LogiCORE IP SPI-4.2 v10.4

User Guide

UG153 March 1, 2011

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/30/04	1.0	Initial Xilinx release.
11/11/04	1.1	Updated document to support SPI-4.2 core v7.1.
04/28/05	1.2	Updated document to support SPI-4.2 core v7.2.
8/31/05	2.0	Updated document to support SPI-4.2 core v7.2, ISE v7.1i with SP 3, added information for Sink and Source core clock updates.
1/18/06	3.0	Updated for ISE v81.i support
7/13/06	4.0	Added support for Virtex-5, updated ISE to v8.2i, core version to 8.1
9/21/06	5.0	Minor updates for IP minor release.
2/15/07	5.5	Added information about DPA training word feature, updated system requirements for IP1 Jade release, and release date.
8/08/07	6.0	Updated for the IP1 Jade Minor release, added new SnkIdelayCtlRdy Input signal.
3/24/08	6.5	Modified ScrAFThresAssert[8:0] signal description, updated core version to 8.5, and updated supported tools.
9/19/08	7.0	Updated for ISE Service Pack 3 release, minor text edits.
4/24/09	7.5	Updated core to v9.1 and ISE to v11.1.
6/24/09	8.0	Updated core to v9.2 and ISE to v11.2.
9/16/09	9.0	Updated core version to v9.3 and ISE to version 11.3. Added support for Virtex-6 -1L and Virtex-6 CXT devices.
4/19/10	9.1	Updated core version to v10.1 and ISE to version 12.1; added support for Spartan-6 in -3 and higher speed grades.
9/21/10	10.0	Updated core version to v10.2 and ISE to version 12.3.
12/14/10	10.1	Updated core version to v10.3 and ISE to version 12.4; removed Spartan-6 support, updated Virtex-6 performance numbers and supported clocking scheme.
3/1/11	11.0	Updated core version to v10.4 and ISE to version 13.1.

Table of Contents

Chapter 4: Generating the Core

Chapter 5: Designing with the Core

Chapter 6: Constraining the Core

Chapter 7: Special Design Considerations

Chapter 8: Simulating and Implementing the Core

Chapter 9: Quick Start Example Design

Chapter 10: Detailed Example Design

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Licensing the Core

Chapter 3: Core Overview

Chapter 4: Generating the Core

Chapter 5: Designing with the Core

Chapter 6: Constraining the Core

Chapter 7: Special Design Considerations

Schedule of Tables

Chapter 1: Introduction

Chapter 2: Licensing the Core

Chapter 3: Core Overview

Chapter 4: Generating the Core

Chapter 5: Designing with the Core

Chapter 6: Constraining the Core

Chapter 7: Special Design Considerations

Chapter 8: Simulating and Implementing the Core

Chapter 9: Quick Start Example Design

Chapter 10: Detailed Example Design

Appendix A: Messages and Warnings

Appendix B: VHDL Details

Appendix C: Verilog Details

About This Guide

The *LogiCORE SPI-4.2 User Guide* describes the function and operation of the Xilinx LogiCORE SPI-4.2 (PL4) core, and provides information about designing, customizing and implementing the core.

Contents

This guide contains the following chapters:

- Preface, About this Guide describes the organization and purpose of the user guide, and the conventions used in this document.
- Chapter 1, Introduction introduces the SPI-4.2 core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, Licensing the Core provides information about installing and licensing the core.
- Chapter 3, Core Overview describes the SPI-4.2 core architecture and interface signals.
- Chapter 4, Generating the Core describes how to generate the SPI-4.2 core using the Xilinx CORE Generator system.
- Chapter 5, Designing with the Core describes how to use the Xilinx SPI-4.2 core in a user application.
- Chapter 6, Constraining the Core describes how to constrain the core.
- Chapter 7, Special Design Considerations describes other considerations when designing with the core, including multi-core instantiations and clocking schemes.
- Chapter 8, Simulating and Implementing the Core provides instructions for simulating and implementing the SPI-4.2 core with their design.
- Chapter 9, Quick Start Example Design provides instructions to quickly generate the core and run the example design through implementation and simulation using the default settings.
- Chapter 10, Detailed Example Design describes the files and directories created by the CORE Generator. It also contains detailed information about the demonstration test bench and directions for customizing it for use in a user application.
- Appendix A, Messages and Warnings describes common warnings and errors.
- Appendix B, VHDL Details provides details about the VHDL demonstration test bench and how to customize it.
- Appendix C, Verilog Details provides details about the Verilog demonstration test bench and how to customize it.

- Appendix D, SPI-4.2 File Descriptions describes the files generated by the CORE Generator system.
- Appendix E, SPI-4.2 Control Word defines the SPI-4.2 control word format.
- Appendix F, SPI-4.2 Calendar Programming lists examples that describe how to program calendars for the Source FIFO status and Sink FIFO status of the SPI-4.2 core.
- Appendix G, SPI-4.2 Core Verification describes the software verification of the SPI-4.2 core.
- Appendix H, SPI-4.2 Source Interface Timing Budget contains examples on how to create and analyze a SPI-4.2 source interface timing budget.

Additional Resources

To find additional documentation, see the Xilinx website at:

http://www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

http://www.xilinx.com/support/mysupport.htm.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported

Convention	Meaning or Use	Example
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **bus[7:0]**, they are required.	**ngdbuild** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**lowpwr ={on|off}**
Vertical bar |	Separates items in a list of choices	**lowpwr ={on|off}**
Angle brackets < >	User-defined variable or in code samples	<directory name>
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Repetitive material that has been omitted	**allow block** *block_name loc1 loc2 ... locn;*
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	**usr_teof_n** is active low.

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Chapter 1

Introduction

The Xilinx SPI-4.2 (PL4) LogiCORE™ IP core implements the *OIF-SPI-4-02.1 System Packet Interface Phase 2* specification and supports both VHDL and Verilog design environments.

This chapter introduces the SPI-4.2 core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.

System Requirements

Windows

- Windows XP® Professional 32-bit/64-bit
- Windows Vista® Business 32-bit/64-bit

Linux

- Red Hat® Enterprise Linux WS v4.0 32-bit/64-bit
- Red Hat® Enterprise Desktop v5.0 32-bit/64-bit (with Workstation Option)
- SUSE Linux Enterprise (SLE) v10.1 32-bit/64-bit

Software

- ISE® 13.1

About the Core

The SPI-4.2 core is a Xilinx CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP center. For detailed information about this core, see:

www.xilinx.com/products/ipcenter/DO-DI-POSL4MC.htm

For information about licensing options, see Chapter 2, Licensing the Core

Recommended Design Experience

Although the SPI-4.2 core is a fully verified solution, the challenge associated with implementing a complete design varies, depending on the configuration and functionality required. For best results, previous experience with building high-performance, pipelined FPGA designs using Xilinx implementation software and the user constraints files (UCF) is recommended.

Contact your local Xilinx representative for a closer review and estimate of the effort required to meet your specific design requirements.

Additional Core Resources

For detailed information and updates about the SPI-4.2 core, see the following documents, located on the SPI-4.2 product page at:

www.xilinx.com/products/ipcenter/DO-DI-POSL4MC.htm

- SPI-4.2 *Data Sheet*
- SPI-4.2 *Release Notes*

Technical Support

To obtain technical support specific to the SPI-4.2 core, visit support.xilinx.com/. Questions are routed to a team of engineers with expertise specific to using the SPI-4.2 core.

Xilinx will provide technical support for use of this product as described in this *SPI-4.2 User Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that deviate from the guidelines provided in these documents.

Feedback

Xilinx welcomes comments and suggestions about the SPI-4.2 core and the documentation provided with the core.

SPI-4.2 Core

For comments or suggestions about the SPI-4.2 core, please submit a webcase from support.xilinx.com/. Be sure to include the following information:

- Product name
- Core version number
- Explanation of your comments

Document

For comments or suggestions about this document, please submit a WebCase from support.xilinx.com/. Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

XILINX®

Chapter 2

Licensing the Core

This chapter provides instructions for obtaining a license for the core so that you can use the core in a design. The SPI-4.2 core is provided under the terms of the Xilinx LogiCORE Site License Agreement. This license agreement conforms to the terms of the SignOnce IP License standard defined by the Common License Consortium. Purchase of the core entitles you to technical support and access to updates for a period of one year.

Before you Begin

This chapter assumes that you have installed the core using either the CORE Generator™ IP Update installer or by performing a manual installation after downloading the core from the web. For information about installing the core, see the SPI-4.2 product page.

Before installing the core, you must have a Xilinx.com account and the ISE v13.1 software installed on your system.

To set up an account and install the ISE software:

1. Click Sign in to Access Account at the top of the Xilinx home page; then follow the instructions to create a support account.
2. Install the ISE 13.1 software with the applicable service pack.

License Options

The SPI-4.2 core provides three licensing options. After installing the required Xilinx ISE software and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess core functionality with either the example design provided with the SPI-4.2 core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the SPI-4.2 core using the example design and demonstration test bench provided with the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Full implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time outs

Obtaining Your License Key

Note: No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Obtaining a Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license, do the following:

1. Navigate to the product page for this core:
   www.xilinx.com/products/ipcenter/DO-DI-POSL4MC.htm
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

Obtaining a Full License

To obtain a Full license key, you must purchase a license for the core. After doing so, click the "Access Core" link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

XILINX®

Chapter 3

Core Overview

This chapter describes the SPI-4.2 core architecture and interface signals.

System Overview

The SPI-4.2 core is comprised of two separate cores that enable the transmission (Source core) and reception (Sink core) of data.

- **Sink core**: Receives data from the SPI-4.2 interface. It takes the 16-bit interface and maps it to a 64-bit or 128-bit interface enabling the internal logic to run at a quarter of the line rate.
- **Source core**: Transmits data on the SPI-4.2 interface. Payload data written into the core as 64-bit or 128-bit words (four or eight 16-bit SPI-4.2 words, respectively) is mapped onto the 16-bit SPI-4.2 interface.

Figure 3-1 illustrates the interfaces of the SPI-4.2 core and shows it in a typical link-layer application.

In the link-layer example, the SPI-4.2 interface connects an external physical-layer device to a link-layer implemented in a Virtex®-6, Virtex-5, or Virtex-4 FPGA. The user logic reads data from the Sink core and writes data into the Source core. A standard FIFO interface is provided for this data access and facilitates integration within a system. Dedicated signals are used to configure the Sink and Source cores in circuit and monitor a suite of status registers.

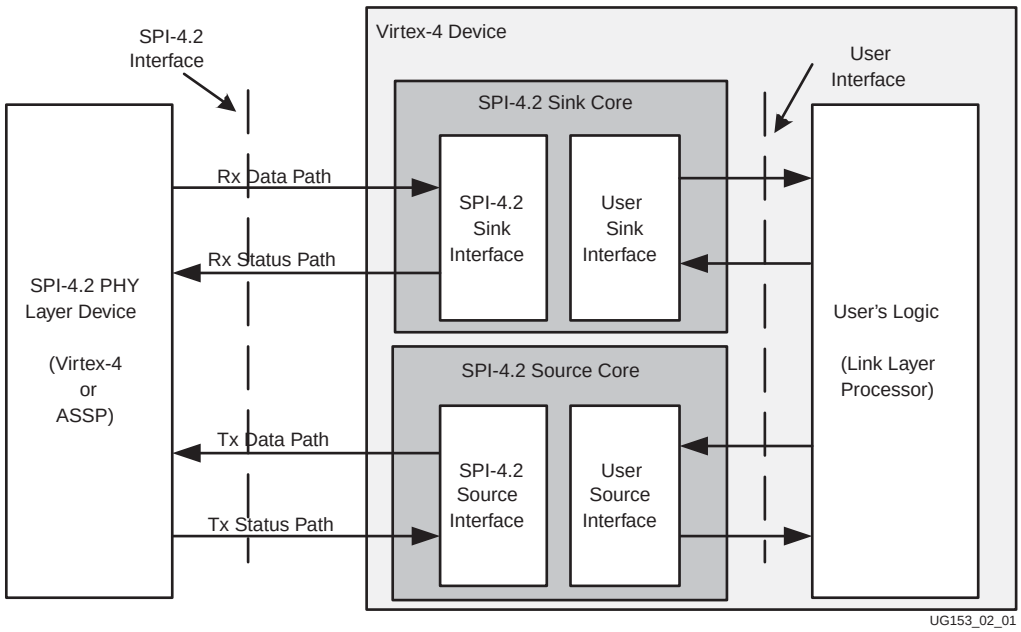


Figure 3-1: **SPI-4.2 Core in a Typical Link Layer Application**

Sink Core

The Sink core receives data from the SPI-4.2 interface. It takes the 16-bit interface and maps it to a 64-bit or 128-bit interface enabling the internal logic to run at a quarter (for 64-bit) or an eighth (for 128-bit) of the line rate. The user data and the corresponding control signals are accessed with a standard FIFO interface. The FIFO read and write operations are performed in independent clock domains.

The Sink core implements the following features.

- Supports 64-bit or 128-bit user data width
- Embedded dynamic alignment support in Virtex-6, Virtex-5 and Virtex-4 FPGAs I/O (embedded SERDES, delay chain, and bitslip module) that runs at data rates exceeding 1 Gbps.
- Dedicated output signal indicating loss of valid `RDClk`.
- Provides a FIFO reset signal for clearing contents of the data pipe during operation.
- Provides support for forcing the insertion of DIP-2 errors for system testing.
- Regional clocking option (saves global clocking resources).

For more information about core features, see Chapter 5, Designing with the Core.

Source Core

The Source core transmits data on the SPI-4.2 interface. Payload data written into the core as 64- or 128-bit words (four or eight 16-bit SPI-4.2 words, respectively) are mapped onto the 16-bit SPI-4.2 interface. Although packet data written into the core may not be 64- or 128-bit aligned, the core optimally maps the data to 16-bit words such that no filler idle cycles are inserted. The data along with the control signals are written into the core via a standard FIFO interface, and the FIFO read and write operations are performed in independent clock domains.

The Source core implements the following features:

- Supports 64-bit or 128-bit user data width
- Optionally transmits only complete data bursts
- Provides both master and slave clocking to facilitate multiple core implementations (Virtex only)
- Enables addressable or transparent access to SPI-4.2 flow control data
- Provides a FIFO reset signal for clearing contents of the data pipe during operation
- Provides support for forcing the insertion of DIP-4 errors for system testing

For more information on core features, see Chapter 5, Designing with the Core.

Sink Core Interfaces

The Sink core contains five functional modules:

- Sink Data FIFO
- Sink Data Receive
- Sink Status Registers
- Sink Calendar
- Sink Status Transmit

The Sink core has the following interfaces:

- Sink SPI-4.2 Interface
- Sink User Interface
  - Sink Control and Status Interface
  - Sink FIFO Interface
  - Sink Status and Flow Control Interface
    - Calendar Control Interface
    - Status FIFO Interface
  - Sink Configuration Interface
  - Sink Clocking Interface

The functional modules and signals which comprise the different interfaces are shown in Figure 3-2 and defined in tables in the following sections.

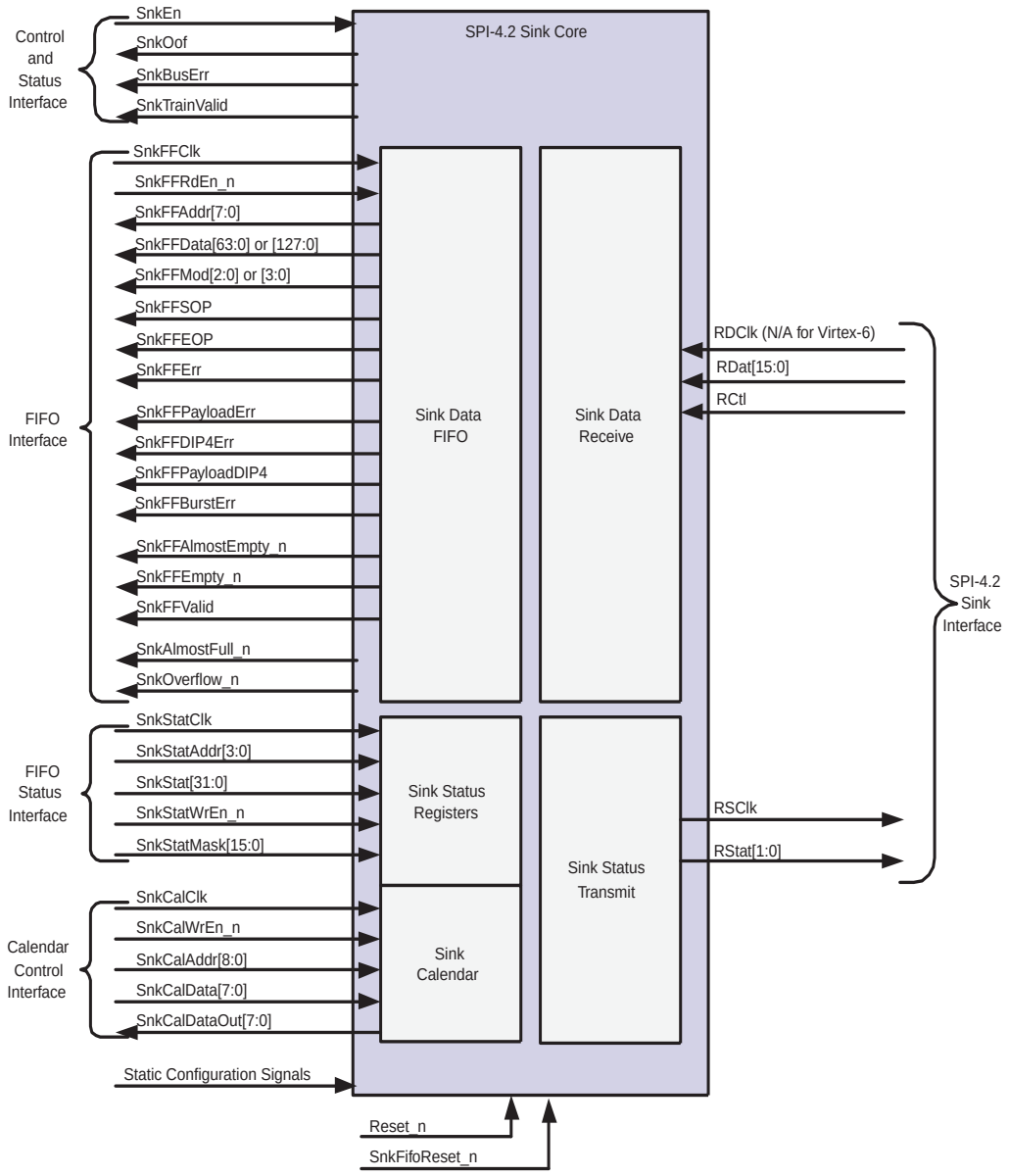


Figure 3-2: **Sink Core Block Diagram**

Sink SPI-4.2 Interface

The Sink SPI-4.2 Interface uses LVDS I/O buffers paired with embedded SERDES to receive 16-bit data words. The SPI-4.2 Sink core runs at the following frequencies:

- Up to 500 MHz LVDS DDR I/O (1 Gbps) in Virtex-4 FPGAs
- Above 500 MHz LVDS DDR I/O (1+ Gbps) in Virtex-6 and Virtex-5 FPGAs

The 16-bit data words received on the SPI-4.2 Interface are combined into 64-bit or 128-bit data words by the SPI-4.2 core. This allows the user interface to run at a quarter (64-bit interface) or an eighth (128-bit interface) of the data rate. For example, with an 800 Mbps SPI-4.2 data rate and a 64-bit interface, you can read data from the Sink core at 200 MHz. If 128-bit interface is used, you can read data from the Sink core at 100 MHz and maintain the same data rate.

The resulting data words are written into an asynchronous FIFO. The received 16-bit control words are stored out of band in the FIFO, along with the corresponding data word. The received control words that are not idle or training words can contain the information listed below.

- Start or continuation of the following packet
- Link address of the following packet
- End of the preceding packet
- Number of valid bytes in the last word of the preceding packet
- Error conditions in the preceding packet

In addition to receiving 16-bit data words, the SPI-4.2 interface also sends flow control data at 1/4 rate (or 1/8 rate) of its data interface. The 32-bit status (2-bit status for each channel) from the user interface is processed and formatted by the SPI-4.2 core to be transmitted on `RStat`. The signals of the Sink SPI-4.2 interface are defined in Table 3-1.

Table 3-1: **Sink SPI-4.2 Interface Signals**

Name	Direction	Clock Domain	Description
RDClk_P RDClk_N	Input	n/a	**SPI-4.2 Receive Data Clock** (LVDS). Source synchronous clock received with RDat and RCtl. The rising and falling edges of this clock (DDR) are used to clock RDat and RCtl. For Virtex-6 designs, the RDClk_P/N ports will not be an input to the Sink core since the clocking module is external to the core.
RDat_P[15:0] RDat_N[15:0]	Input	RDClk	**SPI-4.2 Receive Data Bus** (LVDS). The 16-bit data bus used to receive SPI-4.2 data and control information.
RCtl_P RCtl_N	Input	RDClk	**SPI-4.2 Receive Control** (LVDS). SPI-4.2 Interface signal that indicates whether data or control information is present on the RDat bus. When RCtl is deasserted, data is present on RDat. When RCtl is asserted, control information is present on RDat.

Table 3-1: **Sink SPI-4.2 Interface Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
RSClk	Output	n/a	**SPI-4.2 Receive Status Clock**. Source synchronous clock transmitted with RStat at 1/4 or 1/8 rate of the RDClk. The rate of the status clock is controlled by the static configuration signal RSClkDiv. The user can select this signal to be transmitted as LVTTL or LVDS. For Virtex-6 designs, only LVDS IO standard is supported.
RStat[1:0]	Output	RSClk	**SPI-4.2 Receive FIFO Status**. FlFO Status Channel flow control interface. The user can select this bus to be transmitted as LVTTL or LVDS. For Virtex-6 designs, only LVDS IO standard is supported.

Sink User Interface

The Sink User Interface includes all signals other than those on the SPI-4.2 Interface. With a 64-bit data interface, the user interface can operate at:

- 250 MHz in Virtex-4 FPGAs
- 312 MHz in Virtex-5 FPGAs
- Up to 350 Mhz in Virtex-6 FPGAs

The high performance logic on the Sink back-end enables the user interface to run at higher frequencies than the SPI-4.2 Interface. This is sometimes required if a large percentage of the traffic consists of small packets.

The User Interface is subdivided into five smaller interfaces. Each of the five interfaces are described in this section, and are listed below.

Control and Status Interface. The signals of this interface apply to the operation of the Sink core

FIFO Interface. The signals of this interface allow access to data received on the SPI-4.2 Interface

Status and Flow Control Interface. The signals of this interface send flow control information on the SPI-4.2 Interface

Static Configuration Interface. The signals of this interface configure the core.

Clocking Interface. The signals of this interface report the status of the clocks and include the general purpose clocks.

Sink Control and Status Interface

The Sink core control and status signals either control the operation of the entire Sink core, or provide status information that is not associated with a particular channel (port) or packet. The signals of this interface is defined in Table 3-2.

Table 3-2: **Sink Control and Status Signals**

Name	Direction	Clock Domain	Description
Reset_n	Input	n/a	**Reset**. Active low signal that asynchronously initializes internal flip-flops, registers, and counters. When Reset_n is asserted, the Sink core will go out-of-frame and the entire data path is cleared (including the FIFO). The Sink core will also assert SnkOof, and deassert SnkBusErr and SnkTrainValid. When Reset_n is asserted, the Sink core will transmit framing "11" on RStat and continue to drive RSClk. Following the deassertion of Reset_n, the Sink calendar should be programmed if the calendar is initialized in-circuit.
SnkFifoReset_n	Input	SnkFFClk	**Sink FIFO Reset.** Active low signal that enables the user to reset the Sink FIFO and the associated data path logic. This enables the FIFO to be cleared while remaining in-frame. Coming out of SnkFifoReset_n, the Sink core will discard all data on the SPI-4.2 interface until a valid SOP control word is received.

Table 3-2: **Sink Control and Status Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
SnkEn	Input	SnkStatClk	**Sink Enable.** Active high signal that enables the Sink core. When SnkEn is deasserted, the Sink core will go out-of-frame and will not store any additional data in the FIFO. The current contents of the FIFO remain intact. The Sink core will also assert SnkOof, and deassert SnkBusErr and SnkTrainValid. When SnkEn is deasserted, the Sink core will transmit framing "11" on RStat and continue to drive RSClk.
SnkIdelayCtlRst (optional)	Input	n/a	**Sink Dedicated IDELAYCTRL Reset**. Active high signal that asynchronously resets all the IDELAYCTRL primitives instantiated in the Sink core. If not used, the Reset_n signal resets all IDELAYCTRL primitives. This signal is present when the "Include IDELAYCTRL modules" option is selected.
SnkIdelayRefClk (optional)	Input	n/a	**Reference Clock.** User-supplied 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 SnkIdelayRefClk signal is present when option "Include IDELAYCTRL modules" is selected and must be connected to a user-supplied 200 MHz clock.
SnkIdelayCtlRdy	Input	RDClkDiv_GP	**IDELAYCTRL Ready.** Active high signal that indicates the IDELAY modules are calibrated. The SnkIdelayCtlRdy signal is present when option "Include IDELAYCTRL modules" is not selected. When option "Include IDELAYCTRL modules" is not selected, the IDELAYCTRLs must be instantiated in the wrapper by the user to calibrate the IDELAYs connected to RDat[15:0], RCtl, RDClk. Additionally, all the ready signals from these IDELAYCTRLs must be ANDed together to provide the signal that connect to SnkIdelayCtlRdy signal.
SnkOof	Output	SnkFFClk	**Sink Out-of-Frame**. Active high signal that indicates that the SPI-4.2 Sink block is not in-frame. This signal is asserted when SnkEn is deasserted or the Sink block loses synchronization with the data received on the SPI-4.2 Interface. This signal is deasserted once the Sink block reacquires synchronization with the received SPI-4.2 data.
SnkBusErr	Output	SnkFFClk	**Sink Bus Error.** Active high signal that indicates SPI-4.2 protocol violations or bus errors that are not associated with a particular packet. Information on the specific error condition that caused the SnkBusErr assertion is provided on SnkBusErrStat

Table 3-2: **Sink Control and Status Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
SnkBusErrStat[7:0]	Output	SnkFFClk	**Sink Bus Error Status**. Each bit of this bus corresponds to a specific Sink Bus Error condition and is asserted concurrently with SnkBusErr. The error conditions detected are reported as follows: SnkBusErrStat [0]: Minimum SOP spacing violation SnkBusErrStat [1]: Control word with EOP not preceded by a data word SnkBusErrStat [2]: Payload control word not followed by a data word SnkBusErrStat [3]: DIP4 error received during training or on idles SnkBusErrStat [4]: Reserved control words received SnkBusErrStat [5]: Non-zero address bits on control words received (except on payload and training control words) SnkBusErrStat [6:7]: Reserved bits (tied low) When dynamic phase alignment configuration is used, SnkBusErrStat can be used to monitor the alignment status. See Dynamic Phase Alignment (Virtex Devices Only), page 92
SnkTrainValid	Output	SnkFFClk	**Sink Training Valid.** Active high signal that indicates that a valid training pattern has been received. This signal is asserted for the duration of the training pattern (20 SPI-4.2 bus clock cycles or 5 RDClkDiv_GP clock cycles), if the training pattern received is successfully decoded.

Sink FIFO Interface

The Sink FIFO Interface signals allow access to the data (received on the SPI-4.2 Interface) that is stored in the FIFO. Table 3-3 describes the signals on this interface.

Table 3-3: **Sink FIFO Signals**

Name	Direction	Description
SnkFFClk	Input	**Sink FIFO Clock**. All Sink FIFO Interface signals are synchronous to the rising edge of this clock.
SnkFFRdEn_n	Input	**Sink FIFO Read-Enable**. When detected low at the rising edge of SnkFFClk, data and status information is available from the FIFO on the next rising edge of SnkFFClk.
SnkFFAddr[7:0]	Output	**Sink FIFO Channel Address**. Channel number associated with the data on SnkFFData.
SnkFFData[63:0] or SnkFFData[127:0]	Output	**Sink FIFO Data Out**. The Sink FIFO data bus. Bit 0 is the LSB. The core can be configured to have a 64-bit or 128-bit Interface. The 128-bit interface enables the user to run at half the clock rate required for a 64-bit interface.
SnkFFMod[2:0] or SnkFFMod[3:0]	Output	**Sink FIFO Modulo**. This signal indicates which bytes on the SnkFFData bus are valid when the SnkFFEOP signal is asserted. SnkFFMod[2:0] is used with a 64-bit interface. SnkFFMod[3:0] is used with a 128-bit interface.

Table 3-3: **Sink FIFO Signals *(Cont'd)***

Name	Direction	Description
SnkFFSOP	Output	**Sink FIFO Start of Packet**. Active high signal indicating that the start of a packet is being read out of the Sink FIFO.
SnkFFEOP	Output	**Sink FIFO End of Packet**. Active high signal indicating that the end of a packet is being read out of the Sink FIFO.
SnkFFErr	Output	**Sink FIFO Error**. Active high signal indicating that the current packet is terminated with an EOP abort condition. This signal is only asserted when SnkFFEOP is asserted.
SnkFFEmpty_n	Output	**Sink FIFO Empty**. Active low signal indicating that the Sink FIFO is empty. No data can be read until this signal is deasserted. This signal is asserted with the last data word read out of the FIFO.
SnkFFAlmostEmpty_n	Output	**Sink FIFO Almost Empty**. Active low signal indicating that one word remains in the FIFO, and the user should deassert the read enable signal on the next clock cycle. The user's read logic should evaluate the SnkFFEmpty_n signal to verify that there is no data in the FIFO in case an additional word was simultaneously written into the FIFO. An example of the behavior of this interface signal is provided with the SPI-4.2 core in the Design Example (see the pl4_fifo_loopback_read.v/vhd file.)
SnkFFValid	Output	**Sink FIFO Read Valid**. Active high signal indicating that the information on SnkFFData, SnkFFAddr, SnkFFSOP, SnkFFEOP, SnkFFBurstErr, SnkFFMod, SnkFFErr, SnkFFDIP4Err, SnkFFPayloadDIP4, and SnkFFPayloadErr is valid.
SnkFFDIP4Err	Output	**Sink FIFO DIP-4 Error.** Active high signal indicating that a DIP-4 parity error was detected with the SPI-4.2 control word ending a packet or burst of data. This signal is asserted at the end of that packet or burst of data.
SnkFFPayloadDIP4	Output	**Sink FIFO Payload DIP4 Error.** Active high signal indicating that a DIP-4 parity error was detected with the SPI-4.2 control word starting a packet or burst of data. This signal is asserted at the end of that packet or burst of data.
SnkFFBurstErr	Output	**Sink FIFO Burst Error.** Active high signal indicating that the Sink core has received data that was terminated on a non-credit boundary without an EOP. SnkFFBurstErr may be used by the user's logic to indicate missing EOPs, or incorrectly terminated bursts. In this case the Sink core does not assert SnkFFEOP or SnkFFErr.
SnkFFPayloadErr	Output	**Sink FIFO Payload Error.** Active high signal indicating that the received data was not preceded by a valid payload control word. Since it is not clear what the packet Address and SOP should be, it is flagged as an error. This is asserted with each data word coming out of the FIFO, and will remain asserted until a valid payload control word is followed by data.
SnkAlmostFull_n	Output	**Sink Almost Full**. Active low signal indicating that the Sink core is approaching full (as defined by the parameter SnkAFThresAssert), and that immediate action should be taken to prevent overflow.
SnkOverflow_n	Output	**Sink Overflow.** Active low signal indicating that the Sink core has overflowed and is in an error condition. Data will be lost if SnkOverflow_n is asserted, since no data is written into the FIFO when the overflow signal is asserted.

Sink Status and Flow Control Interface (Calendar Control and Status FIFO)

The Sink Status and Flow Control interface enables you to send flow control data on the SPI-4.2 Interface. The status information is sent based on the channel order and channel frequency defined in the programmable calendar. The calendar interface and status FIFO interface signals are defined in Table 3-4 and Table 3-5.

Table 3-4: **Sink Calendar Control Signals**

Name	Direction	Clock Domain	Description
SnkCalClk	Input	n/a	**Sink Calendar Clock.** All Sink calendar signals are synchronous to this clock.
SnkCalWrEn_n	Input	SnkCalClk	**Sink Calendar Write Enable.** Active low signal indicating the Sink Calendar is written with the data on the SnkCalData bus on the rising edge of SnkCalClk. When the signal is deasserted, the Sink Calendar data can be read on SnkCalDataOut.
SnkCalAddr[8:0]	Input	SnkCalClk	**Sink Calendar Address**. Indicates the calendar address to which the data on SnkCalData is written. When SnkCalWrEn_n is deasserted, this bus indicates the calendar address from which the channel number on SnkCalDataOut is driven.
SnkCalData[7:0]	Input	SnkCalClk	**Sink Calendar Data**. Contains the channel number to write into the calendar buffer when SnkCalWrEn_n is enabled. The channel numbers written into the calendar indicate the order that status is sent on RStat.
SnkCalDataOut[7:0]	Output	SnkCalClk	**Sink Calendar Data Output**. Contains the channel number read from the calendar buffer when SnkCalWrEn_n is disabled. The channel numbers read from the calendar indicate the order that status is sent on RStat.

Table 3-5: **Sink Status FIFO Signals**

Name	Direction	Clock Domain	Description
SnkStatClk	Input	N/A	**Sink Status Clock.** All Sink Status write signals are synchronous to this clock.
SnkStat[31:0]	Input	SnkStatClk	**Sink Status Bus.** This 32-bit bus is used to write status information into the Status FIFO. The user can write the status for 16 channels each clock cycle. The 16-channel status that are accessed simultaneously are grouped in the following manner: channels 15 to 0, channels 31 to 16, channels 47 to 32, . . . , channels 255 to 239.
SnkDIP2ErrRequest	Input	SnkStatClk	**Sink DIP2 Error Request.** Active high signal that requests an incorrect DIP-2 to be sent out of the RStat bus. When this signal is asserted, Sink Status FIFO responds by inverting the next DIP2 value that it transmits.

Table 3-5: **Sink Status FIFO Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
SnkStatAddr[3:0]	Input	SnkStatClk	**Sink Status Address bus**. The Sink Status Address determines the group of 16-channel status that SnkStat updates. Bank 0: SnkStatAddr=0, channels 15 to 0 Bank 1: SnkStatAddr=1, channels 31 to 16 Bank 2: SnkStatAddr=2, channels 47 to 32 . . . Bank 15: SnkStatAddr=15, channels 255 to 239
SnkStatWr_n	Input	SnkStatClk	**Sink Status Write**. Active low signal that qualifies the SnkStatMask signal. When SnkStatWr_n is asserted (active low), status for the different channels is updated. When SnkStatWr_n is deasserted (active high), SnkStat input is ignored.
SnkStatMask[15:0]	Input	SnkStatClk	**Sink Status Mask Bus**. Determines if the 2-bit status among the corresponding group of 16 channels of status on SnkStat (being addressed by SnkStatAddr) will be updated when SnkStatWr_n is asserted (active low): SnkStatMask[x] = 1, status for channel (x+(SnkStatAddr*16)) will be updated. SnkStatMask[y] = 0, status for channel (y+(SnkStatAddr*16)) will not be updated. For example, if SnkStatMask[15] = 1 and SnkStatAddr = 1, then SnkStat[31:30] = 00 will overwrite the current status on channel 31. If SnkStatMask is all zeros, none of the sixteen 2-bit status values will be updated. If SnkStatMask is all ones, all sixteen of the 2-bit status values will be updated.

Sink Static Configuration Interface

Sink Static Configuration signals are inputs to the core that are statically driven by setting them to a constant value in the top-level wrapper file. The SPI-4.2 release includes a wrapper file that has the static configuration signals connected to the values selected in the CORE Generator GUI. The user can customize these signals using the GUI.

Two of the Sink Static Configuration signals can be changed in circuit; they are static registers for `SnkCalendar_M` and `SnkCalendar_Len` and are synchronous to `SnkStatClk`. To change these parameters while the core is operational, SnkEn must first be deasserted.

All Sink static configuration signals can also be changed in-circuit when the core is *not* in operation (disabled and in reset state).

The following steps are recommended when changing static configuration signals:

1. Disable the sink core (SnkEn signal).
2. Assert core reset (Reset_n = 0).
3. Change the desired static configuration signals.
4. Deassert reset (Reset_n=1).
5. Wait at least 10 clock cycles of RDClkDiv_GP for the sink static configuration signals to settle and propagate to the Sink core's logic.

6. Enable the core and wait for the core to achieve synchronization, then continue normal operation.

If the configuration signal is set to an illegal number, the core will automatically set it to the minimum value. The Sink Static Configuration signals are defined in Table 3-6.

Table 3-6: **Sink Static Configuration Signals**

Name	Direction	Range	Description
NumDip4Errors[3:0]	Static Input	1-15 Value of 0 gets set to 1.	**Number of DIP-4 Errors**. The Sink Interface will go out-of-frame (assert SnkOof) and will stop accepting data from the SPI-4.2 bus after receiving NumDip4Errors consecutive DIP-4 errors.
NumTrainSequences[3:0]	Static Input	1-15 Value of 0 gets set to 1.	**Number of Complete Training Sequences**. A complete training pattern consists of 10 training control words and 10 training data words. The Sink interface requires NumTrainSequences consecutive training patterns before going in-frame (deasserting SnkOof) and accepting data from the SPI-4.2 bus.
SnkCalendar_M[7:0]	Input	0-255 (effective range 1-256)	**Sink Calendar Period**. The SnkCalendar_M parameter sets the number of repetitions of the calendar sequence before the DIP-2 parity and framing words are inserted. The core implements this parameter as a static register synchronous to SnkStatClk, and it can be updated in circuit by first deasserting SnkEn. Note that the Sink Calendar Period equals SnkCalendar_M + 1. For example, if SnkCalendar_M=22, the Sink Calendar Period will be equal to 23.
SnkCalendar_Len[8:0]	Input	0-511 (effective range 1-512)	**Sink Calendar Length**. Sets the length of the calendar sequence. The core implements this parameter as a static register synchronous to SnkStatClk, and it can be updated in circuit by first deasserting SnkEn. Note that the Sink Calendar Length equals SnkCalendar_Len + 1. For example, if SnkCalendar_Len=15, the Sink Calendar Length will be equal to 16.

Table 3-6: **Sink Static Configuration Signals *(Cont'd)***

Name	Direction	Range	Description
SnkAFThresAssert[8:0]	Static Input	1-508 Values less than 1 get set to 1. Values greater than 508 get set to 508.	**Sink Almost Full Threshold Assert**. Defines the minimum number of empty FIFO locations that exist when SnkAlmostFull_n is asserted. Note that the assert threshold must be less than or equal to the negate threshold (SnkAFThresNegate). When SnkAlmostFull_n is asserted, the core initiates the flow control mechanism selected by the parameter FifoAFMode. The FifoAFMode defines when the interface stops sending valid FIFO status levels and begins sending flow control information on RStat. This indicates to the transmitting device that the core is almost full and additional data cannot be sent.
SnkAFThresNegate[8:0]	Static Input	SnkAFThresAssert to 508 Values less than SnkAFThresAssert get set to SnkAFThresAssert. Values greater than 508 get set to 508.	**Sink Almost Full Threshold Negate**. Defines the minimum number of empty FIFO locations that exist when SnkAlmostFull_n is deasserted. Note that the negate threshold must be greater or equal to the assert threshold (SnkAFThresAssert). When SnkAlmostFull_n is deasserted, the core stops sending flow control (deasserts SnkAlmostFull_n) and resumes transmission of valid FIFO status levels. This indicates to the transmitting device that additional data can be sent.

Table 3-6: **Sink Static Configuration Signals *(Cont'd)***

Name	Direction	Range	Description
RSClkDiv	Static Input	n/a	**Sink Status Clock Divide**. This static input is used to determine if the RSClk is 1/4 of the data rate, which is compliant with the OIF specification, or 1/8 of the data rate, which is required by some PHY ASSPs: 0: RSClkDiv = 1/4 rate (default value) 1: RSClkDiv = 1/8 rate
RSClkPhase	Static Input	n/a	**Sink Status Clock Phase**. This static input determines whether the *FIFO Status* Channel data (RStat[1:0]) changes on the rising edge of RSClk or the falling edge of RSClk: 0: RSClkPhase = rising edge of RSClk (default value) 1: RSClkPhase = falling edge of RSClk
FifoAFMode[1:0]	Static Input	n/a	**Sink Almost Full Mode**. Selects the mode of operation for the Sink interface when the Sink core reaches the Almost Full threshold (SnkAFThresAssert). If FifoAFMode is set to "00," the Sink interface goes out-of-frame when the core is almost full, and the Sink Status logic sends the framing sequence "11" until Sink core is not almost full. If FifoAFMode is set to "01," the Sink interface remains in-frame (SnkOof deasserted), and the Sink Status logic sends satisfied "10" on all channels until SnkAlmostFull_n is deasserted. If FifoAFMode is set to "10" or "11," the Sink interface will remain in-frame (SnkOof deasserted), and the Sink Status logic continues to drive out the status information (*i.e.*, continues in normal operation). In this case, take immediate action to prevent overflow and loss of data.

Sink Clocking Interface for Virtex-5 and Virtex-4

A list of the Sink Clocks and their description is given in Table 3-7, the DCM reset and clock status signals are defined in Table 3-8.

Table 3-7: **Sink Clock Signals for Virtex-5 and Virtex-4**

Clock Pins	Direction	Description	Minimum Frequency		Max Frequency
			Dynamic Phase Alignment	Static Phase Alignment	
RDClkDiv_GP	Output (User Interface)	**RDClkDiv General Purpose**: This clock is half the frequency of RDClk. It is used for clocking the internal logic of the core and is routed to the User Interface for use by the user's logic.	Virtex-5: 110 Mhz Virtex-4: 110 MHz	DCM or BUFR Minimum	Virtex-5: 250+ MHz Virtex-4: 250 MHz
RDClk0_GP	Output (User Interface)	**RDClk0 General Purpose**: This clock is the full Rate Receive Data Clock.	Virtex-5: 220 Mhz Virtex-4: 220 MHz	DCM or BUFR Minimum	Virtex-5: 500+ MHz Virtex-4: 500 MHz

Table 3-8: **Sink Clock Status Signals for Virtex-5 and Virtex-4**

Name	Direction	Clock Domain	Description
DCMReset_RDClk	Input	N/A	Reset of the RDClk DCM. If DCM is not used, tie this signal to "0".
Locked_RDClk	Output	N/A	Locked status of the RDClk DCM. This signal is tied to "1" internally when DCM is not used.
DCMLost_RDClk	Output	N/A	Indicates RDClk input has stopped (status bit one of RDClk DCM). This signal is tied to "0" internally when DCM is not used.
SnkClksRdy	Output	N/A	Indicates all Sink core clocks are ready for use.

Sink Clocking Interface for Virtex-6 Devices

For Virtex-6 devices, the Sink core's clocking module is not embedded in the core. So, the clock interface to the core consists of input clocks required by the internal logic and output general purpose clock signals that can be used for user logic. A list of the Sink clocks and their description is given in Table 3-9 and clock status signals are defined in Table 3-10.

Table 3-9: **Sink Clock Signals for Virtex-6 Devices**

Name	Direction	Description	Minimum Frequency		Maximum Frequency
			Dynamic Phase Alignment	Static Phase Alignment	
RDClk0_User	Input (User Interface)	**RDClk0 User :** This clock is the full Rate Receive Data Clock.	220 Mhz	MMCM or BUFR Minimum	500+ MHz
RDClkDiv_User	Input (User Interface)	**RDClkDiv User:** This clock is half rate of the RDClk. It is used for clocking the internal logic of the core. This clock runs at half the frequency of RDClk0_User frequency with zero degree phase offset.	110 Mhz	MMCM or BUFR Minimum	250+ MHz
RDClk0_GP	Output (User Interface)	**RDClk0 General Purpose**: Same signal as RDClk0_User.	220 Mhz	MMCM or BUFR Minimum	500+ MHz
RDClkDiv_GP	Output (User Interface)	**RDClkDiv General Purpose**: Same signal as RDClkDiv_User.	110 Mhz	MMCM or BUFR Minimum	250+ MHz

Table 3-10: **Sink Clock Status Signals for Virtex-6 Devices**

Name	Direction	Clock Domain	Description
SnkClksRdy_User	Input	N/A	Indicates all Sink core clocks are ready for use.

Source Core Interfaces

The Source core has five functional modules:

- Source Data FIFO
- Source Data Transmit
- Source Status Registers
- Source Calendar
- Source Status Receive

The Source core has the following interfaces:

- Source SPI-4.2 Interface
- Source User Interface
  - Source Control and Status Interface
  - Source FIFO Interface

- Source Status and Flow Control Interface
  - Calendar Control Interface
  - Status FIFO Interface
- Source Configuration Signals Interface
- Source Clocking Interface

The functional modules and signals which comprise the different interfaces are shown in Figure 3-3 and defined in tables in the following sections.

