XAPP737 (v1.0) June 12, 2007

SPI-4.2 to Quad SPI-3 Bridge in Virtex-4 FPGAs

Author: Zhe Xia

Summary

Often in communication systems, data must be moved between different protocols. This application note describes a reference design used to bridge one four-channel Xilinx SPI-4.2 (PL4) core (v8.1) to four single-channel SPI-3 (PL3) Link Layer cores (v4.1), implemented in a single Virtex™-4 device.

Software and IP Requirements

The software and IP used in the design are listed below:

- ISE™ 8.2.03i IP update1
- Xilinx SPI-3 Link Layer cores (v4.1)
- Xilinx SPI-4.2 core (v8.1)

Introduction

System Packet Interface, Level 3, (SPI-3) provides a link-layer interface for transferring packets at the OC48 data rate (2488.32 Mb/s) while System Packet Interface, Level 4 Phase 2, (SPI-4.2) provides a link-layer interface for transferring packets at the OC192 data rate (9953.28 Mb/s). In many communication applications, a bridge between two systems supporting different interfaces (for example, SPI-3 and SPI-4.2) is required. This application note presents a reference design bridging one, four-channel Xilinx SPI-4.2 (PL4) core (v8.1) to four, single-channel SPI-3 (also known as packet-over-SONET/SDH physical layer or POS-PHY™ Level-3) Link Layer cores (v4.1) in a Virtex-4 device.

Note: Full versions of the SPI-4.2 and SPI-3 Link-Layer cores used with this design are available for a licensing fee. Full-system hardware evaluation licenses are available at no additional cost. For more information regarding these and other cores, please contact a Xilinx distributor.

Design Overview

The reference design provides a bridge between a single, four-channel SPI-4.2 core and four, single-channel SPI-3 cores. All necessary functionality to transfer packets between the two sets of interfaces and appropriate flow control information is provided in this design. The reference design is based on the Xilinx SPI-4.2 (PL4) v8.1 and the POS-PHY Level-3 (SPI-3) Link Layer cores v4.1. Also included with the reference design files are the `.xco` files used to create the final design. Generating the reference design use options other than those described in this application note can induce errors in the resulting design.

Note: The source code provided is for the bridging function only (indicated as Bridge Reference Design in Figure 1). The other portions of the design must be purchased from Xilinx. An evaluation license is also available.

The design is divided into two distinct sections. The section "SPI-4.2 Core to Quad SPI-3 Cores" describes passing packets from the single SPI-4.2 sink core to the four SPI-3 transmit cores. The other section, "Four SPI-3 Cores to One SPI-4.2 Core," describes passing packets from the four SPI-3 receive cores to the single SPI-4.2 source core.

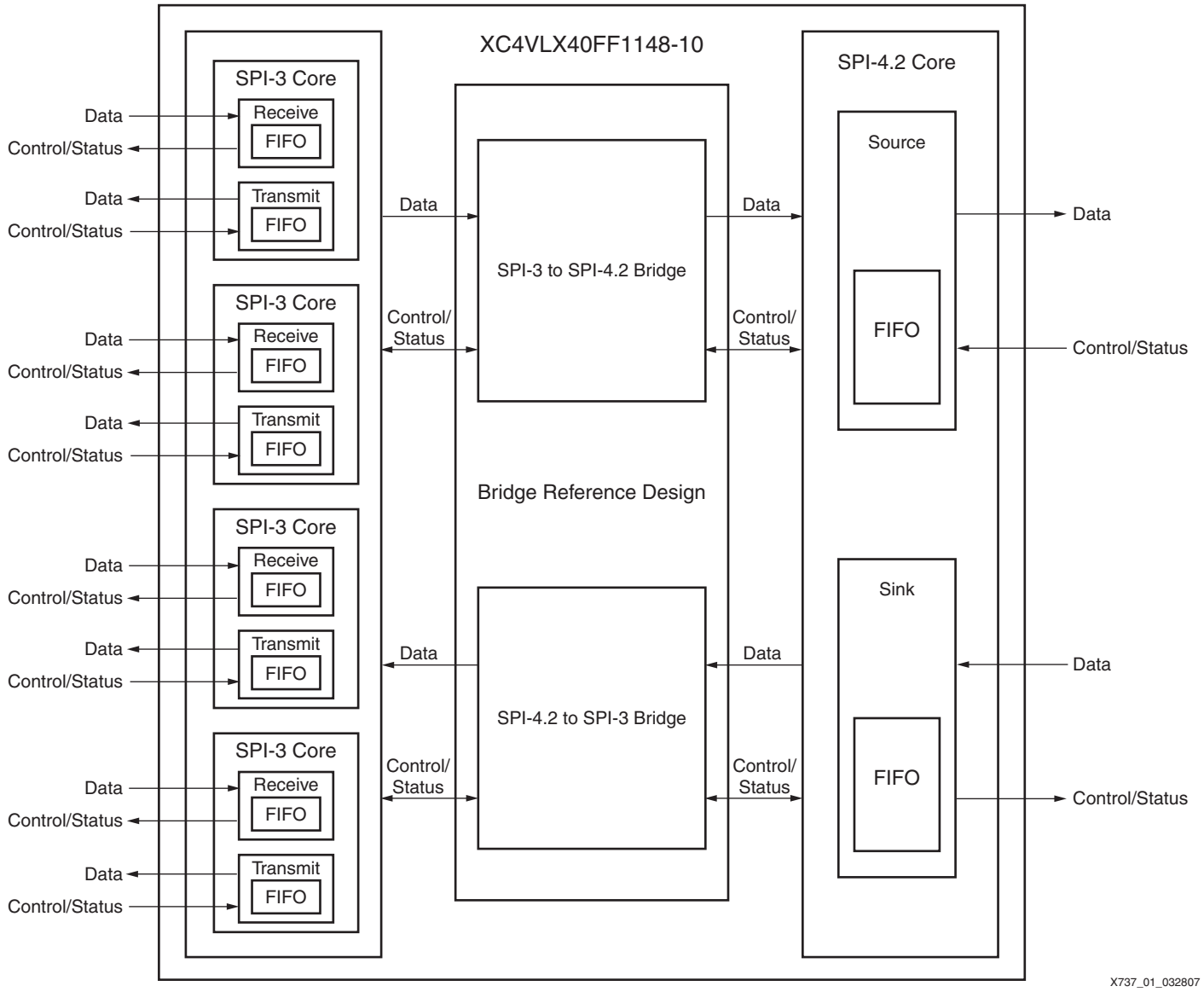

Figure 1: **SPI-4.2 to SPI-3 Bridge Top-Level Block Diagram**

SPI-4.2 Core to Quad SPI-3 Cores

At the top level, the SPI-4.2 sink core, four SPI-3 transmit cores, and spi4_to_spi3_core (SPI-4.2-to-SPI-3 Bridge in Figure 1) are instantiated. The spi4_to_spi3_core module contains the bridging logic for passing packets from the single SPI-4.2 sink core to four SPI-3 transmit cores (Figure 2).

