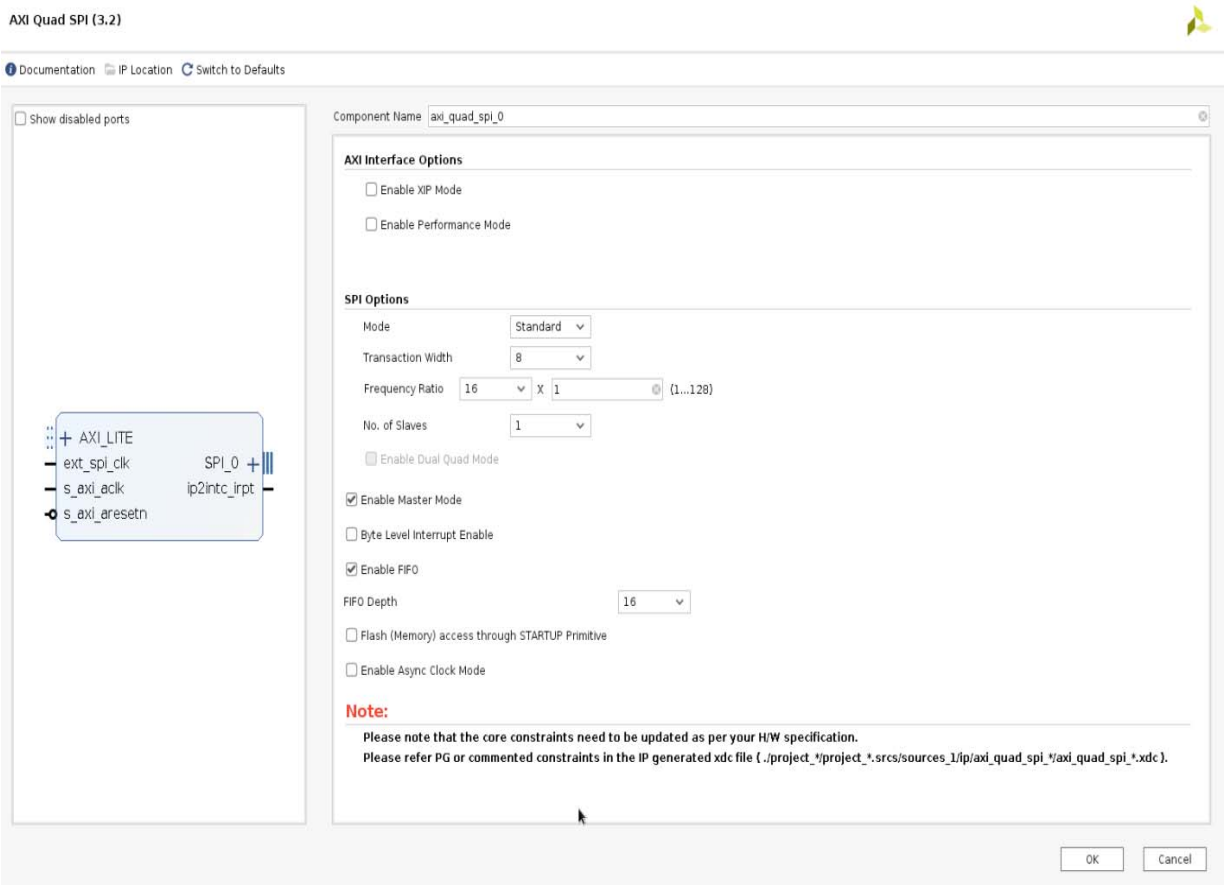

Figure 4-1: **AXI Quad SPI Vivado IDE Screen**

AXI Interface Options

- **Enable XIP Mode** – Enables the eXecute In Place (XIP) mode. This option also enables the AXI4 and AXI4-Lite interfaces. The choice of 24-bit or 32-bit addressing mode should be selected based on the downstream SPI device.

Figure 4-2: **Enable XIP Mode**

- **Enable Performance Mode** – Enables the AXI4 interface. Using the AXI4 interface also enables the burst capability at the transmit and receive FIFO addresses of the core. When this option is not selected, the AXI4-Lite interface is used.

SPI Options

- **Mode** – Selects standard, dual or quad mode. The correct mode is selected based on the targeted SPI slave device and application.
- **Transaction Width** – Selects 8, 16, or 32-bit transactions. Each SPI transaction incorporates the selected number of bits. In dual and quad SPI modes, the transaction width is restricted to 8 bits. In XIP mode the transaction width is restricted to 8 in std, dual and quad modes.
- **Frequency Ratio** – Selects a power of two divisor value from 2 to 2,048. The resulting SPI clock frequency is the quotient of the frequency associated with the `ext_spi_clk` signal divided by the selected value. In dual or quad SPI mode of the legacy and enhanced mode, the divisor is restricted to 2. In XIP mode the frequency ratio is restricted to 2 in std, dual and quad modes.
- **No. of Slaves** – Selects the number on non-XIP mode SPI slave devices from 1 to 32. In XIP mode, the number of SPI slave devices is restricted to 1.
- **Slave Device** – Selects the dual or quad SPI mode slave device category from:
  - **Mixed** – Selects a command subset in common with Winbond, Micron, and Spansion memory specifications.
  - **Winbond** – Selects a Winbond-specific memory command set.
  - **Micron** – Selects a Micron-specific memory command set.
  - **Spansion** – Selects a Spansion-specific memory command set.

  The slave device parameter is ignored in non XIP standard SPI mode. If SPI Mode is selected as Dual/Quad then the slave device option will be visible in Vivado IDE.

Remaining Options

- **Enable Master Mode** – Enables master SPI mode when checked, slave SPI mode when not checked. The enable master mode parameter is applicable only in standard SPI mode. In dual or quad SPI mode, only master SPI mode is supported.
- **Enable FIFO** – Includes the transmit or receive FIFO in the design when checked, omits the FIFO when not checked. The enable FIFO parameter is applicable only in standard SPI mode. In dual or quad SPI mode, the FIFO is always included in the design.
- **FIFO Depth** – Selects the depth of the included FIFO from 0, 16 or 256 beats. In standard SPI mode, this parameter is only available when the FIFO is included. In dual or quad SPI mode, the FIFO depth is limited to either 16 or 256 beats. The FIFO width is fixed at eight bits.