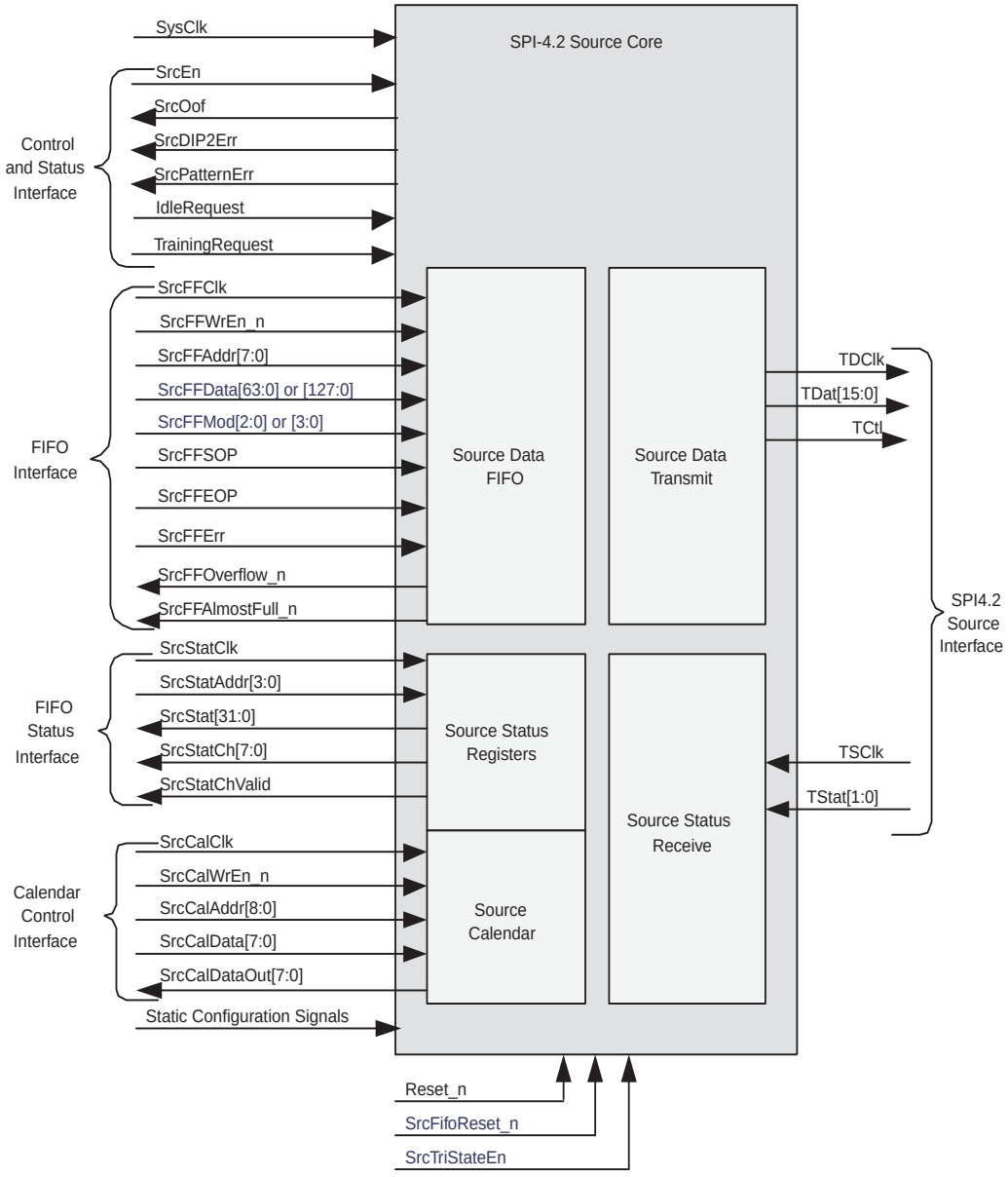


Figure 3-3: **Source Core Block Diagram and I/O Interface Signals**

Source SPI-4.2 Interface

The SPI-4.2 interface uses LVDS I/O buffers paired with embedded SERDES to transmit 16-bit data words. The SPI-4.2 Source core runs at frequencies up to:

- Above 500 MHz LVDS DDR I/O (1+ Gbps) in Virtex-6 and Virtex-5 FPGAs
- 500 MHz LVDS DDR I/O (1 Gbps) in Virtex-4 FPGAs

The data words received on the user interface and the out-of-band control words are multiplexed onto the SPI-4.2 16-bit databus. The Source core supports a 64-bit and 128-bit user interface. This allows the user interface to run at a quarter (64-bit interface) or an eighth (128-bit interface) of the data rate. For example, an 800 Mbps SPI-4.2 data rate and a 64-bit interface can write data into the Source core at 200 MHz. If a 128-bit interface is used data is written into the Source core at 100 MHz and maintain the same data rate.

In addition to transmitting 16-bit data words, the SPI-4.2 interface also receives flow control data at 1/4 rate of its data interface. The 2-bit status received can be presented in two interfaces: transparent and addressable.

The signals on the SPI-4.2 Source Interface are defined in Table 3-11.

Table 3-11: **Source SPI-4.2 Interface Signals**

Name	Direction	Clock Domain	Description
TDClk_P TDClk_N	Output	n/a	**SPI-4.2 Transmit Data Clock** (LVDS): Source synchronous clock transmitted with TDat. The rising and falling edges of this clock (DDR) are used to clock TDat and TCtl.
TDat_P[15:0] TDat_N[15:0]	Output	TDClk	**SPI-4.2 Transmit Data Bus** (LVDS): The 16-bit data bus is used to transmit SPI-4.2 data and control information.
TCtl_P TCtl_N	Output	TDClk	**SPI-4.2 Transmit Control** (LVDS): SPI-4.2 Interface signal that defines whether data or control information is present on the TDat bus. When TCtl is Low, data is present on TDat. When TCtl is High, control information is present on TDat.
TSClk	Input	n/a	**SPI-4.2 Transmit Status Clock**: Source synchronous clock that is received by the Source core with TStat at 1/4 rate (or 1/8 rate) of TDClk. For Virtex-4 and Virtex-5, the user can select this signal to be transmitted as LVTTL or LVDS. For Virtex-6 designs, only LVDS IO standard is supported, and the TSClk_P/N ports will not be an input to the source core since the clocking module is external to the core.
TStat[1:0]	Input	TSClk	**SPI-4.2 Transmit FIFO Status**: FIFO-Status-Channel flow control interface. For Virtex-4 and Virtex-5 designs, the user can select this bus to be transmitted as LVTTL or LVDS. For Virtex-6 designs, only LVDS IO standard is supported.

Source User Interface

The Source User Interface includes all signals other than those on the SPI-4.2 Interface. The user interface can operate up to:

- 350 Mhz in Virtex-6 FPGAs
- 312 MHz in Virtex-5 FPGAs
- 250 MHz in Virtex-4 FPGAs

The high performance logic on the Source back-end enables the user interface to run at higher frequencies than the SPI-4.2 Interface. This is sometimes required if a large percentage of the traffic consists of small packets.

The user interface is subdivided into 5 smaller interfaces. Each of these signal types are presented in detail below.

- **Control and Status Interface**: Apply to the operation of the Sink core.
- **FIFO Interface**: Allows access data received on the SPI-4.2 Interface.
- **Status and Flow Control Interface**: Send flow control information on the SPI-4.2 Interface.
- **Static Configuration Interface**: Allows configuration of the core.
- **Clocking Interface**: Report the status of the clocks and include the general purpose clocks.

Source Control and Status Interface

The Source Control and Status signals either control the operation of the entire Source core or provide status information that is not associated with a particular channel (port) or packet. Table 3-12 defines the signals of this interface.

Table 3-12: **Source Control and Status Signals**

Name	Direction	Clock Domain	Description
Reset_n	Input	n/a	**Reset_n**. Active low, asynchronous control signal that enables the user to restart the entire Source core. This means that the core will go out-of-frame. While Reset_n is asserted, the Source core transmits idles cycles on TDat. Coming out of Reset_n, the Source core transmits training patterns. Following the release of Reset_n, the Source Calendar should be programmed if the calendar is to be initialized in-circuit.
SrcFifoReset_n	Input	SrcFFClk	**SrcFifoReset_n**. Active low control signal that enables the user to reset the Source FIFO and the associated data path logic. This enables the FIFO to be cleared while remaining in-frame. Upon Source FIFO Reset, the Source core sends idle cycles until the user writes data into the FIFO.
SrcEn	Input	SrcStatClk	**Source Enable**. Active high signal that enables the Source core. When SrcEn is deasserted, the Source core will not store or verify received status information. The Source core will also assert SrcOof, and deassert SrcDIP2Err, SrcStatFrameErr and SrcPatternErr. When SrcEn is deasserted, the Source core will transmit training patterns on TDat.

Table 3-12: **Source Control and Status Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
SrcOof	Output	SrcFFClk	**Source Out-of-Frame:** Active high signal indicating that the SPI-4.2 Source block is not in-frame. This signal is asserted when the Source block has lost synchronization on the transmit FIFO status interface. This is caused by the receipt of consecutive DIP-2 parity errors (determined by the parameter NumDip2Errors), invalid received status frame sequence (of four consecutive frame words "11"), or when SrcEn is deasserted. This signal is deasserted once the Source block reacquires synchronization with the SPI-4.2 transmit Status Channel. Synchronization occurs when consecutive valid DIP2 words (determined by the Static Configuration signal NumDip2Matches) are received and SrcEn is asserted.
SrcDIP2Err	Output	SrcFFClk	**Source DIP-2 Parity Error.** Active high signal indicating that a DIP-2 parity error was detected on TStat. This signal is asserted for one clock cycle each time a parity error is detected.
SrcStatFrameErr	Output	SrcFFClk	**Source Status Frame Error**: When this signal is asserted (active high), it indicates that a non "11" frame word was received after DIP2 on TStat. This signal is asserted for one clock cycle each time a frame word error is detected.
SrcPatternErr	Output	SrcFFClk	**Source Data Pattern Error.** Active high signal indicating that the data pattern written into the Source FIFO is illegal. Illegal patterns include the following: Burst of data terminating on a non-credit boundary (not a multiple of 16 bytes) with no EOP Non-zero value on SrcFFMod when SrcFFEOP is deasserted This signal is asserted for one clock cycle each time an illegal data pattern is written into the Source FIFO.
IdleRequest	Input	SrcFFClk	**Idle Request**. Active high signal that requests idle control words be sent out of the Source SPI-4.2 interface. The Source core responds by sending out idle control words at the next burst boundary. This signal overrides normal SPI-4.2 data transfer requests, but it does not override training sequence requests (TrainingRequest). Activating the request for idle cycles does not affect the Source FIFO contents or the user side operation.
TrainingRequest	Input	SrcFFClk	**Training Pattern Request**. Active high signal that requests training patterns be sent out of the Source SPI-4.2 interface. The Source core responds by sending out training patterns at the next burst boundary. This signal overrides idle requests (IdleRequest) and normal SPI-4.2 data transfers. Activating the request for training cycles does not affect the Source FIFO contents or the user side operation.

Table 3-12: **Source Control and Status Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
SrcTriStateEn	Input	SrcFFClk	**SrcTriStateEn**. Active high control signal that enables the user to tri-state the IOB drivers for the following Source core outputs: TDClk, TDat[15:0], and TCtl. When SrcTriStateEn=0 the outputs are not tri-stated. When SrcTriStateEn=1 the outputs are tri-stated. Default setting for this signal is disabled (SrcTriStateEn=0.)
SrcOofOverride	Input	SrcFFClk	**Source Out-of-Frame Override**: When this signal is asserted, the Source core behaves as if in-frame, and sends data on TDat regardless of the status received on TStat. This signal is used for system testing and debugging.

Source FIFO Interface

The Source FIFO Interface signals write data into the FIFO to be transmitted on the SPI-4.2 Interface. Table 3-13 defines the signals of this interface.

Table 3-13: **Source FIFO Signals**

Name	Direction	Clock Domain	Description
SrcFFClk	Input	n/a	**Source FIFO Clock**. All Source FIFO Interface signals are synchronous to the rising edge of this clock.
SrcFFWrEn_n	Input	SrcFFClk	**Source FIFO Write-Enable**. When asserted (active low) at the rising edge of SrcFFClk, data and packet information is written into the FIFO.
SrcFFAddr[7:0]	Input	SrcFFClk	**Source FIFO Channel Address**. Channel number associated with the data on SrcFFData.
SrcFFData[63:0] or SrcFFData[127:0]	Input	SrcFFClk	**Source FIFO Data**. The Source FIFO data bus. Bit 0 is the LSB. The core can be configured to have a 64-bit or a 128-bit interface. The 128-bit interface enables the user to run at half the clock rate required for a 64-bit interface.
SrcFFMod[2:0] or SrcFFMod[3:0]	Input	SrcFFClk	**Source FIFO Modulo:** Indicates which bytes on the SrcFFData bus are valid when the SrcFFEOP or SrcFFErr signal is asserted. When SrcFFEOP is deasserted, SrcFFMod should always be zero. SrcFFMod[2:0] is used with a 64-bit interface. SrcFFMod[3:0] is used with a 128-bit interface.
SrcFFSOP	Input	SrcFFClk	**Source FIFO Start of Packet**. Active high signal indicating that the start of a packet is being written into the Source FIFO.
SrcFFEOP	Input	SrcFFClk	**Source FIFO End of Packet**. When asserted (active high), this signal indicates that the end of a packet is being written into the Source FIFO. May be concurrent with SrcFFSOP.
SrcFFErr	Input	SrcFFClk	**Source FIFO Error**. When asserted (active high) simultaneously with the SrcFFEOP flag, the current packet written into the FIFO contains an error. This causes an EOP abort to be sent on the SPI-4.2 Interface. SrcFFErr can be used in combination with SrcFFEOP to insert erroneous DIP-4 values for testing purposes. When SrcFFErr is asserted and SrcFFEOP is not asserted, the core inserts an EOP (1 or 2 bytes depending on the SrcFFMod value) with an erroneous DIP-4 value. The erroneous DIP4 value is an inversion of the correctly calculated value.
SrcFFAlmostFull_n	Output	SrcFFClk	**Source FIFO Almost Full**. Active low signal indicating that the FIFO is approaching full, and no more data should be written.
SrcFFOverflow_n	Output	SrcFFClk	**Source FIFO Overflow.** Active low signal indicating that the FIFO has overflowed and is in an error condition. No more data can be written until it is deasserted. SrcFFWrEn_n is ignored if SrcFFOverflow_n is asserted.

Source Status and Flow Control Interface (Calendar Control and Status FIFO)

The Source Status and Flow Control Interface allows flow control data to be received from the SPI-4.2 interface. Status information is received based on the channel order and frequency defined in the programmable calendar. The calendar interface and Status FIFO signals are defined in Table 3-14 and Table 3-15.

Table 3-14: **Source Calendar Control Signals**

Name	Direction	Clock Domain	Description
SrcCalClk	Input	n/a	**Source Calendar Clock.** All Source calendar signals are synchronous to this clock.
SrcCalWrEn_n	Input	SrcCalClk	**Source Calendar Write Enable.** When this signal is asserted (Active Low), the Source Calendar is loaded with the data on the SrcCalData bus on the rising edge of SrcCalClk.
SrcCalAddr[8:0]	Input	SrcCalClk	**Source Calendar Address.** When asserted, this bus indicates the calendar address to which the data on SrcCalData is written. When SrcCalWrEn_n is deasserted, this bus indicates the calendar address from which the data on SrcCalDataOut is driven.
SrcCalData[7:0]	Input	SrcCalClk	**Source Calendar Data.** This bus contains the channel number to write into the calendar buffer when SrcCalWrEn_n is enabled. The channel numbers written into the calendar indicate the order that status is updated on the SrcStat bus.
SrcCalDataOut[7:0]	Output	SrcCalClk	**Source Calendar Data Output.** This Source Calendar Data Output bus contains the channel number that is read from the calendar buffer when SrcCalWrEn_n is disabled. The channel numbers read from the calendar indicates the order that status is updated on SrcStat bus.

Table 3-15: **Source Status FIFO Signals**

Name	Direction	Clock Domain	Description
SrcStatClk (Addressable I/F Only)	Input	n/a	**Source Status Clock.** For the *Addressable Interface*, all Source Status read signals are synchronous to this clock. For the *Transparent Interface*, this clock signal is not present. For this interface, all signals are synchronous to TSClk_GP.
SrcStat[31:0] (Addressable I/F Only) SrcStat[1:0] (Transparent I/F Only)	Output	SrcStatClk (Addressable I/F only) TSClk_GP (Transparent I/F only)	**Source Status**. For the *Addressable Interface*, the 32-bit Source Status bus is the dedicated 16-channel interface. The user can read the status for 16-channels each clock cycle. The 16-channel status that are accessed simultaneously are grouped in the following manner: channel 15 to 0, channel 31 to 16, channel 47 to 32, ..., channel 255 to 240. For the *Transparent Interface,* this Source Status bus is two bits wide and represents the last status received.

Table 3-15: **Source Status FIFO Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
SrcStatAddr[3:0] (Addressable I/F Only)	Input	SrcStatClk	**Source Status Address**: For the *Addressable Interface*, the Source Status Address determines which group of 16-channels gets its status driven onto SrcStat on the following clock cycle. The address bus is associated with banks of channels as follows: Bank 0: SrcStatAddr=0 channel 15-0 Bank 1: SrcStatAddr=1, channel 31-16 Bank 2: SrcStatAddr=2, channel 47-32 ... Bank 15: SrcStatAddr=15 channel 255-240 For the *Transparent Interface*, this signal is not present.
SrcStatCh[7:0]	Output	TSClk_GP	**Source Status Channel**. An 8-bit bus containing the channel address that is being updated on the SrcStatAddr bus in the current clock cycle.
SrcStatChValid	Output	TSClk_GP	**Source Status Channel Valid:** When asserted, indicates that the value on SrcStatCh is valid. When the core is processing DIP-2 or frame words, SrcStatChValid is deasserted. Note that a transition of the SrcStatChValid from 0 to 1 indicates that the core has started a new calendar sequence.

Source Static Configuration Interface

Source Static Configuration Interface signals are inputs to the core that are statically driven by setting them to a constant value in the top-level wrapper file. The SPI-4.2 release includes a wrapper file that has the static configuration signals connected to the values selected in the CORE Generator GUI. Customization of these signals is done using the GUI.

Three of the Source Static Configuration signals can be changed in circuit. There are static registers for `SrcBurstLen` (synchronous to `SrcFFClk`), and `SrcCalendar_M` and `SrcCalendar_Len` (synchronous to `SrcStatClk`.) To change these parameters while the core is operational, first deassert `SrcEn`.

All Source Static Configuration signals can also be changed in-circuit when the core is *not* in operation (disabled and in reset state).

The following steps are recommended when changing static configuration signals:

1. Disable the source core (SrcEn signal).
2. Assert core reset (Reset_n = 0).
3. Change the desired static configuration signals.
4. Deassert reset (Reset_n=1).
5. Wait at least 10 clock cycles of SysClkDiv_GP for the source static configuration signals to settle and propagate to the Source core's logic.
6. Enable the core and wait for the core to achieve synchronization, then continue normal operation.

If the configuration signal is set to an illegal number, the core will automatically set it to the minimum value. The Source Static Configuration signals are defined in Table 3-16.

Table 3-16: **Source Static Configuration Signals**

Name	Direction	Range	Description
SrcBurstMode	Static Input	0 or 1	**Source Burst Mode**. When set to zero, the Source core transmits data in the FIFO if the data is terminated by an EOP, or if there is a a burst of data greater than 1 credit followed by a no write (the burst of data must be multiples of credits). When SrcBurstMode is set to1, the Source core only transmits data that is terminated by an EOP or when there is data in the FIFO equal to the maximum burst length defined by SrcBurstLen, or when the channel address changes.
SrcBurstLen[9:0]	Input	1-1023 (SrcBurstMode = 1) 1–256 (SrcBurstMode = 0)	**Source Burst Length**. The Source core automatically segments packets larger than this parameter into multiple bursts, which are each SrcBurstLen in length. This parameter is defined in credits (16 bytes). For SrcBurstLen is greater than or equal to 256, the write data rate has to be faster or equal to the read data rate for the core to send an unsegmented data burst. The core implements this parameter as a static register synchronous to SrcFFClk, and it can be updated in circuit by first deasserting SrcEn.
SrcAFThresAssert[8:0]	Static Input	If SrcBurstMode = 0 1–508 Values less than 1 are set to 1 Values greater than 508 are set to 508 If SrcBurstMode = 1 1–508	**Source Almost Full Threshold Assert**. This parameter specifies the minimum number of empty FIFO locations that exist in the Source FIFO before the Almost Full signal (SrcFFAlmostFull_n) is asserted. SrcAFThresNegate is greater than or equal to SrcAFThresAssert.
SrcAFThresNegate[8:0]	Static Input	SrcAFThresAssert to 508 Values less than SrcAFThresAssert are set to SrcAFThresAssert Values greater than 508 are set to 508	**Source Almost Full Threshold Negate**. This parameter specifies the minimum number of empty FIFO locations that exist in the Source FIFO before the Almost Full signal (SrcFFAlmostFull_n) is deasserted. If SrcBurstMode=0, then: SrcAFThresNegate is greater than or equal to SrcAFThresAssert. If SrcBurstMode=1, then: SrcAFThresNegate ≥ SrcAFThresAssert

Table 3-16: **Source Static Configuration Signals *(Cont'd)***

Name	Direction	Range	Description
SrcCalendar_M[7:0]	Input	0-255 (effective range 1–256)	**Source Calendar Period**. This parameter sets the number of repetitions of the calendar sequence before the DIP-2 parity and framing words are received. The Source core implements this parameter as a static register synchronous to SrcStatClk, and it can be updated in circuit by first deasserting SrcEn. Note that the Source Calendar Period equals SrcCalendar_M + 1. For example, if SrcCalendar_M=22, the Source Calendar Period will be equal to 23.
SrcCalendar_Len[8:0]	Input	0-511 (effective range 1-512)	**Source Calendar Length**. Parameter that sets the length of the calendar sequence. The Source core implements this parameter as a static register synchronous to SrcStatClk, and it can be updated in circuit by first deasserting SrcEn. Note that the Source Calendar Length equals SrcCalendar_Len + 1. For example, if SrcCalendar_Len=15, the Source Calendar Length will be equal to 16.
DataMaxT[15:0]	Static Input	0, 16–65535	**Maximum Data-Training Interval.** Maximum interval between scheduling of training sequences on the SPI-4.2 data path (in SPI-4.2 bus cycles). Note that setting DataMaxT to zero configures the core to never send periodic training.
AlphaData[7:0]	Static Input	0–255	**Data Training Pattern Repetitions**. Number of repetitions of the 20-word data training pattern. Note that setting AlphaData to zero configures the core to not periodically send training patterns. In this case, the user can manually send training patterns by asserting the TrainingRequest command.
NumDip2Errors[3:0]	Static Input	1–15 Value equal to 0 are set to 1	**Number of DIP-2 Errors**. The Source Interface will go out-of-frame (SrcOof asserted) and stop transmitting SPI-4.2 data across the data bus after receiving NumDip2Errors consecutive DIP-2 errors.
NumDip2Matches[3:0]	Static Input	1–15 Value equal to 0 are set to 1	**Number of DIP-2 Matches**. The Source Interface requires NumDip2Matches consecutive DIP-2 matches before going in-frame and beginning to transfer SPI-4.2 data across the SPI-4.2 data bus.

Source Clocking Interface for Virtex-5 and Virtex-4

The Source core supports two clocking implementations: master clocking and slave clocking. The master clocking configuration provides a complete solution with clock circuitry embedded within the Source core. The slave clocking configuration allows the clocking scheme to be implemented external to the Source core.

A list of the Source clocks for master clocking and their description is given in Table 3-17, the DCM reset and clock status signals are defined in Table 3-18, and descriptions of the slave clocking signals are provided in Table 3-19.

The minimum frequency for all clocks in Table 3-17 is dependent on the minimum frequency of the DCM or BUFR (depending on which clocking configuration is selected). See the appropriate specification for Virtex-5 or Virtex-4 devices for additional information.

It is required that the source core reference clock (SysClk) has less than 50 ps of jitter because any jitter present on the SysClk input would appear on the TDClk output. Similarly, the duty cycle distortion on the SysClk should also be minimized. For source cores that use DCM to generate the internal clocks, it is a requirement that the SysClk duty cycle is 45/55 or tighter. For source cores that use regional clocking scheme or PMCD to generate the internal clocks, it is a requirement that SysClk duty cycle is 48/52 or tighter. To reduce the duty cycle distortion of the output TDClk, place the clocking and output component as close to each other as possible. See Source Core Required Constraints for Virtex-5 and Virtex-4 FPGAs in Chapter 6 for more information.

Table 3-17: **Source Core Clocks for Virtex-5 and Virtex-4: Master Configuration**

Clock Pins	Direction	Description	Max Frequency
SysClk_P SysClk_N	Input (differential)	**SysClk**. A The frequency of TDClk is the same as SysClk.	Virtex-5 FPGAs: 500+ MHz Virtex-4 FPGAs: 500 MHz
SysClk0_GP	Output (user interface)	**SysClk0 General Purpose.** This clock is generated from SysClk. It is used to clock the DDR flops of the SPI-4.2 data bus (TDat[15:0]).	Virtex-5 FPGAs: 500+ MHz Virtex-4 FPGAs: 500 MHz
SysClkDiv_GP	Output (user interface)	**SysClkDiv General Purpose**. This clock is generated from SysClk at half the frequency. It is used to clock the internal Source core's logic. This clock runs at half the SysClk0_GP frequency with a zero degree phase offset.	Virtex-5 FPGAs: 250+ MHz Virtex-4 FPGAs: 250 MHz
TSClk_GP	Output (user interface)	**TSClk General Purpose**. This clock is generated from TSClk.	N/A as long as timing constraint is met

Table 3-18: **Source Core Clock Status Signals for Virtex-5 and Virtex-4: Master Configuration**

Signal Name	Direction	Clock Domain	Description
DCMReset_TDClk	Input	n/a	Reset of the TDClk DCM. Tie this signal to "0" when DCM is not used.
Locked_TDClk	Output	n/a	Locked status of the TDClk DCM. This signal is tied to "1" internally when DCM is not used.
DCMReset_TSClk	Input	n/a	Reset of the TSClk DCM. Tie this signal to "0" when DCM is not used.

Table 3-18: **Source Core Clock Status Signals for Virtex-5 and Virtex-4: Master Configuration *(Cont'd)***

Signal Name	Direction	Clock Domain	Description
Locked_TSClk	Output	n/a	Locked status of the TSClk DCM. This signal is tied to "1" internally when DCM is not used.
SrcClksRdy	Output	n/a	Indicates all Source core clocks are ready for use.

Table 3-19: **Source Core Clocks for Virtex-5 and Virtex-4: Slave Configuration**

Clock Pins	Direction	Description	Max Frequency
SysClk0_GBSLV	Input (user interface)	**SysClk-GBSLV**. Used to clock the DDR flops of the SPI-4.2 data bus and Clk (TDClk)(TDat[15:0])	Virtex-5 FPGAs: 500+ MHz Virtex-4 FPGAs: 500 MHz
SysClkDiv_GBSLV	Input (user interface)	**SysClkDiv-GBSLV**. Half the frequency of SysClk0. It is used to clock the internal Source core logic.	Virtex-5 FPGAs: 250+ MHz Virtex-4 FPGAs: 250 MHz
TSClk_GBSLV	Input (user interface)	**TSClk-GBSLV**. Clocks the status logic.	N/A as long as timing constraint is met

Source Clock Interface for Virtex-6 Devices

For Virtex-6 devices, the Source core's clocking module is not embedded into the core. So, the clock interface to the Source core consists of inputs clocks required by the internal logic and output general purpose clock signals that can be used for user logic. A list of the Source clocks and their description is given in Table 3-20.

The minimum frequency for all clocks in Table 3-20 is dependent on the minimum frequency of the MMCM and BUFR (depending on which clocking configuration is selected). See the *Virtex-6 FPGA Product Specification* for more details.

It is required that the source core reference clock (SysClk) has less than 50 ps of jitter because any jitter present on the SysClk input would appear on the TDClk output. Similarly, the duty cycle distortion on the SysClk should also be minimized. For source cores that use MMCM to generate the internal clocks, it is a requirement that the SysClk duty cycle is 45/55 or tighter. For source cores that use regional clocking scheme to generate the internal clocks, it is a requirement that SysClk duty cycle is 48/52 or tighter. To reduce the duty cycle distortion of the output TDClk, place the clocking and output component as close to each other as possible. See Source Core Required Constraints for Virtex-6 FPGAs in Chapter 6.

Table 3-20: **Source Clock Signals for Virtex-6 Devices**

Name	Direction	Description	Maximum Frequency
SysClk0_User	Input (User Interface)	**SysClk User :** This clock is generated from SysClk_P/N. It has the same frequency as TDClk.	500+ MHz
SysClkDiv_User	Input (User Interface)	**SysClkDiv User:** This clock is generated from SysClk_P/N at half the frequency. It is used for clocking the internal logic of the Source core. This clock runs at half the rate of SysClk0_User frequency with a zero degree phase offset.	250+ MHz
TSClk_User	Input (User Interface)	**TSClk User:** This clock is generated from TSClk_P/N. It has the same frequency as TSClk. It is used to clock the internal status and calendar logic.	125+ MHz
SysClk0_GP	Output (User Interface)	**SysClk0 General Purpose**: Same signal as SysClk0_User.	500+ MHz
SysClkDiv_GP	Output (User Interface)	**SysClkDiv General Purpose**: Same signal as SysClkDiv_User.	250+ MHz

Chapter 4

Generating the Core

The SPI-4.2 core is a fully configurable implementation of the *OIF-SPI4-02.1* specification. Using the CORE Generator GUI, you can configure the core and customize the delivered files including the example wrapper and UCF files.

Note: After the core is generated, only static configuration signals options can be modified by changing the input values. If other modifications are required, the core must be regenerated with the new options.

CORE Generator Graphical User Interface

The SPI-4.2 CORE Generator graphical user interface (GUI) consists of seven screens:

- The main screen provides options to generate specific hardware components (using dedicated logic resources) that apply to both the Sink and Source cores.
- The next three screens provide configuration options specific to the Sink core.
- The last three screens provide configuration options specific to the Source core.

Main Screen

The main SPI-4.2 screen defines the component name, core options, and UCF File options.

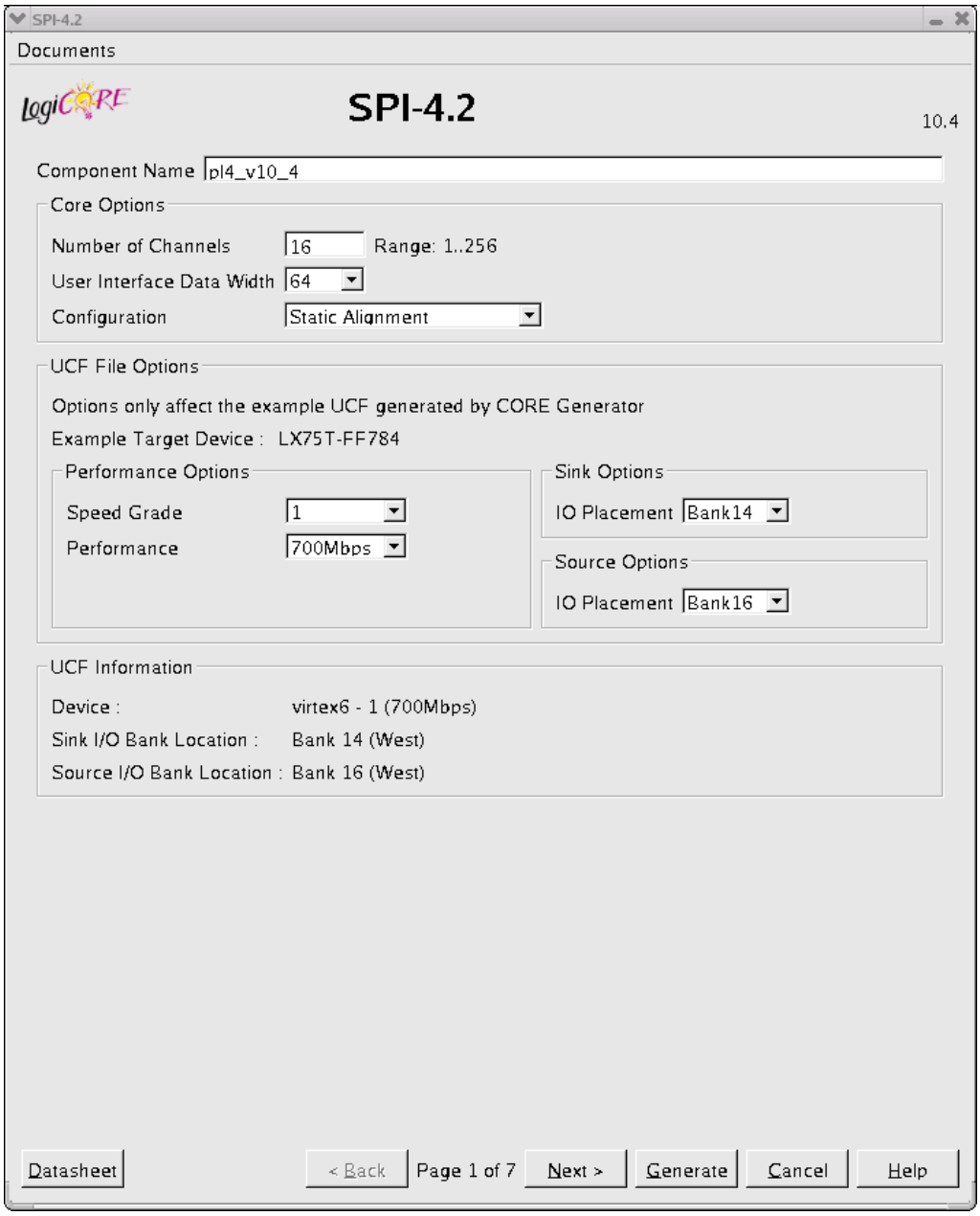


Figure 4-1: **SPI-4.2 Sink and Source Main Customization Screen**

Component Name

The Component Name is the base name of the output files generated for the core. The name must begin with a letter and be composed of the following characters: a to z, 0 to 9, and "_".

Core Options

Number of Channels

The SPI-4.2 core supports between 1 and 256 channels.

User Data Interface

The SPI-4.2 core supports either 64-bit or 128-bit user data interface.

Configuration

Either static or dynamic phase alignment configurations can be used to capture data on the SPI-4.2 Sink interface. See Sink Data Capture Implementation in Chapter 5.

UCF File Options

Selections in the UCF file section of the GUI affect generation of the example UCF file. The UCF file can be edited to change options, or to regenerate the core with the CORE Generator system to create a new example UCF file. See Chapter 6, Constraining the Core for more information on the constraints file.

Performance Options

The speed grade selected determines the data rates available in the drop-down menu. The data rate determines the relevant timing constraints in the example UCF file.

Sink and Source Options

The I/O placement selection configures the pin-location constraints in the example UCF file. These are editable, as described in Chapter 6, Constraining the Core.

UCF Information

This section displays a summary of the contents of the example UCF file that will be generated.

Sink Status Options Screen

This screen contains options for the static configuration parameters of the Sink core. The static configuration parameters listed below determine the behavior of the status interface.

Calendar

The options listed below affect the behavior of the calendar and status interfaces of the Sink core.

Iterations of Calendar Sequence Before DIP2

This is the value of the static configuration signal SnkCalendar_M; it is the number of times the Sink core will repeat the calendar sequence before sending a DIP2 value and frame word on `RStat`. The valid range is 1 to 256.

Length of Calendar Sequence

This is the value of the static configuration signal SnkCalendar_Len; it is the number of entries in the calendar sequence. The valid range is 1 to 512.

Load Init File

If this option is selected, the Sink core calendar block RAM will be initialized at startup with a sequence loaded from a COE file. The sequence can be overwritten at runtime via the calendar interface.

Load Coefficients

For this option, select the name of the COE file with the calendar programming information. For more information, see Calendar COE File Format.

Show Coefficients

This shows the contents of the loaded COE file.

Flow Control

This option selects the value of static configuration signal `FifoAFMode`; it determines the behavior of the Sink core status interface when the internal FIFO is almost full. See FIFO Almost Full Mode and Sink Almost Full, page 90.

Send Satisfied on All Channels

This option causes the Sink core to send the satisfied ("10") status on *`RStat`* for each channel.

Send Framing

This option causes the Sink core to send framing ("11") on *`RStat`* and go out-of-frame.

Send Current Status

This causes the Sink core to continue sending the stored status value on *`RStat`* for each channel.

Status Interface

This option selects the default static configuration parameters for Sink core status channel clocking and I/O type.

Rate

This is the value of static configuration signal `RSClkDiv`; it selects the frequency of *`RSClk`* with respect to RDClk.

Alignment

This is the value of static configuration signal `RSClkPhaseAlign`; it determines whether *`RStat`* transitions on the rising or falling edge of `RSClk`.

Status I/O

This controls whether *RStat* and *RSClk* I/O in the generated wrapper file use LVDS or LVTTL I/O. For Virtex-6 devices, only LVDS I/O option is supported.

Sink Other Options Screen

This configuration screen contains options that affect the FIFO flags, clocking implementation, status channel behavior, and I/O type.

Synchronization

Number of Training Sequences

This is the value of static configuration signal `NumTrainSequences`; it is the number of training sequences the Sink core must receive on RDat before going in-frame and transiting from framing to status on `RStat`. The valid range is 1 to 15.

Number of DIP4 Errors

This is the value of static configuration signal `NumDIP4Errors`; it is the number of consecutive control words with invalid DIP4 values the Sink core must receive on `RDat` before going out-of-frame and sending framing on `RStat`. The valid range is 1 to 15.

FIFO Threshold

This option selects the default static configuration parameters for Sink core FIFO Threshold behavior.

Almost Full Assert

This is the value of static configuration signal `SnkAFThresAssert`; it is the internal FIFO level at which the Sink core will assert `SnkFFAlmostFull_n` and take the specified flow control action. The valid range is 1 to 508 and is measured from the full level. For example, if the value chosen is 10, `SnkFFAlmostFull_n` will be asserted when there are 10 FIFO locations empty.

Almost Full Negate

This is the value of static configuration signal `SnkAFThresNegate`; it is the internal FIFO level at which the Sink core will deassert `SnkFFAlmostFull_n` and return `RStat` behavior to normal. The valid range is the *Almost Full Assert* value to 508 and is also measured from the full level.

Read Mode

Full Burst

Enabling this option causes the Sink core to receive a complete burst of data before processing and passing it to the user read interface (FIFO interface). This behavior ensures that the user always reads full burst of data on the user interface. Using this feature limits the size of unsegmented burst received by the sink core to the size of the FIFO (510 words).

Partial Burst

Low latency mode for reading available data. This feature enables the user to read data from the user interface as soon as it is available. If the read data rate is faster than the write data rate, the user will read partial bursts from the user interface.This feature also enables the user to receive a data burst larger than the sink core FIFO (510 words).

Clocking for Virtex®-5 and Virtex-4 devices

This option enables you to choose the clocking implementation that is most suitable for your system. For more information, see Sink Clocking Options for Virtex-5 and Virtex-4 Devices, page 153.

RDClk Clock Distribution

Depending on the option selected, the `RDClk` internal clocking implementation uses either global clock buffers or regional clock buffers.

RDClk Internal Clock Generator

Depending on the option selected, the `RDClk` internal clocking implementation uses either a DCM or PMCD. This option is only valid if `RDClk` Distribution is set to global clock. The PMCD feature is only available for Virtex-4 FPGAS and when the sink core is configured to use Dynamic Phase Alignment.

Insert IDELAY on RDClk

Enabling this option will cause an IDELAY to be inserted in the `RDClk` input path. This feature is only available for cores with Dynamic Phase Alignment configuration. When this option is selected, the Generate Dedicated Input IDELAYCTRL Reset option is recommended. This is because the IDELAYCTRL reset pin is by default connected to the Sink Core reset, and may cause a lockup issue.

Specify Initial Tap Setting

Found under Insert IDELAY on `RDClk`. This option allows you to pick the initial tap setting for the inserted IDELAY. If using DPA clock adjustment features, see DPA Clock Adjustment, page 96 for information about setting this value.

Include IDELAYCTRL Modules

Select this option if you want the IDELAYCTRLs to be included in the core. If this option is not selected, the IDELAYCTRLs must be instantiated in the wrapper by the user. For Virtex-5 devices, this option is disabled, and the user must instantiate the IDELAYCTRLs in the user design.

Generate Dedicated Input IDELAYCTRL Reset

Select this option to use a dedicated input reset pin to reset all the `IDELAYCTRL` primitives instantiated in the Sink core. This option is available when the option “Include IDELAYCTRL Modules” is selected.

DCM Options

The following options are only applicable if `RDClk` is generated using DCM.

Include DCM Standby Logic

Select this option if you want a DCM Standby Logic to be included in the core. This feature is only available for Virtex-4 FPGAs.

Use DCM to Generate Full Rate Clock

This option is always enabled causing the DCM to always generate the Full Rate Clock (RDCLK0_GP) to sample data on the I/O. This is recommended for all DCM clocking schemes.

Clocking for Virtex-6 Devices

For designs targeting the Virtex-6 family, the Sink core only supports user clocking mode. In user clocking mode, the clocking logic is defined external to the core. An example of the external clocking module is provided as part of the provided example design. For more information, see Sink Clocking Options for Virtex-6 Devices in Chapter 7.

RDClk Distribution

The RDClk clock can be distributed using a global buffer (which includes using an MMCM to generate the full and half rate internal RDClk) or a regional buffer. Depending on the option selected, the RDClk internal clocking implementation uses either global clock buffers or regional clock buffers.

Sink Other Options (II) Screen

This configuration screen contain options relating to the SPI-4.2 sink data bus and dynamic phase alignment.

DPA Options

The following options configure the dynamic phase alignment logic. For information about how to configure these options, see Dynamic Phase Alignment (Virtex Devices Only), page 92.

Master-Slave IDELAY Offset

This option determines the offset between the initial tap settings for the master and reference (slave) IDELAY controller.

Alignment Test Interval

This option determines the number samples taken at each potential alignment point during alignment.

Enable Auto-retry

Enabling this option causes the automatic reinitiation of dynamic phase alignment (when it fails) until alignment is achieved.

Enable Continuous Alignment

Enabling this option causes the alignment logic to continuously monitor alignment during operation, adapting to the data sample point to system timing changes.

Generate Continuous Alignment Halt Pin

Enabling this option allows you to use a dedicated input pin to halt the continuous alignment process during operation.

Enable DPA Clock Adjustment

Enabling this option allows the alignment logic to adjust the incoming `RDClk` to reduce jitter introduced in the IDELAY module. For Virtex-6 this feature is only available when the RDClk clock distribution is set to regional buffer.

DPA Wait for Training Control

Enabling this option allows the DPA logic to wait for the presence of a training control word on the SPI-4.2 bus before starting the alignment process.

Enable DPA Status Monitoring

This option is related to DPA diagnostic signals and enables the user to monitor the dynamic phase alignment logic using the `SnkBusErrStat` signal.

Report IDELAY on SPI-4.2 Bus Index

This option is related to DPA diagnostic signals. With DPA status monitoring and Report IDELAY enabled, the `SnkBusErrStat` signal will track the IDELAY setting during alignment for the chosen SPI-4.2 bus index. Valid index is 0 to 16 where 16 is the control bit.

Generate Advance DPA Diagnostic Ports

This option is related to DPA diagnostic signals. Enabling it allows you to use Advance DPA Diagnostic Ports to measure and capture the data valid window for each channel during operation.

SPI-4.2 Sink Bus Options

These options configure the SPI-4.2 sink bus.

Invert SPI-4.2 input

This option determines whether the SPI-4.2 data and control bits need to be inverted. The XOR mask which consist of a binary string of 17 bits determines which SPI-4.2 bus signals will be inverted. From the left the first bit determines the inversion of the control bit (*`RCtl`*), followed by 15th data bit (`RDat[15]`), etc. The rightmost bit determines the inversion of the 0 data bit (`RDat[0]`). Use this option when the P and N pins for the SPI-4.2 bus is swapped.

Note: The demonstration test bench provided with the core does not support generating stimulus that is compatible to this feature.

Source Status Options Screen

This configuration screen contains options for static configuration parameters of the Source core. The following static configuration parameters determine the behavior of the status interface.

Calendar

This describes the status pattern that the Source core expects on its status interface.

Iterations of Calendar Sequence Before DIP2

This is the value of static configuration signal `SrcCalendar_M;` it is the number of times the Source core will expect the calendar sequence to repeat before seeing a DIP2 value and framing on `TStat`. The valid range is 1 to 256.