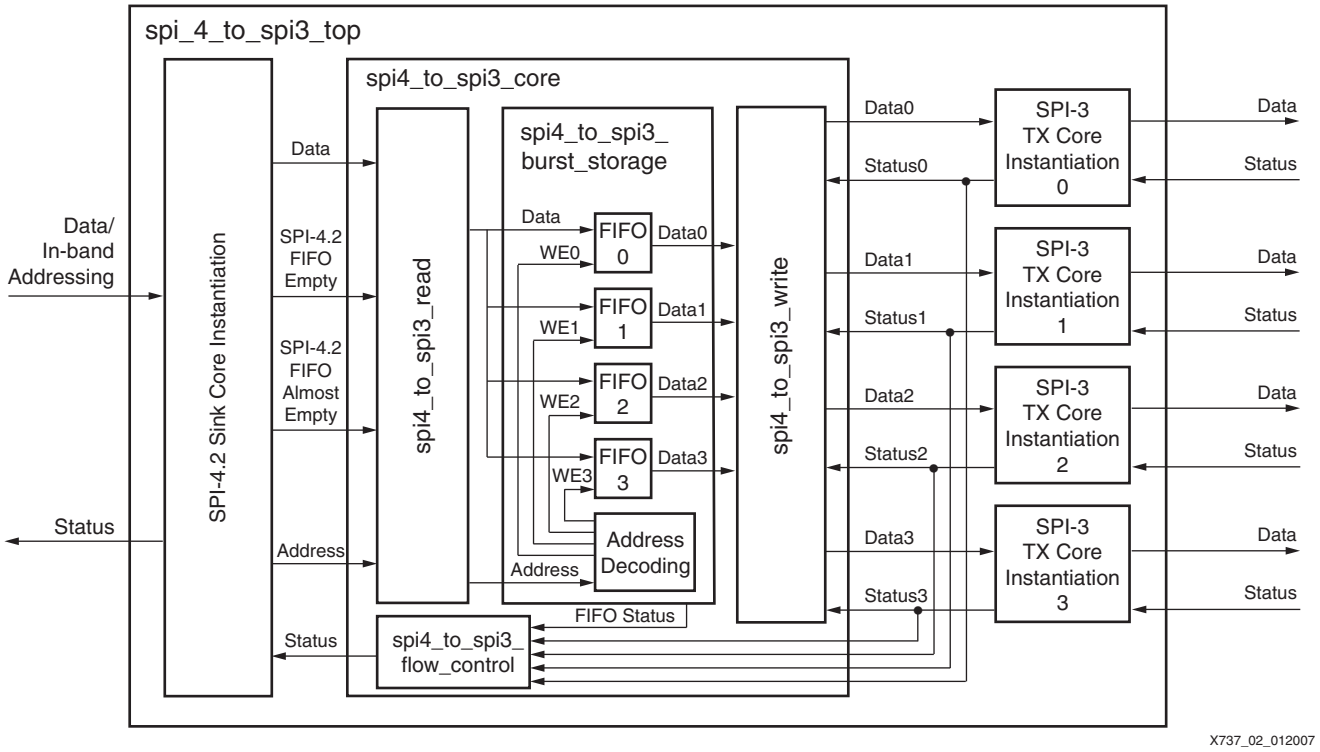

Figure 2: **SPI-4.2 Sink Core to SPI-3 Transmit Core Path Block Diagram**

Table 1 shows the major top signals in spi4_to_spi3_core (the SPI-4.2 and SPI-3 core interface signals are not listed).

Table 1: **Major Signals in spi4_to_spi3_core**

Signal Name	Description
SPI4_Addr[7:0]	Sink FIFO Channel Address. Same functionality as Snk_FFAddr[7:0] in SPI-4 sink core.
SPI4_Data[63:0]	Sink FIFO Data Out. Same functionality as FFData[63:0] in SPI-4 sink core.
SPI4_Mod[2:0]	Sink FIFO Modulo. Same functionality as FFMod[2:0] in SPI-4 sink core
SPI4_SOP	Sink FIFO Start of Packet. Same functionality as SnkFFSOP in SPI-4 sink core.
SPI4_EOP	Sink FIFO End of Packet. Same functionality as SnkFFEOP in SPI-4 sink core.
SPI4_Err	Sink FIFO Error. Same functionality as SnkFFErr in SPI-4 sink core
SPI4_Valid	Sink FIFO Read Valid. Same functionality as SnkFFValid in SPI-4 sink core.
SPI3_0_BrstData[69:0]	Data read from burst storage FIFO 0. Corresponding to SPI-3 Core 0.
SPI3_1_BrstData[69:0]	Data read from burst storage FIFO 1. Corresponding to SPI-3 Core 1.
SPI3_2_BrstData[69:0]	Data read from burst storage FIFO 2. Corresponding to SPI-3 Core 2.
SPI3_3_BrstData[69:0]	Data read from burst storage FIFO 3. Corresponding to SPI-3 Core 3.

Table 1: **Major Signals in spi4_to_spi3_core *(Continued)***

Signal Name	Description
SPI3_0_BrstRdEn	Read enable for burst storage FIFO 0.
SPI3_1_BrstRdEn	Read enable for burst storage FIFO 1.
SPI3_2_BrstRdEn	Read enable for burst storage FIFO 2.
SPI3_3_BrstRdEn	Read enable for burst storage FIFO 3.
SPI3_0_Brst_Empty	Empty flag for burst storage FIFO 0.
SPI3_1_Brst_Empty	Empty flag for burst storage FIFO 1.
SPI3_2_Brst_Empty	Empty flag for burst storage FIFO 2.
SPI3_3_Brst_Empty	Empty flag for burst storage FIFO 3.
SPI3_0_Brst_HalfFull	Indicates burst storage FIFO 0 is half full.
SPI3_1_Brst_HalfFull	Indicates burst storage FIFO 1 is half full.
SPI3_2_Brst_HalfFull	Indicates burst storage FIFO 2 is half full.
SPI3_3_Brst_HalfFull	Indicates burst storage FIFO 3 is half full.

The SPI-4.2 to SPI-3 core consists of four submodules: the read module, the burst storage module, the write module, and the flow control module. The hierarchy of the SPI-4.2 core to four SPI-3 cores design is illustrated in Figure 3.