- **Enable STARTUPE2 Primitive** – Includes the STARTUP primitive in the design when checked, omits the primitive when not checked. The STARTUPE2 primitive is featured in 7 series FPGAs and the STARTUPE3 primitive is featured in UltraScale™ devices. It is useful in sharing the SPI clock with an external SPI slave device. This primitive is always disabled in the slave mode of the device.
- **Enable Async Clock Mode** – Enable this option only when the core is in standalone mode and the AXI interface and external SPI clocks differ in terms of phase/polarity and frequency. This option is disabled in the IP integrator. This parameter is auto propagated depending on the clocks connected to the core. If the AXI clock and `ext_spi_clk` are synchronous to each other, this parameter is set to 0, It is set to 1 if they are asynchronous to each other.
- **Enable Byte Level Interrupt** -Enabling this parameter, the "DRR NOT Empty Interrupt" will be triggered on the basis of Byte level instead of transaction level. By default, the value will be set to "0" which means the "Interrupt" will be triggered with respect to the transaction.

User Parameters

Table 4-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Enable XIP Mode	C_XIP_MODE	0
Enable Performance Mode	C_TYPE_OF_AXI4_INTERFACE	0
Mode	C_SPI_MODE	0
Transaction Width	C_NUM_TRANSFER_BITS	8
Frequency Ratio	C_SCK_RATIO	16
Multiplying Factor	Multiples16	1
Number of Slaves	C_NUM_SS_BITS	1
Enable Master Mode	Master_mode	1
Enable FIFO	FIFO_INCLUDED	1
FIFO Depth	C_FIFO_DEPTH	16
Enable Startup Primitive	C_USE_STARTUP	1
Enable Async Clock Mode	Async_Clk	0
ID_WIDTH	C_S_AXI4_ID_WIDTH	4
SPI Flash Address Bits	C_SPI_MEM_ADDR_BITS	24
Slave Device	C_SPI_MEMORY	1
AXI Read Channel Mode	C_XIP_PERF_MODE	1

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Byte Level Interrupt Enable	C_BYTE_LEVEL_INTERRUPT_EN	0

Notes:
1. Share the unused startup Ports C_SHARED_STARTUP 0.
2. Use Startup internal to IP C_USE_STARTUP_INT 0.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1].

Constraining the Core

IP specific constraints are delivered when the core is generated.

Required Constraints

The system-level constraints mentioned in Constraining the IP section have to be added in the top XDC files. The numbers mentioned are arbitrary values and update them according to your design.

STARTUP is enabled: The frequency of operation of AXI-QSPI is affected when a STARTUP primitive is enabled. This primitive has its own delay that is not accounted in the entire implementation/timing process.

The STARTUPE2 primitive is used to drive the SCK on the CCLK pin and the STARTUPE3 primitive is used to drive the SCK and data pins.

When using the STARTUP primitive, the clock-data relationship changes due to the extra delay added in the SCK path. This delay can vary from as low as 0.1 ns to as high as 7.5 ns based on the device and speed grade. This value can be found in the FPGA data sheet.

Because this delay is not timed by the tool, you must consider the maximum delay for all calculations. Assume this delay is called CCLK_DELAY. For K7-2 this value can range from 0.5 ns to 6.7ns. This puts the first restriction on the frequency of the clock. The frequency of operation cannot exceed F_{max1} = (1/CCLK_DELAY) MHz

Write Operation to SPI

In SCK ratio = 2 scenarios, a successful write operation should be completed within two clock cycles of `ext_spi_clk`.

This means:

- The rising-edge of SCK must be between these two active edges of `ext_spi_clk`.
- The position of the rising-edge of SCK should be such that it meets the T_{setup} and T_{hold} time of the SPI device.
- T_{setup} analysis should be done by taking into account the minimum delay on SCK (that is, the minimum delay of STARTUP), the maximum delay of the datapath, and the board routing delay.
- Similarly T_{hold} analysis should be done by taking into account the maximum delay on SCK (that is, the maximum delay of STARTUP), the minimum delay of data, and the board routing delay.

This requirement of the SPI device further brings down the operational frequency of the IP.

Based on this you obtain two numbers. F_{max} that will break T_{su} and F_{max} that will break T_h:

F_{max2} = fn {STARTUP delay min, max datapath, board routing delay, T_{su}}

F_{max3} = fn {STARTUP delay max, min datapath, board routing delay, T_h}

The more restrictive number of the above two has to be considered.

Read Operation

In SCK ratio = 2 scenarios, a successful read operation occurs when the SPI data, launched on the falling-edge of `SCK`, is captured on the second rising-edge of `ext_spi_clk`. This means that the data should be available and stable at the time the capture happens on the second rising edge.

The worst case scenario occurs when:

- `SCK` has max delay on the line (that is, Max delay of STARTUP)
- SPI drives the data with max T_{co}

F_{max4} = fn {STARTUP delay max, $T_{co\text{-}max}$, board routing}

Thus the actual F_{max} is the one that is most restrictive amongst F_{max2}, F_{max3} and F_{max4}.

Constraining the IP

To understand how the constraints are added, it is important to understand the logic structure around the clock and data in AXI Quad SPI flash memory.

The following figure gives an idea of the logic structure:

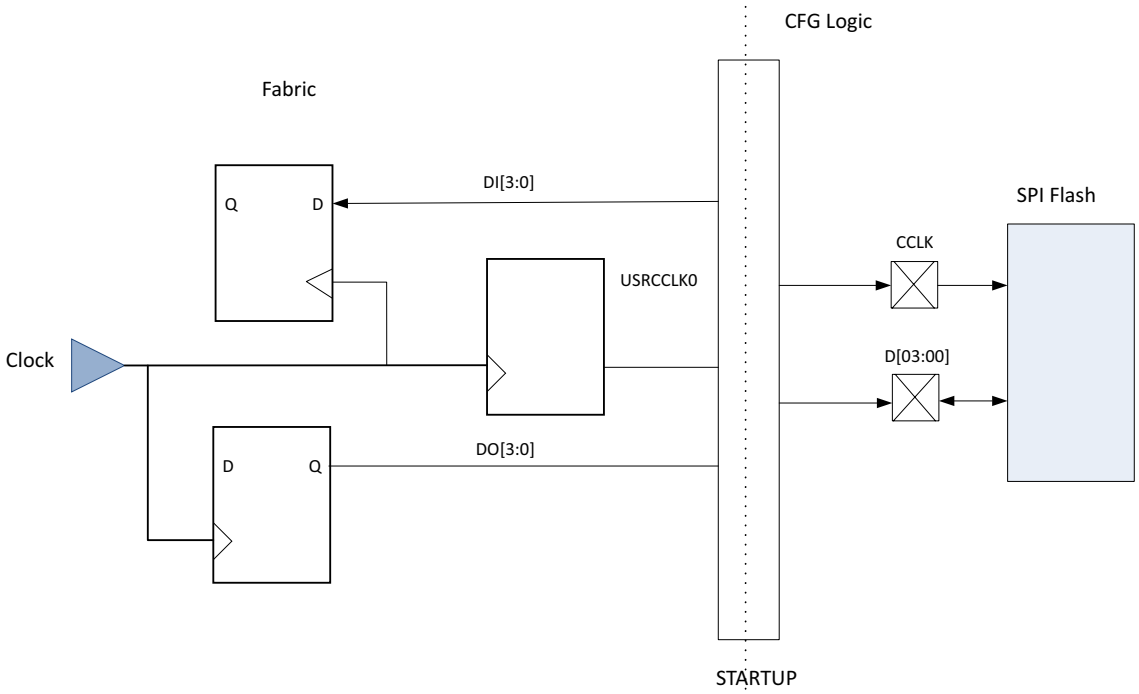

Figure 4-3: **Logic Structure**

Note: Refer to *UltraScale FPGA Post-Configuration Access of Parallel NOR Flash Memory using STARTUPE3 (XAPP1282)* [Ref 22].