Length of Calendar Sequence

This is the value of static configuration signal `SrcCalendar_Len`; it is the number of entries in the calendar sequence. The valid range is 1 to 512.

Load Init File

When this option is selected, the Source core calendar block RAM is initalized at startup with a sequence loaded from a COE file.

Load Coefficients

This option lets you select the name of the COE file with calendar programming information. For more information, see Calendar COE File Format.

Show Coefficients

This option lets you view the contents of the loaded COE file.

Status Interface

Status FIFO Interface

This option determines whether the Source core netlist is generated with an addressable or a transparent user status interface. For more information, see the Source Status and Flow Control Signals, page 115.

Status I/O

This option controls whether the Source core status I/O in the generated wrapper file uses LVDS or LVTTL I/O. For Virtex-6 devices, only LVDS I/O option is supported.

Synchronization

This option defines the synchronization parameters of the core.

Number of DIP2 Matches

This is the value of static configuration signal `NumDIP2Matches`; it is the number of consecutive valid DIP2 words the Source core must observe on `TStat` before it goes in-frame, deasserts `SrcOof`, and begins to transmit data on `TDat`. The valid range is 1 to 15.

Number of DIP2 Errors

This is the value of static configuration signal `NumDip2Errors`; it is the number of consecutive invalid DIP2 words the Source core must observe on *`TStat`* before going out-of-frame. The valid range is 1 to 15.

Source Other Options Screen

This screen contains options that affect data burst behavior, FIFO flag behavior, and clocking implementation.

Bursting

This selects the static configuration parameters that determine Source core transmit behavior.

Number of Data Cycles Before Training

This is the value of static configuration signal `DataMaxT`; it is the approximate number of cycles of data the Source core will transmit on `TDat` between periodic training sequences. The valid range is from 0, 16 to 65535. A value of 0 indicates that the core will not send periodic training.

Number of Training Patterns During Training

This is the value of static configuration signal `AlphaData`; it is the number of training patterns the Source core will transmit on `TDat` each time periodic training is sent. The valid range is from 0 to 255. A value of 0 indicates that the core will not send periodic training.

Burst Size in Credits

This is the value of static configuration signal `SrcBurstLen`; it is the maximum burst length (unsegmented packets) in credits. The valid range is from 1 to 1023.

Burst Mode

This is the value of static configuration signal `SrcBurstMode`. It specifies how the Source core transmits data. Complete Bursts Only causes the core to send only data bursts that are of Burst Size (as defined above) or terminated by an EOP, or when the channel address changes. Segmentation of Bursts **at** Credit Boundary causes the core to send data bursts that terminate at any credit boundary or with an EOP. See Source Burst Mode, page 123.

FIFO Threshold

This option lets you select the default static configuration parameters for Source core FIFO Threshold behavior, as described below.

Almost Full Assert

This is the value of static configuration signal `SrcAFThresAssert`; it is the internal FIFO level at which the Source core will assert `SrcFFAlmostFull_n`. The *Almost Full Assert* value is measured from the full level. For example, if the value chosen is 40, then `SrcFFAlmostFull_n` is asserted when there are 40 FIFO locations empty.

Almost Full Negate

This is the value of static configuration signal `SrcAFThresNegate`; it is the internal FIFO level at which the Source core will deassert `SrcFFAlmostFull_n`. The valid range is the *Almost Full Assert* value to 508 and is also measured from the full level.

Source Other Options (II) Screen

This configuration screen enables you to choose the clocking implementation that is most suitable for your system.

Clocking for Virtex-5 and Virtex-4 Devices

Clock Mode

The Source core netlist will contain a complete clocking solution if Master Clocking is selected. If Slave Clocking is selected, you must provide a clock generation method external to the Source core. For more information, see Source Clocking Options for Virtex-5 and Virtex-4 Devices, page 164.

SysClk Settings

Clock Distribution

The `SysClk` internal clocking implementation can use either global clock buffers or regional clock buffers. This option is only applicable for Master Clocking mode.

Internal Clock Generator

The `SysClk` internal clocking implementation can use either a DCM or PMCD. This applies to Master Clocking Mode, and when `SysClk` distribution is global clock. The PMCD option is only available for Virtex-4 FPGAs.

Use DCM to Generate Full Rate Clock

This option is always enabled causing the DCM to always generate the Full Rate Clock (`SysClk0_GP`) to sample data on the I/O. This is recommended for all DCM clocking schemes.

Include DCM Standby Logic

Select this check box if a DCM standby logic should be included in the core. This feature is only available in Virtex-4 FPGAs.

TSClk Settings

Clock Distribution

The `TSClk` internal clocking implementation can use either the global clock buffers or the regional clock buffers. This only applies to Master Clocking mode.

Internal Clock Generator

The `TSClk` internal clocking implementation can use a DCM, PMCD, or None. The None option enables the user to not regenerate the internal clock. This applies to Master

Clocking mode, and when SysClk distribution is global clock. The PMCD option is only available for Virtex-4 FPGAs.

Include DCM Standby Logic

Select this check box if a DCM standby logic should be included in the core. This feature is only available in Virtex-4 FPGAs.

Clocking for Virtex-6 FPGAs

For designs targeting the Virtex-6 family, the source core only supports user clocking mode. In user clocking mode, the clocking logic is defined external to the core. An example of the external clocking module is provided as part of the provided example design. For more information, see Source Clocking Options for Virtex-6 Devices in Chapter 7.

SysClk Clock Distribution

The SysClk clock can be distributed using a global buffer or a regional buffer. For source core interfacing with a sink core configured with static alignment, only regional buffer is supported.

TSClk Clock Distribution

The TSClk clock can be distributed using a global buffer or a regional buffer.

SPI-4.2 Source Bus Options

These options configure the SPI-4.2 source bus.

Invert SPI-4.2 Output

This option determines whether the SPI-4.2 data and control bits need to be inverted. The XOR mask which consist of a binary string of 17 bits determines which SPI-4.2 bus signals will be inverted. From the left the first bit determines the inversion of the control bit (*RCtl*), followed by 15th data bit (RDat[15]), etc. The rightmost bit determines the inversion of the 0 data bit (RDat[0]). Use this option when the P and N pins for the SPI-4.2 bus is swapped.

Note: The demonstration test bench provided with the core does not support generating stimulus that is compatible to this feature.

Calendar COE File Format

The initial contents of the calendar can be assigned by placing the information into a separate text file called a *coe* file. To select and load a *coe* file, first create the desired *coe* file, select *Load Coefficients* on the parameterization screen, and choose the desired file from the file dialog box. An example *coe* file for a 12-channel SPI-4.2 core with a round-robin calendar and a calendar length of 12 (SnkCalendar_Len = "11" or SrcCalendar_Len = "11") is shown below:

```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=00,01,02,03,04,05,06,07,08,09,0A,0B;
```

When specifying the initial contents for the calendar in a *coe* file, the keywords MEMORY_INITIALIZATION_RADIX and MEMORY_INITIALIZATION_VECTOR are used.

The MEMORY_INITIALIZATION_VECTOR takes the form of a sequence of comma-separated values, one value per calendar entry, terminated by a semicolon.

These values are listed in ascending order, where the first entry in the MEMORY_INITIALIZATION_VECTOR is the first entry in the calendar. Any amount of white space, including new lines, can be included in the vector to enhance readability. The format of an individual value in the vector depends on the MEMORY_INITIALIZATION_RADIX value, which can be 2, 10, or 16 (the default value is 10). The vector must be consistent with the MEMORY_INITIALIZATION_RADIX value and each value must fall within the range of 0 to 255 (base 10).

The number of entries in the *coe* file need not be the same as the calendar length specified in the GUI. If the calendar length is smaller than the number of entries, the calendar sequence used in the core will be a subset of the calendar sequence specified in the *coe* file. This subset will contain calendar entries *0* to *Calendar Length-1* from the *coe* file. If the calendar length is larger than the number of entries, the calendar sequence specified in the *coe* file will be padded with zeros to match the calendar length.

XILINX®

Chapter 5

Designing with the Core

This chapter contains general design guidelines, detailed descriptions about the behavior of each interface, example waveforms, and implementation considerations. Use the guidelines provided in this chapter to design an application using the SPI-4.2 core.

General Design Guidelines

This section describes the steps required to implement each feature of the SPI-4.2 core into a fully-functioning design that is integrated with user application logic. While Xilinx recommends that you follow the guidelines listed below for optimum results, not all designs will require that you perform all steps listed in this chapter.

Know the Degree of Difficulty

A fully compliant SPI-4.2 core is challenging to implement in any technology. That degree of difficulty is significantly influenced by the following:

- Maximum system clock frequency
- Targeted device architecture
- Specific user application

All implementations require careful attention to system performance requirements. Pipelining, placement constraints, and logic duplication are all methods that can be used to improve system performance.

Understand Signal Pipelining

Due to the nature of packet protocols, it is important to understand that the SPI-4.2 Sink and Source cores have been pipelined to maximize performance. The 64-bit data written into the Source core user interface takes several clock cycles before appearing on the SPI-4.2 interface. This is due to the pipelining required to format the packet, create control words, and calculate DIP4.

Similarly, SPI-4.2 packets that are received by the Sink core take several clock cycles before appearing on the user interface. This is due to the pipelining required to convert the streaming input bus to an aligned output with packet information, error signals, etc. The exact latency of the Sink and Source cores varies, based on the configuration of the core. It can be best determined through simulation.

Keep it Registered

The best method to simplify timing and increase system performance in an FPGA design is to keep everything registered. That is, all inputs and outputs from your application should

come from, or connect to, a flip-flop. While registering signals may not be possible for all paths, it simplifies timing analysis and helps achieve timing closure.

Recognize Timing Critical Signals

Watch the timing and loading on the signals listed below. Some of these signals are part of the critical timing path. The following signals are timing-critical and may require special attention when used in the user application.

- `SnkFFRdEn_n`
- `SrcFFWrEn_n`

Use Supported Design Flows

The SPI-4.2 core has been tested with a variety of design flows. While other design tools can be used to simulate and synthesize the user design with the core, they have not been tested and functionality cannot be guaranteed. See Chapter 8, Simulating and Implementing the Core for information about supported design tools.

Make Only Allowed Modifications

All modifications to the SPI-4.2 core must be made using the Xilinx CORE Generator system. Modifications made not using the CORE Generator may have adverse effects on system timing and SPI-4.2 protocol compliance.

Initializing the SPI-4.2 Core

The SPI-4.2 Sink and Source cores require initialization before receiving and transmitting data. The initialization steps are:

1. Assert core reset.

   To reset the cores, `Reset_n` must be asserted. The reset signal for each core must remain asserted until the clocks are ready for use.

2. Disable the cores (`SrcEn = 0, SnkEn = 0`).
3. Assert and release the DCM or MMCM resets.

   This step is only applicable if TSClk, SysClk, or RDClk is distributed with global clocking using a DCM or MMCM. If regional clocking is selected for all clocks, this step can be skipped. If one or more DCMs or MMCMs are used, reset each DCM or MMCM in the core while the core is in reset. Reset the DCM or MMCM by asserting the DCM or MMCM reset signal (for example, `DCMReset_RDClk`). Once the DCM or MMCM reset is asserted, wait for the assertion of the DCM or MMCM locked signal (ex: `Locked_RDClk`). When the locked signal is asserted, the clock is ready for use.

   See Chapter 7, Special Design Considerations for more information on the regional and global clocking options.

4. If the Generate Dedicated Input IDELAYCTRL Reset option is enabled, reset the IDELAY controllers in the Sink core using the `SnkIdelayCtlRst` input.

   To reset the IDELAY controllers in the Sink core, `SnkIdelayCtlRst` must be pulsed after configuration and the `SnkIdelayRefClk` (reference clock input of the IDELAYCTRLS) has stabilized to ensure proper IDELAY operation.

5. Deassert core reset.

Once all the clocks are ready for use, the `SnkClksRdy` and `SrcClksRdy` signals assert. `Reset_n` can be deasserted only when these signals are asserted.

6. Initialize the Status Calendar.

   After the core exits the reset mode, the Sink and status calendars must be initialized or programmed. There are two ways to do this:

   a. Initialize calendar with a default value.

      Using the CORE Generator GUI load an initialization file with the calendar contents. See Chapter 4, Generating the Core for more information.

   b. Programming calendar after reset.

      Using the calendar control interface to program the calendar contents. See Sink Calendar Initialization, page 84 and Source Calendar Initialization, page 116 sections for more information.

7. Enable the cores (`SrcEn = 1, SnkEn = 1`).

After initializing the core, it can be enabled for operation.

Sink Core

Basic Operation

The Sink core receives data across the SPI-4.2 interface and converts the 16-bit data into 64-bit or 128-bit data words which can be accessed on the user interface. It also transmits flow control information on the SPI-4.2 interface by converting a 32-bit status bus to a 2-bit status word.

The following sections explain how the Sink core interfaces operate. See Sink Core Interfaces, page 29 for the signal list of each interface.

SPI-4.2 Interface

The SPI-4.2 data path combines 16-bit data words received on the SPI-4.2 Interface into 64- or 128-bit data words. This allows the user interface to run at a quarter (64-bit interface) or an eighth (128-bit interface) of the data rate. For example, with a 700 Mbps SPI-4.2 data rate and a 64-bit user interface, you can read data from the Sink core at 175 MHz. With a 128-bit user interface, you can read data from the Sink core at 87.5 MHz and maintain the same data rate.

Once the data path combines the 16-bit data words received on the SPI-4.2 interface, the data words are written into an asynchronous FIFO. The received 16-bit control words are stored out of band in the FIFO, along with the corresponding data word. The received control words that are not idle (training words) can contain the information listed below.

- Start or continuation of the following packet
- Link address of the following packet
- End of the preceding packet
- Number of valid bytes in the last word of the preceding packet
- Error conditions in the preceding packet

For details on assignment of each bit in the control word, as defined by the OIF SPI-4.2 specification, see Appendix E, SPI-4.2 Control Word.

Sink Data Path: Example 1

Figure 5-1 provides an example of the data received on the SPI-4.2 Interface and read on the user interface. In this illustration, the first received control word (C1) is a payload resume (with no SOP) for channel 1 followed by two 16-bit words (channel 1, packet A and packet B). The second control word (C2) is an EOP for channel 1 and a payload resume for channel 2 (with no SOP) followed by two 16-bit words. The third control word (C3) is an EOP for channel 2 and an SOP for channel 1 followed by three 16-bit words. The last control word (C4) is an EOP for channel 1.

The data that is received on the SPI-4.2 Interface is processed and stored in the Sink FIFO. Figure 5-1 also shows the data being read out of the FIFO and indicates, with forward slashes, that there is latency in processing and storing the SPI-4.2 data. The first 64-bit word on the FIFO interface contains the two 16-bit words received for channel 1 with an EOP. The second 64-bit word contains the two words received for channel 2 with an EOP. The last 64-bit word on the FIFO interface contains the three words written for channel 1. When the last word is read out of the FIFO, both the `SnkFFSOP` and `SnkFFEOP` for channel 1 are asserted.

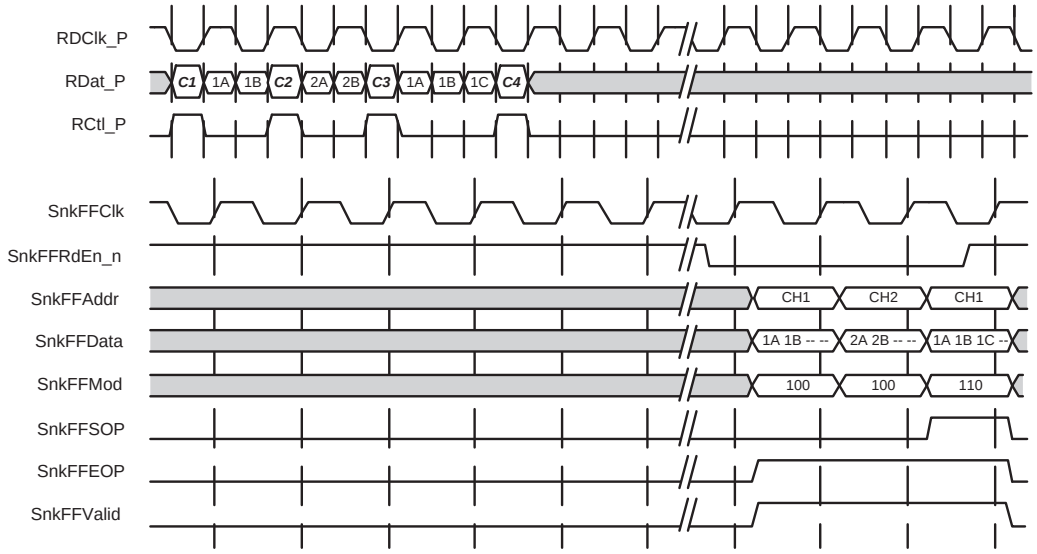


Figure 5-1: **SPI-4.2 Interface to User Interface**

Sink Data Path: Example 2

The Sink core optimally handles any size packet including short packets (less than eight cycles apart), which have multiple SOPs or payload control words.

There are two scenarios in which short packets can be received:

- **Received SOPs that are less than eight cycles apart**. The data is passed through the core as received, and a `SnkBusErr` will be flagged indicating a protocol violation.
- **Received Payload Control words that are less than eight cycles apart**. Though the SPI-4.2 specification requires that successive SOPs must occur not less than eight cycles apart, there is no restriction on payload control words, which are not SOPs. The Sink core can process single payload control words followed by single data words (CTL-DATA-CTL-DATA-CTL, etc.). Since this is not a protocol violation, no `SnkBusErr` is asserted.

Figure 5-2 shows the transfer of short packets from the SPI-4.2 Interface through the Sink FIFO to the user interface. Because each of these packets contains less than 14 bytes, or seven clock cycles of data, idle control word insertion is necessary to meet the start-of-packet spacing requirement of eight cycles. The transfer on the SPI-4.2 Interface begins with a payload control word (C1), indicating a start of packet (SOP) on channel 1. Next, two clock cycles, two bytes each, are used to transfer the data associated with channel 1. The transfer concludes with an end-of-packet control word (C2). The transfer being less than 14 bytes, four idle cycles are required to meet the SOP spacing requirement. After the four idle cycles, the transfer begins with a start-of-packet control word (C3) for channel 2. Next, three clock cycles (of two bytes each) are used to transfer the data associated with channel 2. The transfer concludes with an end-of-packet control word (C4).

Figure 5-2 also shows the data being read out of the FIFO and indicates with forward slashes that there is latency in processing and storing the SPI-4.2 data. The first 64-bit word on the FIFO interface contains the four bytes of valid data received for channel 1. The control signals `SnkFFSOP` and `SnkFFEOP` are active, indicating that this is the start and end of the packet for channel 1. The second 64-bit word contains the six bytes of valid data for channel 2, and the control signals `SnkFFSOP` and `SnkFFEOP` are both asserted.

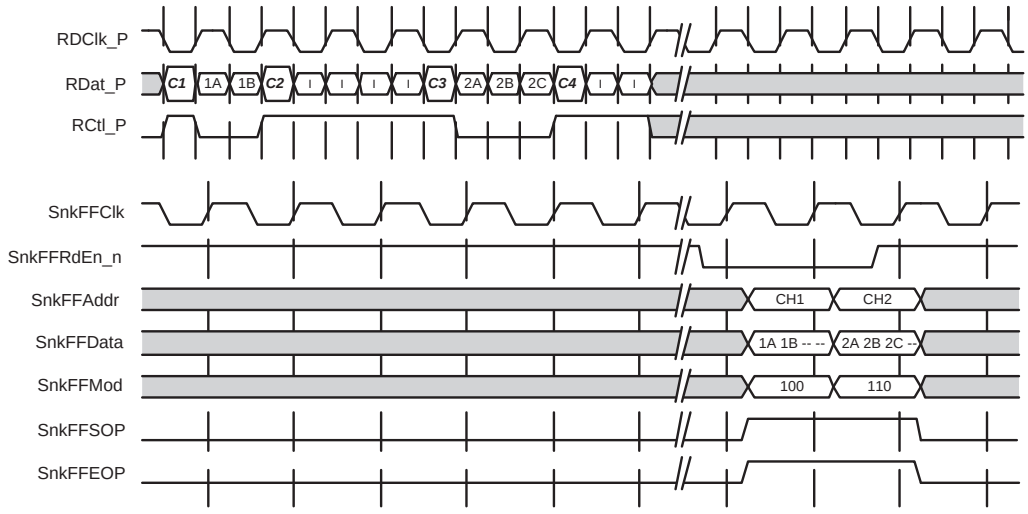


Figure 5-2: **Sink Data Path - Short Packet Transfers with Minimum SOP Spacing Enforced**

Table 5-1 shows how the data and control received on the SPI-4.2 Interface is formatted and presented on the 64-bit Sink FIFO Interface. (Note that control words are shown in binary and payload transfers are shown as hexadecimal.) After an SOP is received, the following 16-bit word transfer is left justified (that is, written to the most significant 16 bits) when written into the FIFO. For the 64-bit interface, the 16 bits will be in the `SnkFFData[63:48]`. The table shows the receipt of an SOP for channel 2, then a series of payload word transfers. The DIP-4 parity depends on this control word and any proceeding transfer, and it is shown in the table as "pppp."

Following this example, there are two more tables that show the mapping between SPI-4.2 Control Words and packet status signals for a 64-bit user interface (Table 5-2) and for a 128-bit user interface (Table 5-3).

Table 5-1: **Formatting SPI-4.2 Interface Data (RDat) for a 64-bit User Interface**

Data Received on the SPI-4.2 Interface (RDat [15:0])	RCtl	RDClk cycle	Data Read from the Sink FIFO (SnkFFData[63:0])	SnkFFClk cycle	Control Bits Read from the Sink FIFO
SOP b:[1001.0000.0010.pppp]	1	1	N/A	n	N/A
SPI-4.2 Word 0 (P0) F1E2	0	2			
SPI-4.2 Word 1 (P1) D3C4	0	3			
SPI-4.2 Word 2 (P2) B5A6	0	4			
SPI-4.2 Word 3 (P3) F9E8	0	5	SnkFFData[63:0] = P0.P1.P2.P3 = [F1E2.D3C4.B5A6. F9E8]	n + 1	SnkFFSOP = 1 SnkFFEOP = 0 SnkFFMod = 000 SnkFFErr = 0 SnkFFAddr = 00000010
SPI-4.2 Word 4 (P4) 1F2E	0	6			
SPI-4.2 Word 5 (P5) 3D4C	0	7			
SPI-4.2 Word 6 (P6) 5B6A	0	8			
SPI-4.2 Word 7 (P7) 9F8E	0	9	SnkFFData[63:0] = P4.P5.P6.P7 = [1F2E.3D4C.5B6A. 9F8E]	n + 2	SnkFFSOP= 0 SnkFFEOP = 0 SnkFFMod = 000 SnkFFErr = 0 SnkFFAddr = 00000010
SPI-4.2 Word 8 (P8) ABCD	0	10			
SPI-4.2 Word 9 (P9) 1200	0	11			
EOP / MOD b:[0110.aaaa.aaaa.pppp]	1	12			
			SnkFFData[63:0] = P8.P9 = [ABCD.1200.0000. 0000]	n + 3	SnkFFSOP= 0 SnkFFEOP = 1 SnkFFMod = 011 SnkFFErr = 0 SnkFFAddr = 00000010

Table 5-2: **SPI-4.2 Control Word Mapping to 64-bit User Interface**

Control Word	Associated SPI-4.2 Control Word bits on RDat (Qualified by RCtl=1)	Associated Sink FIFO Signals
Start of Packet (SOP)	RDat[15] = 1, RDat[12] = 1	SnkFFSOP, SnkFFAddr[7:0] <== RDat[11:4]
New Burst (address change)	RDat[15] = 1, RDat[12] = 0	SnkFFAddr[7:0] <== RDat[11:4]
End of Packet (EOP, 2 bytes valid)	RDat[14:13] = 10	SnkFFEOP, SnkFFMod[2:0] When RDat[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
End of Packet (EOP, 1 bytes valid)	RDat[14:13] = 11	SnkFFEOP, SnkFFMod[2:0] When RDat[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
End of Packet (EOP Abort, error condition)	RDat[14:13] = 01	SnkFFErr & SnkFFEOP

Table 5-3: **SPI-4.2 Control Word Mapping to 128-bit User Interface**

Control Word	Associated SPI-4.2 Control Word bits on RDat (Qualified by RCtl=1)	Associated Sink FIFO Signals
Start of Packet (SOP)	RDat[15] = 1, RDat[12] = 1	SnkFFSOP, SnkFFAddr[7:0] <== RDat[11:4]
New Burst (address change)	RDat[15] = 1, RDat[12] = 0	SnkFFAddr[7:0] <== RDat[11:4]

Table 5-3: **SPI-4.2 Control Word Mapping to 128-bit User Interface *(Cont'd)***

Control Word	Associated SPI-4.2 Control Word bits on RDat (Qualified by RCtl=1)	Associated Sink FIFO Signals
End of Packet (EOP, 2 bytes valid)	RDat[14:13] = 10	SnkFFEOP, SnkFFMod[2:0] When RDat[14:13] = 10: MOD = 0000 if data bits 127-0 have valid data MOD =1110 if data bits 127-16 have valid data MOD =1100 if data bits 127-32 have valid data MOD = 1010 if data bits 127-48 have valid data MOD = 1000 if data bits 127-64 have valid data MOD = 0110 if data bits 127-80 have valid data MOD = 0100 if data bits 127-96 have valid data MOD = 0010 if data bits 127-112 have valid data
End of Packet (EOP, 1 byte valid)	RDat[14:13] = 11	SnkFFEOP, SnkFFMod[2:0] When RDat[14:13] = 11: MOD = 1111 if data bits 127-8 have valid data MOD = 1101 if data bits 127-24 have valid data MOD = 1011 if data bits 127-40 have valid data MOD = 1001 if data bits 127-56 have valid data MOD = 0111 if data bits 127-72 have valid data MOD = 0101 if data bits 127-88 have valid data MOD = 0011 if data bits 127-104 have valid data MOD = 0001 if data bits 127-120 have valid data
End of Packet (EOP Abort, error condition)	RDat[14:13] = 01	SnkFFErr & SnkFFEOP

Sink User Interface

The Sink user interface includes all the signals to the core other than those on the SPI-4.2 Interface (See SPI-4.2 Interface, page 73). With a 64-bit and 128-bit data interface, this user interface can operate up to:

- 350 MHz in Virtex®-6 FPGAs
- 312 MHz in Virtex-5 FPGAs
- 250 MHz in Virtex-4 FPGAs

The high performance Sink back-end enables the user interface to run at higher frequencies than the SPI-4.2 Interface. This is sometimes required if a large percentage of the traffic consists of small packets.

The user interface has three major sections:

- **Control and Status Signals**. Apply to the operation of the entire Sink core.
- **FIFO Interface Signals**. Allow access to data received on the SPI-4.2 Interface.
- **Status and Flow Control Signals**. Used to send flow control information on the SPI-4.2 Interface.

Sink Control and Status Signals

The Sink core control and status signals either control the operation of the entire Sink core or provide status information that is not associated with a particular channel (port) or packet. These signals are defined in Table 3-2.

There are six global status signals:

- **Sink Out-of-Frame** (`SnkOof`). Asserted active high whenever the core loses synchronization with the SPI-4.2 interface.
- **Sink Bus Error Status** (`SnkBusErrStat[7:0]`). Asserted when a SPI-4.2 protocol violation or an error that is not associated to one particular data packet occurs. These signals do not align with `SnkFFData`. These signals are triggered concurrently with the `SnkBuErr` signal. Each bit of the `SnkBusErrStat` bus corresponds to one of the following detected conditions:
  - `SnkBusErrStat[0]`: Minimum SOP spacing was violated.
  - `SnkBusErrStat[1]`: EOP control word not immediately preceded by data. (Example: EOP followed immediately by another EOP)
  - `SnkBusErrStat[2]`: Payload control word not immediately followed by data. (Example: A payload control word is followed immediately by another payload control word.)
  - `SnkBusErrStat[3]`: DIP4 error received during idles or training patterns.
  - `SnkBusErrStat[4]`: Reserved control words received.
  - `SnkBusErrStat[5]`: Control word with payload bit not set and non-zero address (excluding Training Control word).
  - `SnkBusErrStat[7:6]`: Tied to zero. (reserved)

  If the core receives two (or more) back-to-back payload control words, the last one received is used and the others are discarded. If the core receives two (or more) EOPs back-to-back, the first one is used and the others are discarded. For more information see Error Handling, page 101.

- **Sink Bus Error** (`SnkBusErr`). Asserted active high when any error condition triggers the Sink Bus Error Status bus. *`SnkBusErr`* is triggered concurrently with *`SnkBusErrStat`*.

  For each SPI-4.2 protocol violation or error that triggers `SnkBusErr` or `SnkBurErrStat`, these signals will be asserted for at least one `RDClkDiv` clock cycle translated into the `SnkFFClk` domain.

- **Sink Training is Valid** (`SnkTrainValid`). Asserted when valid training data is received. Figure 5-3 shows this signal in the timing diagram. As is shown, the *`SnkTrainValid`* signal is driven high for the duration of a complete training pattern after it has successfully been received.

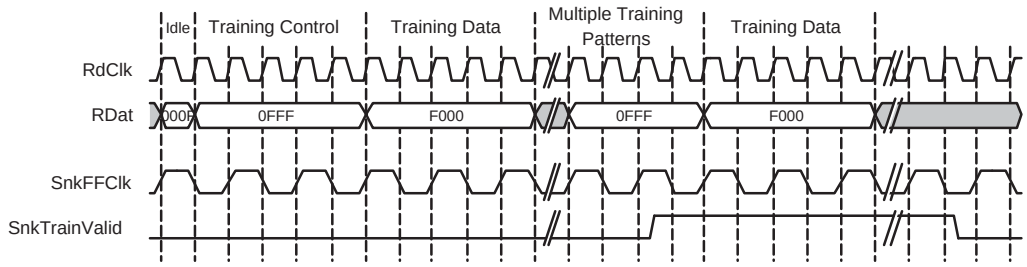


Figure 5-3: **Sink Training Valid Status**

- **SnkFifoReset_n.** Used to clear the FIFO (and the associated data path logic) while remaining in-frame. When `SnkFifoReset_n` is deasserted, the Sink data path will not write data into the FIFO until a packet with a valid SOP is received.
- **Reset_n.** Use to restart the entire Sink core. It causes the interface to go out-of-frame. When `Reset_n` is deasserted, the Sink core initiates the synchronization startup sequence.

Sink FIFO Interface Signals

The Sink FIFO Interface signals allow access to data (received on the SPI-4.2 Interface) stored in the FIFO. These signals are defined in Table 3-3, page 35. Waveforms illustrating the handshaking and FIFO status signals are shown in Figure 5-4 and Figure 5-5. The Sink FIFO Interface signals are synchronous to `SnkFFClk`, and the FIFO is 510 words deep. For 64-bit interface, a FIFO word is 1 credit wide and for 128-bit interface, a FIFO word is 2 credits wide.

Sink FIFO Almost Empty

Figure 5-4 shows the Almost Empty (`SnkFFAlmostEmpty_n`) status signal. As is shown in this waveform, the Almost Empty flag is asserted with the second-to-last word read out of the FIFO. When this signal is asserted (active low), it indicates that one word remains in the FIFO, and you should deassert the read enable signal on the next clock cycle. The Sink FIFO read logic will then evaluate the `SnkFFEmpty_n` signal to verify that there is no data in the FIFO, should an additional word have been simultaneously written into the FIFO. An example of this is provided with the SPI-4.2 core in the Design Example (see the `pl4_fifo_loopback_read.v/vhd file`.) This example also illustrates the Sink FIFO Valid signal, which is asserted while there is valid data on the data bus.

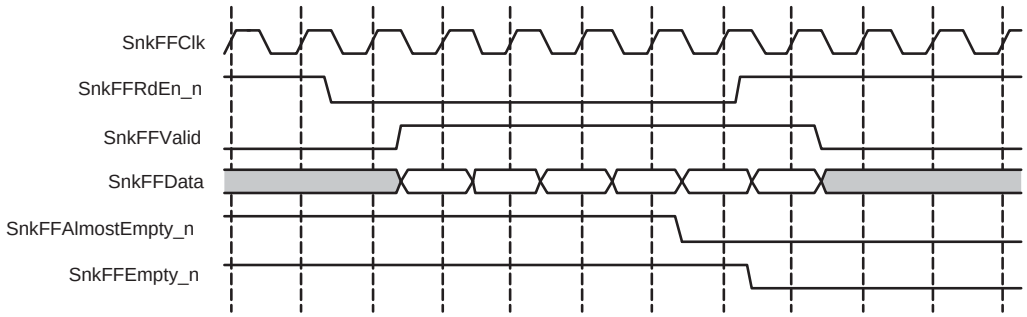


Figure 5-4: **Sink FIFO Almost Empty**

Sink FIFO Empty

Figure 5-5 shows the Empty (`SnkFFEmpty_n`) status signal. As shown in the waveform, the empty flag is asserted with the last word read out of the FIFO. In this example, the Almost Empty flag is asserted prior to a read access being initiated. In this case, there is one data word remaining in the FIFO. To access this word, assert the Sink FIFO Read Enable (`SnkFFRdEn_n`) signal for one cycle.

The Sink FIFO Empty and Sink FIFO Almost Empty are not actual indicators of the status of the Sink FIFO. These flags indicate whether data is ready to be read out from the read pipeline of the FIFO interface. Hence, its possible that a number of data have been written into the FIFO, but the Sink FIFO Empty and Almost Empty flag is asserted. Data written into the FIFO may not be ready to be read out until a number of cycles later.

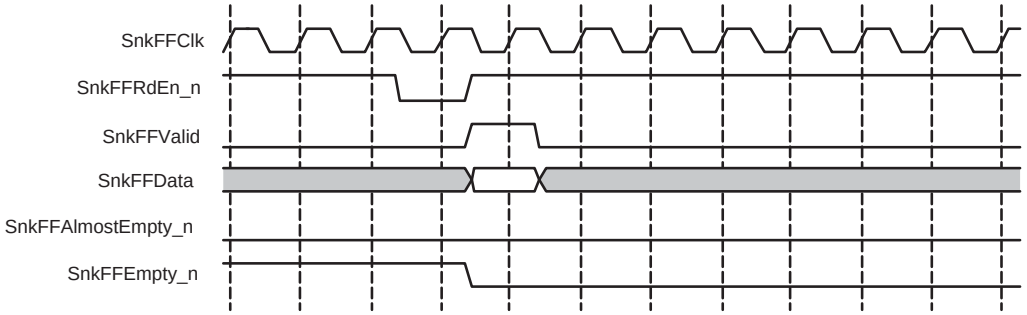


Figure 5-5: **Sink FIFO Empty**

Sink Almost Full

The behavior of Sink Almost Full flag (`SnkAlmostFull_n`) is dependent on the static configuration signals `SnkAFThresAssert` and `SnkAFThresNegate`. When the `SnkAlmostFull_n` flag is asserted, `SnkAFThresAssert` specifies the number of empty FIFO locations available. For a 64-bit user interface, each FIFO location can contain up to one credit (16 bytes) worth of data from a single packet. For a 128-bit user interface, each FIFO location can contain up to two credits (32 bytes) worth of data from a single packet. `SnkAFThresNegate` specifies when the `SnkAlmostFull_n` flag is deasserted.

The number of bytes that can be written into the SPI-4.2 Sink interface after the Sink Almost Full flag is asserted depends on the received packet sizes, data patterns, and the operations occurring on the Sink user interface. Configure the `SnkAFThresAssert` value based on your system requirements.

See FIFO Almost Full Mode and Sink Almost Full, page 90 for a description of the behavior of Sink FIFO interface when the Sink Almost Full flag is asserted.

Sink Overflow

The assertion of Sink Overflow flag (`SnkOverflow_n`) indicates that there is a write operation attempted on the FIFO when there are no empty FIFO locations available. This results in data loss, since no more data will be written into the FIFO until it is no longer in a full state. When the overflow condition occurs, reset the FIFO, because data corruption has occurred. To avoid the overflow condition, use the Sink Almost Full flag to gauge the readiness of the Sink core to receive data. See FIFO Almost Full Mode and Sink Almost Full, page 90.

There is a special case of overflow that occurs when the SOP spacing of the SPI-4.2 data is violated. This special case can be differentiated from the normal overflow condition by the absence of an asserted Almost Full Flag. See Error Handling, page 101.

FIFO Flag Consideration

The `SnkAlmostFull_n` output is an indicator of the amount of data in the Sink FIFO, or more correctly it indicates if the number of unused FIFO storage locations is below the programmed threshold. In systems requiring implementation of flow control this output should be used for user flow control algorithms.

In contrast, the `SnkFFEmpty_n` and `SnkFFAlmostEmpty_n` outputs are not actually indicators of an absence of any data in the Sink FIFO. Rather, these two signals indicate that the Sink (user) FIFO interface does not, or after the next read operation will not, have new data available. Consequently it is possible for Empty/Almost Empty to be asserted when there is actual data in the Sink FIFO. Under certain conditions, these flags may appear to contradict each other, even though they are both active and correct.

There are two related conditions that can occur in Full Burst Read mode:

1. Sink FIFO under run.

In any system where the absolute bandwidth of the read interface exceeds (or even matches) the bandwidth of the SPI-4.2 interface, it is possible for the interface to Sink FIFO to empty the FIFO faster than new data can be written. This can potentially cause the read pipeline in the Sink core to become empty or starved for data, which in turn will cause assertions of Empty/Almost Empty, indicating that new data is not (will not) be available. Due to internal logic in the core that prevents data starvation from occurring in the middle of a SPI-4.2 burst, this condition can occur at the end of a data burst, even if a new data burst has already commenced on the SPI-4.2 bus interface. In fact, the second burst will not become available (at the Sink FIFO read interface) until sometime after its terminating control word is received and processed by the SPI-4.2 Sink core.

2. Sink FIFO under run with SnkAlmostFull_n asserted.

This condition will only occur if the Almost Full Threshold is very large (activates even with minimal data storage). If the Almost Full Threshold is set to a level below the maximum burst size on the SPI-4.2 bus, then large (long) SPI-4.2 burst will cause the Sink FIFO to fill up to a point where `SnkAlmostFull_n` may be asserted even though the Empty/Almost Empty flags are indicating data not ready on the read interface. This occurs because new burst are not indicated to the read logic until the burst is terminated by a SPI-4.2 control word. If this behavior is deemed undesirable, it can be avoided by setting the Almost Full threshold so it not asserted if the FIFO contains less that the largest system burst size plus 24 (for internal pipeline delays).

Following is an example:

Largest Burst of data received: 16 credits = 256 bytes = 16 FIFO words (64-bit interface)

FIFO depth = 510 words

Max Threshold = 510 - (16 + 24) = 470

As already stated, the flags are always correct in their context even if they appear contradictory. Read interfaces with significantly higher bandwidth than the SPI-4.2 interface will periodically become starved for data causing `SnkFFEmpty` and `SnkFFEmpty_n` assertions regardless to the presence or absence of additional (incomplete) data bursts in the Sink FIFO. The `SnkAlmostFull_n` is the correct indicator of the absolute state of the Sink FIFO (within small clock cycle processing latency) and consequently is the correct flag for automated or user-implemented flow control.

Sink Status and Flow Control Signals

The Sink Status FIFO interface allows flow control data to be sent on the SPI-4.2 Interface. The channel order and frequency that status is sent is user-programmed in the calendar. A two-bit register is provided for each location in the calendar to store the channel status information (hungry=01, starving=00, satisfied=10). Figure 5-6 illustrates how calendar information determines the order and frequency that FIFO channel status information is transmitted on *`RStat`*. A detailed description of the calendar interface and the Status FIFO interface follows. A summary of Sink Status Path signals and definitions is provided in Table 3-4 and Table 3-5.

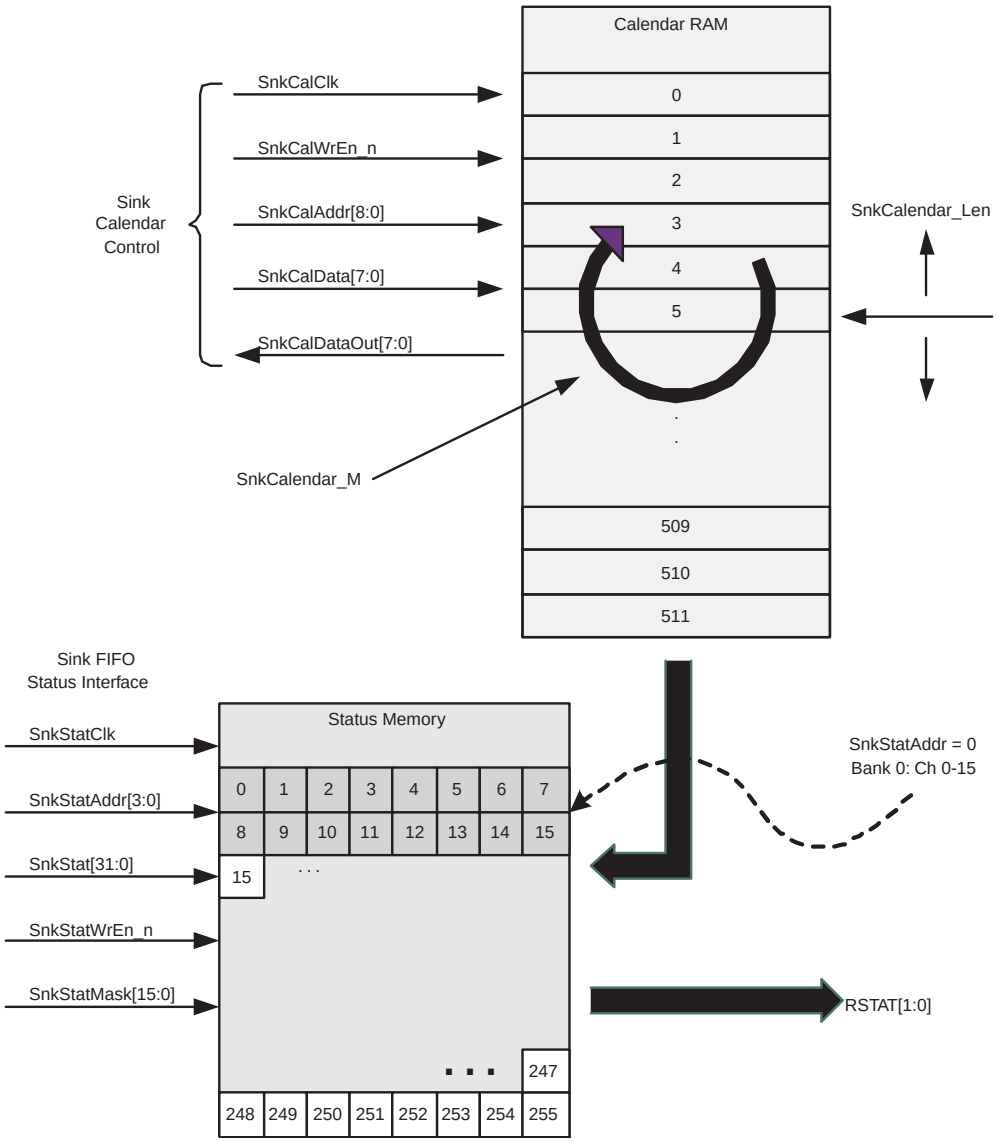


Figure 5-6: **Status FIFO Calendar and Status Memory Block Diagram**

Sink Calendar Initialization

There are two ways to initialize the Sink Calendar. One way is to load a COE file in the CORE Generator GUI, which puts the contents of the COE file into the UCF file (see Chapter 4, Generating the Core) The other way is to initialize the Calendar in-circuit at startup, described in the next section.

Initializing the Calendar In-Circuit

At startup, the Sink Calendar buffer can be programmed by first deasserting Sink Enable (`SnkEn`), then using the calendar write enable, address bus, and data bus. `SnkCalAddr` is used to indicate the location in the calendar buffer, and `SnkCalData` is used to indicate the channel number that should be written into that location. When outputting `RStat`, the status for the channel written to `SnkCalAddr`=0 is output first, followed by

SnkCalAddr=1, etc., until the end of the Calendar is reached, as defined by SnkCalendar_Len.

The waveform in Figure 5-7 illustrates the programming of the Sink Calendar. In this example, SnkCalendar_Len is set to five and SnkCalendar_M is set to zero (indicating that the calendar length is six, and should be repeated once). This means that the Sink Calendar will be expected to drive the FIFO Status Channel data (onto the SPI-4.2 bus) in the following sequence: status for channel 3, status for channel 0, status for channel 1, status for channel 2, status for channel 3, and status for channel 0.

Use the Sink Calendar Data Out bus (SnkCalDataOut[7:0]) to verify what has been programmed into the calendar buffer. When the calendar write enable signal is deasserted, the data stored in the location specified by the calendar address is driven onto the SnkCalDataOut bus. See Appendix F, SPI-4.2 Calendar Programming for more examples of programming the SPI-4.2 Calendar.

Note: For a 1-channel system, you do not need to program the Calendar. By default, all locations are set to zero.