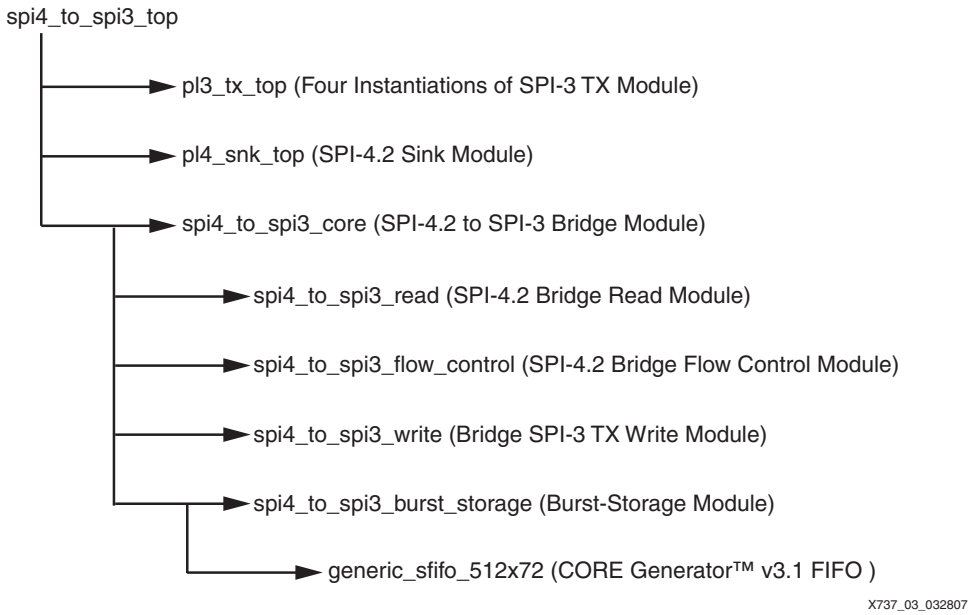

Figure 3: **SPI-4.2 to SPI-3 Bridge Virtex-4 Design Hierarchy**

Read Module

The read module reads packet data/address information from the SPI-4.2 sink core along with packet control signals such as start of packet (SOP), end of packet (EOP), error (Err), and modulo (Mod). The data read from the SPI-4.2 core is 64 bits wide. The address information indicating channel information is 2 bits wide.

Data is not read from the SPI-4.2 core if SnkFFEmpty_n signal from SPI-4.2 sink core is asserted, indicating that the SPI-4.2 sink core's internal FIFO is empty. Otherwise, data is read from the core. When data is read from the SPI-4.2 sink core and written into the FIFOs in burst-storage module, the full flags of the FIFOs are not monitored by the read module. Once the FIFO is half full, the flow control module sends a halt-data-transfer (a status of satisfied)

request. Therefore, if the device supplying data to the SPI-4.2 core in the user's system is slow to respond to a request to halt data transfer, then overflow is possible within the bridge.

Burst-Storage Module

The burst-storage module contains the address decoding logic plus four FIFOs for holding packet data. Each FIFO holds the data for one SPI-3 transmit core and one SPI-4.2 channel. The instantiated FIFOs are LogiCORE™ modules created using FIFO Generator v3.1 (provided free with the ISE 8.2i). See "Modifying the Reference Design" for details on modifying the burst-storage FIFOs.

Data is both read and written to the FIFOs within the burst-storage module as 70-bit data: 64 bits of packet data, SOP, EOP, 3 bits of the Mod signal, plus the Err signal (Table 2). Each FIFO can hold up to 4 KB of data, corresponding to 512 72-bit words.

In the reference design, address 0 for the SPI-4.2 sink core corresponds to instantiation 0 of the SPI-3 transmit core, address 1 corresponds to instantiation 1, and so on. The user can modify the address decode according to the needs of the application (see "Changing Channel Mapping"). Individual write enable signals are provided to each FIFO submodule after address decode.

Table 2: **Signals Mapping of Burst-Storage FIFOs**

	Reserved	Data[63:0]	SOP	EOP	MOD[2:0]	ERR
Bit location	71:69	68:6	5	4	3:1	0

Write Module

The write module reads data out of each of the FIFOs in the burst-storage module, writing it to the corresponding SPI-3 transmit core. Data is read out of each of the FIFOs of the burst-storage module if the FIFO is not empty (SPI3_Brst_Empty is not active), and the corresponding SPI-3 transmit core is ready to receive data (SPI_3_DST_RDY of SPI-3 TX core is asserted). After the packet data is read from the burst-storage FIFO, the data is converted from 64 bits to 32 bits. In addition, the Mod signal must be examined to determine if all of the 64 bits of data are valid and to determine the correct time to send EOP, SOP, Mod, and Err signals.

The Mod signal is a modulo signal used to indicate valid bytes on the last cycle of a transfer (as specified by the OIF SPI3-01.0 standard). Table 3 shows Mod signaling versus the valid bits on the 32-bit data bus.

Table 3: **Example Mod Signaling**

Mod[1:0]	Data Valid
00	[31:0]
01	[31:8]
10	[31:16]
11	[31:24]

The Mod signal must also be converted into a 2-bit signal called REM, used in SPI-3 interfacing, before being sent on. The REM signal on the SPI-3 user interface and Mod signal on the SPI-3 packet interface are not coded identically. The Xilinx LocalLink standard specifies that the remainder signal indicates the number of valid bytes on the data bus as REM + 1, with the valid bytes MSB justified. Table 4 shows the REM signaling versus the valid bits on the 32-bit data bus.

Table 4: **Example REM Signaling**

REM[1:0]	Data Valid
00	[31:24]
01	[31:16]

Table 4: **Example REM Signaling**

REM[1:0]	Data Valid
10	[31:8]
11	[31:0]

Flow Control Module

The flow control module controls the status sent back to the SPI-4.2 sink core for transmission on the SPI-4.2 bus. The status for a particular SPI-4.2 channel is in satisfied status when the corresponding SPI-3 transmit core's FIFO is almost full or the corresponding burst storage FIFO is half full (the alternate status is starving). The half-full level was chosen for the burst-storage FIFO module to increase the amount of latency that can be tolerated before the device supplying data to the SPI-4.2 sink core must respond to the satisfied status.

Four SPI-3 Cores to One SPI-4.2 Core

At the top-level, one spi3_to_spi4_core, four SPI-3 RX cores, and one SPI-4.2 source cores are instantiated (Figure 4), handling SPI-3 traffic from the four single-channel SPI-3 cores and transferring it to a single four-channel SPI-4.2 source core. Flow control received from the SPI-4.2 core determines which channel is active and how much data is transferred for each channel. This block also provides 4 KB of intermediate buffering for each channel in the bridge.