Note: Refer to *UltraScale FPGA Post-Configuration Access of SPI Flash Memory using STARTUPE3 (XAPP1280)* [Ref 23].

The STARTUP primitive adds delay on the USRCCLKO to the CCLK pin. This delay is unaccounted for in the tool and is not considered in the timing calculation. For the tool, the timing path ends at USRCCLKO.

To obtain the required constraints you must account for the STARTUP primitive delay. As the timing path ends at USRCCLKO, you cannot create a clock on the CCLK pin.

To emulate the SCK clock, you must create a generated clock such that the STARTUP primitive delay is taken into account. This can be achieved by creating a generated clock on the USRCCLKO pin.

This takes into account the delay of the flip-flop that generates the SCK, the routing delay from that flip-flop to USRCCLKO pin, and the STARTUP primitive delay. Further, to reduce the delay on SCK you must ensure that the delay from the flip-flop to USRCCLKO is as low as possible. This can be constrained by using `set_max_delay`.

The datapaths can then be constrained using `set_output_delay` and `set_input_delay` along with `set_multicyle_path` constraints. All the following constraints are provided for SCK_ratio = 2:

1. STARTUPE3 (UltraScale) primitive included inside IP:

```
# You must provide all the delay numbers
# CCLK delay is 0.1, 6.7 ns min/max for ultra-scale devices; refer Data sheet
# Consider the max delay for worst case analysis
set cclk_delay 6.7
create_generated_clock -name clk_sck -source [get_pins -hierarchical
*axi_quad_spi_0/ext_spi_clk] -edges {3 5 7} -edge_shift [list $cclk_delay
$cclk_delay $cclk_delay] [get_pins -hierarchical *USRCCLKO]
set_multicycle_path -setup -from clk_sck -to [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]] 2
set_multicycle_path -hold -end -from clk_sck -to [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]] 1
set_multicycle_path -setup -start -from [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]] -to clk_sck 2
set_multicycle_path -hold -from [get_clocks -of_objects [get_pins -hierarchical */
ext_spi_clk]] -to clk_sck 1
set_max_delay -datapath_only -from [get_pins -hier {*STARTUP*_inst/DI[*]}] 1.000
set_max_delay -datapath_only -from [get_clocks -of_objects [get_pins -hierarchical
*/ext_spi_clk]] -to [get_pins -hier *STARTUP*_inst/USRCCLKO] 1.000
set_max_delay -datapath_only -from [get_clocks -of_objects [get_pins -hierarchical
*/ext_spi_clk]] -to [get_pins -hier *STARTUP*_inst/DO[*]] 1.000
set_max_delay -datapath_only -from [get_clocks -of_objects [get_pins -hierarchical
*/ext_spi_clk]] -to [get_pins -hier *STARTUP*_inst/DTS[*]] 1.000
```

2. STARTUPE3 (UltraScale+) primitive included inside IP: Vivado times the STARTUPE3 differently for UltraScale+ devices. For the purpose of timing the STARTUPE3, certain internal pins have been modeled which can be used to time the primitive. Below lines show the set of constraints that can be used with UltraScale+ devices.

```
set tdata_trace_delay_max 0.25
set tdata_trace_delay_min 0.25
set tclk_trace_delay_max 0.2
set tclk_trace_delay_min 0.2
create_generated_clock -name clk_sck -source [get_pins -hierarchical
*axi_quad_spi_0/ext_spi_clk] [get_pins -hierarchical */CCLK] -edges {3 5 7}
set_input_delay -clock clk_sck -max [expr $tco_max + $tdata_trace_delay_max +
$tclk_trace_delay_max] [get_pins -hierarchical *STARTUP*/DATA_IN[*]] -clock_fall;
set_input_delay -clock clk_sck -min [expr $tco_min + $tdata_trace_delay_min +
$tclk_trace_delay_min] [get_pins -hierarchical *STARTUP*/DATA_IN[*]] -clock_fall;
set_multicycle_path 2 -setup -from clk_sck -to [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]]
set_multicycle_path 1 -hold -end -from clk_sck -to [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]]
set_output_delay -clock clk_sck -max [expr $tsu + $tdata_trace_delay_max -
$tclk_trace_delay_min] [get_pins -hierarchical *STARTUP*/DATA_OUT[*]];
set_output_delay -clock clk_sck -min [expr $tdata_trace_delay_min -$th -
$tclk_trace_delay_max] [get_pins -hierarchical *STARTUP*/DATA_OUT[*]];
set_multicycle_path 2 -setup -start -from [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]] -to clk_sck
set_multicycle_path 1 -hold -from [get_clocks -of_objects [get_pins -hierarchical */
ext_spi_clk]] -to clk_sck
set_max_delay -datapath_only -from [get_pins -hier {*STARTUP*_inst/DI[*]}] 1.000
set_max_delay -datapath_only -from [get_clocks -of_objects [get_pins -hierarchical
*/ext_spi_clk]] -to [get_pins -hier *STARTUP*_inst/USRCCLKO] 1.000
set_max_delay -datapath_only -from [get_clocks -of_objects [get_pins -hierarchical
*/ext_spi_clk]] -to [get_pins -hier *STARTUP*_inst/DO[*]] 1.000
```

```
set_max_delay -datapath_only -from [get_clocks -of_objects [get_pins -hierarchical
*/ext_spi_clk]] -to [get_pins -hier *STARTUP*_inst/DTS[*]] 1.000
```

Note: [1] Multi Cycle Path constraints may be removed or updated on getting any timing violations.

Note: [2] Regarding set_max_delay constraints, recent Vivado tool flow takes care of adding set_max_delay constraints for US/US+. In this case, do not add explicit constraints mentioned above and make sure to comment the `set_max/min_delay` constraints.