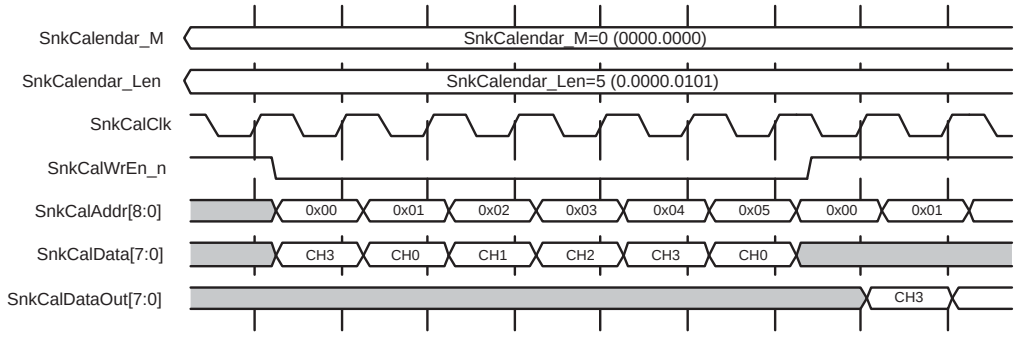


Figure 5-7: **Sink Calendar Initialization**

Sink Flow Control

Typically, there are two ways to implement the SPI-4.2 Sink flow control:

- **Automatic**. The SPI-4.2 Sink core can be configured to perform flow control automatically for a single channel system or for a system that does not require flow control on a per-channel basis. See FIFO Almost Full Mode and Sink Almost Full, page 90.
- **Manual**. The Sink core provides an interface that can be fully-customized if per-channel flow control is required. A typical implementation is shown in Figure 5-8. In this implementation, external FIFOs are used to provide additional per-channel storage and to facilitate per-channel flow control. A programmable full indication on a user's individual FIFOs can be used to drive the status interface of the Sink core. This provides flexibility in implementing optimal flow control for your system requirements.

  When implementing large channel solutions, individual FIFOs may be shared by sets of channels or alternative approaches may be implemented to minimize external logic requirements.

The Sink Status FIFO interface has a 32-bit bus for all channel configurations; for example, whether the core is configured for 4, 128, or 256 channels. This allows you to write the FIFO Status Channel data for 16 channels at a time. There are four address lines for selecting which 16 channels you are accessing.

Note: Address lines can be permanently set to zero for systems using 1-16 channels.

The latency between the user interface and SPI-4.2 Interface for the Sink Status Path is seven `RSClk` cycles and one `SnkStatClk` cycle.

You can configure the Sink core to write status for 16 channels per clock cycle. Use the `SnkStatAddr` bus to select the 16 channels. The core supports configurations of 1-256 channels.

The 16 channels of FIFO Status are addressed as follows:

`Bank 0: SnkStatAddr[3:0]`=0 for channels 15 to 0

`Bank 1: SnkStatAddr[3:0]`=1 for channels 31 to 16

`Bank 2: SnkStatAddr[3:0]`=2 for channels 47 to 32

`Bank 3: SnkStatAddr[3:0]`=3 for channels 63 to 48

...

`Bank 14: SnkStatAddr[3:0]`=14 for channels 239 to 224

`Bank 15: SnkStatAddr[3:0]`=15 for channels 255 to 240

The status is mapped to the 16-bit bus as follows:

For Bank 0: `SnkStatAddr[3:0]`=0

`SnkStat[1:0]` => Channel 0, where `SnkStat[1]` is the MSB of the 2-bit status

`SnkStat[3:2]` => Channel 1

`SnkStat[5:4]` => Channel 2

...

`SnkStat[11:10]` => Channel 13

`SnkStat[13:12]` => Channel 14

`SnkStat[15:14]` => Channel 15

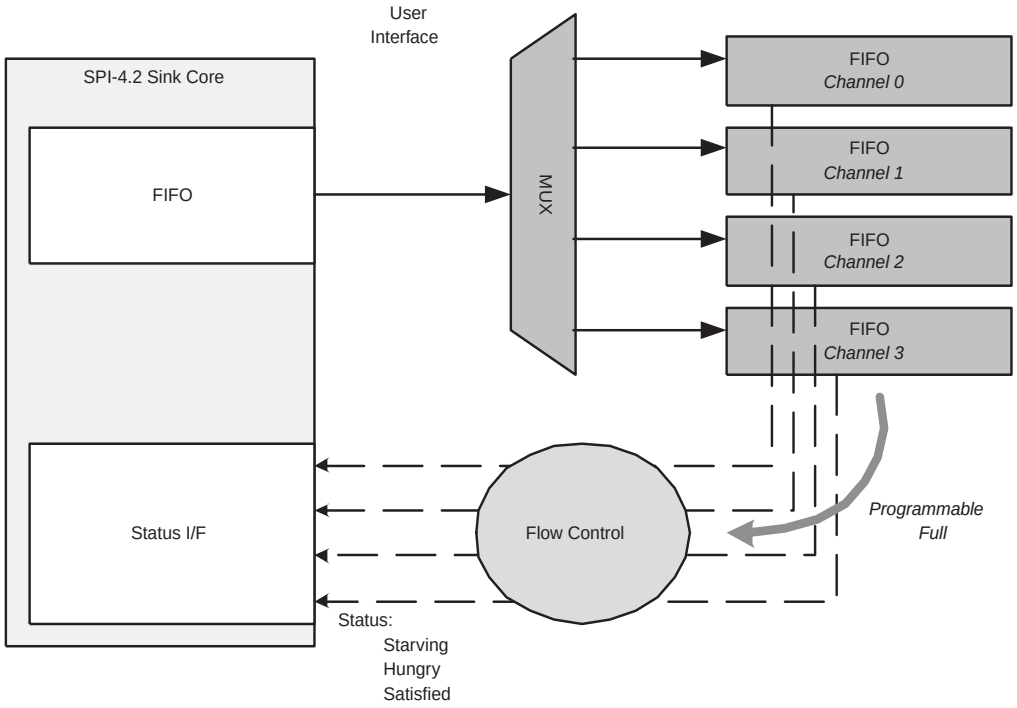


Figure 5-8: **Typical Flow Control Implementation for 4-Channel System**

Sink Status FIFO Interface: Example 1

Figure 5-9 illustrates writing to the Status FIFO interface for a 10-channel SPI-4.2 Sink core. Because there are fewer than 17 channels, the Sink Status Address bus (`SnkStatAddr[3:0]`) is permanently tied to zero. In this example, the mask functionality is utilized to indicate that only 10 channels have valid status. The mask can change from clock cycle to clock cycle, but in this illustration it is fixed (`SnkStatMask`= 0x03FF).

The Sink Status Write signal (`SnkStatWr_n`) is used to write status values to be transmitted on the SPI-4.2 Interface in the order specified by the calendar buffer. The status written in this example. The status data on `SnkStat[31:0]` is represented in hexadecimal.

Write Cycle	Starving Status	Satisfied Status
0,1,2,3	CH 0-9	none
4	CH 1-9	CH 0
5	CH 1,2, 4-9	CH 0,3
6-7	CH 4-9	CH 0,1,2,3
8	CH 0	CH 1-9

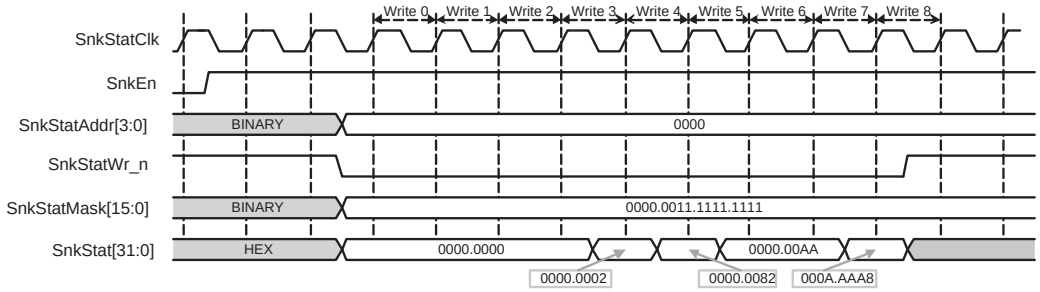


Figure 5-9: **Sink Status FIFO Interface Example 1: 10-channel Configuration**

Sink Status FIFO Interface: Example 2

This example illustrates writing to the Status FIFO Interface for a 64-channel SPI-4.2 Sink core (Figure 5-10). Address the following four banks, depending on the status of the channel that is being updated, to write status for 64 channels:

- `Bank 0: SnkStatAddr[3:0]`= 0000, for channels 15 to 0
- `Bank 1: SnkStatAddr[3:0]`= 0001, for channels 31 to 16
- `Bank 2: SnkStatAddr[3:0]`= 0010, for channels 47 to 32
- `Bank 3: SnkStatAddr[3:0]`= 0011, for channels 63 to 48

The mask (`SnkStatMask[15:0]`) is used to update only the channels for which FIFO status has changed. The status written is shown below.

Write Cycle	Status Address	Status Mask	Starving Status	Satisfied Status
0-1	Bank 0	1111.1111.1111.1111	CH 0-15	none
2	Bank 0	0000.0000.0000.0001	none	CH 0
3	Bank 1	1000.0000.0000.0000	none	CH 31
4-5	Bank 2	1111.1111.1111.1111	CH 32-47	none
6-7	Bank 3	1111.1111.1111.1111	CH 48-63	none
8-9	Bank 0	1111.0000.0000.0000	none	CH 12-15

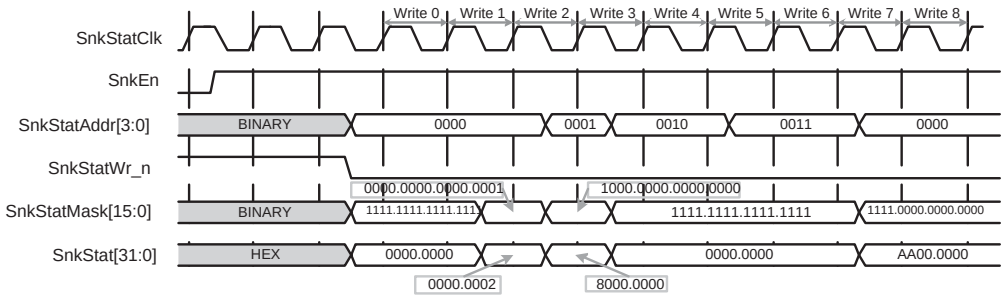


Figure 5-10: **Sink Status FIFO Interface Example 2: 64-channel Configuration**

Sink Status FIFO Status Interface: Example 3

This example illustrates status received on the user interface and written to the SPI-4.2 bus. Figure 5-11 shows a *`RStat`* waveform for a calendar length of four

(SnkCalendar_Len=3) and calendar repetition value of one (SnkCalendar_M=0). FIFO status information is periodic, repeating the sequence of a framing pattern (11), a repeated set of FIFO status words (SnkCalendar_M + 1 times) in accordance with the programmed calendar order, and a DIP-2 value. The programmed calendar sequence is channel 0, 1, 2, 3, and the following RStat[1:0] sequence is as follows.

- Sequence #: CH0, CH1, CH2, CH3
- Sequence 1: 10, 00, 00, 00
- Sequence 2: 10, 00, 10, 10
- Sequence 3: 10, 10, 10, 10

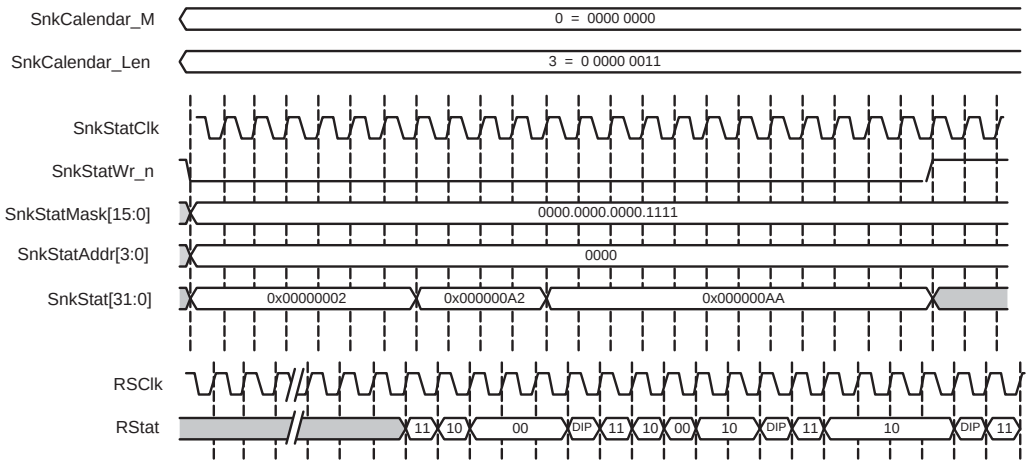


Figure 5-11: **Sink Status Path - User Interface to SPI-4.2 Interface**

Insertion of DIP2 Errors

Use the Sink core to force the insertion of DIP2 errors for use during system testing and debugging. This is supported by utilizing the SnkDIP2ErrRequest signal. When the SnkDIP2ErrRequest signal is asserted, the next DIP2 value sent on RStat will error. The erroneous DIP2 value is an inversion of the correctly calculated DIP2.

Sink Static Configuration Signals

The Sink static configuration signals are inputs to the Sink core, statically driven to determine the behavior of the core. See Table 3-6 for a full list of static configuration signals.

Two of the Sink static configuration signals can be changed in-circuit. There are static registers for SnkCalendar_M and SnkCalendar_Len that are synchronous to SnkStatClk. To change these parameters while the core is operational, you must first deassert SnkEn.

All Sink Static Configuration signals can also be changed in-circuit when the core is *not* in operation (disabled and in reset state).

The following steps are recommended when changing static configuration signals:

1. Disable the sink core (SnkEn signal).
2. Assert core reset (Reset_n = 0).

3. Change the desired static configuration signals.
4. Deassert reset (Reset_n=1).
5. Wait at least 10 clock cycles of RDClkDiv_GP for the sink static configuration signals to settle and propagate to the Sink core's logic.
6. Enable the core and wait for the core to achieve synchronization, then continue normal operation.

FIFO Almost Full Mode and Sink Almost Full

You can select the behavior of the Sink core when it is almost full by setting the static configuration signal Sink FIFO in Almost Full Mode (`FifoAFMode[1:0]`). Figure 5-12 through Figure 5-14 are timing diagrams illustrating the behavior of the core for each of the three modes.

FIFO Almost Full Mode “00”

When the FIFO Almost Full Mode (`FifoAFMode`) is set to “00” and the Sink core becomes Almost Full, the Sink interface will go out-of-frame, and the Sink Status logic will send the framing sequence “11” until `SnkAlmostFull_n` is deasserted and the Sink core transitions back to in-frame. This is illustrated in Figure 5-12.

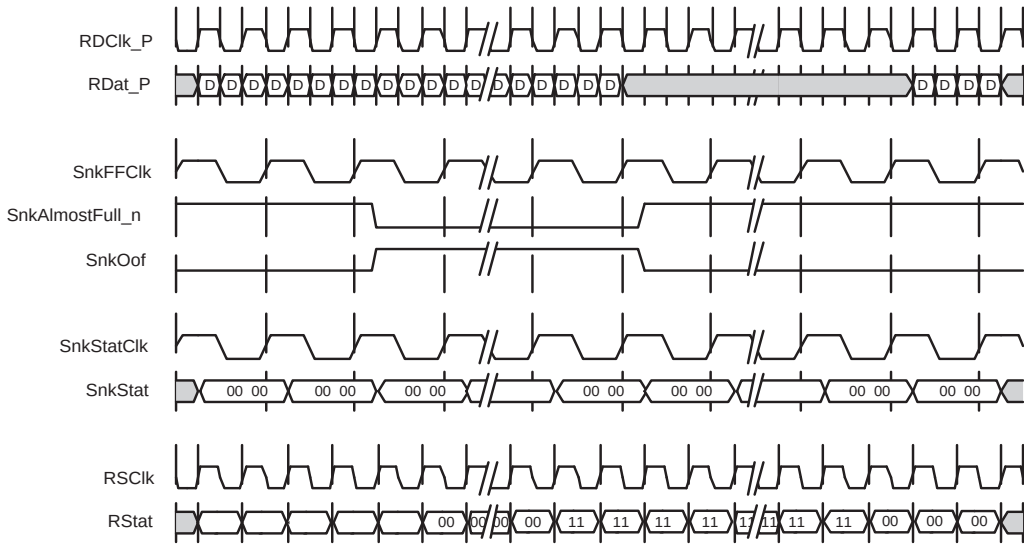


Figure 5-12: **FIFO Almost Full Mode “00”**

FIFO Almost Full Mode “01”

When the FIFO Almost Full Mode (`FifoAFMode`) is set to “01” and the Sink core becomes Almost Full, the Sink interface will remain in-frame (`SnkOof` deasserted), and the Sink Status logic will send satisfied (“10”) on all channels until `SnkAlmostFull_n` is deasserted. This is illustrated in Figure 5-13.

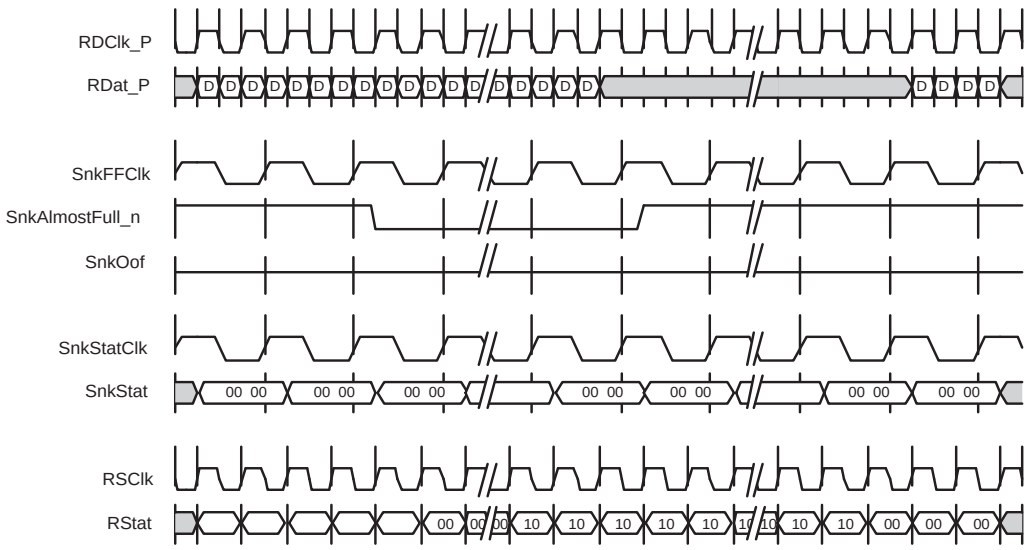


Figure 5-13: **FIFO Almost Full Mode "01"**

FIFO Almost Full Mode 10 or 11

When the FIFO Almost Full Mode (`FifoAFMode`) is set to 10 or 11 and the Sink core reaches the almost-full state, the Sink Status logic will continue in normal operation (Figure 5-14). In this case, take immediate action to prevent FIFO overflow and loss of data.

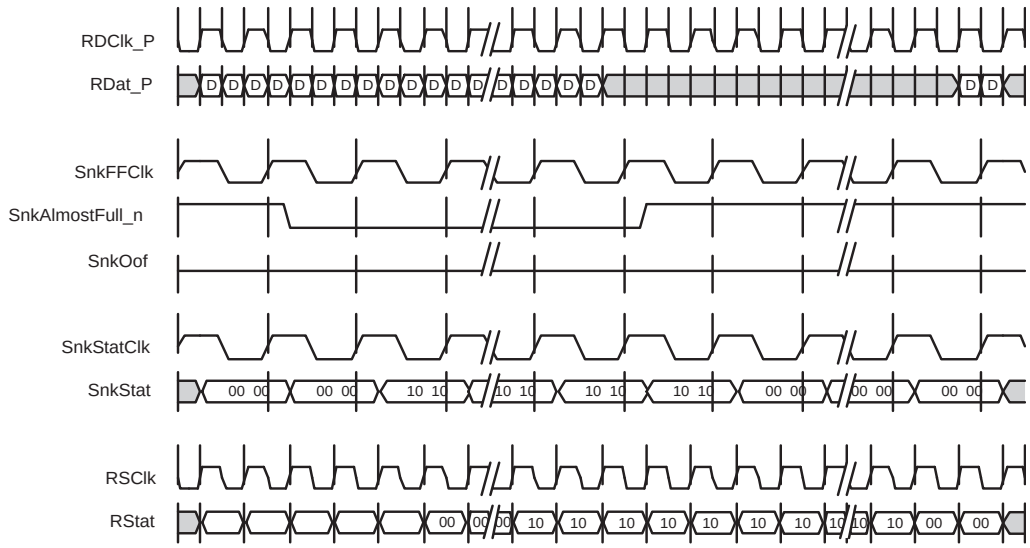


Figure 5-14: **FIFO Almost Full Mode "10" or "11"**

Sink Data Capture Implementation

The SPI-4.2 Sink supports static alignment and two variations of dynamic phase (automatic) alignment of the ingress *`RDClk`* to *`RDat[15:0]`* and `RCtl` signals using the training patterns defined in the SPI-4.2 standard. Selection of the alignment scheme only modifies the Sink core. The Source core does not change, regardless of the alignment method selected. The Source core does support configuration of appropriate training

intervals for a corresponding receiver alignment. Static alignment should not be used in systems where *RDClk* will exceed 350 MHz and either of the two Dynamic Phase Alignment (DPA) schemes is recommended for all Virtex-6, Virtex-5 and Virtex-4 FPGA designs.

The new IDELAY feature in Virtex-6, Virtex-5 and Virtex-4 FPGAs allows you to define the SPI-4.2 I/O pinouts without concern for the chosen alignment scheme. The ChipSync™ technology coupled with an enhanced DPA design result in an automatic alignment implementation that uses significantly fewer FPGA resources and power than DPA implementations seen in prior Virtex FPGA architectures. Performing alignment with one of the two DPA options is recommended for all SPI-4.2 designs above the nominal 311Mhz `RDClk` frequency.

Dynamic Phase Alignment (Virtex Devices Only)

The Sink core can be configured to support dynamic phase alignment (DPA) on the incoming source synchronous SPI-4.2 data stream (`RCtl` and `RDat[15:0]` with respect to the `RDClk`) for all supported `RDClk` frequencies greater than, or equal to 220 MHz. From a system timing standpoint, each bit of the SPI-4.2 bus is aligned to the `RDClk` independent of any other bit of the bus. This increases the system timing margin by completely removing bit-to-bit data skew as a detractor from I/O timing calculations. After the optimal sampling point (to within one IDELAY tap) of each SPI-4.2 bus bit is determined, the proper bus timing is reconstructed using the SPI-4.2 training pattern as an expected reference. Provided each bus bit has a sufficient data eye, the DPA solution can recover and reconstruct the SPI-4.2 bus even in the presence of bit-to-bit skew on the bus of over +/- 1 Unit Interval (UI).

The FPGA fabric logic resources required for DPA have been significantly reduced. The default DPA configuration requires approximately 330 slices, with continuous DPA requiring an additional 60 slices. Due to the new smaller footprint, and the fact that this logic now operates at one-half the `RDClk` clock frequency, the added power dissipation is less than 200 mW when compared to a static alignment configuration.

From a black-box perspective, there is no difference in the Sink core interface definition for static and dynamic alignment options (aside from the DPA ports). The use of dynamic alignment does not relax the PCB design requirements for high-speed differential signals. Each bit-pair of the bus requires the same PCB routing (matched length and controlled impedance traces), given the high speed of the SPI-4.2 LVDS pairs.

For more information about DPA operation and guidelines, see the *SPI-4.2 Dynamic Phase Alignment White Paper*, available on the SPI-4.2 Product Page.

Dynamic Alignment Implementation Considerations

The Sink user interface contains one input and five output signals that are used for dynamic alignment, and are summarized in Table 5-4. The dynamic alignment algorithm uses a training sequence on the SPI-4.2 bus to complete the alignment.

If the “DPA Wait For Training Control” feature is enabled, DPA will check for the presence of a training control word (`RDat[15:0]` = 0fff and `Rctl` = 1) on the SPI-4.2 bus before starting the alignment process. Undefined data "X" or HiZ should not be sent on RDat or RCtl even if option "DPA Wait For Training" is selected. If the training control word is not detected within a time interval (Alignment Test Interval * 32) after `PhaseAlignRequest` is pulsed `SnkDPAFailed` is asserted. If “DPA Wait for Training Control” feature is not enabled, the DPA logic will start the alignment process right after `PhaseAlignRequest` is pulsed.

If PhaseAlignRequest is activated when the Sink logic is not receiving a SPI-4.2 training sequence (continuos training patterns), the logic may fail to achieve lock. There is a remote possibility that PhaseAlignComplete could be asserted without being properly aligned. If PhaseAlignRequest is activated when the Sink logic is only sending idles (for example, the transmitting PHY is not powered), the DPA logic will not assert PhaseAlignComplete if any of the SPI-4.2 inputs are not toggling.

Instead, the SnkDPAFailed signal will assert upon completion of the alignment sequence. Cores configured with auto-retry will immediately initiate a new alignment sequence at this point and continue to attempt alignment unless PhaseAlignRequest is held high. The "auto-retry" features is only applicable after PhaseAlignRequest has been initiated and SnkDPAFailed has been asserted. Enabling this option does not automatically initiate the DPA alignment when the core becomes out of frame. In this event, you need to perform the startup sequence, and initiate PhaseAlignRequest.

SnkDPAFailed is asserted for one SnkFFClk cycle at the end of each unsuccessful alignment attempt. Note all DPA logic operates on RDClk input and if RDClk is not toggling, PhaseAlignRequest transitions will not initiate alignment.

Every time PhaseAlignRequest is asserted (transitions from low-to-high), the core is forced to go out of frame (SnkOof is asserted) and aborts the current alignment. This state causes the core to send framing patterns (all 11s) on its FIFO status channels. The SPI-4.2 specification requires that the corresponding PHY transmit device respond to this condition by sending continuos training sequences. Alignment is initiated only on high-to-low transition of PhaseAlignRequest. This behavior of PhaseAlignRequest enables the user to force to go out of frame (asserting PhaseAlignRequest), wait for the corresponding Transmit device respond with training patterns (hold PhaseAlignRequest high) and restart alignment when training patterns are received (deassert PhaseAlignRequest).

Table 5-4: **Dynamic Alignment Signals**

Signal Name	Direction	Clock Domain	Description
PhaseAlignComplete	Output	SnkFFClk	**Phase Alignment Complete.** Active high signal that indicates phase alignment is complete.
PhaseAlignRequest	Input	SnkFFClk	**Phase Alignment Request.** Initial DPA commences by asserting and deasserting PhaseAlignRequest. DPA starts on a high-to-low transition. When PhaseAlignRequest transitions from low-to-high SnkOof will be driven high (core goes out of frame).
SnkDPAFailed	Output	SnkFFClk	**Phase Alignment Failed**. Active high signal that indicates phase alignment has failed.
SnkDPARamAddr [5:0]	Output	RDClkDiv_GP	**Phase Alignment RAM Address.** Bus indicating the ISERDES tap value that corresponds to the data on SnkDPARamData.

Table 5-4: **Dynamic Alignment Signals *(Cont'd)***

Signal Name	Direction	Clock Domain	Description
SnkDPARamData [16:0]	Output	RDClkDiv_GP	Phase Alignment RAM Data. Initial data collected during alignment. Used to find the valid data window for each bit of the SPI-4.2 bus. An active high on the bus indicates that sampling on the ISERDES tap corresponding to SnkDPARamAddr will result in sampling within a valid data window. Each index corresponds to a bit on the SPI-4.2 bus; RDat(0) is index 0, RDat(1) is index 1, ..., RCtl is index 16.
SnkDPARamValid	Output	RDClkDiv_GP	Phase Alignment RAM Valid. Active high signal indicating the information on SnkDPARamData and SnkDPARamAddr is valid.
SnkCDPAHalt (optional)	Input	RDClkDiv_GP	Phase Alignment DPA Halt. Active high signal halts the pointer adjustment of continuous DPA operation.
SnkDPADiagWin (optional)	Input	RDClkDiv_GP	Phase Alignment DPA Diagnostics. Active high signal that enables the user to find the valid data window during operation for each bit of the SPI-4.2 bus. After SnkDPADiagWin is pulsed, the valid data window information is presented on SnkDPARamAddr [5:0] and SnkDPARamData [16:0] when SnkDPARamValid is asserted.
SnkDPAAddrRst (optional)	Input	RDClkDiv_GP	Phase Alignment DPA Address Reset. Active high signal that clears the SnkDPARamAddr counter.
SnkDPAAddrEn (optional)	Input	RDClkDiv_GP	Phase Alignment DPA Address Enable. Active high signal that enables the SnkDPARamAddress counter.

Table 5-4: **Dynamic Alignment Signals *(Cont'd)***

Signal Name	Direction	Clock Domain	Description
SnkDPAClkDlyRst (optional)	Output	RDClkDiv_GP	Phase Alignment IDELAY Reset. Active high signal used to reset the IDELAY inserted in the RDClk path of the external sink clocking module. This option is only available when the "DPA Clock Adjustment" feature is enabled and the Virtex-6 family is selected.
SnkDPAClkDlyCe (optional)	Output	RDClkDiv_GP	Phase Alignment IDELAY Clock Enable. Active high signal used to enable the increment of the IDELAY inserted in the RDClk path of the external sink clocking module. This option is only available when the "DPA Clock Adjustment" feature is enabled and the Virtex-6 family is selected.
SnkDPAClkDlyInc (optional)	Output	RDClkDiv_GP	Phase Alignment IDELAY Increment. Active high signal used to increment the delay value in the IDELAY inserted in the RDClk path of the external sink clocking module. This option is only available when the "DPA Clock Adjustment" feature is enabled and the Virtex-6 family is selected.

All bits of the SPI-4.2 bus (v7.4 or higher) are now aligned in parallel (rather than sequentially). This significantly reduces the alignment time both in simulation and in the actual hardware. Alignment time is only dependent on the `RDClk` frequency and the chosen alignment test interval and is approximately.

Time ~= `RDClk` period * 128 * (Alignment Test Interval + 7)

Lower test intervals may be selected through the GUI. However, test intervals below 128 are not recommended for hardware implementations. Smaller values are allowed in the CORE Generator GUI to support faster simulation of DPA alignment. It is also recommended that the product of the Alignment test interval and the Master-Slave IDELAY offset always exceed 240. The Master-Slave offset relates inversely to the expected data eye (in IDELAY taps) and therefore should not be set to large values in fast or noisy systems. For example, choosing an offset of 10 may result in alignment failures unless the data eye of all the inputs is at least 825 ps (10 + 1 IDELAY taps). Best initial alignments will be achieved with small Master-Slave IDELAY offsets and larger alignment test intervals.

Users are strongly recommended to use the alignment test interval of 128 and master-slave IDELAY offset of 2. These settings have been tested to work in hardware for systems up to 1 Gbps. These values should be changed only after consulting Xilinx.

The DPA logic has the following debug ports: `SnkDPARamAddr`, `SnkDPARamData` and `SnkDPARamValid.` These ports present the user with the data collected by the logic while finding the data valid window for each of the SPI-4.2 data and control bits (between assertion of `PhaseAlignRequest` to `PhaseAlignComplete`). See Table 5-4 for more information. Additional debugging information is also available when invoking the DPA status monitoring feature. See DPA Status Monitoring, page 97.

The DPA also has the following advance diagnostic ports: `SnkDPADiagWin`, `SnkDPAAddrRst`, and `SnkDPAAddrEn`. These input ports enable you to measure and examine the valid data window for each channel during normal operation. See the section Advance DPA Diagnostic Portsfor more information.

DPA Clock Adjustment

Selection of this feature will cause the DPA alignment to perform a clock alignment phase prior to the alignment of the individual data bits sampling points. This feature should only be invoked if there is an aggregate data window; for example, a sampling point (clock delay) that is valid for all bits of the bus. DPA adjustment of the clock delay prior to per-bit alignment will produce lower final IDELAY tap settings for the data bits. This minimizes the effects of IDELAY tap pattern jitter on system timing. If there is no clock delay that provides valid data for the entire bus, the clock adjustment phase will fail and assert `SnkDPAFailed`. The DPA logic may choose to adjust the clock up or down, depending on the initial data test. Consequently, the initial clock delay should be in the middle of the IDELAY range (default is 32) or at least as many IDELAY taps as correspond to 1 UI.

Continuous Alignment Considerations

This option is an enhancement that maybe useful in a system where the SPI-4.2 bus timing is changing over time due to voltage temperature or other variations. This is not typical, as most SPI-4.2 sources have a data/clock phase relationship that is fixed by design.

The default DPA implementation performs alignment only in response to `PhaseAlignRequest` after it has achieved alignment as indicated by the assertion of `PhaseAlignComplete` and `SnkTrainValid`. No further alignment is performed even when additional training patterns are received. Continuous alignment addresses this by performing a constant non-disruptive monitoring of the ingress data samples by using a second sample offset in time. The reference sample is skewed early and late relative to the data sample and differences are stored. If the reference matches the data at one test offset but not the other, the data timing will be bumped by one IDELAY tap to adjust the data sampling point. This process occurs continuously (and independently) on each bit of the bus, and does not depend on the presence of SPI-4.2 training patterns. Any data pattern with transitions will exercise the tracking feature.

The adjustment opportunity (for any, or all bits) occurs periodically at a rate of:

DPA update rate ~= UI * (Alignment Test Interval * 8 + 136)

Xilinx recommends that you configure the transmitting PHY device to send periodic training patterns to the Xilinx SPI-4.2 sink core once every DPA update interval to guarantee that data patterns with transitions are received to exercise the tracking feature. This especially relevant in systems where the transmitting SPI-4.2 device have long idle cycles.

The continuous alignment process can be halted by asserting the input port `SnkCDPAHalt`. When this port is deasserted, continuous alignment will continue.

The initial alignment requirements and sequences are identical to the default DPA configuration, and the additional logic overhead is extremely low (60 slices) when this feature is enabled.

DPA Status Monitoring

If invoked, this option will output DPA alignment information on `SnkBusErrStat` while the DPA logic is attempting initial alignment and before `PhaseAlignComplete` is asserted. `SnkBusErrStat` reverts back to its normal function as soon as `PhaseAlignComplete` is asserted. The function is intended to support diagnosis of alignment failures with Chipscope or similar logic probes. The bus index allows the probing to be targeted to a particular bit of the SPI-4.2 bus 0-15, or 16 for the `RDCtl` bit.

For more information about DPA operation and guidelines, see the *SPI-4.2 Dynamic Phase Alignment White Paper*, available in the SPI-4.2 Product Page.

Advance DPA Diagnostic Ports

Selection of this feature allows you to use Advance DPA Diagnostic Ports to measure and capture the data valid window for each channel during operation. After the `SnkDPADiagWin` is pulsed, the DPA logic traverses the 64-tap IDELAY to determine if the sampling for an IDELAY tap is within a valid window. The sampling information for each channel is captured and presented on the `SnkDPARamAddr` and `SnkDPARamData` bus when `SnkDPAValid` is asserted.

The signal `SnkDPADiagWin` should be used for diagnostics only as the normal continuous alignment process is halted when `SnkDPADiagWin` is asserted, and will not function correctly after that. The ports `SnkDPAAddrRst` and `SnkDPAAddrEn` can be used to clear the `SnkDAPRamAddr` counter, and enable the address counter to present captured data valid window information on the `SnkDPARamAddr` and `SnkDPARamData` ports.

Static Alignment

For Virtex architectures, the Sink core performs static alignment by shifting the clock relative to the 16-bit data so that the incoming clock edge is centered to the data eye of `RDat/RCtl`. For Virtex-5 and Virtex-4 device designs using global clocking distribution, this alignment can be performed by using a DCM. For Virtex-6 designs using global clocking distribution, the alignment can be performed by using a MMCM. For designs using regional clocking distribution, the IDELAY function is used to shift the clock in relation to the data bits.

The `PhaseAlignRequest` and `PhaseAlignComplete` signals are present but not used for the Static Alignment core. `PhaseAlignRequest` should be tied to a constant zero, and `PhaseAlignComplete` should be ignored.

Alignment Implementation Considerations for Global Clocking Configuration

The core also supports legacy static alignment, which uses a DCM or MMCM to phase shift the `RDClk`. The ability of the DCM or MMCM to shift the internal clock in small increments, enables `RDClk` to be shifted relative to the sampled data. For statically-aligned systems, the DCM or MMCM output clock phase offset is a critical part of the system. The static alignment solution, using the DCM or MMCM, assumes that the PCB is designed with precise delay and impedance matching for all LVDS differential pairs of the data bus. This assumption is critical as the DCM or MMCM does not compensate for deviations in delay between bits.

Determine the optimal DCM or MMCM setting (`PHASE_SHIFT`) to ensure that the target system will have the maximum system margin and performance across voltage,

temperature, and process (chip to chip) variations. Testing the system to determine the best DCM or MMCM PHASE_SHIFT setting has the added advantage of providing a benchmark of the system margin based on the UI (unit interval or bit time).

System Margin (ps) = UI(ps) * (working phase shift range/128)

Xilinx makes no recommendation for a single DCM or MMCM PHASE_SHIFT value that will be effective across all hardware platforms. Xilinx also does not recommend that you attempt to determine the PHASE_SHIFT setting empirically. In addition to the clock-to-data phase relationship, other factors such as package flight time (package skew) and clock routing delays (internal to the device) affect the clock data relationship at the sample point (in the IOB) and are difficult to characterize.

The optimal PHASE_SHIFT setting should be investigated during hardware integration and debugging. The phase shift setting provided with the SPI-4.2 core in the constraints file is only a place-holder, and has been determined to work on the Xilinx SPI-4.2 hardware platform. This default setting has changed for the various SPI-4.2 releases to account for changes to the DCM DESKEW ADJUST attribute. For information about finding the ideal phase shift value for your system, see the Xilinx SPI-4.2 Solution Record 16112.

Note: This alignment method should only be used with global clock distribution.

ISERDES Alignment Implementation Considerations for All Device Architectures

Static alignment can be performed using the IDELAY function when regional clock distribution is used. For Virtex devices, the ability of the IDELAY function to delay its input by small increments (75 ps) enables the internal RDCLK to be shifted relative to the sample data.

For statically aligned systems, the delay chain length is a critical path of the system. The static alignment solution assumes that the PCB is designed with precise delay and impedance matching for all LVDS differential pairs of the data bus. In this case, the primary alignment mechanism is time, shifting the internal RDCLK relative to the data bits using the IDELAY function.

Determine the optimal delay in the IDELAY function (IOBDELAY or IDELAY) to ensure that the target system will have the maximum system margin and performance across voltage, temperature, and process (chip to chip) variations. Xilinx cannot recommend a single delay value that will be effective across all hardware platforms. Xilinx also does not recommend that you attempt to determine the delay setting empirically. In addition to the clock-to-data phase relationship, other factors such as package flight time (package skew) and clock routing delays (internal to the device) affect the clock data relationship at the sample point (in the IOB) and are difficult to characterize. The optimal delay setting should be investigated during hardware integration and debugging. The delay setting provided with the SPI-4.2 core in the constraints file is only a place-holder.

Note: This alignment method should be used only with regional clock distribution.

Alignment Guidelines

There are several parameters to consider when choosing the best alignment option for your system:

- clock frequency
- available resources
- board layout

The Dynamic Phase Alignment feature operates on systems ranging from 220 MHz to 500+ MHz RDClk frequency. The Static Alignment feature operates on systems ranging from the minimum DCM or MMCM frequency for the architecture to 350MHz.

In terms of resources, the Dynamic Phase Alignment logic and its features adds approximately 330 slices to the design when compared to the Static Alignment.

The Dynamic Phase Alignment solution increases the system timing margin by completely removing bit-to-bit data skew as a detractor from data eye calculations and can recover and reconstruct the SPI-4.2 bus in the presence of a bit-to-bit skew on the bus over ± 1 UI. Additionally the Continuous Alignment feature enables the DPA logic to continuously track the data eye in systems where the SPI-4.2 bus timing is changing over time due to voltage and temperature or other variations.

In general, for systems operating above the nominal 350 MHz RDClk frequency, it is recommended that the Dynamic Phase Alignment with Continuous Alignment is used. The DPA Clock Adjustment feature should only be used if the traces for the data and control bits are approximately matched.

Synchronization and Startup

Once the Sink core has been initialized (Initializing the SPI-4.2 Core, page 72), the Sink core must be synchronized before data and status can be received and transmitted. Figure 5-15 shows a state machine diagram illustrating a Sink core startup sequence and error condition processing.

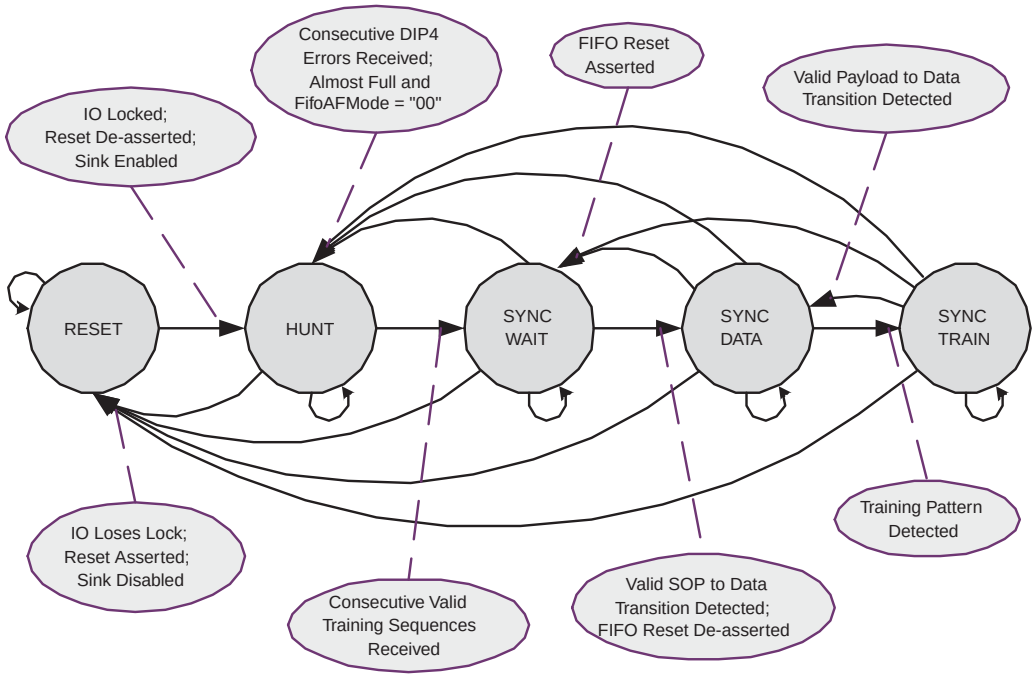


Figure 5-15: **Sink Startup Sequence State Machine**

Reset

The Sink core remains in the Reset state until the following conditions are true.

- SnkClksRdy is asserted.
- Reset_n is deasserted.

- SnkEn is asserted.
- Clock-Data Alignment is complete.
  - Static Alignment: Alignment is assumed to be correct.
  - Dynamic Alignment: Per bit deskew is performed, the eye of RDat[15:0] is aligned to the RDClk.
    - Assert PhaseAlignRequest for two user clock cycles.
    - Wait for PhaseAlignComplete to assert.

In Reset state, the Sink core transmits framing patterns (11) on RStat[1:0]. The core is out-of-frame in this state.

Hunt

The Sink core remains in the hunt state until a set number of consecutive training patterns are received, as defined by the parameter NumTrainSequences. In this state, the Sink core transmits framing patterns (11) on RStat[1:0]. The core is out-of-frame in this state.

Sync Wait

In the Sync Wait state, the Sink core has completed the startup sequence and is waiting to receive the first valid SOP to data transition on RDat.

The Sink core will remain in this state until the following conditions are true.

- SnkFifoReset_n is deasserted.
- The first valid SOP-to-data transition is received on RDat.

In this state, the Sink core continuously checks DIP-4 parity, and sends FIFO Channel status on RStat. The core is in-frame in this state. After the core is in frame, RStat will send satisfied status "10" until the user has written in the Fifo Channel Status.

Sync Data

In the Sync Data state, normal core operation is enabled.

In this state, the Sink core continuously checks DIP-4 parity, stores data received on RDat[15:0] into the Sink FIFO, and sends FIFO Channel status on RStat. The core is in-frame in this state.

Sync Train

The Sink core enters the Sync Train state when a training pattern is detected on RDat[15:0]. The Sink core stops storing data to the Sink FIFO while in this state. The core will remain in this state until the first valid Payload-to-data transition is received on RDat.

In this state, the Sink core continuously checks DIP-4 parity, and sends FIFO Channel status on RStat. The core is in-frame in this state.