All the bridge functionality is performed by the spi3_to_spi4_core block with four sub blocks:

- spi3_to_spi4_read
- spi3_to_spi4_burst_storage
- spi3_to_spi4_arbiter
- spi3_to_spi4_write

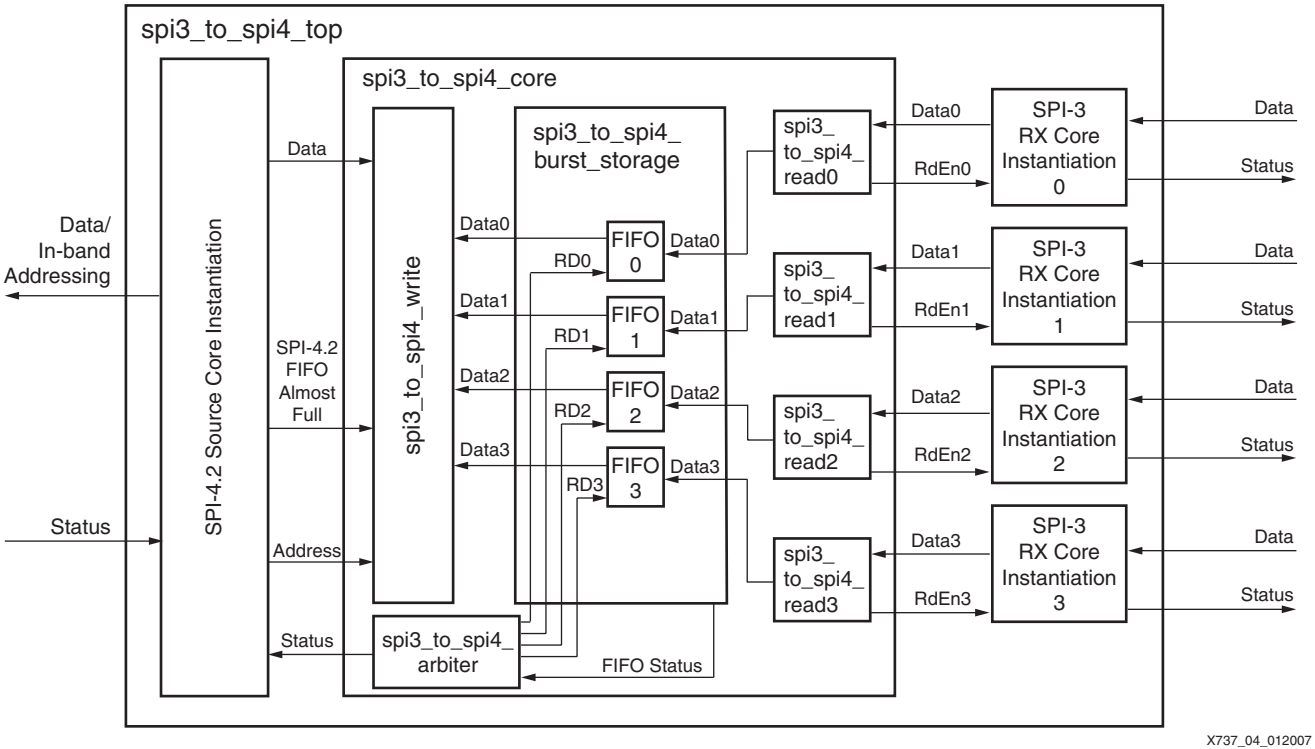

Figure 4: **SPI-3 Receive Core to SPI-4.2 Source Core Path Block Diagram**

The hierarchy of the four SPI-3 cores to one SPI-4.2 core path portion of the design is illustrated in Figure 5.

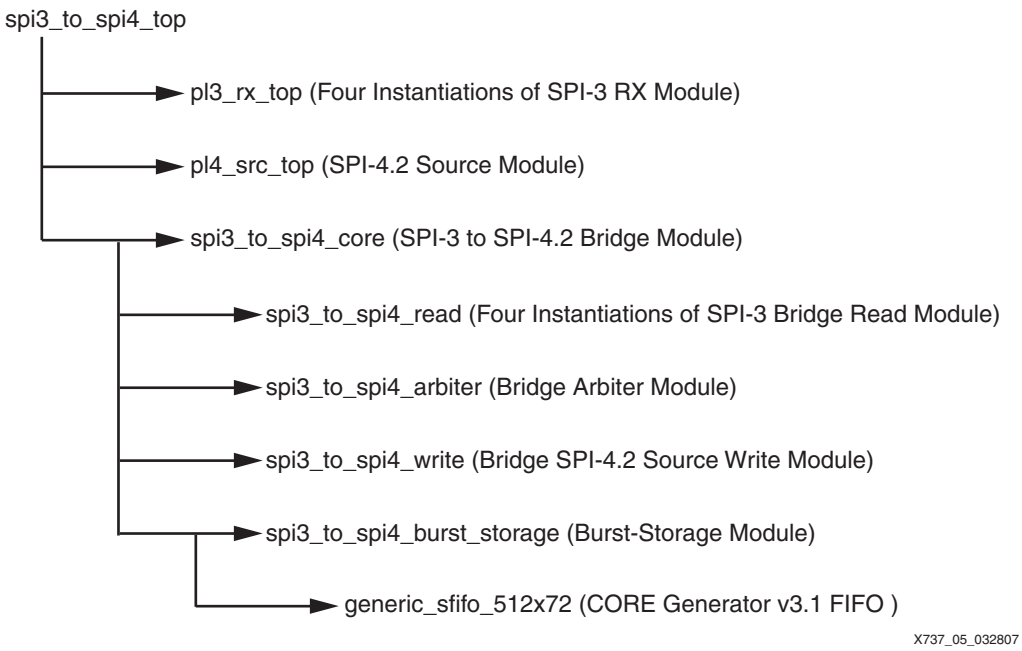

Figure 5: **SPI-3 to SPI-4.2 Bridge Hierarchy**

Read Module

Spi3_to_spi4_read blocks are used to read data from four SPI-3 RX cores. These blocks also write data into FIFOs of the burst-storage modules. If SPI-3 RX side is ready (indicated by SPI3_SRC_RDY of SPI-3 RX side), the read module reads from the SPI-3 FIFO and transfers to the burst-storage FIFO, provided the burst-storage FIFOs is not full. The end application could stop the transaction when almost_full flags of the FIFOs are asserted, allowing for more latency.

Burst-Storage Module

The burst-storage modules provide intermediate buffering of packet data for each SPI-3 core. There are four instantiations of this module, one for each SPI-3 RX core. These burst storage modules have 4 KB of capacity and store all the packet information, including SOP, EOP, Err, Mod, and packet data from the SPI-3 RX core. The burst-storage module merges two 32-bit SPI-3 data words into one 64-bit FIFO word and converts the 2-bit SPI-3 REM signals to 3-bit SPI-4.2 Mod signals, resulting in 72-bit data (Table 5).