3. STARTUPE2 Enabled:

Use the set of constraints for STARTUPE2.

```
# You must provide all the delay numbers
# CCLK delay is 0.5, 6.7 ns min/max for K7-2; refer Data sheet
# Consider the max delay for worst case analysis

set cclk_delay 6.7
# Following are the SPI device parameters
# Max Tco
set tco_max 7
# Min Tco
set tco_min 1

# Setup time requirement
set tsu 2

# Hold time requirement
set th 3

# Following are the board/trace delay numbers
# Assumption is that all Data lines are matched
set tdata_trace_delay_max 0.25
set tdata_trace_delay_min 0.25
set tclk_trace_delay_max 0.2
set tclk_trace_delay_min 0.2

### End of user provided delay numbers


# this is to ensure min routing delay from SCK generation to STARTUP input
# User should change this value based on the results
# having more delay on this net reduces the Fmax

set_max_delay 1.5 -from [get_pins -hier *SCK_O_reg_reg/C] -to [get_pins -hier
*USRCCLKO] -datapath_only
set_min_delay 0.1 -from [get_pins -hier *SCK_O_reg_reg/C] -to [get_pins -hier
*USRCCLKO]

# Following command creates a divide by 2 clock
# It also takes into account the delay added by STARTUP block to route the CCLK

create_generated_clock  -name clk_sck -source [get_pins -hierarchical
*axi_quad_spi_1/ext_spi_clk] [get_pins -hierarchical *USRCCLKO] -edges {3 5 7}
-edge_shift [list $cclk_delay $cclk_delay $cclk_delay]

# Data is captured into FPGA on the second rising edge of ext_spi_clk after the SCK
falling edge
```

```
# Data is driven by the FPGA on every alternate rising_edge of ext_spi_clk

set_input_delay -clock clk_sck -max [expr $tco_max + $tdata_trace_delay_max +
$tclk_trace_delay_max] [get_ports IO*_IO] -clock_fall;

set_input_delay -clock clk_sck -min [expr $tco_min + $tdata_trace_delay_min +
$tclk_trace_delay_min] [get_ports IO*_IO] -clock_fall;

set_multicycle_path 2 -setup -from clk_sck -to [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]]
set_multicycle_path 1 -hold -end -from clk_sck -to [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]]

# Data is captured into SPI on the following rising edge of SCK
# Data is driven by the IP on alternate rising_edge of the ext_spi_clk

set_output_delay -clock clk_sck -max [expr $tsu + $tdata_trace_delay_max -
$tclk_trace_delay_min] [get_ports IO*_IO];
set_output_delay -clock clk_sck -min [expr $tdata_trace_delay_min -$th -
$tclk_trace_delay_max] [get_ports IO*_IO];

set_multicycle_path 2 -setup -start -from [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]] -to clk_sck
set_multicycle_path 1 -hold -from [get_clocks -of_objects [get_pins -hierarchical */
ext_spi_clk]] -to clk_sck
```

STARTUP is Disabled:

```
# Following are the SPI device parameters
# Max Tco
set tco_max 7
# Min Tco
set tco_min 1

# Setup time requirement
set tsu 2

# Hold time requirement
set th 3

# Following are the board/trace delay numbers
# Assumption is that all Data lines are matched
set tdata_trace_delay_max 0.25
set tdata_trace_delay_min 0.25
set tclk_trace_delay_max 0.2
set tclk_trace_delay_min 0.2

### End of user provided delay numbers

create_generated_clock  -name clk_sck -source [get_pins -hierarchical
*axi_quad_spi_1/ext_spi_clk] [get_ports <SCK_IO>] -edges {3 5 7}


# Data is captured into FPGA on the second rising edge of ext_spi_clk after the SCK
falling edge
# Data is driven by the FPGA on every alternate rising_edge of ext_spi_clk
```

```
set_input_delay -clock clk_sck -max [expr $tco_max + $tdata_trace_delay_max +
$tclk_trace_delay_max] [get_ports IO*_IO] -clock_fall;

set_input_delay -clock clk_sck -min [expr $tco_min + $tdata_trace_delay_min +
$tclk_trace_delay_min] [get_ports IO*_IO] -clock_fall;

set_multicycle_path 2 -setup -from clk_sck -to [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]]
set_multicycle_path 1 -hold -end -from clk_sck -to [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]]

# Data is captured into SPI on the following rising edge of SCK
# Data is driven by the IP on alternate rising_edge of the ext_spi_clk

set_output_delay -clock clk_sck -max [expr $tsu + $tdata_trace_delay_max -
$tclk_trace_delay_min] [get_ports IO*_IO];
set_output_delay -clock clk_sck -min [expr $tdata_trace_delay_min -$th -
$tclk_trace_delay_max] [get_ports IO*_IO];

set_multicycle_path 2 -setup -start -from [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]] -to clk_sck
set_multicycle_path 1 -hold -from [get_clocks -of_objects [get_pins -hierarchical */
ext_spi_clk]] -to clk_sck
```

Constraints in Dual Quad Mode

In dual quad mode, the STARTUPE3 primitive is enabled. There are 2 SPI interfaces, one is connected to the STARTUPE3 block and the other is connected to the SPI1 (io0_1_,io1_1_,io2_1_,io3_1_ and ss_1_* ports).

The SPI_1 ports are constrained the way shown in STARTUP is Disabled. The rest of the constraints are taken from STARTUP3 is Enabled and based on the Ultrascale and Ultrascale + device family. In Ultrascale devices, the constraints file looks like the following:

```
# You must provide all the delay numbers
# CCLK delay is 0.5, 6.7 ns min/max for K7-2; refer Data sheet
# Consider the max delay for worst case analysis
set cclk_delay 6.7
# Following are the SPI device parameters
# Max Tco
set tco_max 7
# Min Tco
set tco_min 1
# Setup time requirement
set tsu 2
# Hold time requirement
set th 3
# Following are the board/trace delay numbers
# Assumption is that all Data lines are matched
set tdata_trace_delay_max 0.25
set tdata_trace_delay_min 0.25
set tclk_trace_delay_max 0.2
set tclk_trace_delay_min 0.2
### End of user provided delay numbers
```

```
create_generated_clock -name clk_sck -source [get_pins -hierarchical
*axi_quad_spi_0/ext_spi_clk] -edges {3 5 7} -edge_shift [list $cclk_delay
$cclk_delay $cclk_delay] [get_pins -hierarchical *startup*/*usrcclko]
set_multicycle_path -setup -from clk_sck -to [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]] 2
set_multicycle_path -hold -end -from clk_sck -to [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]] 1
set_multicycle_path -setup -start -from [get_clocks -of_objects [get_pins
-hierarchical */ext_spi_clk]] -to clk_sck 2
set_multicycle_path -hold -from [get_clocks -of_objects [get_pins -hierarchical */
ext_spi_clk]] -to clk_sck 1
#create_generated_clock -name clk_sck -source [get_pins -hierarchical
*axi_quad_spi_1/ext_spi_clk] [get_ports <SCK_IO>] -edges {3 5 7}
# Data is captured into FPGA on the second rising edge of ext_spi_clk after the SCK
falling edge
# Data is driven by the FPGA on every alternate rising_edge of ext_spi_clk
set_input_delay -clock clk_sck -max [expr $tco_max + $tdata_trace_delay_max +
$tclk_trace_delay_max] [get_ports IO*_1_IO] -clock_fall;
set_input_delay -clock clk_sck -min [expr $tco_min + $tdata_trace_delay_min +
$tclk_trace_delay_min] [get_ports IO*_1_IO] -clock_fall;
# Data is captured into SPI on the following rising edge of SCK
# Data is driven by the IP on alternate rising_edge of the ext_spi_clk
set_output_delay -clock clk_sck -max [expr $tsu + $tdata_trace_delay_max -
$tclk_trace_delay_min] [get_ports IO*_1_IO];
set_output_delay -clock clk_sck -min [expr $tdata_trace_delay_min - $th -
$tclk_trace_delay_max] [get_ports IO*_1_IO];
```