In-Frame and Out-of-Frame Behavior

There are a number of conditions that must be met before the Sink core deasserts SnkOof and starts accepting data. Data will be written to the FIFO when the following conditions are met.

- Clock alignment is complete (PhaseAlignComplete asserted).
- Reset_n is deasserted.

- SnkFifoReset_n is deasserted.
- SnkEn is asserted.
- SnkOof is deasserted (NumTrainSequences consecutive training patterns received).
- First valid SOP-to-data transition detected (after SnkOof or SnkFifoReset_n asserted).
- First valid Payload-to-data transition detected (after training pattern).

There are five conditions that will cause the Sink core to lose synchronization and assert SnkOof.

- PhaseAlignComplete is deasserted.
  - If the I/O loses lock (PhaseAlignComplete = 0) the data written to the FIFO is corrupt and is immediately terminated.
  - PhaseAlignRequest is asserted (specifically, low-to-high transition). This forces the core to go out of frame.
- SnkEn is deasserted.
  - The core will continue to write data to the FIFO and will terminate with the next control word detected.
- SnkAlmostFull_n asserted and SnkFifoAFMode = Send Framing Patterns ("00")
  - The core will continue to write data to the FIFO and will terminate with the next control word detected.
- NumDip4Errors consecutive DIP4 errors are detected
  - The core will terminate writing to the FIFO with the last control word that causes SnkOof.
- When the Sink core is out-of-frame, the FIFO will still contain the old data that were not read out by the user application prior to the core entering the out-of-frame state. It is recommended that the user reset the FIFO using SnkFifoReset_n before the Sink core transits back into frame.

Error Handling

This section describes how the Sink core handles the receipt of non-compliant SPI-4.2 data and subsequent error handling in a number of common scenarios. This section also provides information on the Sink core error status signals.

Short Packet Support (Less Than 16-byte Packet Support)

Though the SPI-4.2 specification requires that successive start-of-packets must occur not less than eight cycles apart, and that there is no restriction on payload control words that are not SOPs. The Sink core automatically handles any size packets, including multiple SOP that are less than eight cycles apart. If SOPs are less than eight cycles apart, the data passes through the core correctly, but the status output SnkBusErr is flagged to indicate that there has been a protocol violation.

Long streams of data that violate SOP spacing can cause the Sink core to overflow quickly. This special case of overflowing as a result of repeated SOP spacing violations is indicated by SnkOverflow_n asserting when SnkAlmostFull_n has not asserted. Any time that SnkOverflow_n is asserted, the integrity of the data in the FIFO is compromised and the FIFO must be reset.

Figure 5-16 illustrates back-to-back short packets. In this example there are four channels that are each sending 17-byte packets with a maximum burst of 16 bytes.

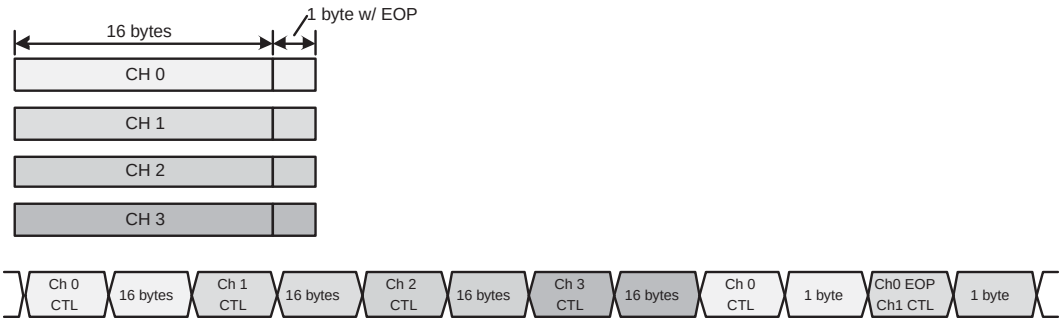


Figure 5-16: **Short Packet Support**

Sink FIFO Burst Error

When data is received on `RDat` that is terminated on a non-credit boundary without an EOP, the Sink core will flag this error at the end of the burst by asserting `SnkFFBurstErr`. `SnkFFBurstErr` may be used by the user's logic to indicate missing EOPs, or incorrectly terminated bursts. In this case the Sink core does not assert `SnkFFEOP` or `SnkFFErr`.

EOP Abort Handling

When an EOP abort is received, the Sink core asserts the output flags `SnkFFEOP` and `SnkFFErr` when the packet is terminated. In this case, the Sink core does not assert `SnkFFBurstErr`.

Loss of RDClk (Virtex-4 or Virtex-5 Devices Only)

The `Locked_RDclk` is connected directly to the LOCKED input of the DCM when the `DCM_Standby` option is not used. The `DCMLost_RDclk` is connected directly to the status bit `DO[1]` of the DCM.

Sink SPI-4.2 Bus Error and Sink Bus Error Status[7:0]

The Sink SPI-4.2 Bus Error and Bus Error Status indicate different information, depending on Sink core status and configuration.

Core In-frame or Configured with Static Alignment

With these conditions and configuration, a Sink SPI-4.2 Bus Error (`SnkBusErr`) is an indication of SPI-4.2 protocol violations or bus errors that are not associated with a particular data packet. Sink Bus Error Status (`SnkBusErrStat[7:0]`) triggers simultaneously with `SnkBusErr` and clarifies which protocol violations have occurred. These signals do not align with `SnkFFData`. Each bit of the `SnkBusErrStat` bus corresponds to one of the following detected conditions:

- `SnkBusErrStat[0]`: Minimum SOP spacing was violated.
- `SnkBusErrStat[1]`: EOP control word not immediately preceded by data.

  (Example: EOP followed immediately by another EOP).
- `SnkBusErrStat[2]`: Payload control word not immediately followed by data.

  (Example: A payload control word is followed immediately by another payload control word.)

- SnkBusErrStat[3]: DIP4 error received during idle or training patterns.
- SnkBusErrStat[4]: Reserved control words received.
- SnkBusErrStat[5]: Control word with payload bit not set and non-zero address (excluding Training Control word).
- SnkBusErrStat[7:6]: Unused and tied to zero (reserved).

If the core receives two (or more) back-to-back payload control words, the last one received is used and the others are discarded. If the core receives two (or more) back-to-back EOP control words, the first one is used and the others are discarded.

Any of the error conditions that flag the Sink Bus Error Status bus also flag SnkBusErr.

The latency from the time the error is received on the SPI-4.2 Interface to the time that the SnkBusErr and SnkBusErrStat signals are asserted is 33 RDClk clock cycles.

Core is Out-of-Frame and Configured with Dynamic Phase Alignment

With these conditions and configuration, the SnkBusErrStat signal can be used to output the DPA alignment information while the DPA logic is attempting initial alignment (SnkOff asserted). See DPA Status Monitoring, page 97. This feature is available if the Enable DPA Status Monitoring, page 64 is selected. The SnkBusErrStat displays the DPA alignment information only during the alignment process. At this time the SnkBusErr signal will not be asserted.

Once the alignment is achieved and the core is in frame, the behavior of SnkBusErrStat and SnkBusErr will operate as described in the previous section.

Sequential Payload Control Words

If back-to-back payload control words are sent, the Sink core only uses the payload control word that precedes a data word. All other payload control words are dropped by the Sink core. Each time a payload control word is dropped, it is flagged on SnkBusErr. This behavior is illustrated in Figure 5-17.

Sequential End-of-Burst Control Words

The Sink core only stores the end-of-burst control word that was preceded by data. It drops any other end-of-burst control words that are not preceded by data and flags SnkBusErr. Figure 5-17 illustrates this behavior.

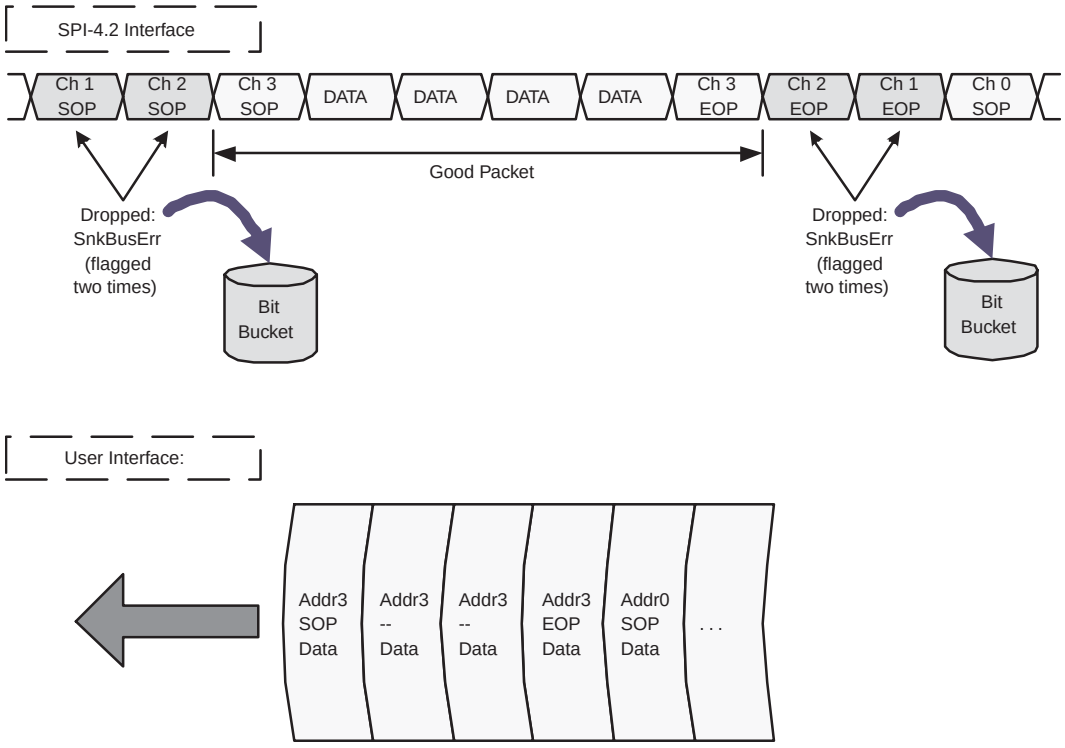


Figure 5-17: **Sequential Payload Control Word Example**

Sink DIP-4 Error Handling

When a DIP-4 error occurs at the end of a burst (for the previous packet), the Sink core stores a `SnkFFDIP4Err` flag. Figure 5-18 illustrates a DIP-4 error that occurred on an end-of-packet control word.

When a DIP-4 error occurs on a payload control word (start of next data burst packet), the Sink core stores a `SnkFFPayloadDIP4` flag. If the payload control word was also the end-of-burst control word for the previous packet, then `SnkFFDIP4Err` would also be asserted for the previous packet. Since the *OIF SPI-4.2* specification does not distinguish between these two DIP-4 errors, the Sink core will tag each packet that was terminated with a DIP-4 error on `SnkFFDIP4Err`, and each packet that was started with a DIP-4 error on `SnkFFPayloadDIP4`. This is illustrated in Figure 5-19 where packet 1 is flagged with a `SnkFFDIP4Err` and packet 2 is flagged with `SnkFFPayloadDIP4`. Note that both DIP-4 errors are asserted at the end of the burst or packet.

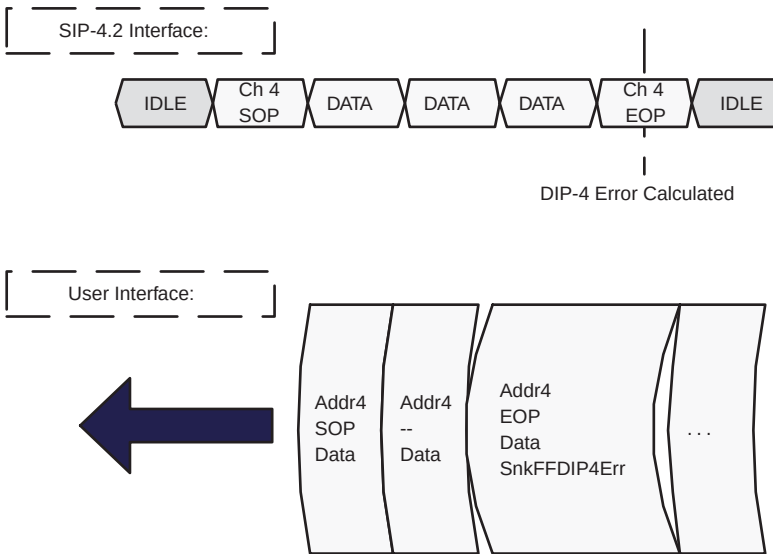


Figure 5-18: **Example of Error Flag SnkFFDIP4Err**

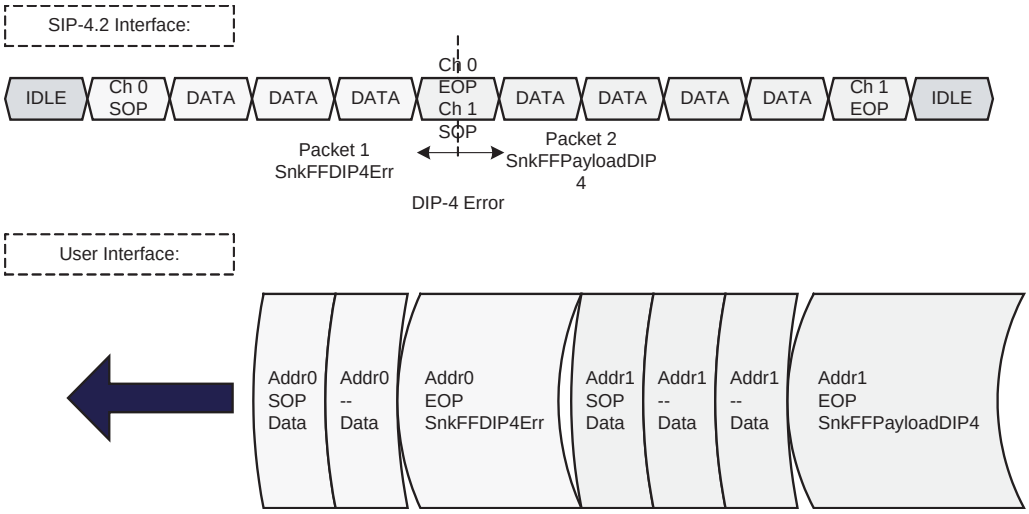


Figure 5-19: **Example of Error Flag SnkFFDIP4Err and SnkFFPayloadDIP4**

Reserved Control Words

As defined by the OIF SPI-4.2 specification, a reserved control word contains an SOP, but the payload control bit (`RDat[15]`) is not set to a one. If this occurs and then is followed by data, the Sink core will assert `SnkFFPayloadErr` for the duration of the burst, indicating that the burst did not have a correct payload control word. This indicates that the SOP and address configuration will not be valid. This error will also be flagged on `SnkBusErr`. This behavior is illustrated in Figure 5-20.

If this behavior occurs and is not followed by data, then the Sink core drops the control word and asserts the output `SnkBusErr`.

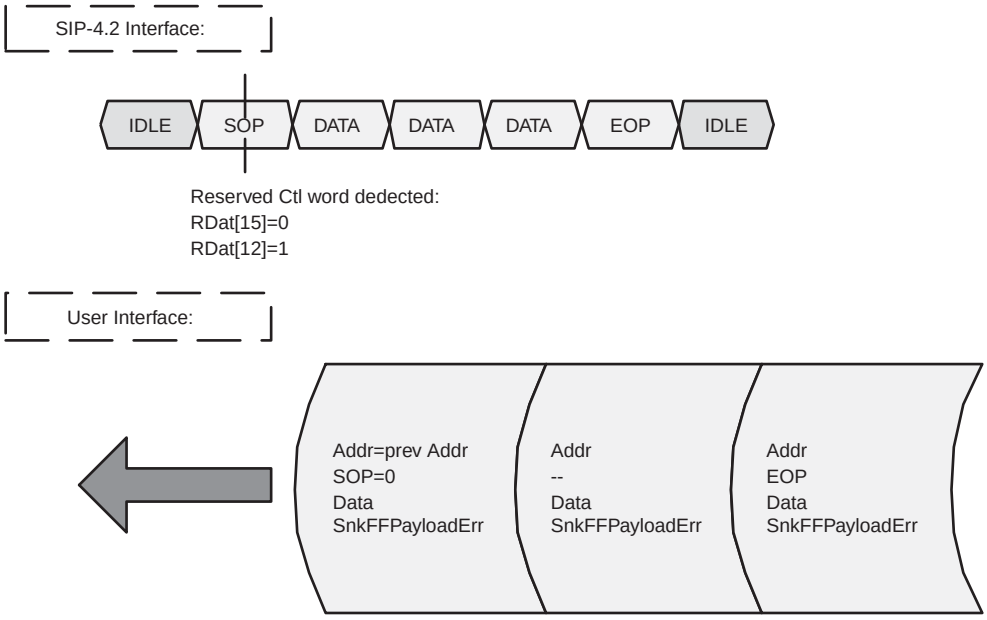


Figure 5-20: **Example of Error Flag SnkFFPayloadErr**

Source Core

Basic Operation

The Source core receives 64- or 128-bit data on the user interface and converts data to 16-bit data which is transferred across the SPI-4.2 interface. It also receives flow control information of the SPI-4.2 interface and processes it into 32-bit or 2-bit status word, depending on the status FIFO interface (accessed on the user interface).

The following sections explain how the Source core operates. See Source Core Interfaces, page 43 for the signal list of the interfaces.

SPI-4.2 Interface

The SPI-4.2 user interface combines data words and out-of-band control signals and multiplexes them to the SPI-4.2 16-bit data bus. This allows the user interface to run at a quarter (64-bit interface) or an eighth (128-bit interface) of the data rate. For example a 944 Mbps SPI-4.2 data rate and a 64-bit user interface can write data into the Source core at 236 MHz. If a 128-bit user interface is used, data can be written into the Source core at 118 MHz and maintain the same data rate.

Source Data Path: Example 1

An example of the data received on the user interface and subsequently transmitted on the SPI-4.2 Interface is shown in Figure 5-21. In this illustration, a 14-byte packet of data is written into channel 1, followed by an 8-byte packet into channel 2. On the SPI-4.2 bus, the transfer begins with a payload control word (C1) indicating the start of packet (SOP), and address of the data to follow. Next, seven SPI-4.2 bus cycles of data, two bytes each, are used to transfer the data associated with channel 1. The transfer on channel 1 is concluded with an end of packet control word (C2). Since the next FIFO location contains the start of

a new packet on channel 2, the SOP and address of that packet are combined with the end of packet information from channel 1 to form one control word (C2). The second packet is terminated with an EOP (C3).

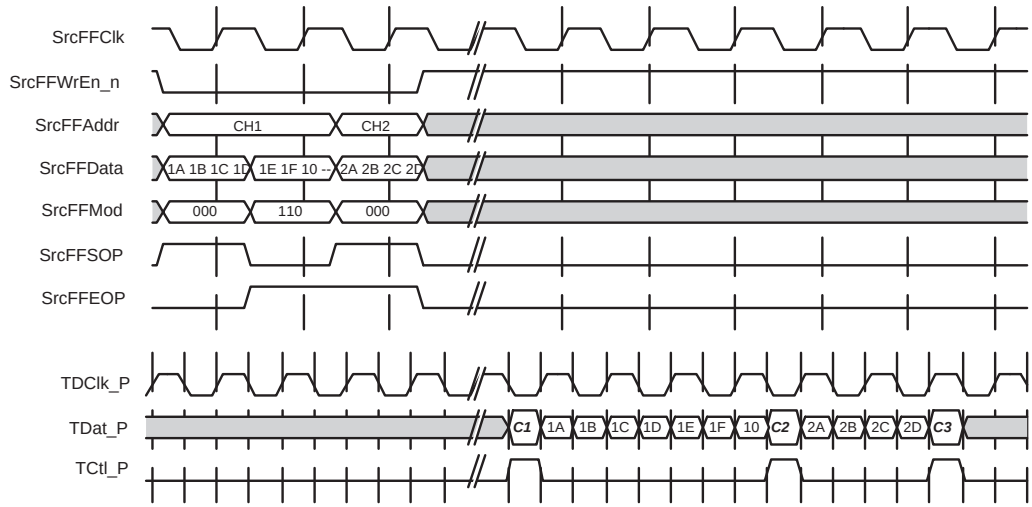


Figure 5-21: **Source Data Path: User Interface to SPI-4.2 Interface**

Source Data Path: Example 2

Figure 5-22 shows the transfer of short packets from the Source FIFO to the SPI-4.2 bus interface. Since each of the packets contain fewer than 14 bytes, or seven SPI-4.2 bus cycles of data, idle word insertion is necessary to meet the start-of-packet spacing requirement of eight cycles.

The transfer begins with a 4-byte packet of data for channel 1 written into the Source FIFO. Next, a 6-byte packet of data is written into the FIFO for channel 2. Finally, a 4-byte packet for channel 3 is written into the FIFO. The transfer on the SPI-4.2 bus begins with a control word (C1) indicating a start-of-packet for channel 1. Next, the four bytes of data for channel 1 are transferred. While the FIFO contains the start of packet information for channel 2, that information cannot be combined with the end-of-packet information from channel 1 because of the 8-cycle start-of-packet spacing requirement.

For this reason, five additional idle control words (I) are sent across the SPI-4.2 bus with the first idle control word containing the end-of-packet information for channel 1. The next SPI-4.2 cycle contains the start-of-packet and address information for channel 2 (C2). This payload control word is followed by the six bytes of data for channel 2.

Again, because of the start-of-packet spacing requirement, another four cycles of idle control words (I) must be sent across the interface with the first idle control word containing the end-of-packet information for channel 2. Finally, the start-of-packet and address information for channel 3 are sent in the payload control word (C3).

If the payload control words did not contain SOP indications (payload resumes), the Source core would not be required to enforce minimum SOP spacing. The Source core will then pack the EOP and Payload Control word into a single cycle and will not insert idle cycles. This behavior is illustrated in Figure 5-23.

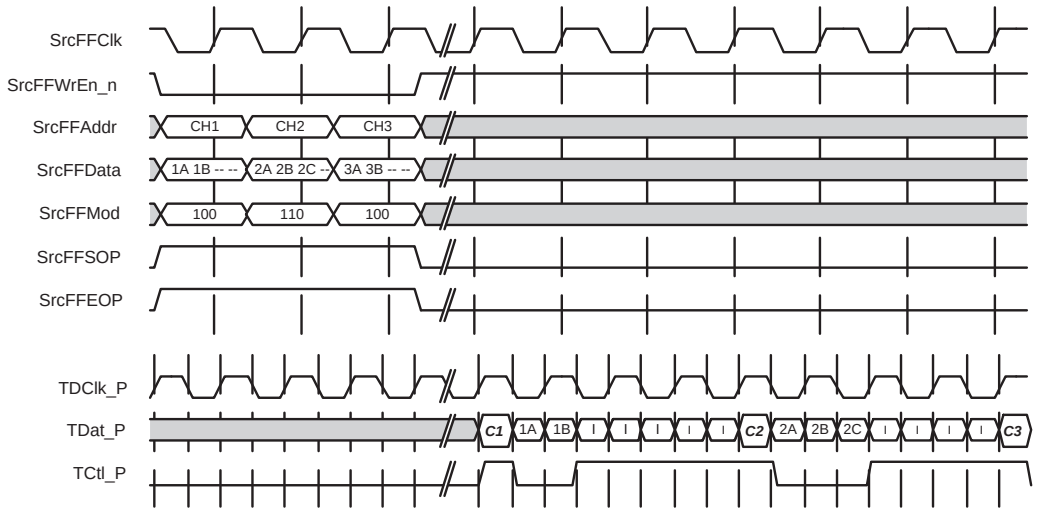


Figure 5-22: **Source Data Path - Minimum SOP Spacing Enforced**

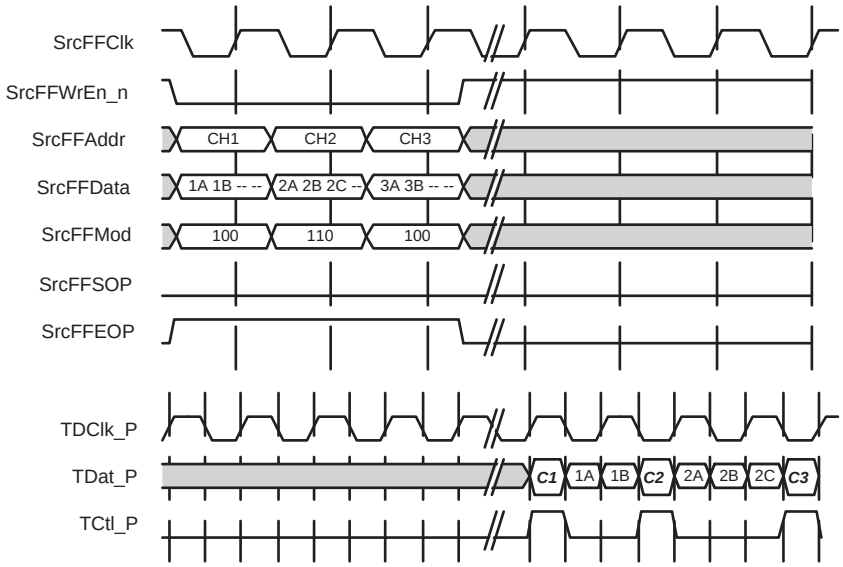


Figure 5-23: **Source Data Path - Short Packet Transfers**

The Source core formats the data to be written onto the SPI-4.2 bus (`TDat`). Table 5-5 shows an example of the formatting that this block does with the data read out of the Source FIFO (note that control words are shown in binary and payload transfers are shown as hexadecimal). When an SOP is read out of the FIFO, the following 16-bit-word transfer sent on the SPI-4.2 data bus is an SOP control word. This example shows the receipt of an SOP for channel 2 and two 64-bit words from the Source FIFO are transmitted on the SPI-4.2 data bus. The DIP-4 parity depends on this control word and any proceeding transfer and therefore it is left as "pppp" (shown in the 13th `TDClk` clock cycle).

The following two tables show the mapping between the packet status signals on the user interface and SPI-4.2 control words for a 128-bit user interface (Table 5-6) and for a 64-bit user interface (Table 5-7).

Table 5-5: **Example of Formatting Source FIFO Data for a 64-bit User Interface**

Data Written to the Source FIFO (SrcFFData[63:0])	SrcFFClk cycle	FIFO Control Bit	Data Transmitted on the SPI-4.2 Interface (TDat [15:0])	TCtl	TDClk cycle
SrcFFData[63:0] = [F1E2.D3C4.B5A6.9F8E]	1	SrcFFSOP = 1 SrcFFEOP = 0 SrcFFMOD = 000 SrcFFAddr = 0000.0010 SrcFFErr = 0	N/A	N/A	n
			SOP b:[1001.0000.0010.pppp]	1	n+1
			SPI-4.2 Word 0 (P0) F1E2	0	n+2
			SPI-4.2 Word 1 (P1) D3C4	0	n+3
SrcFFData[63:0] = [1F2E.3D4C.5B6A.F9E8]	2	SrcFFSOP= 0 SrcFFEOP = 0 SrcFFMOD = 000 SrcFFAddr = 0000.0010 SrcFFErr = 0	SPI-4.2 Word 2 (P2) B5A6	0	n+4
			SPI-4.2 Word 3 (P3) 9F8E	0	n+5
			SPI-4.2 Word 4 (P4) 1F2E	0	n+6
			SPI-4.2 Word 5 (P5) 3D4C	0	n+7
SrcFFData[63:0] [ABCD.1200.0000.0000]	3	SrcFFSOP= 0 SrcFFEOP=1 SrcFFMOD = 011 SrcFFAddr = 0000.0010 SrcFFErr = 0	SPI-4.2 Word 6 (P6) 5B6A	0	n+8
			SPI-4.2 Word 7 (P7) F9E8	0	n+9
			SPI-4.2 Word 8 (P8) ABCD	0	n+10
			SPI-4.2 Word 9 (P9) 1200	0	n+11
	4		EOP / MOD b:[0110.0000.0010.pppp]	1	n+12

Table 5-6: **SPI-4.2 Control Word Mapping to 128-bit Interface**

Control Word	Associated Source FIFO Signal(s)	Associated SPI-4.2 Control Word bits on TDat (Qualified by TCtl=1)
Start of Packet (SOP)	SrcFFSOP, SrcFFAddr[7:0]	TDat[15] =1, TDat[12]=1, TDat[11:4] <== SrcFFAddr[7:0]
New Burst (address change without SOP)	SrcFFAddr[7:0]	TDat[15] = 1, TDat[12] = 0, TDat[11:4] <== SrcFFAddr[7:0]

Table 5-6: **SPI-4.2 Control Word Mapping to 128-bit Interface**

Control Word	Associated Source FIFO Signal(s)	Associated SPI-4.2 Control Word bits on TDat (Qualified by TCtl=1)
End of Packet (EOP, even bytes valid)	SrcFFEOP, SrcFFMOD[3:0] When TDat[14:13] = 10: MOD = 0000 if data bits 127-0 have valid data MOD =1110 if data bits 127-16 have valid data MOD =1100 if data bits 127-32 have valid data MOD = 1010 if data bits 127-48 have valid data MOD = 1000 if data bits 127-64 have valid data MOD = 0110 if data bits 127-80 have valid data MOD = 0100 if data bits 127-96 have valid data MOD = 0010 if data bits 127-112 have valid data	TDat[14:13] = 10
End of Packet (EOP, odd bytes valid)	SrcFFEOP & SrcFFMod[3:0] When TDat[14:13] = 11: MOD = 1111 if data bits 127-8 have valid data MOD = 1101 if data bits 127-24 have valid data MOD = 1011 if data bits 127-40 have valid data MOD = 1001 if data bits 127-56 have valid data MOD = 0111 if data bits 127-72 have valid data MOD = 0101 if data bits 127-88 have valid data MOD = 0011 if data bits 127-104 have valid data MOD = 0001 if data bits 127-120 have valid data	TDat[14:13] = 11
End of Packet (EOP, abort, error condition)	SrcFFErr, SrcFFEOP, SrcFFMOD[3:0]	TDat[14:13] = 01

Table 5-7: **SPI-4.2 Control Word Mapping to 64-bit User Interface**

Control Word	Associated Source FIFO Signal(s)	Associated SPI-4.2 Control Word bits on TDat (Qualified by TCtl=1)
Start of Packet (SOP)	SrcFFSOP, SrcFFAddr[7:0]	TDat[15] =1, TDat[12]=1, TDat[11:4] <== SrcFFAddr[7:0]
New Burst (address change without SOP)	SrcFFAddr[7:0]	TDat[15] = 1, TDat[12] = 0, TDat[11:4] <== SrcFFAddr[7:0]
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

Table 5-7: **SPI-4.2 Control Word Mapping to 64-bit User Interface**

Control Word	Associated Source FIFO Signal(s)	Associated SPI-4.2 Control Word bits on TDat (Qualified by TCtl=1)
End of Packet (EOP, odd bytes valid)	SrcFFWEOP & SrcFFWMod[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
End of Packet (EOP, abort, error condition)	SrcFFErr, SrcFFEOP, SrcFFMOD[2:0]	TDat[14:13] = 01

Transmitting Training Patterns

Training patterns are transmitted at startup (after reset) until the core acquires synchronization on the FIFO Status Channel. Subsequently, if the parameter `DataMaxT` or `AlphaData` are not zero, the core will transmit `AlphaData` training patterns at least every `DataMaxT` cycles.

The core continuously monitors the number of data cycles since the transmission of the last training pattern. Once a `DataMaxT` interval of SPI-4.2 bus cycles has completed, the current transfer will be terminated on the next burst boundary, and training patterns will be transmitted on the SPI-4.2 bus (`AlphaData` number of times). Once the training patterns have completed, the SPI-4.2 core will resume transmission of data on the data bus.

The control signal `TrainingRequest` (see Table 3-12) is provided to request that training patterns be sent out of the SPI-4.2 Source interface. When the `TrainingRequest` signal is asserted, the transmission of data is halted on the next burst boundary and training patterns are transmitted on the SPI-4.2 Interface.

If the static configuration signal `AlphaData[7:0]` (see Table 3-16) is set to zero, and the `TrainingRequest` signal is asserted, the Source core will transmit a complete training pattern sequence. The core will continue to transmit training patterns until `TrainingRequest` is deasserted. When it is deasserted, the core will halt transmission of training patterns after the current sequence is complete.

If the static configuration signal `AlphaData[5:0]` is set to a non zero value, the Source core will send the number of training patterns defined by `AlphaData` every time it detects a rising edge on the `TrainingRequest` signal.

Transmitting Idle Cycles

Idle cycles are sent on the SPI-4.2 Interface only when there is no data in the FIFO. The core will also insert idle cycles when the control signal `IdleRequest` (see Table 3-12) is asserted. When this signal is asserted, the transmission of data is halted on the next burst boundary and idle cycles are forced onto the SPI-4.2 Interface. The insertion of training patterns always takes precedence over the transmission of idle cycles.

Inserting DIP4 Errors

For system diagnostics, you can force DIP4 errors to be inserted with a specific packet. This is supported by using the `SrcFFErr` signal. When `SrcFFErr` is asserted and `SrcFFEOP` is

deasserted, it signals the core to terminate the current packet with an EOP and to force the insertion of an erroneous DIP4 value, see Figure 5-26 for more details.

SPI-4.2 Source User Interface

The Source user interface includes all the signals to the core other than those on the SPI-4.2 Interface (See SPI-4.2 Interface). With a 64-bit or 128-bit data interface, the user interface can operate up to:

- 350 MHz in Virtex-6 FPGAs
- 312Mhz in Virtex-5 FPGAs
- 250 MHz in Virtex-4 FPGAs

The user interface has three types of signals:

Control and Status Signals. Apply to the operation of the Source core.

FIFO Interface Signals. Writes data to the FIFO to be transmitted on the SPI-4.2 Interface.

Status and Flow Control Signals. These signals are used to receive flow control information from the SPI-4.2 Interface

Source Control and Status Signals

The Source core control and status signals either control the operation of the entire Source core or provide status information that is not associated with a particular channel (port) or packet. The description of these signals is summarized in Table 3-12.

The Source core is reset asynchronously by the signal `Reset_n`, and there are three global status signals:

- **Source Out-of-Frame** (`SrcOof`) is asserted whenever the core has lost synchronization with the SPI-4.2 status bus (`TStat`)
- **Source DIP2 Error** (`SrcDIP2Err`) is asserted when a DIP2 error is detected on the SPI-4.2 status bus
- **Source Pattern Error** (`SrcPatternErr`), is asserted when an illegal data pattern is written into the Source FIFO. There are two conditions that will trigger this error signal:
  - *Case 1*: The address was changed on a non-credit boundary, without an EOP: In this case, the remainder of that packet will be terminated with an EOP Abort, and sent out the SPI-4.2 bus.
  - *Case 2*: The `SrcFFMod` signal is non-zero without an EOP: In this case an EOP abort will not be asserted. When this occurs, the Source core will ignore the SrcFFMod value and send the data word with MOD set to zero.

  The source core does not handle per-channel error handling. Handling errors on per-channel basis must be handle by the user logic.
- **Source Status Frame Error** (`SrcStatFramErr`) is asserted when a non-"11" frame word is received on the SPI-4.2 status bus.

The control signal `TrainingRequest` is used to request that training patterns be sent out of the SPI-4.2 Source interface. When this signal is asserted, the transmission of data is halted on the next burst boundary and training patterns are transmitted on the SPI-4.2 Interface. For more information on the behavior of `TrainingRequest`, see Transmitting Training Patterns.

The control signal IdleRequest requests that idle control words be sent on the SPI-4.2 bus. This request overrides payload data transfers, but not training sequence requests. When this signal is asserted, the transmission of data is halted on the next burst boundary and idle cycles are transmitted on the SPI-4.2 Interface. Idle cycles continue to be transmitted until the signal is deasserted. For more information on the behavior of IdleRequest, see Transmitting Idle Cycles.

The control signal SrcTriStateEn allows you to set the IOB drivers to high impedance for the Source core output signals TDClk, TDat[15:0], and TCtl. The Source core is provided such that the default setting for this signal is that the outputs are not tri-stated (SrcTriStateEn=0).

The control signal SrcOofOverride removes the requirement that the Source core must receive consecutive valid DIP2 values on TStat. This signal forces the Source core to go in-frame and begin transmitting data on the SPI-4.2 interface. This signal is intended for system testing and debugging.

The control signals SrcFifoReset_n and Reset_n provide the reset capabilities:

- SrcFifoReset_n is used to clear the FIFO (and the associated data path logic) while remaining in-frame. When SrcFifoReset_n is deasserted, the Source core will send idle cycles until the user writes data into the FIFO. SrcFifoReset_n should only be asserted when there is no FIFO write operation taking place. The core will continue to write data, but it will be an incomplete packet and may not be transmitted to the SPI-4.2 bus.
- Reset_n is used to restart the entire Source core, and it will cause the interface to go out-of-frame. When Reset_n is deasserted, the Source core will initiate the synchronization startup sequence.

Source FIFO Interface Signals

The Source FIFO Interface signals allow data to be written to the FIFO to be transmitted on the SPI-4.2 Interface. The description of each signal is summarized in Table 3-13. The Source FIFO Interface signals are synchronous to SrcFFClk, and the effective FIFO depth is 510 words deep. Each FIFO location can contain up to 16 bytes (one credit) worth of data from a single packet.

The SPI-4.2 Source core offers 64-bit and 128-bit FIFO Interface options for writing data into the FIFO. Waveforms illustrating the handshaking and FIFO status signals are shown in Figure 5-24, Figure 5-25, and Figure 5-26. The Source core also supports the insertion of DIP-4 errors on a per-packet basis, for use in system diagnostics. For more information on the insertion of DIP-4 errors, see Insertion of DIP-4 Errors, page 115.

Source FIFO Almost Full

Figure 5-24 shows the Almost Full response of the Source FIFO. The behavior of the Source Almost Full flag (SrcAlmostFull_n) is dependent on the static configuration signals SrcAFThresAssert and SrcAFThresNegate. When the SrcAlmostFull_n flag is asserted, SrcAFThresAssert specifies the number of empty FIFO locations available. Each FIFO location can contain up to one credit (16 bytes) worth of data from a single packet. SrcAFThresNegate specifies when the SrcAlmostFull_n flag is deasserted.

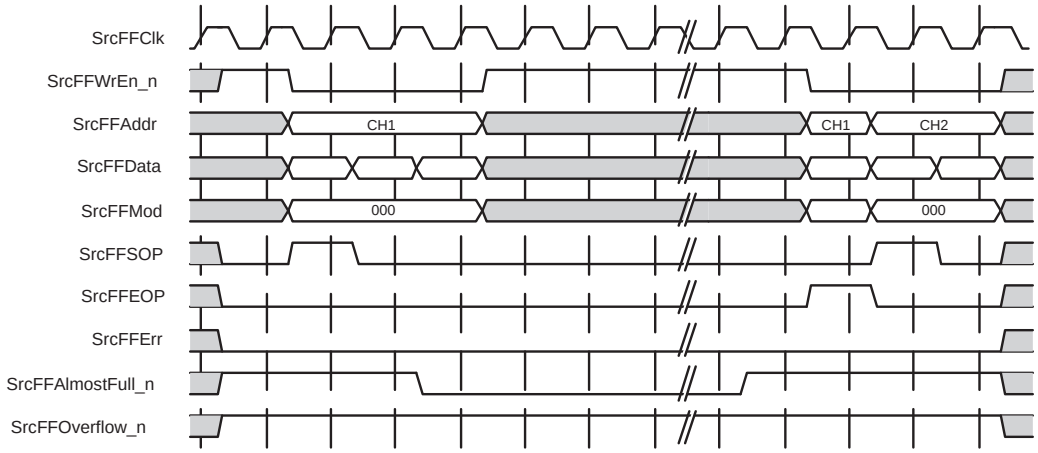


Figure 5-24: **Source FIFO Almost-full Condition**

Source FIFO Overflow

Figure 5-25 shows the overflow response of the Source FIFO. The assertion of the `SrcFFAlmostFull_n` signal indicates that the FIFO is almost full, and that data should no longer be written into the FIFO. If you continue to write data into the FIFO, the FIFO may overflow, resulting in the loss of data. The assertion of the `SrcFFOverflow_n` signal indicates that an overflow has occurred and that the current data (along with any subsequent data written to the FIFO) will be lost. The user may resume writing data to the FIFO when the `SrcFFAlmostFull_n` signal is deasserted.

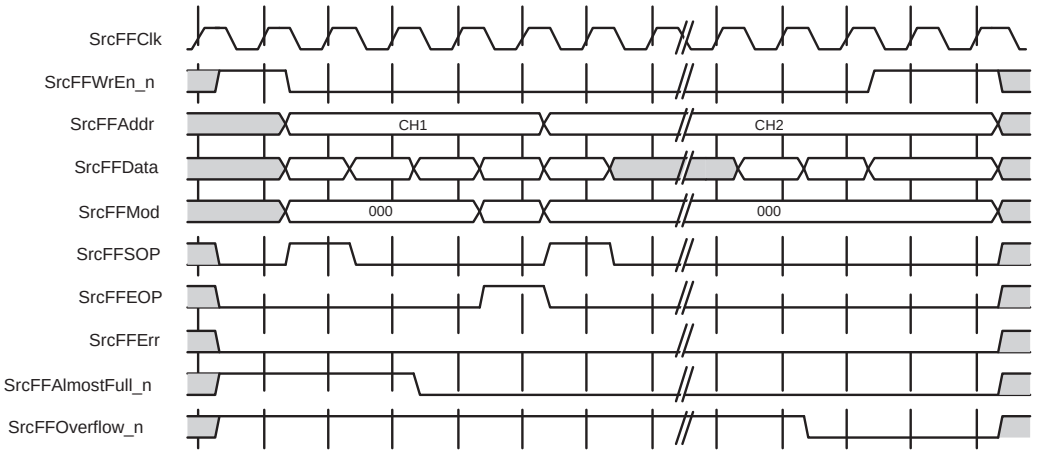


Figure 5-25: **Source FIFO Overflow Condition**

Writing to the Source FIFO

The user may pause a transfer on a credit (16 bytes) boundary, as illustrated Figure 5-26. In the example shown, two packets for unique channels are transferred into the FIFO. The user writes 32 bytes of data for channel 1, followed by 32 bytes of data for channel 2. Next, the final eight bytes of data and associated EOP for channel 1 is written to the FIFO. Finally, the remaining 16 bytes of data and associated EOP is written into the FIFO for channel 2. The data will be transferred across the SPI-4.2 bus in the same order it is written into the FIFO.

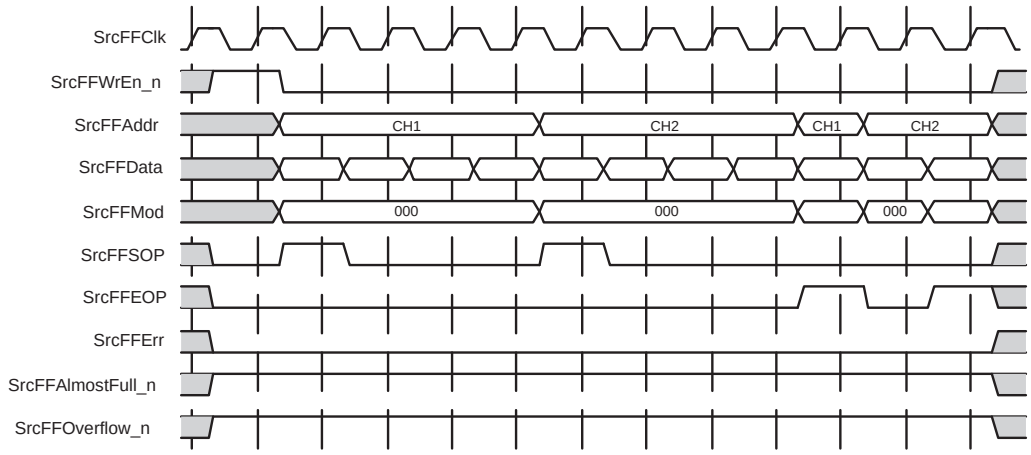


Figure 5-26: **Writing to the Source FIFO**

Insertion of DIP-4 Errors

The Source core lets you force the insertion of DIP4 error for use during system testing and debugging. This is supported using the `SrcFFErr` signal. When a `SrcFFEOP` flag is asserted, `SrcFFErr` is used to indicate that the current packet contains an error and causes the core to transmit an EOP abort with the packet. When `SrcFFEOP` is not asserted, the assertion of `SrcFFErr` causes the core to force the insertion of an EOP (1 byte or 2 bytes depending on `SrcFFMod`) with an erroneous DIP4 value when this data is transmitted on the TDat bus. The erroneous DIP4 value is an inversion of the correctly calculated DIP4 value. Note that the DIP-4 error insertions are independent of `SrcFFSOP`.