Note: The EOP signal is triplicated to decrease fanout and improve timing.

Table 5: **Signals Mapping of Burst-Storage FIFOs**

	EOP	EOP	EOP	SOP	ERR	MOD[2:0]	DATA1[31:0]	DATA2[31:0]
Bit location	71	70	69	68	67	66:64	63:32	31:0

On the SPI-4.2 bus, the transfer begins with a payload control word indicating the start of packet (SOP), and ends with other payload control words indicating the end of packet (EOP), indicated on the user interface of the source core by the signals SrcFFSOP, SrcFFEOP, and SrcFFERRs.

The SPI-4.2 Mod signal is a modulo signal indicating which bytes on the 64-bit SPI-4.2 data bus are valid when the SrcFFEOP or SrcFFErr signal is asserted. When SrcFFEOP is deasserted, the SPI-4.2 Mod signal should always be zero.

Table 6 shows the mapping between the packet status signals on the user interface and SPI-4.2 control words in the reference design ([Ref 1]).

Table 6: **SPI-4.2 Mod Signal Mapping to 64-bit User Interface**

Control Word	Associated Source FIFO Signals	Associated SPI-4.2 Control Word bits on TDat(1,2)
End of Packet (EOP, even bytes valid)	SrcFFEOP, SrcFFMOD[2:0] When TDat[14:13] = 10: • Mod = 000 if data bits 63–0 have valid data • Mod = 110 if data bits 63–16 have valid data • Mod = 100 if data bits 63–32 have valid data • Mod = 010 if data bits 63–48 have valid data	TDat[14:13] = 10
End of Packet (EOP, odd bytes valid)	SrcFFEOP, SrcFFMod[2:0] When TDat[14:13] = 11: • Mod = 111 if data bits 63–8 have valid data • Mod = 101 if data bits 63–24 have valid data • Mod = 011 if data bits 63–40 have valid data • Mod = 001 if data bits 63–56 have valid data	TDat[14:13] = 11

Notes:

1. Qualified by the TCtl signal.
2. TDat is the 16-bit data bus is used to transmit SPI-4.2 data and control information.

The burst-storage module also provides FIFO status information, indicating whether the FIFO is full or almost empty. The burst-storage FIFO almost-empty threshold is defined in the package file `spi_pkg.v/vhd` and must be greater than the MaxBurst1 and MaxBurst2 parameters (defined in "Configuration Parameters"). Finally, the burst-storage module keeps track of the number of EOPs in the FIFO and is used during arbitration, indicated by PktCntEq0 and PktCntEq1 signals shown in Table 7.

Arbiter Module

The arbiter module `spi3_to_spi4_arbiter.v/vhd` arbitrates between the four SPI-3 channels to transfer packet data from burst storage to the SPI-4 source FIFO. Each SPI-3 channel is eligible for transfer when it has at least one EOP in its burst FIFO or if its burst-storage FIFO has more data than almost-empty-threshold words. The arbiter always transfers a multiple of SPI-4.2 credits (1 credit = 16 bytes) as per the SPI-4.2 specification.

In addition, the data transferred for each channel depends on the status of that channel, as monitored by the module. Major signals in arbiter module are defined in Table 7.

The arbiter transfers:

- MaxBurst1 credits when starving status (00) is received for a channel.
- MaxBurst2 credits when hungry status (01) is received for a channel, terminating transfer immediately after EOP is transferred from burst storage to SPI-4.2 source FIFO.

Table 7: **Major Signals in Arbiter Module**

Signal Name	Description
BrstRdEn	Burst Storage Read Enable Signal
BrstRdChan	Burst storage read channel.
BrstRdEOP	Burst storage Read EOP. Indicates if next read burst-storage word has EOP.
BrstRdValid	Burst storage Read Valid. Indicates if burst-storage has valid data.
PktCntEq0	0 = no EOPs are present on the FIFO input/output bus. 1 = at least one EOP is present on the FIFO input/output bus.
PktCntEq1	1 = only one EOP is present in the burst FIFO. 0 = either zero EOPs or more than one EOP is present in the burst-storage FIFO.

Write Module

The write module (`spi3_to_spi4_write`) writes the current channel's data (plus SOP, EOP, Err, Mod and Addr) from its burst-storage FIFO to the SPI-4.2 source FIFO.

Initialization Sequencing

The proper start sequence for SPI-4.2 v8.1 is:

Source

1. Assert source core reset (Reset_n).
2. Assert and release the TDClk DCM Reset (DcmReset_TDClk) and TSClk DCM Reset (DcmReset_TSClk). This step is applicable only for the master clocking core using global clock distribution.
3. Wait for the SrcClksRdy signal to be asserted. If the source core is used in slave-clocking mode and a custom clocking module is used, the module waits until all the clocks necessary for SPI4.2 Core are ready to use.
4. Release the source core reset (Reset_n).
5. Program the source calendar (handled in the `bridge_top.v/vhd` module of the reference design if required).
6. Enable the source core (SrcEn = 1).

Sink

1. Assert sink core Reset (Reset_n) and assert sink IDELAYCTL Reset (SnkIdelayCtlRst) if used.

   Note: SnkIdelayCtlRst is not used in the reference design.
2. Assert and release the RDClk DCM Reset (DcmReset_RDClk).
3. Wait for the SnkClksRdy signal to be asserted.
4. Release sink core Reset (Reset_n) and the sink IDELAYCTL Reset (SnkIdelayCtlRst) if used.
5. Program the sink calendar (handled in the `bridge_top.v/vhd` module of the reference design if required).
6. Enable the sink core (SnkEn = 1).

The following steps are required in dynamic-alignment mode only:

7. Pulse the PhaseAlignRequest signal High for two times the SnkFFClk period (just long enough for PhaseAlignRequest to be recognized).
8. Deassert the PhaseAlignRequest signal
9. Monitor SnkDPAFailed and SnkOof. If SnkDPAFailed = 1 and SnkOof = 1, repeat the PhaseAlignRequest.

   If Auto Retry is enabled under dynamic phase alignment (DPA) option, it is not necessary to repeat the PhaseAlignRequest. The core automatically initiates the alignment again each time the alignment fails. Auto Retry does not initiate the PhaseAlignRequest when the core goes out of frame or loses the lock during the normal operation. In this case, the resynchronization must be manually initiated.