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado Design Suite simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

For information about simulating the example design, see Simulating the Example Design in Chapter 5.

IMPORTANT: *For cores targeting 7 series FPGAs or Zynq-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1].

For information about synthesizing and implementing the example design, see Implementing the Example Design.

Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite.

Overview

The top module instantiates all components of the core and example design that are needed to implement the design in hardware, as shown in Figure 1. This includes clock generator (MMCME2), Register configuration module, Read data checker and Slave.

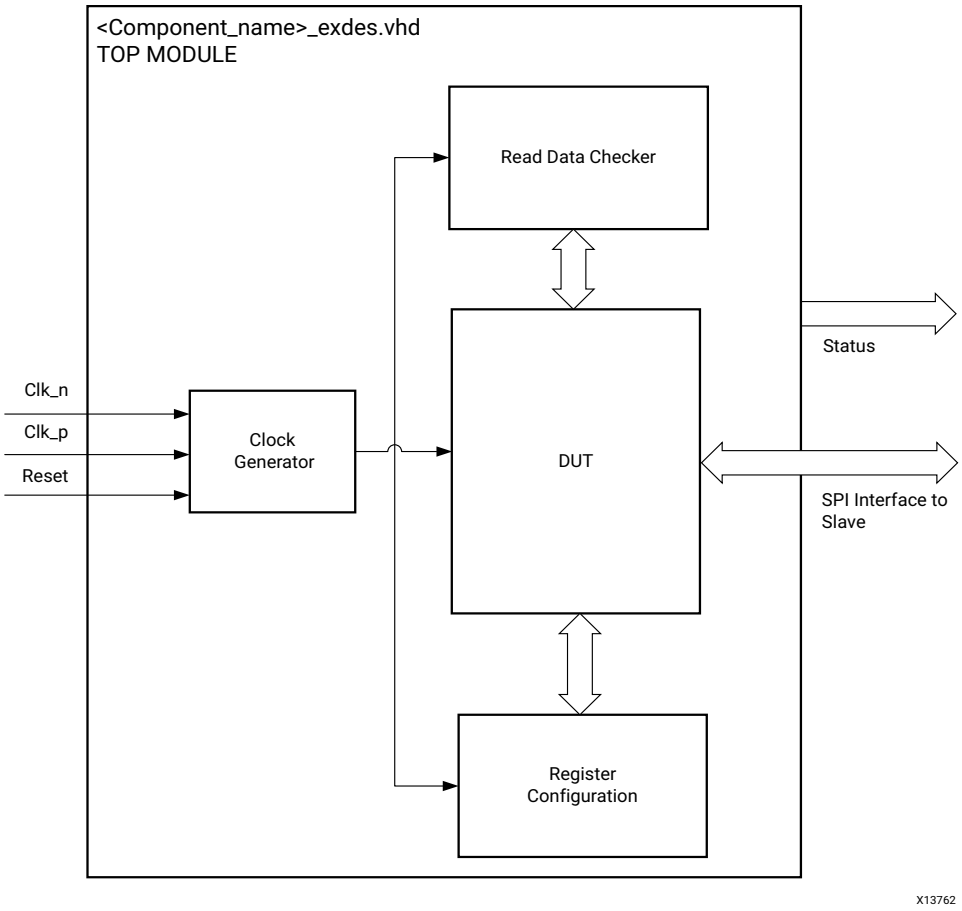

Figure 5-1: **Example Design Block Diagram**

The example design demonstrates transactions of the AXI Quad SPI core in its different configurations. It shows read and write transactions from the AXI Quad SPI to the slave memory in different configured modes.

- **Clock generator:** The MMCME2 is used to generate the clock for the example design. It generates a 200 MHz clock for the device under test (DUT) and other modules used in the example design. The DUT of the example design is kept under reset until MMCME2 is locked.
- **Register configuration module**: The AXI Traffic Generator is used for configuring the internal registers of the DUT and other modules (see Standard SPI Mode Transactions, page 76). After initial configuration of the registers sequence of data is sent to the AXI Quad SPI as AXI transactions, the AXI Quad SPI core generates SPI transactions for the same data and writes into the slave memory connected to it.
- **Read data checker**: The AXI Traffic Generator asks the AXI Quad SPI core to read the data from the connected slave memory by generating the commands, and verifies the correct data which is written previously is read.
- **Slave**: Memory models delivered mimic the behavior of Winbond, Micron, and Spansion Flashes, and are used as slave memory to the AXI Quad SPI core when it is configured in master mode.

In slave mode the AXI Quad SPI core is instantiated itself as a master internally to show the slave behavior of the core.

IMPORTANT: *The example design is not supported when the STARTUPEn block is enabled.*

Implementing the Example Design

After following the steps described in Chapter 4, Customizing and Generating the Core, implement the example design as follows:

1. Right-click the core in the Hierarchy window, and select **Open IP Example Design**.

   A new window pops up where you can specify a new directory name for the example design, or keep the default directory.

   A new project is created in the selected directory and is opened in a new Vivado window.

2. In the Flow Navigator (left side pane), click **Run Implementation** and follow the directions.

In the current project directory, a new project called `<component_name>_example` is created. This directory and its subdirectories contain all the source files that are required to create the AXI Quad SPI example design.

Table 5-1 shows the design files generated.

Table 5-1: **Example Design Directory**

Name	Description
<component_name>_exdes.vhd	Top-level HDL file for the example design.
memory.vhd	Memory model used in XIP mode.
memory_model.vhd	Memory model used in other modes.

Table 5-2 shows the COE files generated for data transmission.

Table 5-2: **COE Design Directory**

Name	Description
qspi_addr_*.coe	Delivers address information to the AXI Traffic Generator.
qspi_data_*.coe	Delivers data information to the AXI Traffic Generator.
qspi_ctrl_*.coe	Delivers control information to the AXI Traffic Generator.
qspi_mask_*.coe	Delivers mask information to the AXI Traffic Generator.
init_data_coe	Initialization data to BMG.