Source Status and Flow Control Signals

The Source core transmits data in the order in which it was written into the FIFO. You can use data transmission by sending idle cycles (using `IdleRequest`) or training (`TrainingRequest`), but unless the FIFO is cleared (`Reset_n` or `SrcFifoReset_n`), the data written into the FIFO will be transmitted in the ordered it is received. Ensure proper data scheduling is implemented to prevent a channel from going hungry or overflowing. This can be accomplished with the status information presented by the Source core to determine which channel data should be written next. A typical user flow-control design is shown in Figure 5-27. This is an illustration of a two channel system. The diagram shows an arbiter that is used to poll the FIFO Status for each channel and then uses this information to determine the data that is written into the Source core FIFO.

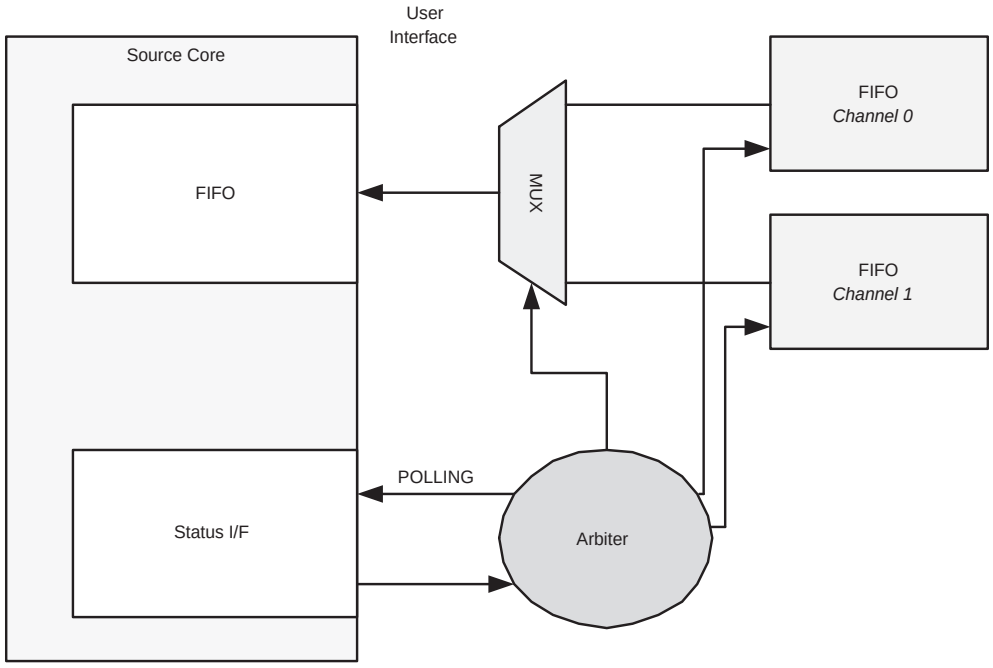


Figure 5-27: **Typical User Design Example**

To design back-end user logic, the Source core presents status information in two ways:

- **Addressable Status Interface**. Lets you access (poll) the status of 16 channels at a time. This polling is synchronous to a user-defined clock (`SrcStatClk`). Additionally, the last channel receiving a status update on `TStat[1:0]` is presented (synchronous to `TSClk`).
- **Transparent Status Interface**. Presents the status information as it is received on *`TStat[1:0]`* with minimal latency. It also provides the ideal interface to customize how the received FIFO status information is processed.

A user-programmable calendar is provided. This calendar stores the order and frequency for each channel status received on `TStat`, which is identical to the sequence defined by the device receiving data from the Source Interface. This is the mechanism that enables the interfaces to determine which channel status is being received on `TStat`. As defined by the SPI-4.2 specification, there are two bits provided for each channel, indicating the channel status (hungry=01, starving=00, satisfied=10).

These interfaces are described in greater detail below, and the description of the signals on the Source Status Path are defined inTable 3-14 and Table 3-15.

Source Calendar Initialization

There are two ways to initialize the Source calendar. First is by loading the COE file in the CORE Generator GUI, which loads the Calendar contents into the UCF file. For more information, see Chapter 4, Generating the Core The other method is to initialize the calendar in-circuit at startup.

Initializing the Calendar In-Circuit

At startup, the user circuitry can program the Source Calendar buffer by first deasserting Source Enable (`SrcEn`), then using the calendar write enable, address bus, and data bus.

SrcCalAddr is used to indicate the location in the calendar buffer, and SrcCalData is used to indicate the channel number that should be written into that location. This programming defines the sequence that the status for each channel will be received. It is programed identically to the device that the Source core has transmitted data.

The waveform in Figure 5-28 illustrates the programming of the Source Calendar. In this example, SrcCalendar_Len is set to five and SrcCalendar_M is set to zero (indicating that the calendar length is six, and should be repeated once). In this example, TStat[1:0] will receive status for each channel in the following sequence: status for channel 3, status for channel 0, status for channel 1, status for channel 2, status for channel 3, and status for channel 0.

To verify what has been programmed into the calendar buffer, read the contents using Source Calendar Data Out (SrcCalDataOut[7:0]). When the calendar write enable signal is deasserted, the data stored in the location specified by the calendar address is driven on the SrcCalDataOut bus. See Appendix F, SPI-4.2 Calendar Programming for more examples of programming the SPI-4.2 Calendar. Note that for a one channel system, it is not necessary to program the calendar since by default all locations are set to zero.

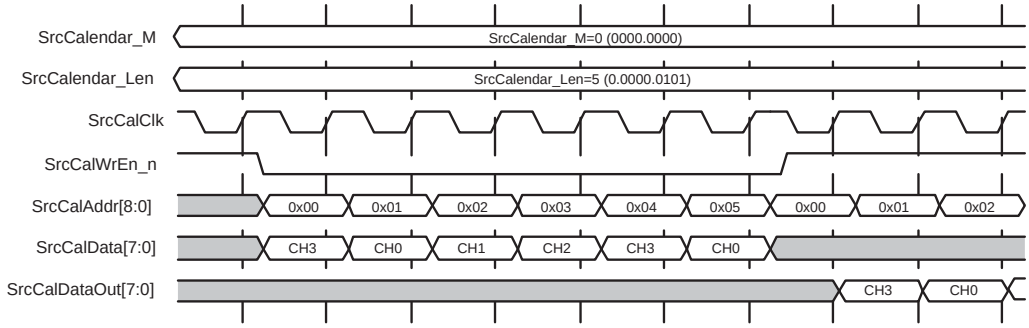


Figure 5-28: **Source Calendar Initialization**

Source Flow Control: Addressable Status Interface

The Addressable Status Interface is 32 bits for all channel configurations, allowing you to read the FIFO Status Channel data for 16 channels at a time. There are four address lines (SrcStatAddr) for selecting which 16 channels. (Note that for systems using 1-16 channels, the address lines can be permanently set to zero.) A block diagram of how the Addressable Interface processes the received SPI-4.2 Status FIFO is shown in Figure 5-29. The minimum latency between the user interface and SPI-4.2 Interface for this Status Path interface is nine TSClk cycles.

You can read the status for 16 channels in each clock cycle. SrcStatAddr bus is used to select which 16 channels are read, and the core supports configurations of 1-256 channels.

The 16-channel status banks that can be accessed are addressed as follows:

- Bank 0: SrcStatAddr[3:0]=0 for channels 15 to 0
- Bank 1: SrcStatAddr[3:0]=1 for channels 31 to 16
- Bank 2: SrcStatAddr[3:0]=2 for channels 47 to 32
- Bank 3: SrcStatAddr[3:0]=3 for channels 63 to 48
- ...
- Bank 14: SrcStatAddr[3:0]=14 for channels 239 to 224
- Bank 15: SrcStatAddr[3:0]=15 for channels 255 to 240

The status that is accessed is mapped to the 16-bit bus as follows (assuming SrcStatAddr [3:0] = 0):

- `SrcStat[1:0]` => Channel 0, where SrcStat[1] is the MSB of the 2-bit status
- `SrcStat[3:2]` => Channel 1
- `SrcStat[5:4]` => Channel 2
- ...
- `SrcStat[11:10]` => Channel 13
- `SrcStat[13:12]` => Channel 14
- `SrcStat[15:14]` => Channel 15

The operation of the Addressable Status interface is explained using three examples described below and illustrated in Figures 5-30, 5-31, and 5-32.

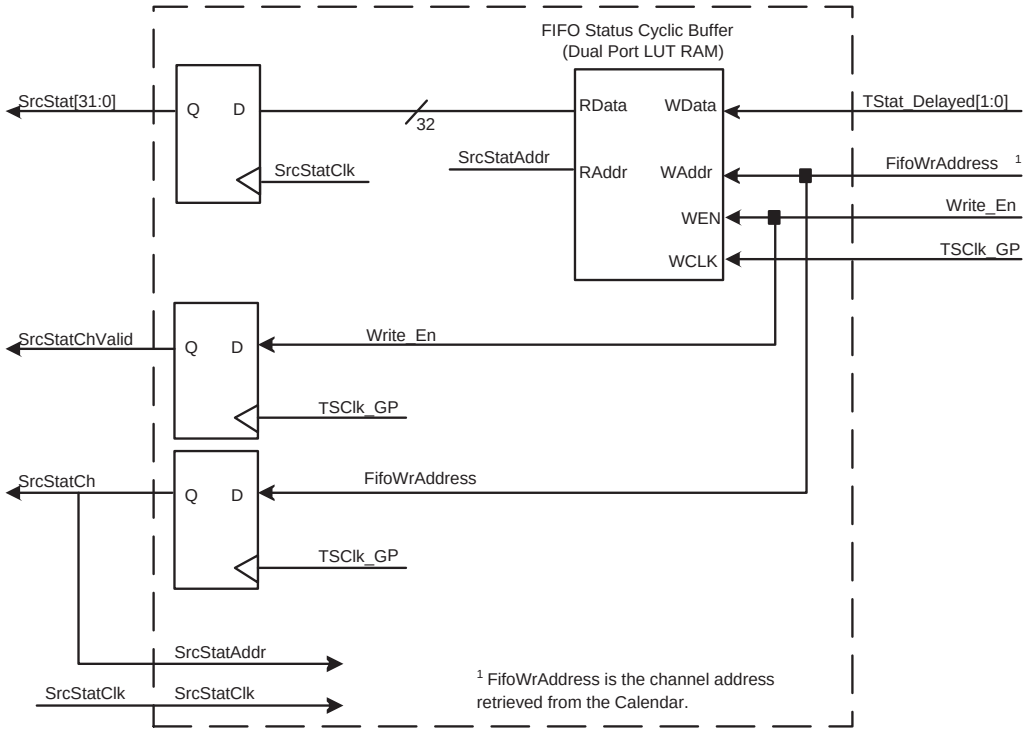


Figure 5-29: **Addressable Status FIFO Interface**

Addressable Status FIFO Interface: Example 1

Example 1 illustrates reading the Status FIFO Interface for a 4-channel SPI-4.2 Source core as shown in Figure 5-30. As there are fewer than 17 channels, the Source Status Address bus (`SrcStatAddr[3:0]`) is permanently tied to zero. The Source Status address (`SrcStatAddr[3:0]`) causes the Source Status data bus (`SrcStat[31:0]`) to be updated on the next clock cycle. Both buses utilize the user-selected clock domain (`SrcStatClk`), which can be tied to the SPI-4.2 Interface clock domain (`TSClk_GP`).

The Source Status Channel (`SrcStatCh[7:0]`) indicates which channel status was last updated on the SPI-4.2 Interface and is qualified by the Source Status Channel Valid signal (`SrcStatChValid`=1). The `SrcStatChValid` signal enables `SrcStatCh[7:0]`to be encoded, and the valid signal is disabled when the core processes a received DIP-2 or

framing word. The `SrcStatusCh[7:0]` and `SrcStatChValid` use the SPI-4.2 Interface clock domain (`TSClk_GP`).

The following status is read for the 4-channel system. In this example, the calendar length is set to six and programmed to round-robin this sequence: Ch0, Ch1, Ch2, Ch3, Ch0, Ch1.

Read Cycle	Starving Status	Satisfied Status
0	CH 0-3	none
1	CH 0-3	none
2	CH 0-3	none
3	CH 0-3	none
4	CH 0-3	none
5	CH 0-3	none
6	CH 0-3	none
7	CH 1-3	CH 0
8	CH 2-3	CH 0-1
9	CH 3	CH 0-2

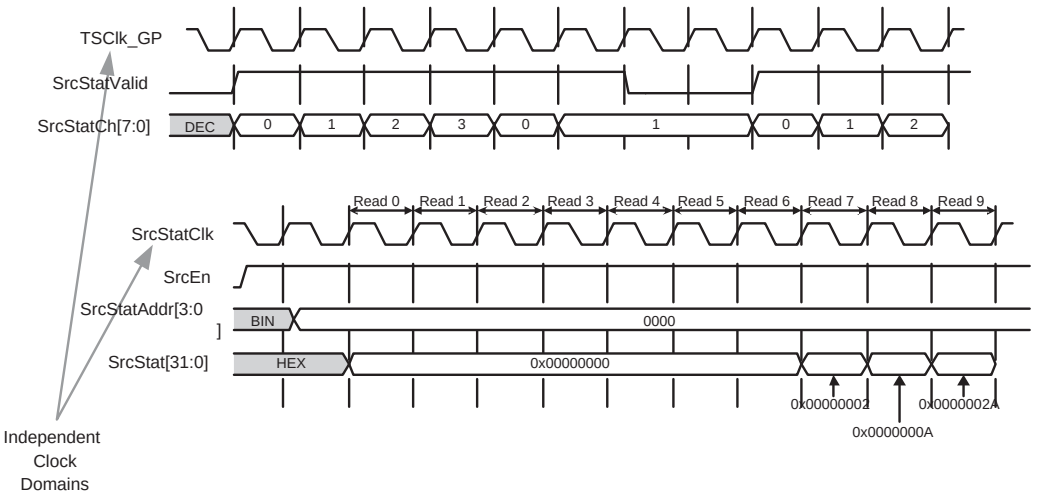


Figure 5-30: **Addressable Status FIFO Interface: 4-Channel Configuration**

Addressable Status FIFO Interface: Example 2

Example 2 illustrates reading the Status FIFO Interface for a 256-channel SPI-4.2 Source core and is shown in Figure 5-36. To read the status for 256 channels, you must address the following sixteen banks depending on which channel status is being read.

- Bank 0: `SrcStatAddr[3:0]`= 0000, for channels 15 to 0
- Bank 1: `SrcStatAddr[3:0]`= 0001, for channels 31 to 16
- Bank 2: `SrcStatAddr[3:0]`= 0010, for channels 47 to 32
- Bank 3: `SrcStatAddr[3:0]`= 0011, for channels 63 to 48
- ...
- Bank 15: `SrcStatAddr[3:0]`= 1111, for channels 255 to 240

The status read in the example shown in Figure 5-31 is summarized in the table below.

Read Cycle	Status Address	Starving Status	Satisfied Status
0	Bank 15	CH 240-255	None
1	Bank 15	CH 240-255	None
2	Bank 15	CH 240-255	None
3	Bank 0	CH 0-15	None
4	Bank 0	CH 0-15	None
5	Bank 0	CH 1-15	CH 0
6	Bank 15	CH 242-255	CH 240-241
7	Bank 15	CH 243-255	CH 240-242
8	Bank 15	CH 241-254	CH 255
9	Bank 0	CH 0-13	CH 14-15
10	Bank 0	CH 0-12	CH 13-15

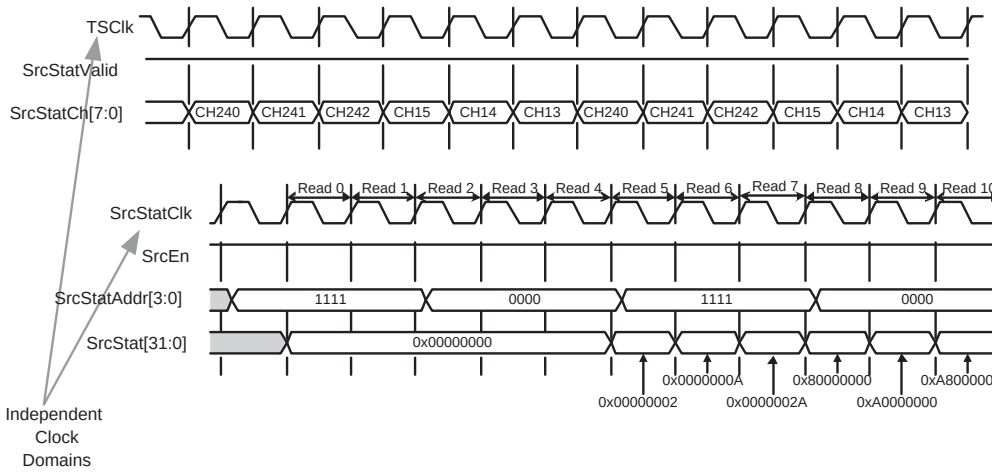


Figure 5-31: **Addressable Status FIFO Interface: 256-channel configuration**

Addressable Status FIFO Interface: Example 3

Example 3 illustrates status received on the SPI-4.2 bus and written to the user interface as shown in Figure 5-32. In the example shown, the calendar length is 17 (`SrcCalendar_Len`=16) and calendar repetition value is one (`SrcCalendar_M`=0). This illustrates a system in which the internal status path clock (`SrcStatClk`) is synchronous to the external status path clock (`TSClk`). In other words, `SrcStatClk` is tied to `TSClk_GP`. This enables access to the latest status information, which is achieved by connecting `SrcStatAddr` directly to the most significant four bits of the `SrcStatCh` bus.

In this example, the status for each channel alternates between starving and satisfied. To read the status for the full sequence, first set the `SrcStatAddr` to zero for channels 0-15,

and then to one to address channel 16. During the DIP-2 and framing cycles, the `SrcStatValid` is deasserted. During this time, the output on the bus is not defined.

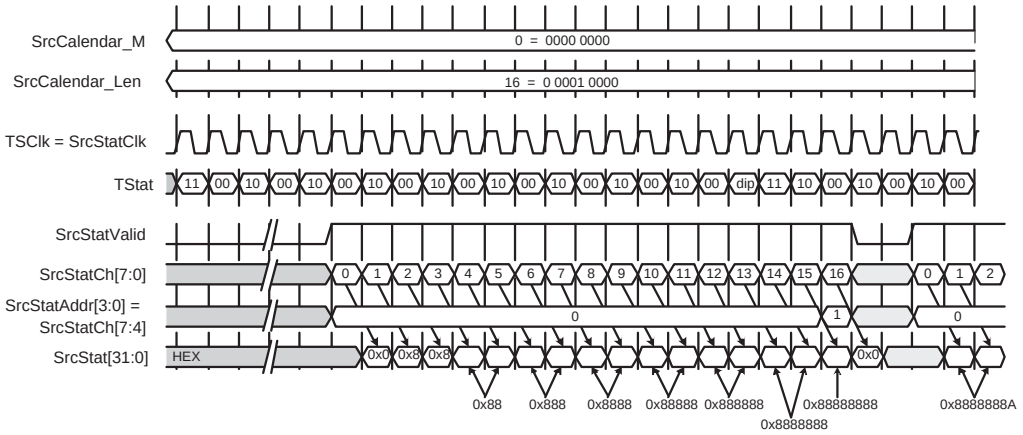


Figure 5-32: **Source Status Path - SPI-4.2 Interface to User Interface**

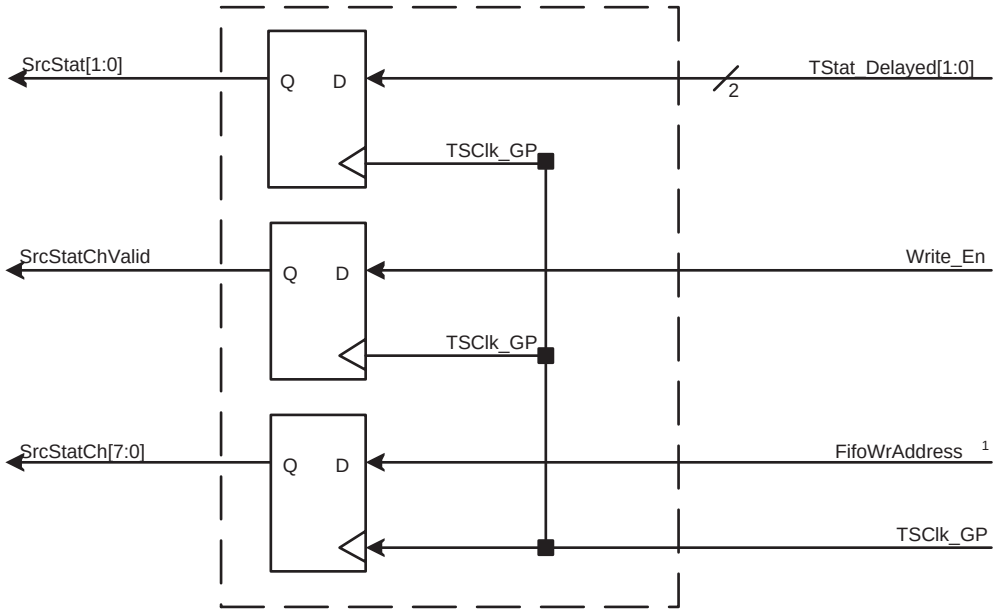
FIFO status information is periodic, repeating the sequence of a frame word (11), a repeated set of FIFO status words (`SrcCalendar_M` + 1 times) in accordance with the programmed calendar order, and a DIP-2 value. Figure 5-32 shows the receipt of one complete calendar sequence followed by the beginning of a second sequence. At startup, the circuitry initializes the Calendar buffer as described (See Source Calendar Initialization, page 116) and asserts the Source Enable signal (`SrcEn`). After reset is deasserted, the Source Interface sends training patterns on the data path (`TDat[15:0]`), and looks for non-framing data on the status path (`TStat[1:0]`). When `NumDip2Matches` number of valid DIP2 values has been received on the status path, valid data can be sent on the SPI-4.2 data path. If there is no data in the Source FIFO to be sent, the core sends idle cycles.

Source Flow Control: Transparent Status Interface

The Transparent Status Interface is 2 bits for all channel configurations. For the Transparent Interface, you are presented with the current status received on the SPI-4.2 Interface. The 2-bit status is presented by a corresponding channel address (`SrcStatCh[7:0]`) and is qualified with the valid signal `SrcStatChValid`. Unlike the Addressable Interface, the transparent interface does not store the received status in a cyclic buffer. This means you cannot access the status of a specific channel, but will receive the status in real-time as it is received by the Source core. Figure 5-33 is a block diagram of how the Transparent Interface processes the received SPI-4.2 FIFO Status. The minimum latency between the user interface and SPI-4.2 Interface for this Status Path interface is 4 `TSClk_GP` cycles.

Figure 5-34 illustrates the output of the Transparent Status FIFO Interface for a 256-channel configuration. On each clock cycle, the status (`SrcStat[1:0]`) and its corresponding channel (`SrcStatCh[7:0]`) is presented. The Source Status and channel address are only valid when `SrcStatChValid` is asserted (equal to one). When the `SrcStatChValid` signal is deasserted (equal to zero), the interface is receiving DIP-2 or framing and the data

on `SrcStat[1:0]` is not valid. For the Transparent Interface, all outputs use the SPI-4.2 Interface clock domain (`TSClk_GP`).



[1] FifoWrAddress is the channel address retrieved from the Calendar.

Figure 5-33: **Transparent Status FIFO Interface Block Diagram**

The following status is shown for the 256-channel Source Calendar Initialization system:

Read Cycle	Channel	Status
0	0	Hungry
1	2	Satisfied
2	128	Starving
3	129	Hungry
4	9	Hungry
5	1	Satisfied
6	Invalid	Invalid (DIP-2)
7	Invalid	Invalid (Frame word)
8	10	Starving
9	79	Hungry
10	16	Satisfied

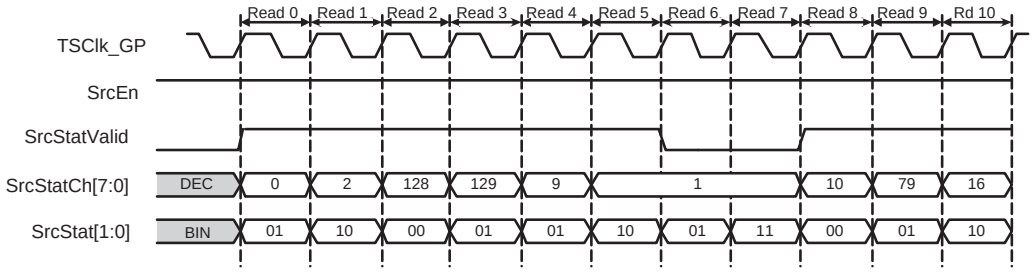


Figure 5-34: **Transparent Source Status FIFO Interface: 256-channel Configuration**

Source Static Configuration Signals

The Source static configuration signals are inputs to the core that are statically driven to determine the behavior of the core. See Table 3-16, page 52 for a full list of static configuration signals.

Three of the Source Static Configuration signals can be changed in circuit. There are static registers for `SrcBurstLen` (synchronous to `SrcFFClk`), and `SrcCalendar_M` and `SrcCalendar_Len` (synchronous to `SrcStatClk`.) To change these parameters while the core is operational, first deassert `SrcEn`.

All Source Static Configuration signals can also be changed in-circuit when the core is *not* in operation (disabled and in reset state).

The following steps are recommended when changing static configuration signals:

1. Disable the source core (`SrcEn` signal).
2. Assert core reset (Reset_n = 0).
3. Change the desired static configuration signals.
4. Deassert reset (Reset_n=1).
5. Wait at least 10 clock cycles of `SysClkDiv_GP` for the source static configuration signals to settle and propagate to the Source core's logic.
6. Enable the core and wait for the core to achieve synchronization, then continue normal operation.

Source Burst Mode

Source Burst Mode (`SrcBurstMode`) is a static configuration signal that allows you to define how data is transmitted by the Source core. If this signal is set to zero, the Source core will transmit data in the FIFO whenever there is a burst of data larger than 1 credit, followed by a no write (the burst of data must be multiples of credits), or when there is an end-of-packet flag (`SrcFFEOP`.)

When `SrcBurstMode` = 0 and a partial credit is written in (such as 12.5 credits), then `SrcBurstLen` will have an effect on whether or not the data is transmitted. If more credits have been written than `SrcBurstLen`, then that data will be written out in bursts the size of `SrcBurstLen`, but only after `WrEn` has been de-asserted. No data is sent out until the number of credits written has exceeded the `SrcBurstLen` setting.

For example:

- If `SrcBurstLen` = 1 and 12.5 credits are written, then 12 credits will be sent out in 1 credit increments with a control word in between.

- If `SrcBurstLen` = 8 and 12.5 credits are written, then the core will only write out an 8 credit burst and will wait until another 8 credits are in the FIFO.
- If `SrcBurstLen` = 12 and 12.5 credits are written, then 12 credits will be written out in a 12-credit burst without control words in between.
- If `SrcBurstLen` = 12 and 10.5 credits are written, then no data will be written out until at least 12 credits have been written.

This behavior is compliant with the transmit operation as defined by the SPI-4.2 OIF specification. If a partial credit is written into the FIFO and then paused, the data in the FIFO will be transmitted up to the last credit boundary.

When `SrcBurstMode` is set to a 1, and `SrcBurstLen` is less than 256, the Source core will only transmit data that is terminated by an EOP or when there is data in the FIFO equal to the maximum burst length defined by the static configuration signal `SrcBurstLen`, or when the channel address changes. If an incomplete burst is written into the FIFO and paused, then data in the FIFO will be transmitted up to the last burst boundary.

When `SrcBurstMode` is set to 1 and `SrcBurstLen` is greater than or equal to 256, the Source core will transmit data that is terminated by an EOP, or when the channel address changes, or when the Source FIFO is approximately one-half full. The write data-rate must be faster or equal to the read data-rate to ensure the size of the unsegmented burst.

Source Burst Mode Example

`SrcBurstLen` equals two credits and 1.5 credits are written into the FIFO followed by a 0.5 credit.

Figure 5-35 illustrates the behavior of the Source core when `SrcBurstMode`=1.

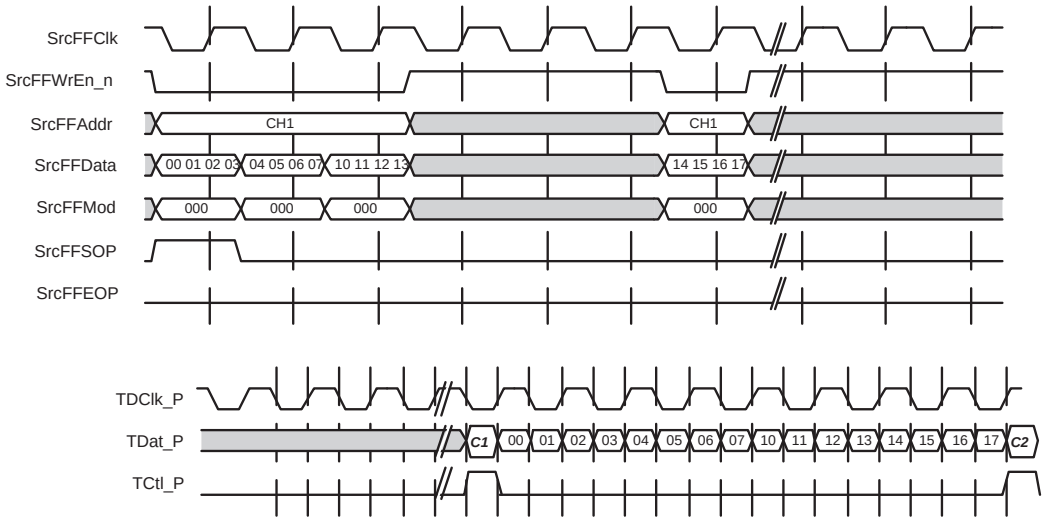


Figure 5-35: **Example Of Source Burst Mode = 1**

Synchronization and Startup

Once the Source core has been initialized (Initializing the SPI-4.2 Core, page 72), the Source core must be synchronized before data and status can be received and transmitted. Figure 5-36 is a state-machine diagram illustrating the Source core startup and error condition processing.

RESET

The Source core remains in the reset state until `SrcClksRdy` is asserted and the `Reset_n` signal is deasserted. When in the reset state, the Source core transmits idle patterns on `TDat[15:0]` and the Status FIFO is driven to be satisfied (“10”) for all channels.

HUNT

When *`Reset_n`* is deasserted, the state machine goes to the hunt state and sends continuous training patterns on the SPI-4.2 Interface. Once the Source core is enabled (`SrcEn`=1), the Source Status Interface attempts to acquire synchronization on the FIFO Status Channel. When the Source Status Interface has found the “11” framing pattern, the Source core and monitors for the programmed number of consecutive DIP-2 correct matches (`NumDip2Matches`). When in the hunt state, the Status FIFO is driven to be satisfied (“10”) for all channels.

SYNC

If the number of correct DIP-2 matches are received (`NumDip2Matches`), the Source core goes into the sync state. In this state, the core transmits the flow control data received on the status path (`TStat[1:0]`) onto the user interface. It also transmits the data that has been written into the FIFO on the SPI-4.2 data bus (`TDat[15:0]`). If an incorrect framing pattern (of four consecutive "11") is received, a set number of consecutive DIP-2 errors (defined by `NumDip2Errors`) are received, or if `SrcEn` is deasserted, the state machine returns to the hunt state.

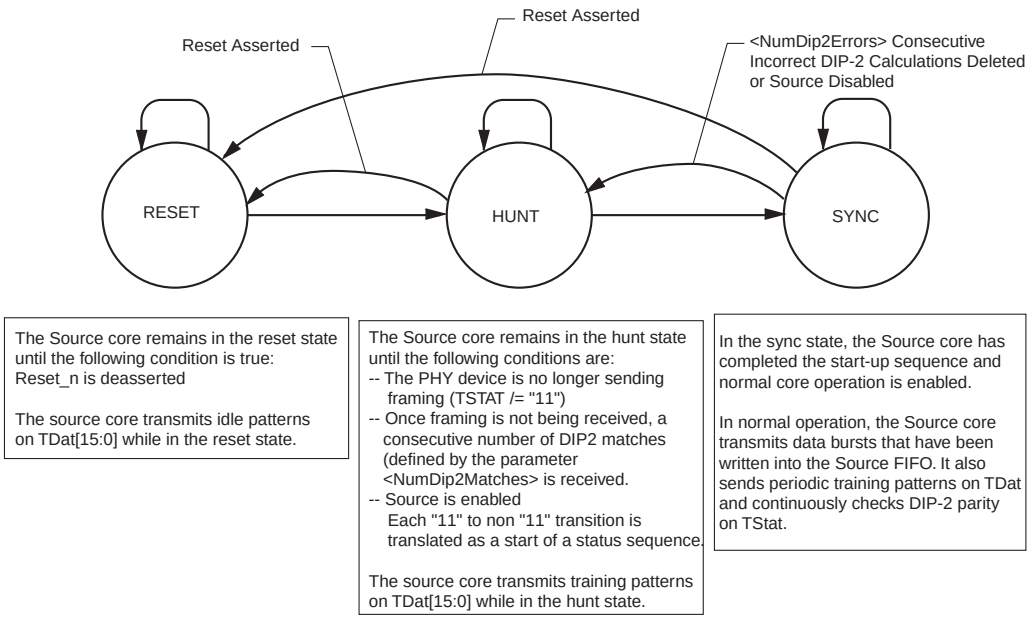


Figure 5-36: **Source Startup Sequence State Machine**

Error Handling

This section describes how the Source core handles the receipt of non-compliant SPI-4.2 and subsequent error handling in a number of common scenarios. This section also provides additional information on the Source core error status signals.

Source Behavior Before Synchronization

- To go into frame, the Source core must receive the number of complete status sequences defined by `NumDip2Matches`.

  Each received status sequence must contain the correct number of entries (defined by `SrcCalendar_Len* SrcCalendar_M`) followed by the DIP-2 calculation and a frame word "11".
- When the core is out of sync, it will re-sync itself to the first 11 to non-11 transition. Once it receives this transition, it will go in-frame once it receives the expected number of correct consecutive DIP-2 words.
- If there is a calendar mismatch with the receiving device, the core may not go into frame. If the mismatch causes DIP-2 errors, then `SrcDIP2Err` is asserted. If a non-”11” frame word is received, then SrcStatFrameErr is asserted.
- When the core is out-of-frame, every 11 to non-11 transition is considered as a start of status sequence.

Source Behavior After Synchronization

- If the core receives an incorrect DIP-2 word, `SrcDIP2Err` flag will be asserted.
- If the core receives an incorrect frame word on `TStat`, the `SrcStatFrameErr` flag will be asserted.
- After a specified number of consecutive DIP-2 Errors (defined by `NumDip2Errors`), the Source core will go out-of-frame.
- If the Source core receives four consecutive frame words ("11"), it will go out-of-frame.
- Once in-frame, the core doesn’t realign to the beginning of a status sequence. The assertion of DIP-2 errors would indicate a possible mismatch with the calendar of the receive device.
- A mismatch with the calendar of the receive device can be detected by polling that you have received a 11 as status on `SrcStat`.

EOP Abort Insertion

An EOP Abort will be inserted when a burst termination on a non-credit boundary without an EOP is followed by an SOP or an address change.

If a burst is paused on a non-credit boundary and then resumed with data (without an SOP) from the same channel, an EOP abort will not be inserted.

Source Out-of-Frame

Source Out-of-Frame (`SrcOof`) is asserted when the Source core is out-of-frame. The following cases can cause the Source core to go out-of-frame:

- Case 1: Resetting the core by asserting `Reset_n`.
  - Action: The Source core will transmit idle cycles when `Reset_n` is asserted. When `Reset_n` is deasserted, the core will initiate the synchronization startup sequence.
- Case 2: If the core receives a number of consecutive DIP-2 errors as defined by `NumDip2Errors`.
  - Action: The Source core terminates the current packet at the next burst boundary, and begins transmitting training patterns on `TDat[15:0]`.

- Case 3: If the core receives four framing sequences 11 in a row on `TStat`.
  - Action: The Source core terminates the current packet at the next burst boundary, and begins transmitting training patterns on `TDat[15:0]`.

After the Source core is in-frame, it will resume transmitting the remaining data stored in the FIFO. (Note that if `SrcFifoReset_n` is asserted, the Source core will remain in-frame (`SrcOof` will be deasserted).

Source DIP-2 Error Handling

The Source core asserts the DIP-2 error flag (`SrcDIP2Err`) when a DIP-2 error is received on *`TStat`*.

Source Status Frame Error Handling

The Source core asserts the frame error flag (`SrcStatFrameErr`) when a non-"11" frame word is received on `TStat`.

Source Pattern Error Handling

Source Pattern Error (`SrcPatternErr`) is asserted when an illegal data pattern is written into the Source FIFO. The two conditions that will trigger this error signal are described below.

- **The address was changed on a non-credit boundary, without an EOP**. In this case, the remainder of that packet will be terminated with an EOP Abort, and sent out the SPI-4.2 bus.
- **The SrcFFMod signal is non-zero without an EOP**. In this case, an EOP abort will not be inserted. When this occurs, the Source core will ignore the `SrcFFMod` value and send the data word with MOD set to zero.

  The source core does not handle per-channel error handling. Handling errors on per-channel basis must be handle by the user logic.

Incorrect Burst Termination

When a burst (that has an odd number of bytes), terminated with an EOP, is not padded with zeros, the Source core sets unused bytes to zero (as required by the SPI4 specification). The Source core will also assert `SrcPatternErr`, but the core will not assert an EOP abort.

XILINX®

Chapter 6

Constraining the Core

This chapter describes the timing and placement constraints required by the SPI-4.2 core to meet performance requirements, including a set of optional constraints. These constraints are provided in an example user constraints file (UCF).

In this chapter, <snk_instance_name> and <src_instance_name> indicate the instance name used to instantiate the Sink and Source cores in HDL respectively. Depending upon where in the user design hierarchy the cores are instantiated, <*instance_name> changes to include the design hierarchy. To illustrate, in the example UCF the cores are instantiated in a top-level wrapper file as <component_name>_pl4_snk_top0 and <component_name>_pl4_src_top0. Therefore, the <snk_instance_name> used for the Sink core is <component_name>_pl4_snk_top0 and the <src_instance_name> used for the Source core is <component_name>_pl4_src_top0. In this context, <component_name> is the name given by the user in the CORE Generator SPI-4.2 GUI.

Overview

The Virtex®-6, Virtex-5, and Virtex-4 FPGAs with the new ChipSync™ I/O technology offer increased flexibility when designing with the SPI-4.2 core, as compared to the SPI-4.2 solution targeting older FPGAs. The following FPGA features allow for designing with minimum core constraints:

- Chip-Sync I/O technology eliminates fixed-pin assignments for SPI-4.2 core
- Increased number of global clocks mitigates the need to perform clock management
- Dedicated regional clock resource further extends the clock capabilities

With these features, core location, choice of I/O banks, and specific pin assignment may be user-specified and driven by external requirements such as PCB routing. In some cases, where the core is using regional clocking, the addition of user-constraints helps guide the implementation tools to a solution.

The large number of possible core implementations makes it impossible for the core to include constraints that cover all implementations. Even if such constraints were generated, they would tend to be less than optimal for any particular FPGA design. The flexibility provided by the core allows constraints to be driven by user-specific design requirements. In many cases, only the timing constraints are required to ensure correct implementation of the core. Any configuration that achieves static-timing closure (for example, meets the timing constraints of the operating clock frequency) is valid and will operate correctly.

The following sections describe how each set of constraints provided in the example UCF interacts with the implementation tool flow. In many cases, the placement constraints are not required; however, when used they must be appropriately modified for the chosen

device and consistent with other constraints. For example, I/O bank locations and Sink and Source clock region constraints must be compatible when used together. For more information about the definition and use of a UCF or specific constraints, see the *Xilinx Libraries Guide* and/or *Development System Reference Guide*.

The descriptions of the constraints provided in the example UCFs in the following sections are divided based on architecture:

- Virtex-5 and Virtex-4 Device Constraints
- Virtex-6 Device Constraints

Virtex-5 and Virtex-4 Device Constraints

Sink Core Required Constraints for Virtex-5 and Virtex-4 FPGAs

Timing Constraints

Timing constraints are crucial to proper operation. The following constraints are provided with the SPI-4.2 core, and you can modify these constraints to meet your system requirements. In the examples below, the target performance is 700 Mbps. Before modifying these constraints, be sure that the modification does not create unconstrained paths.

Time Names for Clocks

The following Sink core clocks constraints are required.

- `NET "RDClk_P" TNM_NET = "RDClk_P";`
- `NET "<snk_instance_name>/U0/cal0/EnRSClk_int*" TNM = FFS snk_cal_flops;`

The following Sink core constraints are for the DCM Standby Logic clock, and are only required when DCM Standby Logic is used.

- `NET "<snk_instance_name>/U0/clk0/rdclk_dcm0/ RINGOSC_INST/CLKWIRE" TNM_NET = “rdclk_clkwire”;`
- `NET "<snk_instance_name>/U0/clk0/rdclk_dcm0/ RINGOSC_INST/_n00006<0>" TNM_NET = “rdclk_clkosc”;`

The following Sink core user-interface clock constraints are required only when the example design is used and the user interface signals are looped back to the Source core interface.

- `NET "CalClk" TNM_NET = "CalClk";`
- `NET "LoopbackClk" TNM_NET = "LoopbackClk";`

The following Sink core user-interface clock constraints are required only when the respective clocks are used.

- `NET "SnkCalClk" TNM_NET = "SnkCalClk";`
- `NET "SnkFFClk" TNM_NET = "SnkFFClk";`
- `NET "SnkStatClk" TNM_NET = "SnkStatClk";`

Timespecs for Clocks

These constraints specify the frequency and duty-cycle of the clock signal. Clock jitter is specified for high frequency clocks. These values can be modified.

The following Sink core clock constraints are always required. Note that the generated SPI-4.2 core may have different timing constraints than the examples provided.

- `TIMESPEC "TS_RDClk_P" = PERIOD "RDClk_P" 350MHz HIGH 50 % INPUT_JITTER 300ps;`
- `TIMESPEC "TS_SnkCalFlops" = FROM "snk_cal_flops" TO "snk_cal_flops" "TS_RDClk_P"/ 4;`

The following DCM Standby Logic clock constraints are required only when DCM Standby Logic is used.

- `TIMESPEC “TS_rdclkclkwire” = PERIOD “rdclk_clkwire” 300 MHz HIGH 50%`
- `TIMESPEC “TS_rdclkclkosc” = PERIOD “rdclk_clkosc” 50 MHz HIGH 50%`

The following Sink core user-interface clock constraints are required when the example design is used, and the user interface signals are looped back to the Source core interface.

- `TIMESPEC "TS_CalClk" = PERIOD "CalClk" 88MHz HIGH 50%;`
- `TIMESPEC "TS_LoopbackClk" = PERIOD "LoopbackClk" 175MHz HIGH 50% INPUT_JITTER 300ps;`

The following Sink core user-interface clock constraints are required only when the respective clocks are used.

- `TIMESPEC "TS_SnkCalClk" = PERIOD "SnkCalClk" 88MHz HIGH 50%;`
- `TIMESPEC "TS_SnkFFClk" = PERIOD "SnkFFClk" 175MHz HIGH 50% INPUT_JITTER 111ps;`
- `TIMESPEC "TS_SnkStatClk" = PERIOD "SnkStatClk" 88MHz HIGH 50%;`

Maxdelay for Reset

The following Sink core reset-signal constraints are always required. The generated SPI-4.2 Core may have different timing constraints than the examples provided below.

- `NET "<snk_instance_name>/U0/core0/queue0/rstcq*/fifo_reset_out" MAXDELAY = 4.56 ns;`
- `NET "<snk_instance_name>/U0/core0/pre0/sfs0/reset_out" MAXDELAY = 4.56 ns;`
- `NET "<snk_instance_name>/U0/core0/post0/sf*/fifo_reset_out*" MAXDELAY = 4.56 ns;`
- `NET "<snk_instance_name>/U0/clkdomain0/sr*/reset_out" MAXDELAY = 4.56 ns;`
- `NET "<snk_instance_name>/U0/cal0/sync_reset_cal/reset_out" MAXDELAY = 4.56 ns;`
- `NET "<snk_instance_name>/U0/io0/s*0/reset_out" MAXDELAY = 4.56 ns;`

These MAXDELAY values may differ depending on target speed grade and core performance.

DCM and Static Alignment Constraints

DCM and BUFR constraints align incoming data (RDat and RCtl) to its clock (RDClk) in the static alignment configuration. The frequency of the DCM clock input must also be specified for complete timing analysis. The following constraints are provided with the SPI-4.2 core, and can be modified.

Input Clock Period for DCM

The following Sink core DCM constraint is always required.

- `INST "<snk_instance_name>/U0/clk0/rdclk_dcm0" CLKIN_PERIOD = 2.85 ;`

This constraint specifies the period of the DCM input clock in nanoseconds. The frequency of the DCM input clock is half that of the *RDClk_P* frequency.

Follow the example below if you are using DCM Standby Logic.