   Note: When dynamic-phase-alignment mode is used, the sink core must be receiving a valid training pattern at the time PhaseAlignRequest is asserted. The source core must be capable of sending a training pattern at the startup and when sink core out of frame (SnkOof) is asserted by the sink core. For subsequent alignment, after the High-to-Low transition of PhaseAlignRequest, the module waits for at least eight SnkFFClk clocks before monitoring SnkDPAFail.

Module spi_clk_startup

Also included at the top level of the reference design is a module called `spi_clk_startup`. This module assures that the SPI-4.2 core source and sink modules plus the DCMs within these modules, as implemented in the reference design, are initialized in the proper sequence.

Configuration Parameters

Table 8 shows the bridge configuration parameters defined in the `spi_pkg.v/vhd` file.

Table 8: **Virtex-4 SPI-4.2 to SPI-3 Bridge Configuration Parameters**

Configuration Signal	Description and Range	Range	Default Value
MaxBurst1	**MaxBurst1 for Starving Channels.** When a SPI4.2 channel receives starving status (status = `00`), it is able to accept a burst length of MaxBurst1 blocks. SPI3-to-SPI4 bridge transfers MaxBurst1 SPI4.2 credits (1 credit = 16 bytes) to the starving channel.	1 to 255 MaxBurst1 < (SPI3_to_SPI4_AEThres parameter)/2	8
MaxBurst2	**MaxBurst2 for Hungry Channels.** When a SPI4.2 channel receives hungry status (status = `01`), it is able to accept a burst length of MaxBurst2 blocks. SPI3-to-SPI4 bridge transfers MaxBurst2 SPI4.2 credits (1 credit = 16 bytes) to the hungry channel.	1 to 254 MaxBurst2 < (SPI3_to_SPI4_AEThres parameter)/2	4
SPI3-to-SPI4_AEThresh	**SPI3-to-SPI4 Burst Storage Almost Empty Threshold.** This parameter specifies the number of FIFO words (64 bit) that must be present in the burst-storage FIFO for a given channel before the FIFO almost-empty signal is asserted. This value must be greater than 2 × MaxBurst1.	4 to 504	255

Synthesis and Implementation

This reference design has all the required VHDL and Verilog files for the bridge logic between four SPI-3 cores and one SPI-4.2 core. Before implementing the reference design, the user must either purchase the full version of the SPI-4.2 and SPI-3 cores from Xilinx, or license the full-system hardware evaluation versions at no additional cost. The netlist files and simulation models of the SPI-4.2 and SPI-3 cores are required for implementation and verification of this bridge.

Within the reference design directory, there are four sub-directories: the `hdl/` directory, the `synth/` directory, the `implement/` directory and the `simulation/` directory. The `hdl/` directory provides all Verilog/VHDL files for the bridge design (excluding the SPI-3 and SPI-4.2 cores). The `synth/` directory provides the synthesis related files. The `implement/` directory provides the implementation related files. The `test/` directory provides the functional simulation related files.

Note: The synthesis and implementation tool used in the reference design is ISE 8.2.03.i, update 1.

Steps to Synthesize and Implement the Bridge Design.

1. Generate the SPI-4.2 and SPI-3 cores using a valid license.
2. Copy the following netlist files and place them in the `\implement\ngc\verilog` or the `\implement\ngc\vhdl` directories:

   ```
   pl4_v8_1_pl4_snk_top.ngc
   pl4_v8_1_pl4_src_top.ngc
   spi3_link_v4_1_spi3_link_rx.ngc
   spi3_link_v4_1_spi3_link_tx.ngc
   ```

3. Run `run_xst.bat` in `synth\verilog` or `synth\vhdl` directory to synthesize the design using XST 8.2.03.
4. Run `run_vhd` or `run_verilog` in `\implement` directory to implement the design with ISE 8.2.03.

Testbench and Simulation

The demonstration test suite (Figure 7) consists of three files:

```
master_clocks.v/vhd
spi3_emulator_phy.v/vhd
spi4_to_4spi3_tb.v/vhd
```

The top-level file of the testbench is `spi4_to_4spi3_tb.v/vhd` (Figure 6). In this file, the two other components of the test suite and the bidirectional bridge design under test (`bridge_top.v/vhd`) are connected. The sink side and source side of SPI-4.2 interface are connected in a loopback manner to allow data to flow through both directions of the bridge. Four SPI-3 PHY emulators (`spi3_emulator_phy.v/vhd`) are instantiated and connected to the SPI-3 RX and TX interfaces of the bridge. Initially, the emulators and the bridge design are held in reset. After reset is deasserted, data is allowed to flow from the SPI-3 emulators through the SPI-3 to SPI-4 core. As the SPI-4.2 interface is in loopback, the data passed through the SPI-3-to-SPI-4 core passes back through the SPI-4-to-SPI-3 bridge to the SPI-3 emulators.

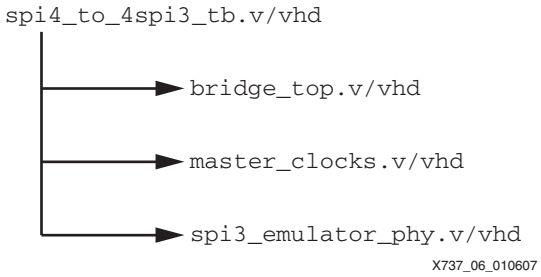

Figure 6: **Testbench Structure**

The file `master_clocks.v/vhd` generates the external clocks necessary to drive the bridge design and the emulators. Four clocks are required:

- One for the SPI-3 RX interfaces.
- One for the SPI-3 TX interfaces.
- Two for the SPI-4.2 interfaces (one is the SPI-4.2 receive data clock and the other is the 200 MHz reference clock used in Virtex-4 SPI-4.2 design).

The `spi3_emulator_phy.v/vhd` file provides a simplified behavioral simulation model of a SPI-3 PHY. The model is configured as a single-channel PHY device transmitting sets of eight DWORD (nominally 32-byte) packets to an SPI-3 port. The SOP and EOP signals are driven as expected to send an eight DWORD packet. During an EOP condition, the values that the Modulos (Mod) is driven with is constantly varied throughout the simulation to create packet lengths varying between 29 and 32 bytes. In addition, some packets are sent with the Err signal asserted to indicate a packet errors. The emulator uses a generic value to seed the starting data value for the packets to be sent to the link layer SPI-3 port, thus allowing different instances of the emulator to send packets containing different data (but data sent in each packet is the same).

For the TX port of the PHY interface emulation, varying polled-status information is provided back to the TX interface on the link-layer SPI-3 port. As the emulator is only designed to be a simple demonstration, no error checking is performed on the data received on its SPI-3 TX interface.