Notes:
1. The range from 1 to 3 as each file corresponds to a particular AXI Traffic Generator.

Table 5-3 shows the test bench file delivered.

Table 5-3: **Simulation Directory**

Name	Description
<component_name>_exdes_tb.vhd	Test bench for Exdes.

Table 5-4 shows the constraints file delivered.

Table 5-4: **Constraints Directory**

Name	Description
exdes.xdc	Top-level constraints file for the example design.

The example design has been verified on KC705 boards. Board constraints are also specified in the `exdes.xdc` file but are commented out by default.

IMPORTANT: *Uncomment the pin constraints while testing on the board.*

Testing the Example Design on a KC705 Board

Follow these steps to test the example design on a KC705 board:

1. Configure the core using the Vivado Integrated Design Environment (IDE) in standalone mode.
2. Implement the example design.
3. Generate the bitstream by selecting the implementation and generate the bitstream option.

The status of the transaction, such as Done, is displayed on F16, while the E18 and G19 pins indicate the AXI Traffic Generator core status. For more reference about these pins, refer to the `exdes_xdc.ttcl` file. It might be necessary to use the `exdes_xdc.ttcl` file after you uncomment the LOC constraints. For more information, see Checking Results, page 108.

With the external memories targeted as slave SPI, the example design is not supported on the board, but you can test the bitstream with the memory model on a given FPGA.

IMPORTANT: *You must set the parameter combinations for checking the core behavior through Vivado IDE options.*

The built-in memory model is used as target slave. The memory model and core are configured with the predefined command, address, and data. When the example design simulations are enabled, both the model and the core exchange data.

Legacy Mode and Performance Mode Example Design Behavior

For Standard SPI mode, Dual and Quad SPI mode, the core and the memory model have predefined commands. The standard page program command (`0x02h`) is used for writing the data in the memory model when the FIFO is enabled in the design. In case of a read from memory model, the commands are chosen based on the SPI mode of the core.

XIP Mode Example Design Behavior

In the XIP mode, the core has built-in commands as per the mode chosen. The memory model is pre-configured with the data and the same data is read by the core when a transaction is initiated.

Simulating the Example Design

Using the example design delivered as part of the AXI Quad SPI core, you can quickly simulate and observe the behavior of the core.

Setting up the Simulation

The Xilinx simulation libraries must be mapped to the simulator. To set up the Xilinx simulation models, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7]. To switch simulators, click **Simulation Settings** in the Flow Navigator (left pane). In the Simulation options list, change **Target Simulator**.

The example design supports functional (behavioral) and post-synthesis simulations. For information how to run simulation, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

Simulation Results

The simulation script compiles the AXI Quad SPI example design, and supporting simulation files. It then runs the simulation and checks that it completed successfully.

If the test passes, the following message is displayed:

```
Test Completed Successfully
```

If the test hangs, the following message is displayed.

```
Test Hanged
```

If the test fails, the following message is displayed.

```
Test Failed.
```

Example Programming Sequence

All SPI transactions in master mode depend upon commands supported by a slave device connected to the AXI QUAD SPI core. Refer to the respective SPI slave data sheet for the supported commands and mode (standard/Dual/Quad). The following steps briefly describe the erase, write and read command sequence for SPI flash.

Note: [1] The WRITE ENABLE command must be issued before every write transaction (erase/write data to the flash/register write of flash) command to the flash.

Note: [2] The following programming sequence is based on the 24-bit addressing mode of flash and assumes there is only one slave connected to the AXI QUAD SPI core.

Write Enable Command Sequence

1. Disable the master transaction by asserting the master inhibit bit of SPICR (60h), and reset the RX and TX FIFOs through SPICR.

   Example: write 0x1E6 to SPICR
2. Issue the write enable command by writing 0x06 into SPIDTR.
3. Issue chip select by writing 0x00 to SPISSR(70h).
4. Enable master transaction by deasserting the SPICR master inhibit bit.
5. Deassert chip select by writing 0x01 to SPISSR.
6. Disable master transaction by asserting the SPICR master inhibit bit.

Erase Command Sequence

1. Reset RX and TX FIFOs through SPICR.
2. Issue sector erase command [(1)] into SPIDTR to erase any specific sector followed by the flash sector address or issue the bulk erase command [(1)] to erase the entire flash followed by the flash base address.

   Example: Write 0xD8 to SPIDTR
3. Issue chip select by writing 0x00 to SPISSR.
4. Enable master transaction by deasserting the SPICR master inhibit bit.
5. Deassert chip select by writing 0x01 to SPISSR.
6. Disable master transaction by asserting the SPICR master inhibit bit.

Write Data Command Sequence

1. Reset RX and TX FIFOs through SPICR.
2. Issue the write data command[(1) (2)] into SPIDTR, to write data into any specific sector followed by the flash sector address.
3. Fill SPIDTR with the data to be written to flash; the maximum data size depends upon the configured QSPI FIFO size.
4. Issue chip select by writing 0x00 to SPISSR.
5. Enable master transaction by deasserting the SPICR master inhibit bit.
6. Deassert chip select by writing 0x01 to SPISSR.
7. Disable master transaction by asserting the SPICR master inhibit bit.

Read Data Command Sequence

1. Reset RX and TX FIFOs through SPICR.
2. Issue the read data command [(1) (2)] into SPIDTR to read data from any specific sector followed by the flash sector address.
3. Fill SPIDTR with the dummy data to read required data from the flash.
4. Issue chip select by writing 0x00 to SPISSR(70h).
5. Enable master transaction by deasserting the SPICR master inhibit bit.
6. Deassert chip select by writing 0x01 to SPISSR.
7. Disable master transaction by asserting SPICR master inhibit bit
8. Read SPIDRR, to get the Read data that is received from the SPI bus.

1. Refer to the respective SPI slave (flash) data sheet to know which commands to issue.
2. Write/Read commands vary with respect to the mode (Standard/Dual/Quad) used.

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

Overview

Figure 6-1 shows test bench for the AXI Quad SPI example design. The top-level test bench generates a 200 MHz clock and drives initial reset to the example design.