- `INST "<snk_instance_name>/U0/clk0/rdclk_dcm0/STANDBY_INST/DCM_ADV_INST" CLKIN_PERIOD = 2.85 ;`

Phase Shift for DCM

The following constraints are used to align the incoming data (RDat and RCtl) to its clock (RDClk) when global clocking is used. This is accomplished by changing the phase of the I/O clock relative to the data. The default PHASE_SHIFT value, 25, is the correct nominal "PHASE_SHIFT," assuming RDClk transitions at the same time as RDat and RCtl inputs.

- `INST " <snk_instance_name>/U0/clk0/rdclk_dcm0" CLKOUT_PHASE_SHIFT = FIXED;`
- `INST " <snk_instance_name>/U0/clk0/rdclk_dcm0" PHASE_SHIFT = 25;`

Follow the example below if you are using DCM Standby Logic.

- `INST " <snk_instance_name>/U0/clk0/rdclk_dcm0/STANDBY_INST/DCM_ADV_INST" CLKOUT_PHASE_SHIFT = FIXED;`
- `INST " <snk_instance_name>/U0/clk0/rdclk_dcm0/STANDBY_INST/DCM_ADV_INST" PHASE_SHIFT = 25;`

Placement Constraints

Although the SPI-4.2 core does not require fixed pinouts, there are several placement constraints that are critical to meet performance requirements and process through the Xilinx tools. The constraints generated in the CORE Generator system is only an example and should be modified. The constraints can be modified to:

- Move the core placement to a different area
- Target a different device (other than the example LX25-FF668)

See *Constraints Migration Guide* for information on how to migrate the core to a different area or device-package.

DCM, PMCD and BUFG placement

The DCM or PMCD and the BUFG(s) it drives need to be placed in the same top or bottom half of the chip. Although there are no constraints in the example UCF that dictate this, this is a requirement for the Xilinx SPI-4.2 solution.

I/O Placement

In the SPI-4.2 core, you can place the SPI-4.2 I/Os according to need. You are not restricted to placing the I/Os in the bank options provided in the GUI. The placement of the I/Os can be defined using 2 kinds of constraints: bank or pin-lock constraints.

An example of how to define I/O bank constraints are given in the example design file:

- `INST "RCtl*" LOC = "Bank5"; # 1 LVDS I/O pair`
- `INST "RDat*" LOC = "Bank5"; # 16 LVDS I/O pairs`

All the SPI-4.2 I/Os do not need to be in a single bank as given in the example UCF. Ensure that there are enough I/Os in the targeted bank (or banks) when using these constraints. For static alignment core, all SPI-4.2 I/Os need to be placed in a single bank to reduce package and clock skew within the data bus.

An example of how to define I/O pin lock constraints are give in the example design file:

- `NET "RDat_P(15)"  LOC = "G18";`
- `NET "RDat_P(14)"  LOC = "B24";`
- `NET "RDat_P(13)"  LOC = "F18";`
- `NET "RDat_P(12)"  LOC = "E21";`
- `NET "RDat_P(11)"  LOC = "A20";`
- `NET "RDat_P(10)"  LOC = "D22";`

To use these constraints, uncomment the example constraints and modify the pinout accordingly. If you are using an area group to define the placement of the Sink core, place the SPI-4.2 pins (`RCtl` and `RDat`) in the same clock regions as the defined area group. This is especially needed if regional clocking is used.

You can also place `RDClk` using the above constraints type. However, there are some general guidelines when using different clocking options. If regional clocking is chosen, `RDClk` must be placed on a clock capable I/O pin that is in the same region as the Sink core logic.

For instance, in the example UCF, a generalized bank constraint is used to guide I/O placement:

- `INST "RDClk" LOC = "Bank5";`

If global clocking is chosen, `RDClk` must be placed on a pin that is connected to a global clock buffer. For instance, in the example UCF, a generalized bank constraint is used to guide I/O placement:

- `INST "RDClk*" LOC = "Bank3";`

IDELAYCTRL Modules Placement for Virtex-4 FPGAs

When option "Include IDELAYCTRL modules" is selected (by default), the IDELAYCTRLs that calibrate the IDELAY connected to `RDat[15:0]`, `RCtl`, and `RDClk` are instantiated in the core. The following constraints defines where the IDELAYCTRLs are placed. These

IDELAYCTRLs must be placed to cover the clock regions where the SPI-4.2 data and control I/Os are placed.

- `INST "<snk_instance_name>/U0/io0/sict1" LOC = IDELAYCTRL_X0Y4;`
- `INST "<snk_instance_name>/U0/io0/sict2" LOC = IDELAYCTRL_X0Y5;`

This `IDelayCtrl` must be placed in the clock region where the SPI-4.2 `RDClk` I/O is placed.

- `INST "<snk_instance_name>/U0/io01/sict3" LOC = IDELAYCTRL_X1Y3;`

This constraint is needed only when IDELAY is inserted in the `RDClk` path.

When option "Include IDELAYCTRL modules" is not selected, the IDELAYCTRLs must be instantiated in the wrapper by the user to calibrate the IDELAYs connected to `RDat[15:0], RCtl, RDClk`. All the ready signals from these IDELAYCTRLs must be ANDed together to provide the signal that connect to `SnkIdelayCtlRdy` signal. See Instantiating IDELAYCTRL Modules for Virtex-4 Devices, page 179 for the HDL examples and required constraints.

IDELAYCTRL Modules Placement for Virtex-5 FPGAs

The option "Include IDELAYCTRL modules" is disabled for Virtex-5 FPGAs. The user must instantiate the IDELAYCTRLs in the user design to calibrate the IDELAY elements connected to the SPI-4.2 interface signals: `RDat[15:0], RCtl`, and `RDClk`. All the ready signals from the IDELAYCTRLs must be *AND*ed together to provide the signal that connects to `SnkIdelayCtlRdy` input of the Sink Core. See Instantiating IDELAYCTRL Modules for Virtex-6 and Virtex-5 Devices, page 181 for the HDL examples and required constraints.

IOB Register Packing

The following constraints are mandatory for the Sink core. It ensures that the output registers of the RStat and *`RSClk`* signals are packed in the IOB. This guarantees that the timing between the output pad and the register is met.

- `INST "<snk_instance_name>/U0/cal0/rstat1_ff" IOB=TRUE;`
- `INST "<snk_instance_name>/U0/cal0/rstat0_ff" IOB=TRUE;`
- `INST "<snk_instance_name>/U0/cal0/rsclk_ff"  IOB=TRUE;`

Sink Core Optional Constraints for Virtex-5 and Virtex-4 FPGAs

This section contains information about the constraints that can be used in addition to the required constraints. Use of these types of constraints depends on individual user-design.

I/O Standards Constraints

You can define different I/O standards for several input and output pins. To change the I/O standards for *`SnkIdelayRefClk`*, *`RSClk`*, and RStat to LVTTL, uncomment the following constraints in the design:

- `NET "SnkIdelayRefClk" IOSTANDARD = LVTTL;`
- `NET  "RSClk" IOSTANDARD = LVTTL;`
- `NET  "RStat" IOSTANDARD = LVTTL;`

To change the I/O standards of the SPI-4.2 data bus, control bit, and clock inputs to LVDS 25 with internal device termination or DCI, uncomment the following constraints in the design:

- `INST "RDat_P(*)" IOSTANDARD = LVDS_25_DCI;`
- `INST "RDat_N(*)" IOSTANDARD = LVDS_25_DCI;`
- `INST "RCtl_P" IOSTANDARD = LVDS_25_DCI;`
- `INST "RCtl_N" IOSTANDARD = LVDS_25_DCI;`
- `INST "RDClk_P" IOSTANDARD = LVDS_25_DCI;`
- `INST "RDClk_N" IOSTANDARD = LVDS_25_DCI;`

To change the I/O standards of the SPI-4.2 data bus, control bit, and clock inputs to LVDS 25 with internal differential termination, uncomment the following constraints in the design for static configurations:

- `INST "<sink_instance_name>/U0/clk0/rdclk_ibufg0" DIFF_TERM = TRUE;`
- `INST "<sink_instance_name>/U0/io0/buffer_data/Dat*" DIFF_TERM = TRUE;`
- `INST "<sink_instance_name>/U0/io0/buffer_data/Ctl" DIFF_TERM = TRUE;`

For dynamic alignment configuration, uncomment the following:

- `INST "<sink_instance_name>/U0/io0/dpa2/dpa_top0/CTLPAIR/INBUF*" DIFF_TERM = TRUE;`
- `INST "<sink_instance_name>/U0/io0/dpa2/dpa_top0/DATAPAIR*/INBUF/*" DIFF_TERM = TRUE;`

Area Group Constraints

Area group constraints define a specific placement of the Sink core. These constraints are recommended for cores using regional clocking, but are not required for Sink cores using global clocking.

For static alignment, the following constraints are used to place the Sink core in two adjacent clock regions in the example UCF:

- `INST <snk_instance_name>/* AREA_GROUP = AG_pl4_snk;`
- `AREA_GROUP "AG_pl4_snk" RANGE = CLOCKREGION_X0Y3,CLOCKREGION_X0Y4;`

For dynamic alignment, use the following constraints to place the Sink core in 3 adjacent clock regions in the example UCF:

- `INST <snk_instance_name>/* AREA_GROUP = AG_pl4_snk;`
- `AREA_GROUP "AG_pl4_snk" RANGE = CLOCKREGION_X0Y3,CLOCKREGION_X0Y4, CLOCKREGION_X0Y5;`

Timing Ignores Constraints

If Source core static configuration signals are driven statically from a register, apply timing ignore attributes (TIG) to the static configuration signals to create proper timing ignore paths. If these are driven statically from a wrapper file, then a TIG is not needed.

In the example UCF, these constraints are commented out. Uncomment to include the following constraints in the design.

- `NET "SnkAFThresAssert(*)"  TIG;`
- `NET "SnkAFThresNegate(*)"  TIG;`
- `NET "FifoAFMode(*)" TIG;`
- `NET "NumDip4Errors(*)" TIG;`
- `NET "NumTrainSequences(*)" TIG;`
- `NET "RSClkPhase" TIG;`
- `NET "RSClkDiv" TIG;`

Source Core Required Constraints for Virtex-5 and Virtex-4 FPGAs

Timing Constraints

Timing constraints are critical for proper operation. The following constraints are provided with the SPI-4.2 core, and the user can modify these constraints to meet their system requirements. In the examples below, the target performance is 700 Mbps. However, the user is responsible for ensuring that any modification to these constraints does not result in paths which are unconstrained.

Timenames for Clocks

The following Source core clock constraints are always required.

- `NET "SysClk_P" TNM_NET = "";`
- `NET "TSClk" TNM_NET = "TSClk" (for Source status I/O type of LVTTL);`
- `NET "TSClk_P" TNM_NET = "TSClk" (for Source status I/O type of LVDS);`

The following Source core DCM Standby Logic clock constraints are only required when DCM Standby Logic is used.

- `NET "<: print(src_top_name); :>/U0/clk0/tsd/RINGOSC_INST/CLKWIRE" TNM_NET = "tsd_clkwire";`
- `NET "<: print(src_top_name); :>/U0/clk0/tsd/RINGOSC_INST/TOGGLE" TNM_NET = "tsd_clkosc";`
- `NET "<: print(src_top_name); :>/U0/clk0/tdd/RINGOSC_INST/CLKWIRE" TNM_NET = "tdd_clkwire";`
- `NET "<: print(src_top_name); :>/U0/clk0/tdd/RINGOSC_INST/TOGGLE" TNM_NET = "tdd_clkosc";`

The following Source core user-interface clock constraints are only required if the user-interface signals are not looped back to the Source core user interface.

- `NET "SrcCalClk" TNM_NET = "SrcCalClk";`
- `NET "SrcFFClk" TNM_NET = "SrcFFClk";`
- `NET "SrcStatClk" TNM_NET = "SrcStatClk";`

Timespecs for Clocks

The following Source core clock constraints are always required. Note the generated SPI-4.2 core may have different timing constraints than the examples provided.

- `TIMESPEC "TS_SysClk_P" = PERIOD "SysClk_P" 350 MHz HIGH 50% INPUT_JITTER 200ps;`
- `TIMESPEC "TS_TSClk" = PERIOD "TSClk" 88MHz HIGH 50%;`

The following DCM Standby Logic clock constraints are required only when DCM Standby Logic is used.

- `TIMESPEC "TS_tddclkwire" = PERIOD "tdd_clkwire" 300 MHz HIGH 50%;`
- `TIMESPEC "TS_tddclkosc" = PERIOD "tdd_clkosc" 50 MHz HIGH 50%;`
- `TIMESPEC "TS_tsdclkwire" = PERIOD "tsd_clkwire" 300 MHz HIGH 50%;`
- `TIMESPEC "TS_tsdclkosc" = PERIOD "tsd_clkosc" 50 MHz HIGH 50%;`

The following Source core user-interface clock constraints are required only when the respective clocks are used.

- `TIMESPEC "TS_SrcCalClk" = PERIOD "SrcCalClk" 88MHz HIGH 50%;`
- `TIMESPEC "TS_SrcFFClk" = PERIOD "SrcFFClk" 175MHz HIGH 50% INPUT_JITTER 300 ps;`
- `TIMESPEC "TS_SrcStatClk" = PERIOD "SrcStatClk" 88MHz HIGH 50%;`

These constraints specify the frequency and duty cycle of the clock signal. For high frequency clocks, clock jitter is also specified. These values can be modified, based on target performance.

Maxdelay for Reset

The following Source core reset-signal constraints are always required. The generated SPI-4.2 core may have different timing constraints than the examples provided in this section.

- `NET "<src_instance_name>/U0/io0/SrcReset_cp*" MAXDELAY = 4.56 ns;`
- `NET "<src_instance_name>/U0/rst0/*/reset_out_*" MAXDELAY = 4.56 ns;`
- `NET "<src_instance_name>/U0/rst0/SrcClk_Reset_sync" MAXDELAY = 4.56 ns;`
- `NET "<src_instance_name>/U0/srcclk_fiforeset" MAXDELAY = 4.56 ns;`
- `NET "<src_instance_name>/U0/srcffclk_fiforeset" MAXDELAY = 4.56 ns;`

- `NET "<src_instance_name>/U0/srcffclk_reset" MAXDELAY = 4.56 ns;`
- `NET "<src_instance_name>/U0/srcclk_u1_reset" MAXDELAY = 4.56 ns;`
- `NET "<src_instance_name>/U0/srcclk_data_reset" MAXDELAY = 4.56 ns;`

The MAXDELAY values differ based on target speed grade and core performance.

Placement Constraints

Although the SPI-4.2 core does not require fixed pinouts, there are several placement constraints that are critical to meet performance requirements and for processing through the Xilinx tools. The constraints generated in CORE Generator system are provided as an example and should be modified. Placement constraints can be modified to:

- Move the core placement to a different area.
- Target a different device (other than the LX25-FF668 example).

See Constraints Migration for information about migrating the core to a different area or device package.

DCM, PMCD and BUFG placement

DCM or PMCD and the BUFG(s) it drives need to be placed in the same top or bottom half of the chip. Although there are no constraints in the example UCF that dictate this, this is a requirement of the Xilinx SPI-4.2 solution.

I/O Placement

The SPI-4.2 core provides the flexibility in placing the SPI-4.2 I/Os. You are not restricted to placing the I/Os in the bank options provided in the GUI. I/Os placement can be defined using two constraints: bank and pin-lock.

An example for defining I/O banks constraints is provided in the example design file:

- `INST "TDClk*" LOC = "Bank9"; #1 LVDS I/O pair`
- `INST "TCtl*" LOC = "Bank9"; #1 LVDS I/O pair`
- `INST "TDat*" LOC = "Bank9"; #16 LVDS I/O pairs`

All the SPI-4.2 I/Os do not need to be located in a single bank as shown in the example. Ensure that there are enough I/Os in the targeted bank (or banks) when using these constraints. For a Source core that is interfacing with a Sink core configured with static alignment, all SPI-4.2 I/Os need to be placed in a single bank to reduce package and clock skew within the data bus. See Appendix H, SPI-4.2 Source Interface Timing Budget. This appendix illustrates how package and clock skew will affect the source interface timing budget.

An example for defining I/O pin lock constraints is provided in the example design file:

- `NET "TDat_P(15)" LOC = "J23";`
- `NET "TDat_P(14)" LOC = "K22";`
- `NET "TDat_P(13)" LOC = "J26";`
- `NET "TDat_P(12)" LOC = "L19";`
- `NET "TDat_P(11)" LOC = "L21";`

- `NET "TDat_P(10)"  LOC = "K24";`

Uncomment the example constraints and modify the pinout accordingly to use these constraints.

If you use an area group to define placement of the Source core, Xilinx recommends that the SPI-4.2 pins (TCtl and TDat) are placed in the same clock regions as the defined area group. This is especially important when using regional clocking.

Note: To reduce the duty cycle distortion of the TDClk output in a global clocking design, place the TDClk in the bank closest to the BUFG that drives it. This means that the chosen bank needs to be at minimum in the same top or bottom half of the chip as the placement of its clocking components.

You can also place SysClk and TSClk using the two constraints above. However, there are some general guidelines for using different clocking options.

- If regional clocking is chosen, SysClk must be placed on a clock capable I/O pin that is in the same clock region as the Sink core logic. For instance, in the example UCF, a generalized bank constraint is used to guide I/O placement:
  - `INST "SysClk" LOC = "Bank9";`
- If global clocking is chosen, SysClk must be placed on a pin that is connected to a global clock buffer. In the example UCF, a generalized bank constraint is used to guide I/O placement:
  - `INST "SysClk*" LOC = "Bank4";.`
- If regional clocking is chosen, TSClk must be placed on a clock-capable I/O pin in the same clock region as the Source core logic. In the example UCF, a generalized bank constraint is used to guide I/O placement:
  - `INST "TSClk*" LOC = "Bank 9";`
- If global clocking is chosen, TSClk must be placed on a pin that is connected to a global clock buffer. In the example UCF, a generalized bank constraint is used to guide I/O placement:
  - `INST "TSClk*" LOC = "Bank4";`

IOB Register Packing

The following constraints are mandatory for the Source core to ensure that the input registers of the `TStat` and `TSClk` signals are packed in the IOB. This guarantees that the timing between the input pad and the register is met.

- `INST "<src_instance_name>/U0/cal0/tstat1_ff" IOB=TRUE;`
- `INST "<src_instance_name>/U0/cal0/tstat0_ff" IOB=TRUE;`

Source Core Optional Constraints for Virtex-5 and Virtex-4 FPGAs

You can add the following optional constraints, based on your specific design requirements.

TSClk DCM Constraints

The SPI-4.2 specification states that `TStat` should be sampled at the rising of edge `TSClk` and should meet (`tS`) setup time and (`tH`) hold time. If the `TStat` changes at the rising edge of the `TSClk`, the setup time and hold time may not meet. This behavior can be modified by skewing the `TSClk` by 180 degrees. This constraint is only applicable when the `TSClk` clocking is generated internally with a DCM.

The following constraints skew the TSClk by 180 degree. These constraints are commented out in the UCF. Uncomment them to include them in your design.

- `INST "<src_instance_name>/pl4_src_clk0/tsd" CLKOUT_PHASE_SHIFT = FIXED;`
- `INST "<src_instance_name>/pl4_src_clk0/tsd" PHASE_SHIFT = 128;`

When using DCM Standby Logic, uncomment the following:

- `INST "<src_instance_name>"/pl4_src_clk0/tsd/STANDBY_INST/ DCM_ADV_INST" CLKOUT_PHASE_SHIFT=FIXED;`
- `INST"<src_instance_name>"/pl4_src_clk0/tsd/STANDBY_INST/ DCM_ADV_INST" PHASE_SHIFT=128;`

I/O Standards Constraints

You can define different I/O standards for several input and output pins. To change the I/O standards for TSClk and TStat to LVTTL, uncomment the following constraints:

- `NET "TSClk" IOSTANDARD = LVTTL;`
- `NET "TStat" IOSTANDARD = LVTTL;`

To change the I/O standards of the Source core input reference clock (SysClk) to LVPECL 33, uncomment the following constraints:

- `NET "SysClk_P" IOSTANDARD = LVPECL_33;`

To change the I/O standards of the Source core input reference clock (SysClk) to LVDS 25 with internal device termination or DCI, uncomment the following constraints:

- `NET "SysClk_P" IOSTANDARD = LVDS_25_DCI;`
- `NET "SysClk_N" IOSTANDARD = LVDS_25_DCI;`

To change the I/O standards of the Source core input reference clock (SysClk) to LVDS 25 with internal differential termination, uncomment the following constraints:

- `INST "<source_instance_name>/U0/clk0/sysbg" DIFF_TERM = TRUE;`

Area Group Constraints

The area group constraints can be used to define a specific placement of the Source core. These constraints are not required for Source cores that use global clocking distribution but are recommended for Source cores that use regional clocking distribution.

Following is an example of an area group constraint for the Source core placed in three adjacent clock regions:

- `INST <src_instance_name>/* AREA_GROUP = AG_pl4_src;`
- `AREA_GROUP " AG_pl4_src" RANGE = CLOCKREGION_X0Y1, CLOCKREGION_X0Y2, CLOCKREGION_X0Y3;`

Timing Ignores Constraints

If Source core static-configuration signals are driven statically from a register, apply the timing ignore attributes (TIG) to the static configuration signals to create proper timing ignore paths. If these signals are driven statically from a wrapper file, then the TIG is not needed.

In the example UCF, these constraints are commented out. Uncomment to include these constraints in your design.

- `NET "SnkAFThresAssert(*)"  TIG;`
- `NET "SnkAFThresNegate(*)"  TIG;`
- `NET "FifoAFMode(*)" TIG;`
- `NET "NumDip4Errors(*)" TIG;`
- `NET "NumTrainSequences(*)" TIG;`
- `NET "RSClkPhase" TIG;`
- `NET "RSClkDiv" TIG;`

Virtex-6 Device Constraints

Sink Core Required Constraints for Virtex-6 FPGAs

Timing Constraints

Timing constraints are crucial to proper operation. The following constraints are provided with the SPI-4.2 core, and you can modify these constraints to meet your system requirements. In the examples below, the target performance is 700 Mbps. Before modifying these constraints, be sure that the modification does not create unconstrained paths.

Time Names for Clocks

The following Sink core clocks constraints are required.

- `NET “RDClk_P” TNM_NET = “RDClk_P”;`
- `NET "<snk_instance_name>/U0/*cal0/EnRSClk_int*" TNM = FFS snk_cal_flops;`

The following Sink core user-interface clock constraints are required only when the example design is used and the user interface signals are looped back to the Source core interface.

- `NET "CalClk" TNM_NET = "CalClk";`
- `NET "LoopbackClk" TNM_NET = "LoopbackClk";`

The following Sink core user-interface clock constraints are required only when the respective clocks are used.

- `NET "SnkCalClk" TNM_NET = "SnkCalClk";`
- `NET "SnkFFClk" TNM_NET = "SnkFFClk";`
- `NET "SnkStatClk" TNM_NET = "SnkStatClk";`

Timespecs for Clocks

These constraints specify the frequency and duty-cycle of the clock signal. Clock jitter is specified for high frequency clocks. These values can be modified.

The following Sink core clock constraints are always required. Note that the generated SPI-4.2 core may have different timing constraints than the examples provided.

- `TIMESPEC "TS_RDClk_P" = PERIOD "RDClk_P" 350MHz HIGH 50 % INPUT_JITTER 300ps;`
- `TIMESPEC "TS_SnkCalFlops" = FROM "snk_cal_flops" TO "snk_cal_flops" "TS_RDClk_P"/ 4;`

The following Sink core user-interface clock constraints are required when the example design is used, and the user interface signals are looped back to the Source core interface.

- `TIMESPEC "TS_CalClk" = PERIOD "CalClk" 88MHz HIGH 50%;`
- `TIMESPEC "TS_LoopbackClk" = PERIOD "LoopbackClk" 175MHz HIGH 50% INPUT_JITTER 300ps;`

The following Sink core user-interface clock constraints are required only when the respective clocks are used.

- `TIMESPEC "TS_SnkCalClk" = PERIOD "SnkCalClk" 88MHz HIGH 50%;`
- `TIMESPEC "TS_SnkFFClk" = PERIOD "SnkFFClk" 175MHz HIGH 50% INPUT_JITTER 300ps;`
- `TIMESPEC "TS_SnkStatClk" = PERIOD "SnkStatClk" 88MHz HIGH 50%;`

Maxdelay for Reset

The following Sink core reset-signal constraints are always required. The generated SPI-4.2 Core may have different timing constraints than the examples provided below.

- `NET "<snk_instance_name>/U0/*core0/queue0/rstcq*/fifo_reset_out*" MAXDELAY = 4.56 ns;`
- `NET "<snk_instance_name>/U0/*core0/pre0/sfs0/reset_out*" MAXDELAY = 4.56 ns;`
- `NET "<snk_instance_name>/U0/*core0/post0/sf*/fifo_reset_out*" MAXDELAY = 4.56 ns;`
- `NET "<snk_instance_name>/U0/clkdomain0/sr*/reset_out*" MAXDELAY = 4.56 ns;`
- `NET "<snk_instance_name>/U0/*cal0/sync_reset_cal/reset_out*" MAXDELAY = 4.56 ns;`
- `NET "<snk_instance_name>/U0/io0/s*0/reset_out*" MAXDELAY = 4.56 ns;`

These MAXDELAY values may differ depending on target speed grade and core performance.

MMCM and Static Alignment Constraints

The MMCM constraint aligns incoming data (RDat and RCtl) to its clock (RDClk) in the static alignment configuration. The frequency of the MMCM clock input is specified for complete timing analysis. The following constraints are provided with the SPI-4.2 core, and can be modified.

Input Clock Period for MMCM

The following Sink core MMCM constraint is in the ucf file. It is dependent on the input frequency:

- `INST "pl4_snk_clk0/mmcm0" CLKIN1_PERIOD = 2.85;`

This constraint specifies the period of the MMCM input clock in nanoseconds.

Phase Shift for MMCM

The following constraints are used to align the incoming data (RDat and RCtl) to its clock (RDClk) when global clocking is used. This constraint can be modified to change the alignment of the RDClk relative to the RDat and RCtl inputs.

- `INST "pl4_snk_clk0/mmcm0" CLKOUT0_PHASE=90;`

Regional Clocks and Static Alignment Constraints

In regional clocking mode, an IODELAY is inserted in the RDClk path. The IDELAY_VALUE constraint on this IODELAY element align incoming data (RDat and RCtl) to its clock (RDClk) in the static alignment configuration. The following constraints are provided with the SPI-4.2 core, and can be modified.

- `INST "pl4_snk_clk0/rdclk_idel" IDELAY_VALUE = 10 ;`

Placement Constraints

Although the SPI-4.2 core does not require fixed pinouts, there are several placement constraints that are critical to meet performance requirements and process through the Xilinx tools. The constraints generated in the CORE Generator system is only an example and should be modified. The constraints can be modified to:

- Move the core placement to a different area
- Target a different device (other than the example LX75T-FF784)

See *Constraints Migration Guide* for information on how to migrate the core to a different area or device-package.

MMCM and BUFG placement

MMCM and the BUFG(s) it drives need to be placed in the same top or bottom half of the chip. Although there are no constraints in the example UCF that dictate this, this is a requirement for the Xilinx SPI-4.2 solution.

I/O Placement

In SPI-4.2 core, you can place the SPI-4.2 I/Os according to need. You are not restricted to placing the I/Os in the bank options provided in the GUI. The placement of the I/Os can be defined using 2 kinds of constraints: bank or pin-lock constraints.

An example of how to define I/O bank constraints are given in the example design file:

- `INST "RCtl*" LOC = "Bank14"; # 1 LVDS I/O pair`
- `INST "RDat*" LOC = "Bank14"; # 16 LVDS I/O pairs`

All the SPI-4.2 I/Os do not need to be in a single bank as given in the example UCF. Ensure that there are enough I/Os in the targeted bank (or banks) when using these constraints. For sink cores configured as static alignment, all SPI-4.2 I/Os need to be placed in a single bank to reduce package and clock skew within the data bus.

An example of how to define I/O pin lock constraints are give in the example design file:

- `NET "RDat_P(15)"  LOC = "G18";`

- `NET "RDat_P(14)"  LOC = "B24";`
- `NET "RDat_P(13)"  LOC = "F18";`
- `NET "RDat_P(12)"  LOC = "E21";`
- `NET "RDat_P(11)"  LOC = "A20";`
- `NET "RDat_P(10)"  LOC = "D22";`

To use these constraints, uncomment the example constraints and modify the pinout accordingly. If you are using an area group to define the placement of the Sink core, place the SPI-4.2 pins (`RCtl` and `RDat`) in the same clock regions as the defined area group. This is especially needed if regional clocking is used.

You can also place `RDClk` using the above constraints type. However, there are some general guidelines when using different clocking options. If regional clocking is chosen, `RDClk` must be placed on a clock capable I/O pin.

For instance, in the example UCF:

- `INST "RDClk" LOC = "U26";`

If global clocking is chosen, `RDClk` must be placed on a pin that is connected to a global clock buffer. For instance, in the example UCF:

- `INST "RDClk*" LOC = "AE13";`

IDELAYCTRL Modules Placement for Virtex-6 FPGAs

The option “Include IDELAYCTRL modules” is disabled for Virtex-6 FPGAs. The user must instantiate the IDELAYCTRLs in the user design to calibrate the IDELAY elements connected to the SPI-4.2 interface signals: RDat[15:0], RCtl, and RDClk. All the ready signals from the IDELAYCTRLs must be ANDed together to provide the signal that connects to SnkIdelayCtlRdy input of the Sink Core. See Instantiating IDELAYCTRL Modules for Virtex-6 and Virtex-5 Devices, page 181 for the HDL examples and required constraints.

IOB Register Packing

The following constraints are mandatory for the Sink core. It ensures that the output registers of the RStat and *`RSClk`* signals are packed in the IOB. This guarantees that the timing between the output pad and the register is met.

- `INST "<snk_instance_name>/U0/*cal0/rstat1_ff" IOB=TRUE;`
- `INST "<snk_instance_name>/U0/*cal0/rstat0_ff" IOB=TRUE;`
- `INST "<snk_instance_name>/U0/*cal0/rsclk_ff"  IOB=TRUE;`

Sink Core Optional Constraints for Virtex-6 FPGAs

This section contains information about the constraints that can be used in addition to the required constraints. Use of these types of constraints depends on individual user-design.

I/O Standards Constraints

To change the I/O standards of the SPI-4.2 data bus, control bit, and clock inputs to LVDS 25 with internal device termination or DCI, uncomment the following constraints in the design:

- `INST "RDat_P(*)" IOSTANDARD = LVDS_25_DCI;`

- `INST "RDat_N(*)" IOSTANDARD = LVDS_25_DCI;`
- `INST "RCtl_P" IOSTANDARD = LVDS_25_DCI;`
- `INST "RCtl_N" IOSTANDARD = LVDS_25_DCI;`
- `INST "RDClk_P" IOSTANDARD = LVDS_25_DCI;`
- `INST "RDClk_N" IOSTANDARD = LVDS_25_DCI;`

To change the I/O standards of the SPI-4.2 data bus, control bit, and clock inputs to LVDS 25 with internal differential termination, uncomment the following constraints in the design for static configurations:

- `INST "pl4_snk_clk0/rdclk_ibuf0"  DIFF_TERM = TRUE;`
- `INST "<sink_instance_name>/U0/io0/StaticAlign.buffer_data/stat_v6.BUFIN*" DIFF_TERM = TRUE;`

For dynamic alignment configuration, uncomment the following:

- `INST "pl4_snk_clk0/rdclk_ibuf0" DIFF_TERM = TRUE;`
- `INST "<sink_instance_name>/U0/io0/DynamicAlign2.dpa2/dpa_top0/LDATA_V6.LDATA*.DATAPAIR/BUFIN.INBUF*" * DIFF_TERM = TRUE;`
- `INST "<sink_instance_name>/U0/io0/DynamicAlign2.dpa2/dpa_top0/HDATA_V6.HDATA*.DATAPAIR/BUFIN.INBUF*" DIFF_TERM = TRUE;`
- `INST "<sink_instance_name>/U0/io0/DynamicAlign2.dpa2/dpa_top0/CTLPAIR_V6.CTLPAIR/BUFIN.INBUF*" DIFF_TERM = TRUE;`

Area Group Constraints

Area group constraints define a specific placement of the Sink core. These constraints are recommended for cores using regional clocking, but are not required for Sink cores using global clocking.

The following constraints are used to place the Sink core in two adjacent clock regions in the example UCF:

- `INST <snk_instance_name>/* AREA_GROUP = AG_pl4_snk;`
- `AREA_GROUP "AG_pl4_snk" RANGE = CLOCKREGION_X0Y0,CLOCKREGION_X0Y1;`

Timing Ignores Constraints

If Source core static configuration signals are driven statically from a register, apply timing ignore attributes (TIG) to the static configuration signals to create proper timing ignore paths. If these are driven statically from a wrapper file, then a TIG is not needed.

In the example UCF, these constraints are commented out. Uncomment to include the following constraints in the design.

- `NET "SnkAFThresAssert(*)"  TIG;`
- `NET "SnkAFThresNegate(*)"  TIG;`
- `NET "FifoAFMode(*)" TIG;`
- `NET "NumDip4Errors(*)" TIG;`
- `NET "NumTrainSequences(*)" TIG;`

- `NET "RSClkPhase" TIG;`
- `NET "RSClkDiv" TIG;`

Modifying HIGH_PERFORMANCE_MODE attribute for IODELAYE1 to Improve Power Consumption

The SPI-4.2 core netlist generated by CORE Generator for Virtex-6 FPGA designs sets the HIGH_PERFORMANCE_MODE attribute on the IODELAYE1 elements to "TRUE". If targeting a performance of less than 700 Mbps, this attribute can be changed to "FALSE" in the UCF file. This will allow the IODELAY to consume less power.

The following constraints are used to change the HIGH_PERFORMANCE_MODE attribute:

- `INST "pl4_snk_clk0/rdclk_idel"  HIGH_PERFORMANCE_MODE = "FALSE" ;`
- `INST "<sink_core_instance_name>/U0/io0/*dpa2/dpa_top0/ *DATAPAIR*/MASTER_DELAY" HIGH_PERFORMANCE_MODE = "FALSE" ;`
- `INST "<sink_core_instance_name>/U0/io0/*dpa2/dpa_top0/ *DATAPAIR*/SLAVE_DELAY"  HIGH_PERFORMANCE_MODE = "FALSE" ;`
- `INST "<sink_core_instance_name>/U0/io0/*dpa2/dpa_top0/ *CTLPAIR*/SLAVE_DELAY"   HIGH_PERFORMANCE_MODE = "FALSE" ;`
- `INST "<sink_core_instance_name>/U0/io0/*dpa2/dpa_top0/ *CTLPAIR*/MASTER_DELAY"  HIGH_PERFORMANCE_MODE = "FALSE" ;`

Source Core Required Constraints for Virtex-6 FPGAs

Timing Constraints

Timing constraints are critical for proper operation. The following constraints are provided with the SPI-4.2 core, and the user can modify these constraints to meet their system requirements. In the examples below, the target performance is 700 Mbps. However, the user is responsible for ensuring that any modification to these constraints does not result in paths which are unconstrained.

Timenames for Clocks

The following Source core clock constraints are always required.

- `NET "SysClk_P" TNM_NET = "SysClk_P";`
- `NET "TSClk_P" TNM_NET = "TSClk";`

The following Source core user-interface clock constraints are only required if the user-interface signals are not looped back to the Source core user interface.

- `NET "SrcCalClk" TNM_NET = "SrcCalClk";`
- `NET "SrcFFClk" TNM_NET = "SrcFFClk";`
- `NET "SrcStatClk" TNM_NET = "SrcStatClk";`

Timespecs for Clocks

The following Source core clock constraints are always required. Note the generated SPI-4.2 core may have different timing constraints than the examples provided.

- `TIMESPEC "TS_SysClk_P" = PERIOD "SysClk_P" 350 MHz HIGH 50% INPUT_JITTER 300ps;`
- `TIMESPEC "TS_TSClk" = PERIOD "TSClk" 88MHz HIGH 50%;`

The following Source core user-interface clock constraints are required only when the respective clocks are used.

- `TIMESPEC "TS_SrcCalClk" = PERIOD "SrcCalClk" 88MHz HIGH 50%;`
- `TIMESPEC "TS_SrcFFClk" = PERIOD "SrcFFClk" 175MHz HIGH 50% INPUT_JITTER 300 ps;`
- `TIMESPEC "TS_SrcStatClk" = PERIOD "SrcStatClk" 88MHz HIGH 50%;`

These constraints specify the frequency and duty cycle of the clock signal. For high frequency clocks, clock jitter is also specified. These values can be modified based on target performance.

Maxdelay for Reset

The following Source core reset-signal constraints are always required. The generated SPI-4.2 core may have different timing constraints than the examples provided in this section.

- `NET "<src_instance_name>/U0/io0/SrcReset_cp*" MAXDELAY = 4.56 ns;`
- `NET "<src_instance_name>/U0/rst0/*/*reset_out_*" MAXDELAY = 4.56 ns;`
- `NET "<src_instance_name>/U0/rst0/SrcClk_Reset_sync" MAXDELAY = 4.56 ns;`

The MAXDELAY values differ based on target speed grade and core performance.

Placement Constraints

Although the SPI-4.2 core does not require fixed pinouts, there are several placement constraints that are critical to meet performance requirements and for processing through the Xilinx tools. The constraints generated in CORE Generator system are provided as an example and should be modified. Placement constraints can be modified to:

- Move the core placement to a different area.
- Target a different device (other than the LX75T-FF784 example).

See Constraints Migration for information about migrating the core to a different area or device package.

MMCM and BUFG placement

MMCM and the BUFG(s) it drives need to be placed in the same top or bottom half of the chip. Although there are no constraints in the example UCF that dictate this, this is a requirement for the Xilinx SPI-4.2 solution.

I/O Placement

The SPI-4.2 core provides the flexibility in placing the SPI-4.2 I/Os. You are not restricted to placing the I/Os in the bank options provided in the GUI. I/Os placement can be defined using two constraints: bank and pin-lock.

An example for defining I/O banks constraints is provided in the example design file:

- `INST "TDClk*" LOC = "Bank16"; #1 LVDS I/O pair`
- `INST "TCtl*" LOC = "Bank16"; #1 LVDS I/O pair`
- `INST "TDat*" LOC = "Bank16"; #16 LVDS I/O pairs`

All the SPI-4.2 I/Os do not need to be located in a single bank as shown in the example. Ensure that there are enough I/Os in the targeted bank (or banks) when using these constraints. For a source core that is interfacing with a sink core configured with static alignment, it is recommended that all SPI-4.2 I/Os are placed in a single bank to reduce package and clock skew within the data bus. See Appendix H, SPI-4.2 Source Interface Timing Budget. This appendix illustrates how package and clock skew will affect the source interface timing budget.

An example for defining I/O pin lock constraints is provided in the example design file:

- `NET "TDat_P(15)"  LOC = "J23";`
- `NET "TDat_P(14)"  LOC = "K22";`
- `NET "TDat_P(13)"  LOC = "J26";`
- `NET "TDat_P(12)"  LOC = "L19";`
- `NET "TDat_P(11)"  LOC = "L21";`
- `NET "TDat_P(10)"  LOC = "K24";`

Uncomment the example constraints and modify the pinout accordingly to use these constraints.

If you use an area group to define placement of the Source core, Xilinx recommends that the SPI-4.2 pins (TCtl and TDat) are placed in the same clock regions as the defined area group. This is especially important when using regional clocking.

Note: To reduce the duty cycle distortion of the TDClk output in a global clocking design, place the TDClk in the bank closest to the BUFG that drives it. This means that the chosen bank needs to be at minimum in the same top or bottom half of the chip as the placement of its clocking components.

You can also place SysClk and TSClk using the two constraints above. However, there are some general guidelines for using different clocking options.

If regional clocking is chosen, SysClk must be placed on a clock capable I/O pin that is in the same clock region as the Sink core logic. For instance, in the example UCF, a generalized bank constraint is used to guide I/O placement:

- `INST "SysClk" LOC = "Bank16";`

If global clocking is chosen, SysClk must be placed on a pin that is connected to a global clock buffer. In the example UCF, a generalized bank constraint is used to guide I/O placement:

- `INST "SysClk*" LOC = "Bank35";`

IOB Register Packing

The following constraints are mandatory for the Source core to ensure that the input registers of the `TStat` and `TSClk` signals are packed in the IOB. This guarantees that the timing between the input pad and the register is met.

- `INST "<src_instance_name>/U0/*cal0/tstat1_ff" IOB=TRUE;`
- `INST "<src_instance_name>/U0/*cal0/tstat0_ff" IOB=TRUE;`

Source Core Optional Constraints for Virtex-6 FPGAs

You can add the following optional constraints, based on your specific design requirements.

TSClk MMCM Constraints

The SPI-4.2 specification states that `TStat` should be sampled at the rising of edge `TSClk` and should meet (`tS`) setup time and (`tH`) hold time. If the `TStat` changes at the rising edge of the `TSClk`, the setup time and hold time may not meet. This behavior can be modified by skewing the `TSClk` by 180 degrees. This constraint is only applicable when the `TSClk` clocking example design is generated with an MMCM.

The following constraints skew the `TSClk` by 180 degree. These constraints are commented out in the UCF. Uncomment them to include them in your design.

- `INST "pl4_src_clk0/mmcm1" CLKOUT0_PHASE = 180;`

I/O Standards Constraints

You can define different I/O standards for several input and output pins. To change the I/O standards of the Source core input reference clock (`SysClk`) to LVDS 25 with internal device termination or DCI, uncomment the following constraints:

- `NET "SysClk_P"  IOSTANDARD = LVDS_25_DCI;`
- `NET "SysClk_N"  IOSTANDARD = LVDS_25_DCI;`

To change the I/O standards of the Source core input reference clock (`SysClk`) to LVDS 25 with internal differential termination, uncomment the following constraints:

- `INST "pl4_src_clk0/sysclk_ibuf*" DIFF_TERM = TRUE;`

Area Group Constraints

Area group constraints can be used to define a specific placement of the Source core. These constraints are not required for Source cores that use global clocking distribution but are recommended for Source cores that use regional clocking distribution.

Following is an example of an area group constraint for the Source core placed in three adjacent clock regions:

- `INST <src_instance_name>/* AREA_GROUP = AG_pl4_src;`
- `AREA_GROUP " AG_pl4_src" RANGE = CLOCKREGION_X0Y1, CLOCKREGION_X0Y2;`

Timing Ignore Constraints

If Source core static-configuration signals are driven statically from a register, apply the timing ignore attributes (`TIG`) to the static configuration signals to create proper timing ignore paths. If these signals are driven statically from a wrapper file, then the TIG is not needed.

In the example UCF, these constraints are commented out. Uncomment to include these constraints in your design.

- `NET "SnkAFThresAssert(*)"  TIG;`
- `NET "SnkAFThresNegate(*)"  TIG;`

- `NET "FifoAFMode(*)" TIG;`
- `NET "NumDip4Errors(*)" TIG;`
- `NET "NumTrainSequences(*)" TIG;`
- `NET "RSClkPhase" TIG;`
- `NET "RSClkDiv" TIG;`

User Constraints

In certain cases, additional constraints may need to be added to cover other logic implemented in the design. While the UCF provided is designed to completely constrain the Xilinx SPI-4.2 core, it may not adequately constrain user-implemented logic that is interfaced to the core.

Constraints Migration

The example UCF must be modified to migrate the core to a different area or target device. The examples in this section illustrate the changes necessary to migrate the Sink and Source cores to a user-defined location on a XC4VLX40-FF1148 Virtex-4 FPGA part. This is achieved by modifying the example UCF. The static alignment example shows the migration of the Sink and Source cores to the south-west region of the part (banks 11 and 7, respectively). The dynamic phase alignment example shows the migration of the Sink and Source cores to the north-west region of the part (banks 5 and 9, respectively).

New Target Region or Device Package

When selecting a new target region or device package, first verify that the new region has adequate resources for the generated core. Specifically, the following resources should be taken into consideration:

- Block RAMs
- I/O Pins (in targeted I/O banks)
- Logic cells
- Clocking resources: DCM or MMCM, regional and global buffers

Below are some typical region selections within a device.

- Source Core: Two adjacent clock regions on the same side of the device, East or West.
- Sink Core (static): Two adjacent clock regions on the same side of the device.
- Sink Core (dynamic): Three adjacent clock regions on the same side of the device.

The east side is the side of the device with even numbered I/O banks: 6, 8, 10, and so forth. The west side is the side of the device with odd numbered I/O banks: 5, 7, 9, and so on.

When choosing a target region, be aware that placing the core in a region that contains a Power PC or other hard-embedded IP may increase the difficulty of the tools to place the core and meet the timing constraints of the core.

A target region or device without adequate resources assigned to it will result in tool errors; not due to portability issues, but to resource issues.