The following additional files must be copied into the simulation directory to use the test suite:

```
spi3_link_v4_1_spi3_link_rx.v/vhd
spi3_link_v4_1_spi3_link_tx.v/vhd
pl4_v8_1_pl4_snk_top.v/vhd
pl4_v8_1_pl4_src_top.v/vhd
```

These files are the gate-level simulation models for the SPI-3 and SPI-4.2 cores used in the bridge design. These models are generated in CORE Generator together with the netlist files, separate from the reference design. These simulation models use alternative names for SPI-3 and SPI-4: POS-PHY Level 3 (PL3) and POS-PHY Level 4 (PL4), respectively.

After the files are copied to the simulation directory, the logic simulator ModelSim should start from `test/<verilog|vhdl>`. From the ModelSim main window, simulation is started by executing `simulate.do`. This DO file compiles the appropriate source files, runs the simulation, and displays the waveforms at the end of the simulation run. Data is visible at all four SPI-3 interfaces and the looped back SPI-4 interface, as seen in the waveform when running the sample ModelSim scripts provided with the demonstration test suite. Only the external interfaces are shown to reduce the complexity of the displayed waveforms.

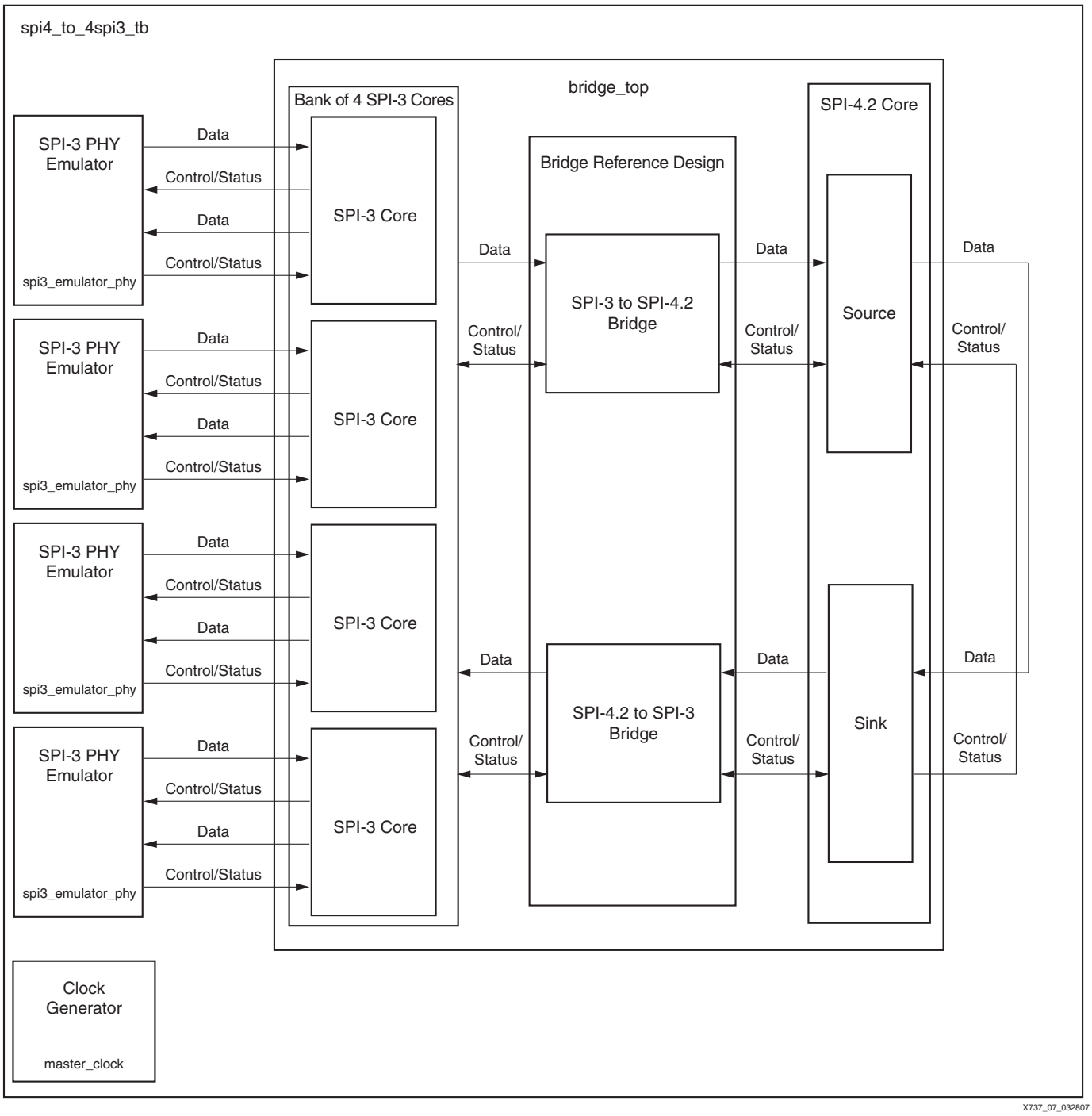

Figure 7: **Demonstration Test Suite**

Modifying the Reference Design

Minor modifications can be made to the Virtex-4 SPI-4.2 to SPI-3 bridge reference design. This section describes how to make a few of these modifications. After the modifications are made, the design might not meet timing using the constraints file provided.

Performance Requirements

This design is specified to work with the SPI-4.2 core at an operating frequency of 350 MHz DDR (175 MHz internal frequency), and the SPI-3 core at an operating frequency of 104 MHz. The constraints file can be modified to support additional data rates, up to the maximum operating frequencies of the SPI-4.2 and SPI-3 cores.

During implementation, the following warning can result:

```
WARNING:Timing:3233 - Timing Constraint "TS_RDClk_P = PERIOD TIMEGRP
"RDClk_P" 351 MHz HIGH 50%;" fails the minimum period check for the input
clock spi4_to_spi3_top0/pl4_snk_top0/U0/clk0/RDClk_ibufgds to DCM
spi4_to_spi3_top0/pl4_snk_top0/U0/clk0/rdclk_dcm0 because the period
constraint value (2849 ps) is less than the minimum internal period limit
of 3332 ps.   Please increase the period of the constraint to remove this
timing failure.
```

Currently, the ISE 8.2 tools use the more restrictive period check for the DFS outputs, which is 3332 ps or 300 MHz for a -10 speed grade device. If the input and output frequencies coincide with the specification of stated in "DCM and PMCD Switching Characteristics" of [Ref 2], these warnings can be ignored.

Clock Requirements

When targeting a Virtex-4 device, the SPI-4.2 core requires a 200 MHz reference clock. This reference clock provides a time reference to the IDELAYCTRL modules to calibrate the individual delay elements (IDELAY) in the clock region. This clock must be routed on a global clock buffer.