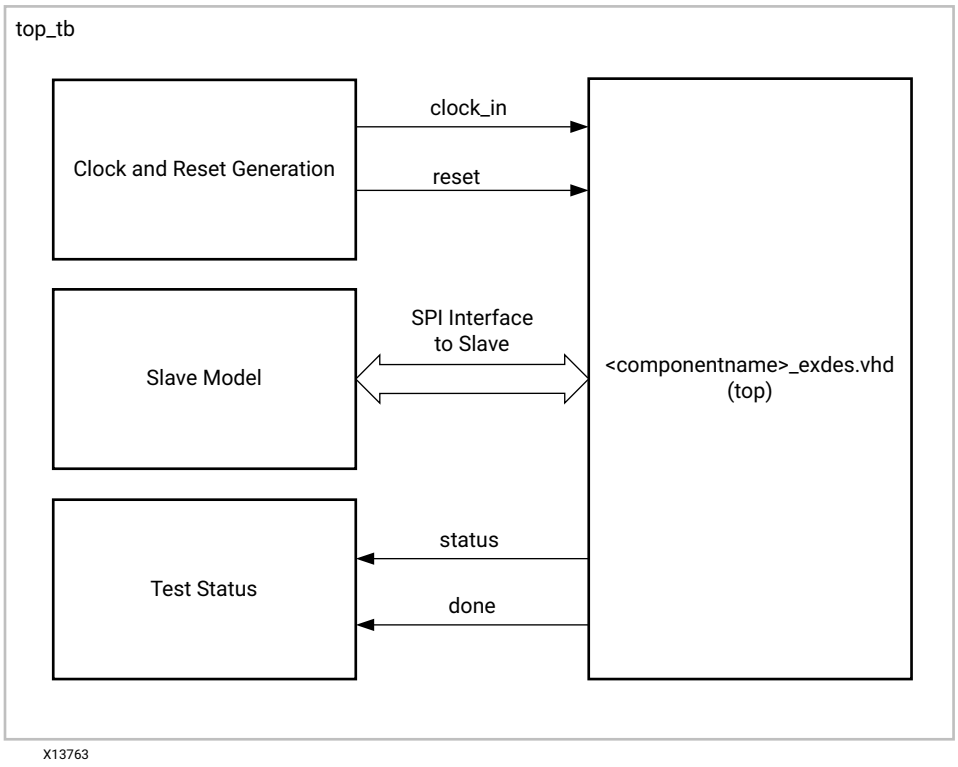

Figure 6-1: **AXI Quad SPI Example Design Test Bench**

Checking Results

The Quad SPI example design does not write much data into the slave device. It reads back and compares the read data with the written data. The signal status is used to convey information about the test case results.

- Status (1 downto 0) indicates whether the test passes, failed or hanged.

  01 - Test completed successfully. This means the read data matches with the written data.

  10 - Test failed. Read data does not match with the written data—data mismatch

  11 - Test hanged. The core is hanged at some particular instruction.

- Status (9 downto 2) gives the index of MIF where the test stopped.

Note: The `Done` signal goes High after the test is complete.

Verification, Compliance, and Interoperability

Table A-1 lists the device part numbers that were used for verifying and validating the AXI Quad SPI core.

Table A-1: **Device Part Numbers Used for Verification and Validation**

Device	Validation	Verification
Micron		
N25Q256A33E	No	Yes
N25Q256A31E	No	Yes
N25Q256A13E	No	Yes
N25Q256A11E	No	Yes
N25Q032A13E	No	Yes
N25Q032A11E	No	Yes
N25Q256A83E	No	Yes
N25Q256A73E	No	Yes
N25Q128A11E	No	Yes
N25Q128A11B	No	Yes
N25Q128A13E	No	Yes
N25Q128A13B	Yes	Yes
Spansion		
S25FL512S	No	Yes
S70FL01GS	Yes	Yes
Winbond		
W25Q80BL	No	Yes
W25Q80BV	No	Yes
W25Q80BW	No	Yes
W25Q80DL	No	Yes
W25Q80DV	No	Yes
W25Q80EW	No	Yes

Table A-1: **Device Part Numbers Used for Verification and Validation**

Device	Validation	Verification
W25Q80V	No	Yes

Notes:

1. See *Migrating from Micron's N25Q to Micron's MT25 technical note* [Ref 21], for compatibility of Micron MT25Q devices with validated N25Q devices.

Upgrading

This appendix contains information about migrating a design from ISE® Design Suite to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information on migrating to the Vivado Design Suite, see *ISE to Vivado Design Suite Migration Methodology Guide* (UG911) [Ref 8].

Upgrading in the Vivado Design Suite

The SPISEL port is hidden when master SPI mode is selected. This port is internally driven to VCC in master mode. In case of slave SPI operation mode, this port is available and should be connected to the Slave Select port of other SPI masters.

There were several I/O ports added that are applicable only in dual quad mode. See Table 2-2 on pages 21 and 22.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the AXI Quad SPI core. the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the AXI Quad SPI core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are listed here, and can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool messages
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

For the AXI Quad SPI Core Master Answer Record see Xilinx Answer 54408

Technical Support

Xilinx provides technical support at the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

Several internal signal are marked as debug signals. These can be easily added to the logic analyzer.

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 9].

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado Design Suite debug feature is a valuable resource to use in hardware debug.

1. Use the Vivado logic analyzer for hardware debug at the SPI interface.
2. Attach the logic analyzer ports on all SPI interface ports like `SS`, `SCK`, `IO0`, and `IO1` (also, `IO2`, and `IO3` if used in Quad mode).
3. Generate the transactions at the AXI interface, and observe whether or not the Slave Select line is asserted, and whether or not the SPI clock is present.
4. Observe whether or not the data from the core provides sufficient setup and hold time.

Interface Debug

AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. Output `s_axi_arready` asserts when the read address is valid, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `s_axi_aclk` and `aclk` inputs are connected and toggling.
- The interface is not being held in reset, and `s_axi_areset` is an active-Low reset.
- If the simulation has been run, verify in simulation and/or a Vivado Design Suite debug feature capture that the waveform is correct for accessing the AXI4-Lite interface.
- At a given time, the core will accept either an AXI read or AXI write transaction.

AXI4 Interfaces

The AXI Quad SPI core supports AXI4 interface in Enhanced mode and in XIP mode.

- In Enhanced mode, only FIXED AXI4 transactions are allowed for Data Transmit and Data Receive FIFO locations. Any other AXI4 transaction is not allowed and core behavior is not guaranteed.
- In XIP mode, all AXI4 transactions are allowed. It is recommended that you use 32-bit AXI4 transactions. The following points cover further debug information.

Enhanced Mode Debug

1. Make sure the clock is connected to the `s_axi4_aclk` port.
2. Make sure the active-Low `reset` signal is connected to the `s_axi4_aresetn` port. The reset should be de-activated to make sure the core is in working condition.
3. All the registers should be accessed with a single AXI transaction and at word boundary.
4. The DTR and DRR FIFO can be accessed using a single transaction or a burst FIXED transaction.
5. The AXI4 transactions are converted in the `bus2ip_*` signal transactions.
6. For each SPI transaction, it is necessary to have data beats are available in DTR FIFO. The SPI transactions are enabled only when the SPICR register is configured, and the slave register should be updated to select a particular slave. The SPI transactions can be observed on SPI interface like Slave Select, SPI clock, IO0 and IO1.
7. The Slave Select line must be asserted for any SPI transaction.

XIP Mode Debug

1. Make sure all the clocks and resets are connected to proper AXI interface.
2. This mode of the core supports all AXI4 transactions like FIXED, WRAP and INCR. Use a 32-bit AXI transaction for the best results. In this mode, the core operates in read-only mode.
3. Based on the number of address bits supported by the downstream device, select 24-bit or 32-bit addressing mode in the Vivado Integrated Design Environment (IDE). Based on this choice, the core considers either 24 or 32-bit from AXI4 address transactions.
4. Based on the choice of SPI mode, the core chooses the suitable SPI to read commands and add dummy cycles before accepting the correct data bits from memory.
5. The SPI transactions are observed on the SPI interface where the Slave Select line is asserted, and the SPI clock is generated by the core.
6. After the data is available in the core, the core supplies the data to an AXI4 read channel through an internal FIFO.

XILINX

Appendix D

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. *UltraScale Architecture Libraries Guide* (UG974)
2. *Vivado Design Suite User Guide: Designing with IP* (UG896)
3. *Vivado AXI Reference Guide* (UG1037)
4. *AMBA AXI4-Stream Protocol Specification* (ARM IHI 0051A)
5. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
6. *Vivado Design Suite User Guide: Getting Started* (UG910)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *ISE to Vivado Design Suite Migration Methodology Guide* (UG911)
9. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
10. *Motorola M68HC11-Rev. 4.0 Reference Manual*
11. *Motorola MPC8260 PowerQUICC II Users Manual 4/1999 Rev. 0*
12. *AXI4-Lite IPIF LogiCORE IP Product Guide* (PG155)
13. *AXI Interconnect LogiCORE IP Product Guide* (PG059)
14. *Winbond memory data sheet* (W25Q64BV)
15. *Micron memory data sheet* (N25Q256-3v)
16. *7 Series FPGAs Data Sheet: Overview* (DS180)
17. *7 Series FPGAs Configuration User Guide* (UG470)

18. *Execute-in-Place (XIP) with AXI Quad SPI Using Vivado IP Integrator Application Note* (XAPP1176)
19. *Throughput Performance Measurement Application Note* (XAPP797)
20. *Vivado Design Suite User Guide: Implementation* (UG904)
21. *Migrating from Micron's N25Q to Micron's MT25 technical note* (TN-25-01)
22. *UltraScale FPGA Post-Configuration Access of Parallel NOR Flash Memory using STARTUPE3* (XAPP1282)
23. *UltraScale FPGA Post-Configuration Access of SPI Flash Memory using STARTUPE3* (XAPP 1280)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/06/2021	3.2	• Updated Figure 4-1. • Added DRR NOT EMPTY feature at Byte level support. • Minor updates based on CR(s).
02/04/2021	3.2	• Added AXI Read Channel Mode in User Parameters. • Updated XIP Mode and added Enable XIP Mode figure. • Updated constraints for STARTUP3.
07/08/2019	3.2	Updated Table 3-3.
04/04/2018	3.2	• Added constraints in dual quad mode. • Updated the supporting commands for different devices. • Updated the FIFO depth values. • Added a note mentioning one byte command mode only support.
04/05/2017	3.2	• XDC updates for UltraScale+ devices.
10/05/2016	3.2	• Updated the SDK directory in Note 2 n the IP Facts table. • Added a note about the AXI4-Lite write access register to the beginning of the Register Space section. • Added the Dual Quad SPI Mode to Chapter 1, Overview. • Added new I/O signals that are applicable only in dual quad mode See Table 2-2. • Added the Using the Dual Quad Mode section to Chapter 3, Designing with the Core.
04/06/2016	3.2	• Added the Example Programming Sequence section to Chapter 5. • Updated the set_max_delay commands in Chapter 4.
11/18/2015	3.2	Added support for UltraScale+ families.

Date	Version	Revision
09/30/2015	3.2	• Added links to the web performance and utilization numbers. • Added information about the new STARTUPE3 primitive used with UltraScale™ devices. • Added EXT_spi_clk column to Table 2-1. • Modified the description of ss_i [(No. of Slaves – 1):0] in Table 2-4. • Added a note to the Master control register description in Table 2-7. • Added two notes to Table 4-1. • Replaced all references of Numonyx with Micron. • Removed all instances of Intel Flash (no longer makes Quad SPI flash S33 family obsolete since S3A) • Removed all instances of ST MicroElectronics as this is replaced by Micron. • Removed instances of Atmel. • Corrected Figure 3-2.
11/19/2014	3.2	Added UltraScale resource numbers.
10/01/2014	3.2	• Document updates only for revision change. • Removed E3 reference throughout document. • Updated Note #2 in Table 1-1: Core Operation Mode and Design Parameter Values. • Updated Table 2-2: Resource Utilization for a Kintex®-7 FPGA. • Updated STARTUP Signals in Table 2-3: I/O Signals. • Updated Bit[8] SPI Control Register. • Updated Bit[2] description in SPI Status Register. • Updated description in SPI Data Receive Register. • Updated Bit[2] description in IP Interrupt Status Register. • Removed opcode 20 in Table 3-1 and Table 3-2. • Added Table 3-3: Supported Commands in AXI Quad SPI Core in XIP Mode Commands section. • Added AXI Quad SPI Supported Devices section in Designing the Core chapter. • Added User Parameter table in Design Flow Steps chapter. • Added description for Required Constraints section. • Updated Fig. 5-1: Example Design Block Diagram. • Updated Fig. 6-1: AXI Quad SPI Example Design Test Bench.
04/02/2014	3.2	• Added Spansion Flash Support (Beta Version). • Added startup interface to bring out the startup signals to be used by other IP cores.
12/18/2013	3.1	Added UltraScale architecture support.

Date	Version	Revision
10/02/2013	3.1	• Updated for the core v3.1. Version in this table aligns to core version. • Added 32-bit address support for Micron Memories in XIP mode. • Added the core design example, and test bench information. • Added information related to simulation, synthesis and implementation, and Vivado IP integrator support. • Added hardware debug and interface debug information.
03/20/2013	1.0	Initial release in product guide format. This document replaces DS843. • Moved Core Mode and Parameter Value table to Chapter 1. • Revised I/O signal widths where affected. • Revised Resets section in Chapter 3. • Added 24-bit addressing restriction note to XIP mode section of Chapter 3. • Added IDE screen documentation and removed obsolete Design Parameter section in Chapter 4.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2013–2021 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. The DisplayPort Icon is a trademark of the Video Electronics Standards Association, registered in the U.S. and other countries. All other trademarks are the property of their respective owners.