The SPI-4.2 user interface, bridge logic, and SPI-3 user interfaces are currently all clocked synchronously from the SPI-4.2 output clock SysClkDiv_GP. This design can be modified to clock these interfaces with a different clock. To clock the SPI-4.2 and SPI-3 user interfaces at different clock frequencies, refer to “Changing to an Asynchronous FIFO.”

To simplify clocking within the design, the bridge design assumes that all four SPI-3 cores use the same RFClk and TFClk clocks. However, the bridge design can be modified to support independent clocks for the SPI-3 interfaces.

Device and Package Requirements

This reference design is targeted to a Virtex-4 XC4VLX40FF1148-10 device. However, the provided constraints file can be modified to support additional Virtex-4 device/package combinations. Any new device/package combination for this design must be supported by the SPI-4.2 v8.1 core. To modify the UCF file provided with the reference design, lines in the UCF file should be replaced with corresponding information from the UCF files generated together with the SPI-4.2 v8.1 core in particular device/package combination.

Resource Requirements

The slice count for the design is 7917 slices (42% of the Virtex-4 LX40 slices) and the block RAM count is 48 (50% of the Virtex-4 LX40 slices). The block RAM usage can be increased or decreased by adjusting the size of the internal bridge FIFOs (see “Changing FIFO Depth” for more details).

Note: The resource utilization above is calculated including the SPI-4.2 and SPI-3 cores.

SPI-4.2 Core: Static versus Dynamic Alignment

The SPI-4.2 core implemented with the bridge design is the dynamic alignment version. If dynamic alignment is used with the bridge design, then no modification is required to the provided code. However, if static alignment is desired, then the file `spi_clk_startup.v/vhd` must be modified by commenting out the entire procedure `snk_fsm`. This procedure uses the PhaseAlignRequest/PhaseAlignComplete signals to ensure the dynamic alignment logic is successfully trained and is not required in the static alignment implementation. Instead, the PhaseAlignRequest signal is driven to a constant 0.

Monitoring the Full Flag of Burst-Storage FIFO

As mentioned in "Read Module," potential data overflow is possible when the device supplying data to the SPI-4.2 core in the user's system is slow to respond to a request to halt data transfer. The user can monitor the almost-full flag of the burst-storage FIFO to eliminate the problem. After the almost-full flag of the burst-storage FIFO is High, the user should not read any more data from the SPI-4.2 sink core.

Changing FIFO Depth

To change the size of the burst-storage FIFO:

1. Open the Xilinx CORE Generator tool.
2. Click **Create a New Project.**
3. Set options as below:
   a. Choose Virtex-4 device and related packages.
   b. Under Design Entry, choose **VHDL** or **Verilog**.
   c. Set Vendor as **ISE**, and click **OK.**
4. Select the menu item **File** → **Recustomize core.**
5. Browse to `hdl/generic_sfifo_512x72.xco`, and click **Open.**
6. The FIFO depth can now be changed to a different size. A larger FIFO requires more block RAM and affects timing.

Changing to an Asynchronous FIFO

Using different clocks for clocking the user side of the SPI-4.2 cores and the user side of the SPI-3 cores causes the bridge logic to cross clock domains. To make this change, the `generic_sfifo_512x72.xco` must be changed to an asynchronous FIFO, generated using the CORE Generator system.

In addition, the module `hdl/<verilog|vhdl>/spi3_to_spi4_burst_storage.v/vhd`, the module `hdl/<verilog|vhdl>/spi4_to_spi3_burst_storage.v/vhd`, and the wrapper files must be modified as follows:

1. Add both user-side clocks to the modules. This extra port must be reflected in all levels of wrapper files.
2. Change the `generic_sfifo_512x72` instantiations in each module must to reflect the new asynchronous FIFO module.
3. Drive the empty, almost-empty, half-full, etc. signals appropriately by signals from the newly instantiated FIFO.
4. Minimize glitching and metastability on the packet count signals (PktCntEq0 and PktCntEq1) generated in the process create_pkt_cnt in the file:

   `hdl/<verilog|vhdl>/spi3_to_spi4_burst_storage.v/vhd`

   This process compares signals crossing clock domains.
5. Change all levels of wrapper files so each module is clocked by the appropriate clock.

Changing Channel Mapping

Currently, the SPI-4.2 channel 0 is mapped to SPI-3 core instantiation 0, channel 1 is mapped to instantiation 1, and so on. This can be changed easily with two modifications:

- In the SPI-4.2 to SPI-3 direction, the channel mapping is determined by the file:

  `hdl/spi4_to_spi3_burst_storage.v/vhd`

  in the sequential block write_enable_proc. To change the mapping, modify the address contained in the if statement within this sequential block.
- In the SPI-3 to SPI-4.2 direction, the mapping is determined by the file:

  `hdl/<verilog|vhdl>/spi3_to_spi4_core.v/vhd`

  in the spi3_to_spi4_burst_storage instantiations. To change the mapping, change all port mappings of the SPI3_<port number> and Brst_<port number> to reflect the desired mapping.

Changing Arbitration Schemes

Currently, in the SPI-3 to SPI-4.2 direction, the arbitration to determine the SPI-3 core data to be transferred is determined in a round-robin fashion. If a different arbitration scheme is desired, modify the schaddr process in the file:

`hdl/<verilog|/vhdl>/spi3_to_spi4_arbiter.v/vhd`.

Reference Design Specification

The reference design for implementing the SPI-4.2 to quad SPI-3-bridge reference design targeted to Virtex-4 devices is available at:

http://www.xilinx.com/bvdocs/appnotes/xapp737.zip

Table 9 provides device, performance, and resource utilization information on the SPI-4.2-to-quad-SPI-3 bridge reference design.

Table 9: **Reference Design Specifications (Virtex-4 Device)**

Parameter		Value
Device supported		XC4VLX40FF1148-10
Performance	SPI-4.2 interface	350 MHz (max)
	SPI-3 interface	104 MHz (max)
	Bridge design	175 MHz (max)
Resource utilization (includes SPI-4.2 and SPI-3 cores)		7917 Virtex-4 slices 48 RAMB16s

Conclusion

The SPI-4.2 to quad SPI-3-bridge reference design implements an effective way to move packets from one data rate to another and constructs a bridge between two standard devices supporting different interfaces standards.

References

1. UG153, *SPI-4.2 v8.3 User Guide*. Download requires registration.
2. DS302, *Virtex-4 Data Sheet: DC and Switching Characteristics*.
3. UG154, *LogiCORE SPI-4.2 v8.3 Getting Started Guide*.
4. DS504, *SPI-3 Link Layer Data Sheet*.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/12/07	1.0	Initial Xilinx release.