Modifying the User Constraints File

Once the target region is selected, the UCF must be modified. While modifying the constraints, ensure that changes are within the specifications described by the Sink and Source core required constraints. The UCF modifications are provided in this section.

Note: Using optional constraints is at user discretion.

Target Device

Change `CONFIG_PART` constraint to a desired device.

Sink Core

Specify pin placements for the SPI-4.2 interface I/Os (`RCtl*` and `RDat*`). If you are using regional clocking, the I/Os must be constraint to pins that coincide with the clock regions of the Sink core. If using I/O bank constraints, verify that the targeted bank can accommodate the total LVDS I/O pairs.

In the following example, Bank 11 must contain at least 17 LVDS I/O pairs:

- `INST "RCtl*" LOC = "Bank11"; # 1 LVDS I/O pair`
- `INST "RDat*" LOC = "Bank11"; #16 LVDS I/O pairs`

Specify pin placement for `RDClk` I/O. See Placement Constraints, page 132 (Virtex-5 and Virtex-4 devices) or Placement Constraints, page 143 (Virtex-6 devices) for more information about placement constraints. For example:

- `INST "RDClk*" LOC = "Bank4";`

Specify an area group constraint if using regional clocking. In the UCF, area group `AG_pl4_snk` is defined to be two (three for dynamic) adjacent clock regions on the same side of the device. For example:

- AREA_GROUP "AG_pl4_snk" RANGE = CLOCKREGION_X0Y2, CLOCKREGION_X0_Y3;

For Virtex-4 device design, place `IDELAYCTRL` components in the same clock regions as the SPI-4.2 Sink Interface I/O (for example, `RDAT_P`, `RCTL_P`, `RDCLK_P`). For example:

- `INST "<sink_instance_name>/U0/io0/sict1" LOC = IDELAYCTRL_X0Y2;`
- `INST "<sink_instance_name>/U0/io0/sict2" LOC = IDELAYCTRL_X0Y3;`
- `INST "<sink_instance_name>/U0/io0/sict3" LOC = IDELAYCTRL_X1Y3;`

Source Core

Specify pin placement for `SysClk` I/O. See Placement Constraints, page 138 (Virtex-5 and Virtex-4 devices) or Placement Constraints, page 147 (Virtex-6 devices) for detailed information. For example:

- `INST "SysClk*" LOC = "Bank 4";`

Specify pin placements for the SPI-4.2 interface I/Os (`TDClk*`, `TDat*` and `TCtl*`). If you are using regional clocking, the I/Os must be constraint to pins that coincide with the clock regions of the Source core. If I/O bank constraints are used, verify that the targeted bank can accommodate the total LVDS I/O pairs. In the following example, Bank 7 contains at least 18 LVDS I/O pairs:

- `INST "TDClk*" LOC = "Bank7"; # 1 LVDS I/O pair`

- `INST "TCtl*" LOC = "Bank7"; # 1 LVDS I/O pair`
- `INST "TDat*" LOC = "Bank7"; # 16 LVDS I/O pair`

Specify pin placement for "TSClk" I/O. See Placement Constraints, page 138 (Virtex-5 and Virtex-4 devices) or Placement Constraints, page 147 (Virtex-6 devices). For example:

- `INST "TSClk" LOC = "Bank3";  # For Virtex-4 and Virtex-5`
- `INST "TSCLK*" LOC = "Bank3"; # For Virtex-6`

Specify an area group constraint when using regional clocking. In the UCF, area group "AG_pl4_src" is defined to be two adjacent clock regions on the same side of the device.

- AREA_GROUP "AG_pl4_src" RANGE = CLOCKREGION_X0_Y0, CLOCKREGION_X0_Y1;

Special Consideration for Dynamic Phase Alignment

If the placement of the Sink core (dynamic phase alignment configuration) is on the top or bottom of the Source core, the following guidelines are recommended.

- When area group constraints are used, the Sink and Source cores both need to be 3 clock regions tall. For this condition the Sink and Source core can share one clock region. This results in a total of 5 required clock regions instead of 6. For example, in the delivered UCF, the area groups are defined as:
- AREA_GROUP "AG_pl4_snk" RANGE = CLOCKREGION_X0Y3, CLOCKREGION_X0Y4, CLOCKREGION_X0Y5;
- AREA_GROUP "AG_pl4_src" RANGE = CLOCKREGION_X0Y1, CLOCKREGION_X0Y2, CLOCKREGION_X0Y3;
- The I/O constraints for the Sink core need to be locked to the I/O bank within the clock regions of the Sink core. This is recommended for regional clocking. For example, in the delivered UCF, the I/O banks placement are defined as:
  - `INST "RCtl*" LOC = "Bank5";`
  - `INST "RDat*" LOC = "Bank5";`

Bank 5 is within clock region X0Y5 and X0Y4. For Virtex-4 devices, the IDELAYCTRL for the Sink core also needs to be locked to the I/O bank within the clock regions of the Sink core.

- The I/O constraints for the Source core need to be locked to the I/O bank associated with the I/O bank within the clock regions on the Source core. This is recommended for regional clocking. For example, in the delivered UCF, the I/O banks placements are defined as:
  - `INST "TDClk*" LOC = "Bank9";`
  - `INST "TCtl*" LOC = "Bank9";`
  - `INST "TDat*" LOC = "Bank9";`

Bank 9 is within clock region X0Y3 and X0Y2. This includes the overlay clock region, X0Y2. I/O Bank 7 is not acceptable because its associated clock regions, X0Y0 and X0Y1, are not in the AG_pl4_src area group.

XILINX®

Chapter 7

Special Design Considerations

This chapter describes the special design considerations to be made when designing with the Xilinx SPI-4.2 core, including:

- Sink Clocking Options for Virtex-5 and Virtex-4 Devices
- Sink Clocking Options for Virtex-6 Devices
- Source Clocking Options for Virtex-5 and Virtex-4 Devices
- Source Clocking Options for Virtex-6 Devices
- Clocking Guidelines
- Instantiating IDELAYCTRL Modules for Virtex-4 Devices
- Instantiating IDELAYCTRL Modules for Virtex-6 and Virtex-5 Devices
- Multiple Core Implementations

Sink Clocking Options for Virtex-5 and Virtex-4 Devices

The SPI-4.2 solution provides four clocking options that are embedded in the Sink core. Depending on the chosen clocking option, different clock resources are used. Table 7-1 provides the clocking resource count for each clocking option.

Table 7-1: **Sink Core Clocking Option Resources**

Clocking Option	BUFR	BUFG	DCM	PMCD
Global clocking with DCM	0	2/3[a]	1	0
Global clocking with DCM Standby Logic (Virtex®-4 FPGAs only)	0	2/3	1	0
Global clocking with PMCD (Virtex-4 FPGAs only)	0	2/3[a]	0	1
Regional clocking	1	0/1[a]	0	0

a. The Sink core requires that the SnkIdelayRefClk clock (200 MHz reference clock) is driven by a global clock buffer. This reference clock provides a time reference to IDELAYCTRL modules to calibrate all the individual delay elements (IDELAY) in the region. Multiple cores need only one global clock buffer to distribute the SnkIdelayRefClk clock.

Global Clocking with DCM

This option uses the DCM and global clock routing to generate and distribute a full-rate clock (`RDCLK0_GP`) and a half-rate clock (`RDClkDiv_GP`). The global clock lines are implemented differentially in Virtex®-5 and Virtex-4 FPGA devices. Figure 7-1 illustrates

this configuration for static alignment, and Figure 7-2, page 155 illustrates a dynamic-alignment configuration.

To use global clocking with DCM in Virtex-4 FPGAs, the following DCM timing parameters must be met to ensure that the DCMs meet maximum frequency specifications: `TCONFIG`, `DCM_INPUT_CLOCK_STOP`, and `DCM_RESET`. See the *Virtex-4 FPGA Data Sheet* and Answer Record 21127 for more information.

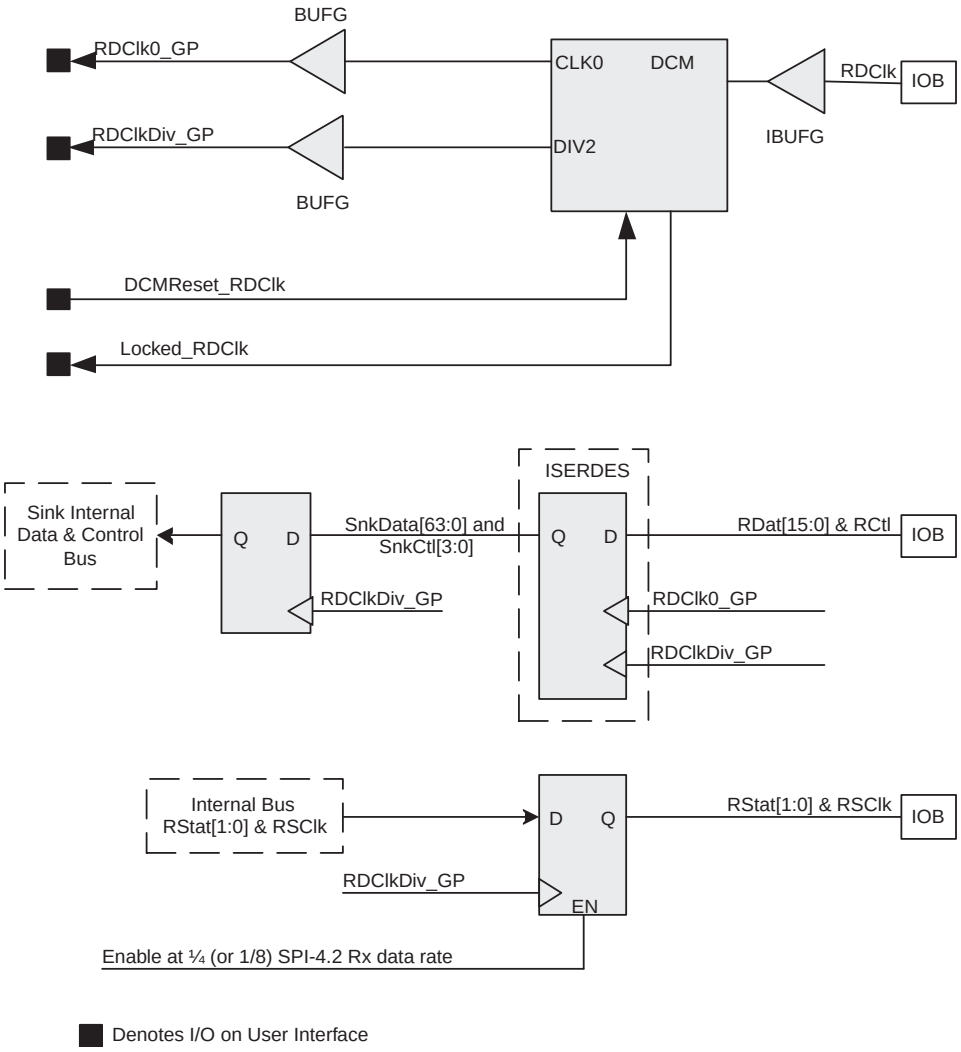


Figure 7-1: **Sink Global Clocking Option with DCM (Virtex-5 and Virtex-4)**

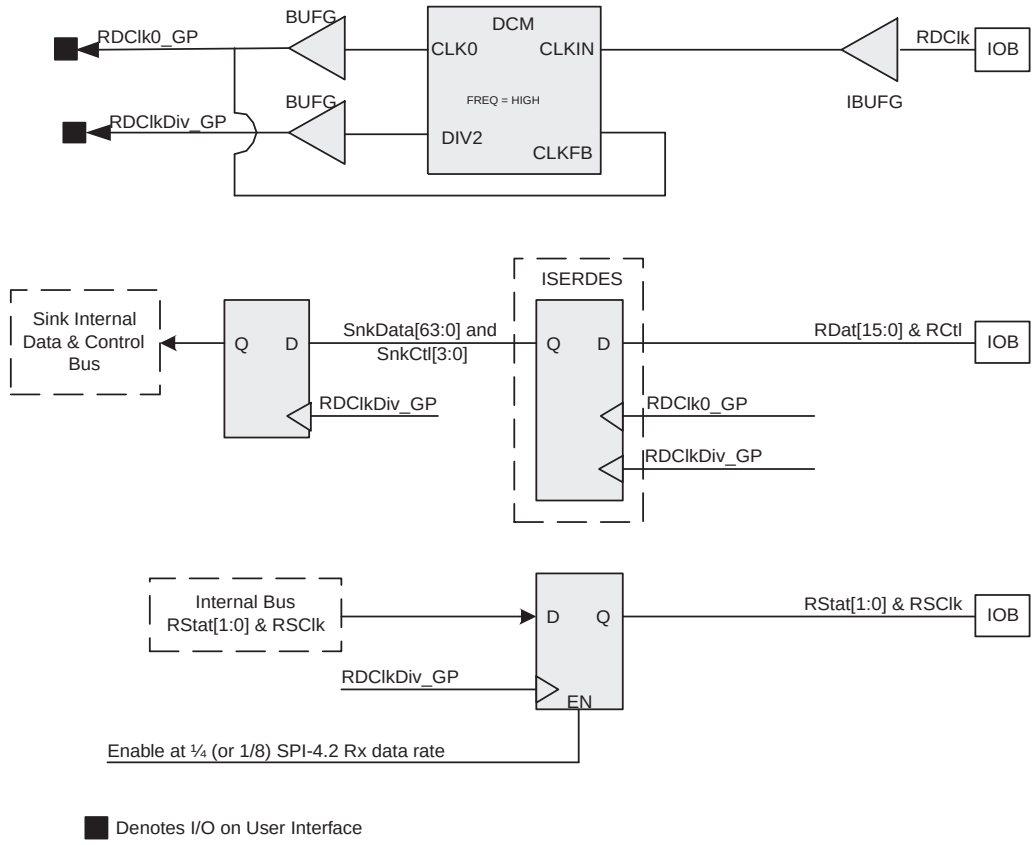


Figure 7-2: **Sink Global Clocking with DCM for Dynamic Phase Alignment (Virtex-5 and Virtex-4)**

Global Clocking with DCM Standby Logic (Virtex-4 FPGAs only)

This option is available only in Virtex-4 FPGA designs. It uses DCM and global clock routing to generate and distribute a full-rate clock (`RDClk0_GP`) and a half-rate clock (`RDClkDiv_GP`). It supports long clock stop and reset times. These conditions occur when the DCM timing parameters, `DCM_INPUT_CLOCK_STOP` and *`DCM_RESET`* are not met. An additional DCM Standby Logic is used to fulfill these requirements. This configuration is illustrated in Figure 7-4, page 158 for static and dynamic alignment configuration. The DCM Standby Logic monitors the input clock, and the reset to the DCM. If the input clock or the feedback clock is not toggling for more than 100ms, or if a reset is asserted, the macro will keep the DCM in continuous calibration mode. When the input clock resumes or the reset is deasserted, the macro will reset the DCM and continue monitoring.

Note: To simulate a core with this feature, use a post-par (SIMPRIM) simulation model. Simulation with a functional (UNISIM) model that is generated with the core should not be used.

Global Clocking with PMCD (Virtex-4 FPGAs only)

This option is available only in Virtex-4 FPGA designs. It uses the PMCD and global clock routing to generate both a full-rate clock (*`RClk0_GP`*) and a half-rate clock (*`RDClkDiv_GP`*). The global clock lines are implemented differentially in Virtex-4 FPGA

devices. This configuration is illustrated in Figure 7-4. The implementation is available in dynamic alignment configurations only.

Regional Clocking

Regional clocking uses the regional clock buffer resources BUFIO and BUFR to generate the full-rate clock (*RDClk0_GP*) and a half-rate clock (*RDClkDiv_GP*). This configuration is illustrated in Figure 7-5 and Figure 7-6. This implementation is the same for both static and dynamic alignment configurations.

A list of the Sink clocks and their description is given in Table 3-7, page 42 of this guide. The DCM reset, lost, and locked signals are defined in Table 3-8, page 42 of this guide.

IDELAY on RDClk

The user also has the option to insert an IDELAY in the RDClk path, if needed. This option is only available when Global Clocking with DCM is used, and must be enabled when using the DPA clock adjustment feature. The user can delay the RDCLK input by defining an IDELAY tap value before it is generated or distributed internally. For Regional Clocking, this feature is always enabled.

When this option is selected, the Generate Dedicated Input IDELAYCTRL Reset option is recommended because the IDELAYCTRL reset pin is by default connected to the Sink Core reset, and may cause a lockup issue.

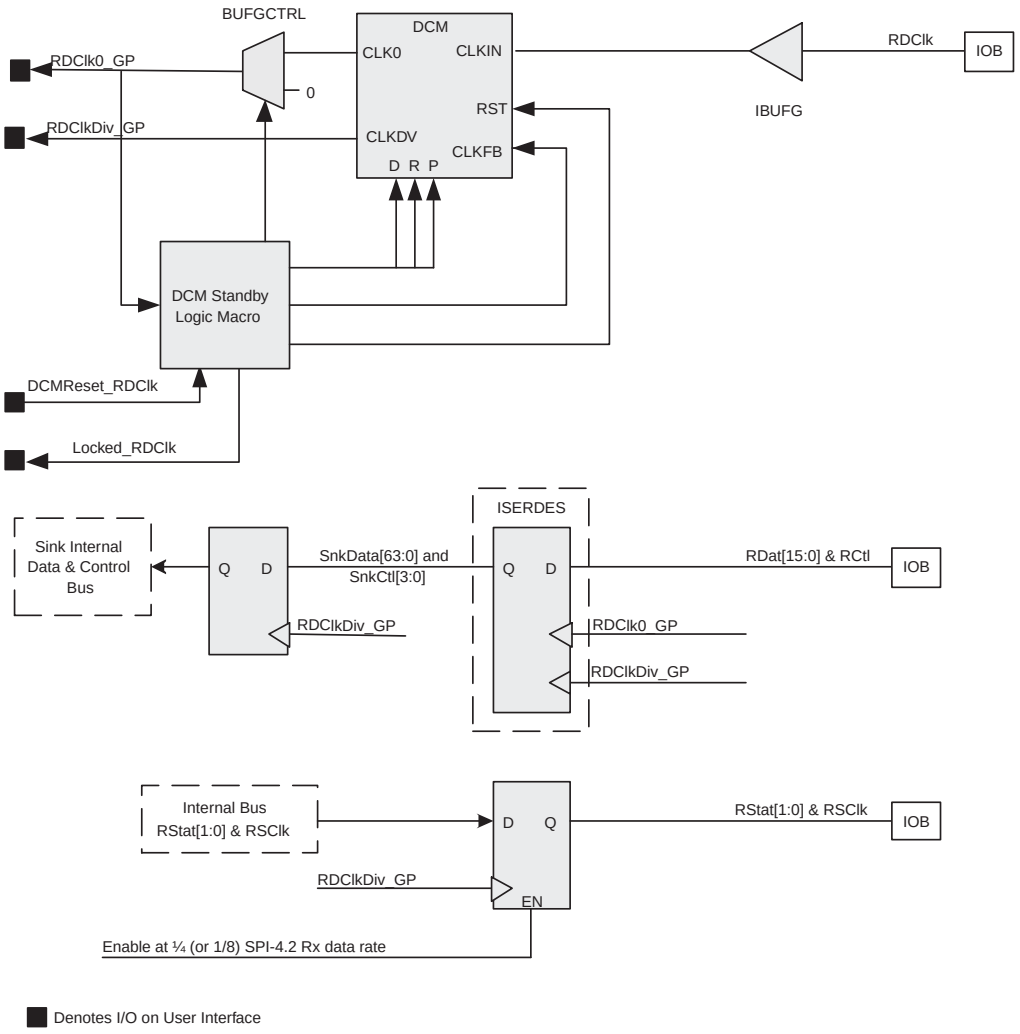


Figure 7-3: **Sink Global Clocking with DCM Standby Logic (Virtex-4 FPGAs Only)**

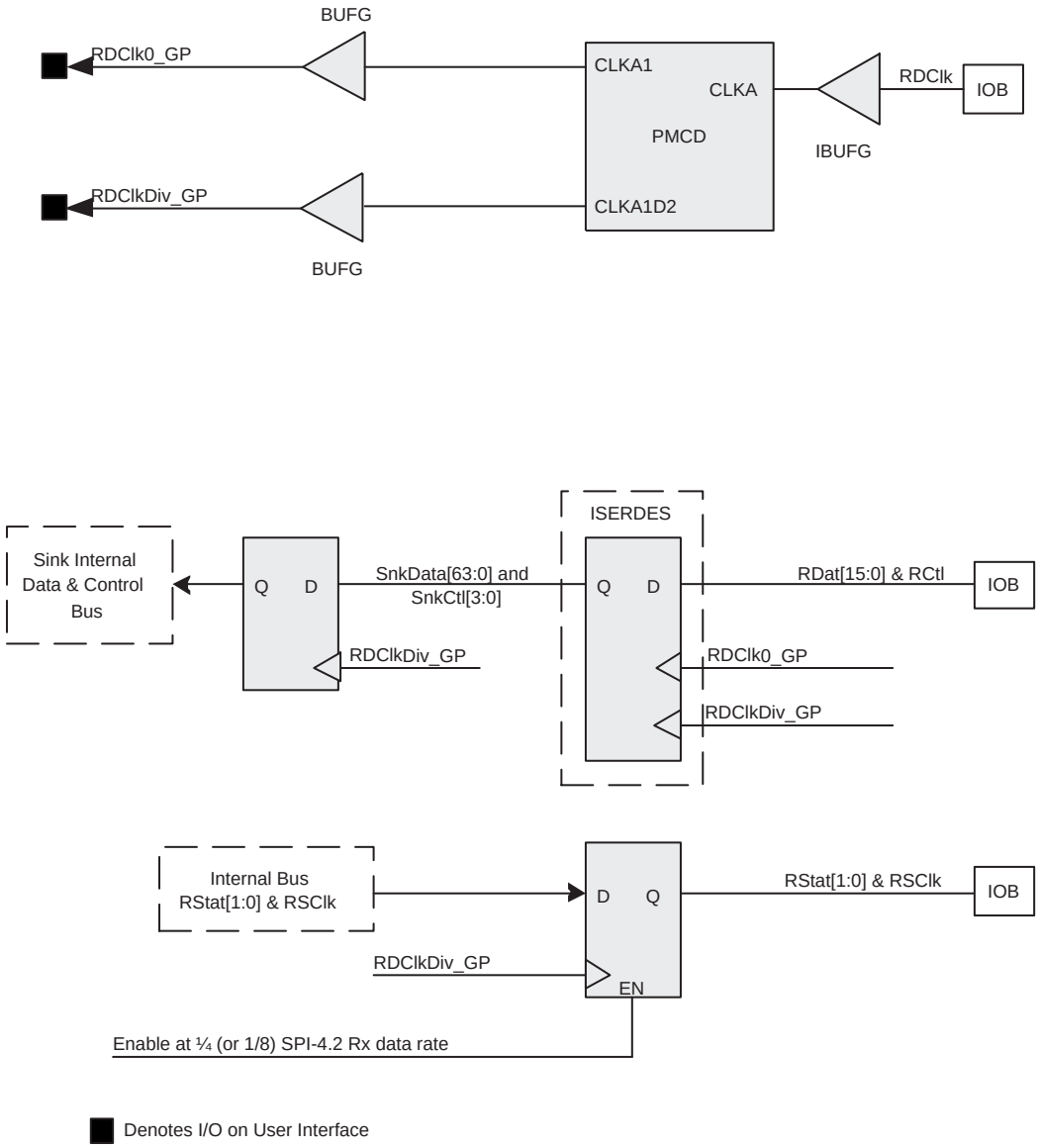


Figure 7-4: **Sink Global Clocking with PMCD (Virtex-4 FPGAs Only)**

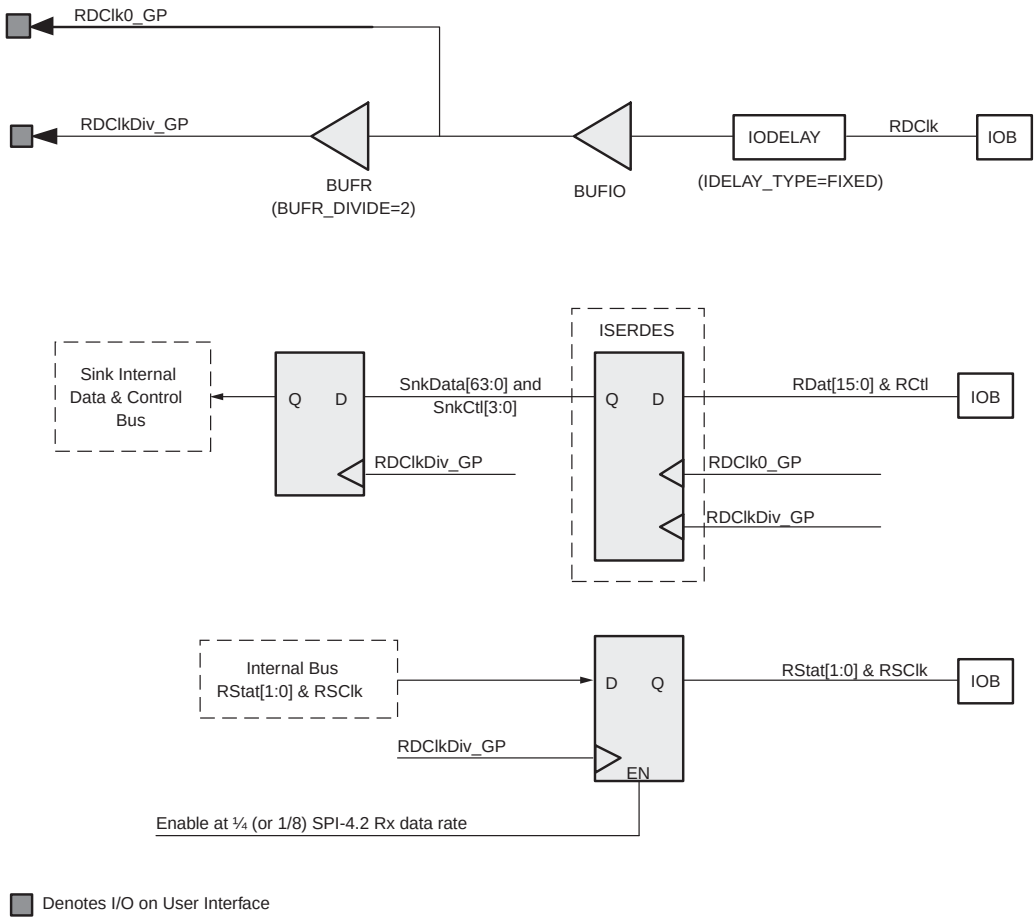


Figure 7-5: **Sink Regional Clocking for Virtex-4**

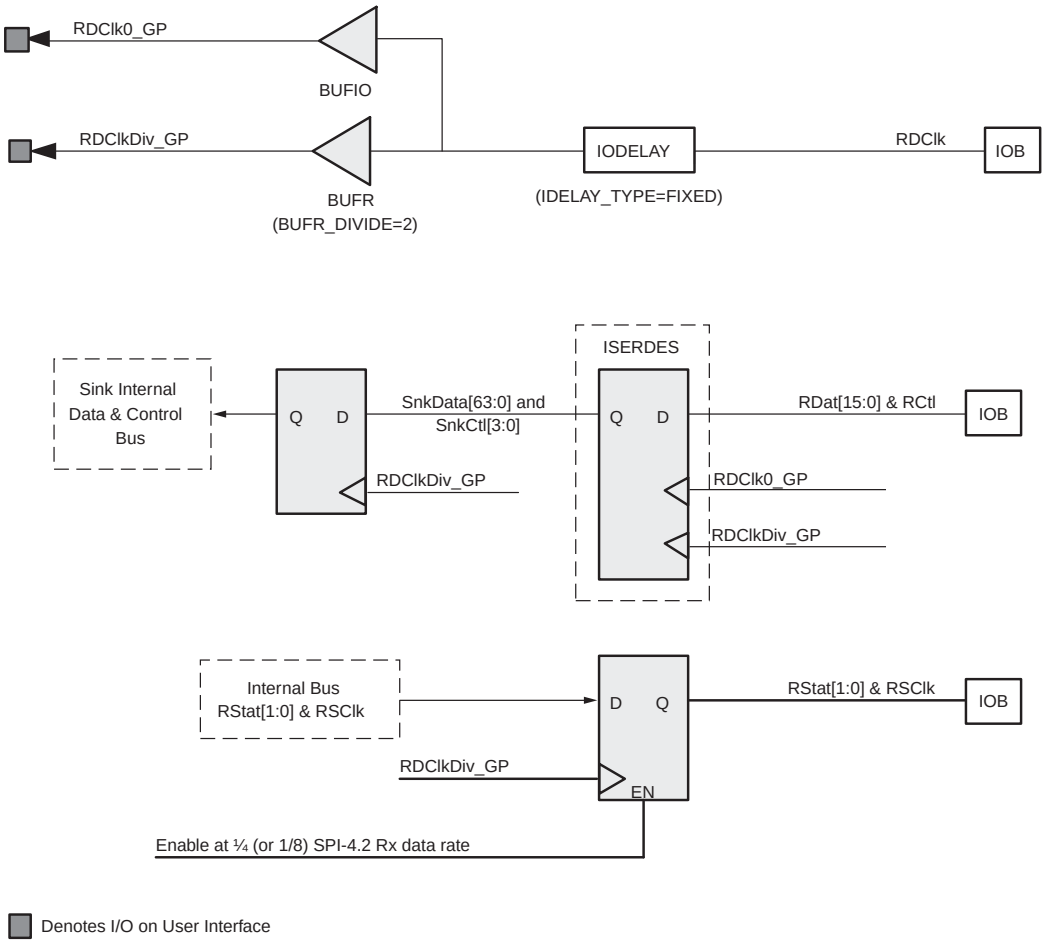


Figure 7-6: **Sink Regional Clocking for Virtex-5**

Sink Clocking Options for Virtex-6 Devices

For Virtex-6, the SPI-4.2 solutions deliver the clocking circuitry external to the Sink core. This is called user clocking mode. This mode allows a customized clocking solution to be created based on individual system requirements.

An example file is provided (`pl4_snk_clk.[v|vhd]`) to demonstrate the implementation of a clocking module for the Sink core.

There are two examples of clocking to choose from: global clocking and regional clock. Depending on the chosen clocking option, different clock resources are used. Table 7-2 provides the clocking resource count for each clocking option.

Table 7-2: **Sink Core Clocking Options Resources for Virtex-6**

Clocking Option	BUFR	BUFIO	BUFG	MMCM
Global Clocking	0	0	4	1
Regional Clocking	1	1	0	0

An illustration of the Sink user clock inputs and the sink clock configurations are shown in Figure 7-7. The inputs are defined inTable 3-9, page 43 and Table 3-10, page 43.

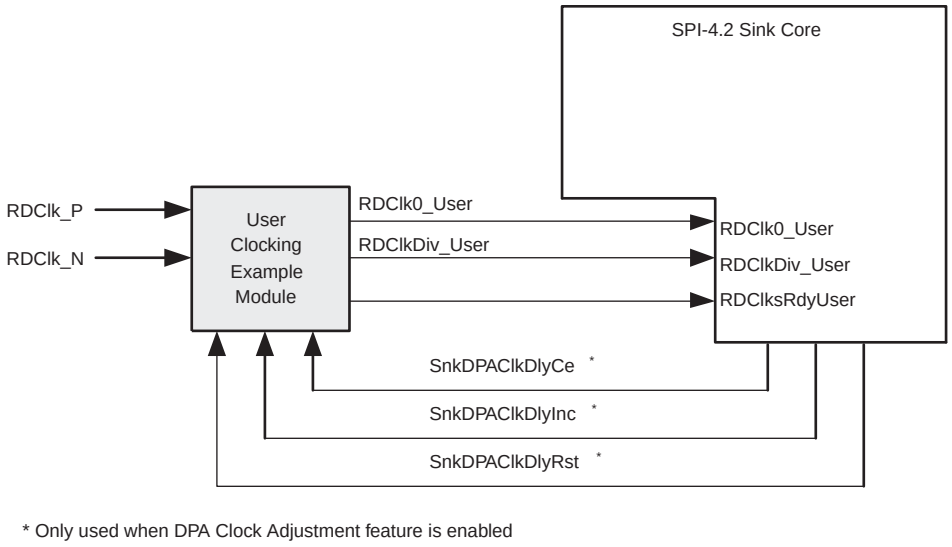


Figure 7-7: **Sink Core User Clocking Example for Virtex-6**

Global Clocking

This option uses the MMCM and global clock routing to generate and distribute a full-rate clock (RDClk0_User) and a half-rate clock (RDClkDiv_User). The global clock lines are implemented differentially in Virtex-6 devices. This configuration is illustrated in

Figure 7-8. This implementation is the same for both static and dynamic alignment configurations.

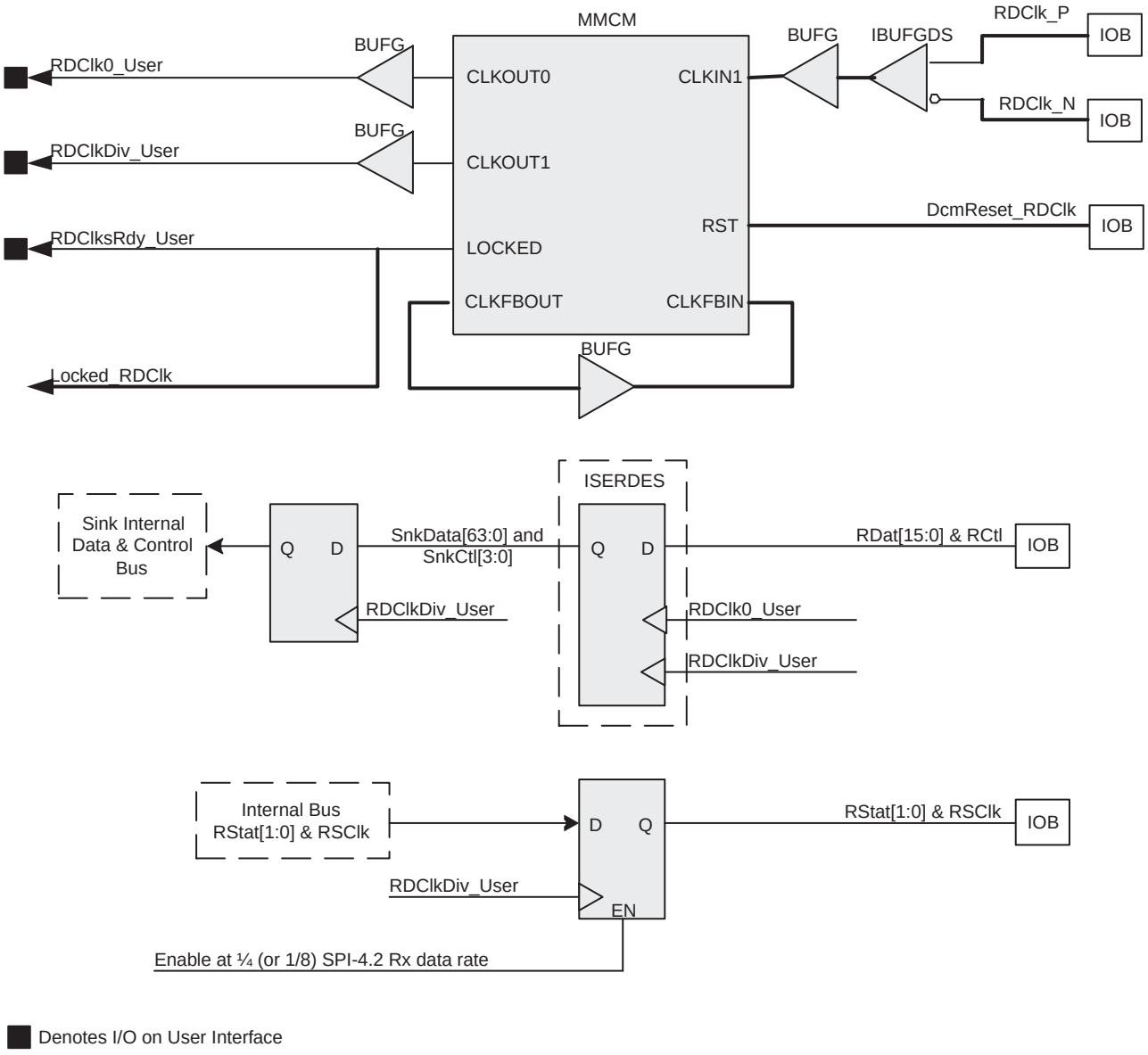


Figure 7-8: **RDClk Global clocking (Virtex-6)**

Regional Clocking

Regional clocking uses the regional clock buffer resources BUFIO and BUFR to generate the full-rate clock (RDClk0_User) and half-rate clock (RDClkDiv_User). This

configuration is illustrated in Figure 7-9. This implementation is the same for both static and dynamic alignment configurations.

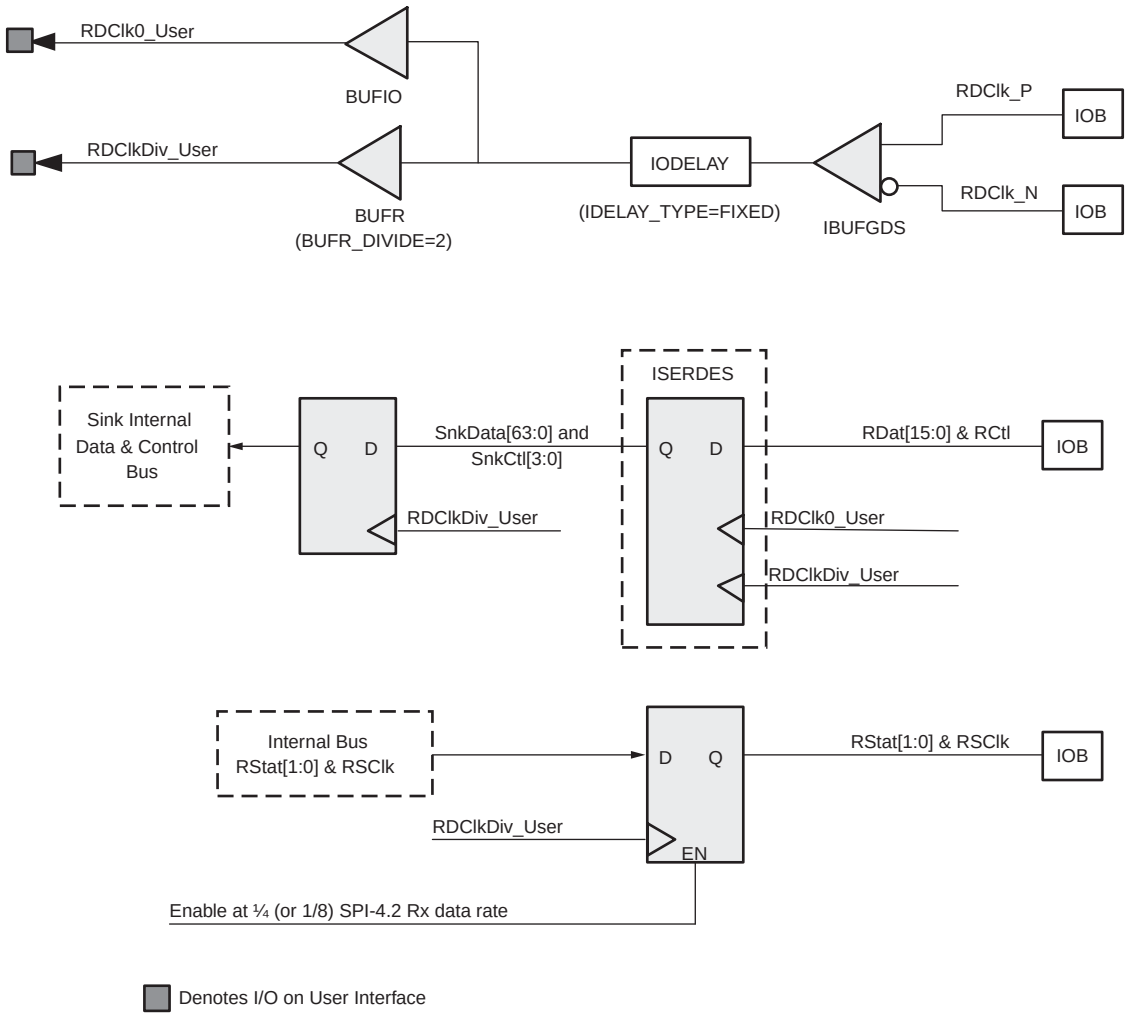


Figure 7-9: **RDClk Regional Clocking (Virtex-6)**

IDELAY on RDClk for DPA Clock Adjustment

The user also has the option to insert an IDELAY in the RDClk path to adjust the incoming RDClk. This is useful to reduce jitter introduced in the IDELAY modules used to capture the incoming RDat databus. Figure 7-10 illustrates how to insert an IDELAY in the RDClk path and connect the DPA clock delay output signals (`SnkDPAClkDlyRst`, `SnkDPAClkDlyInc`, `SnkDPAClkDlyCe`) to the IDELAY.

This feature is only available for regional clocking with dynamic phase alignment and when dynamic phase alignment clock adjustment feature enabled.

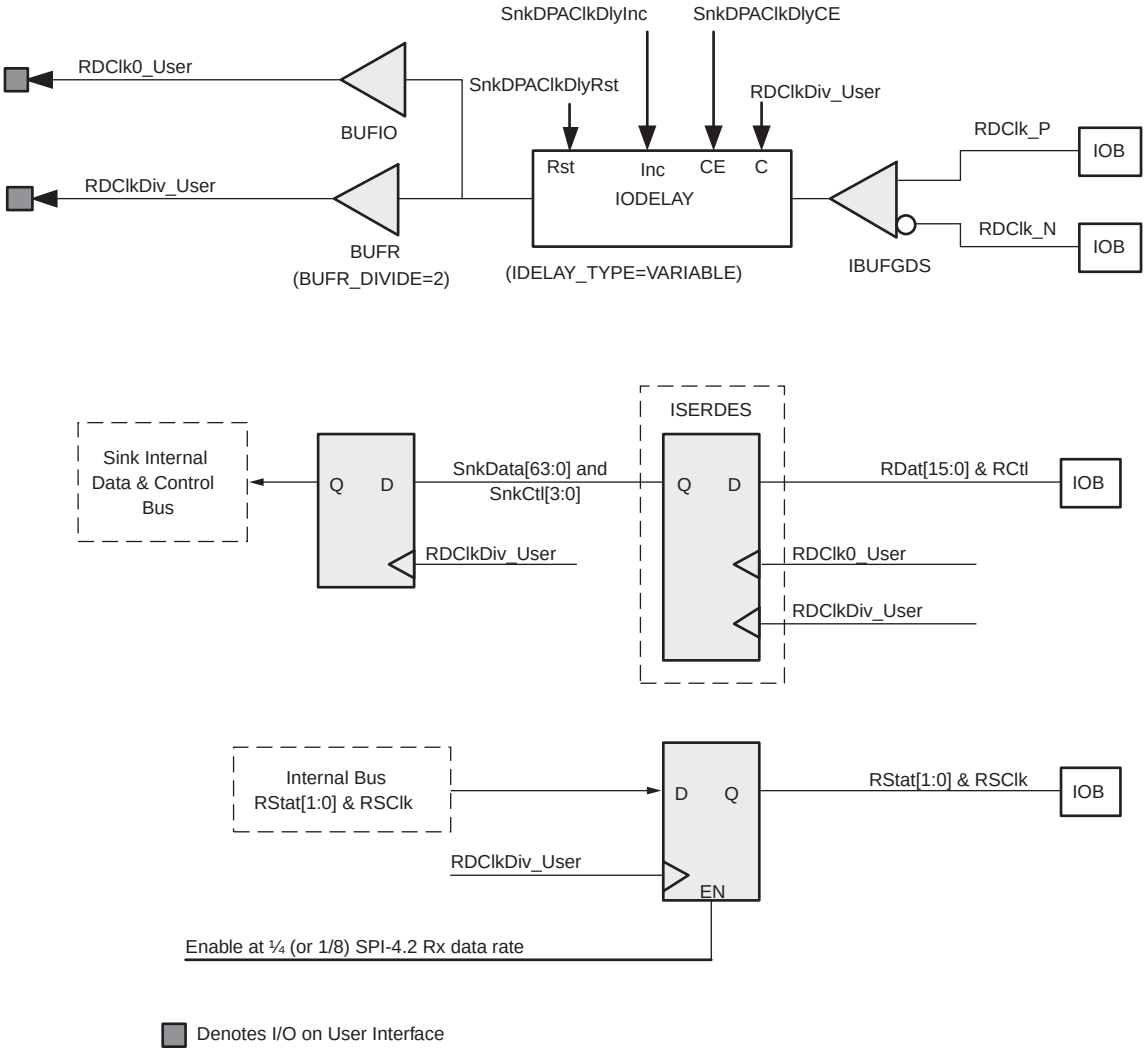


Figure 7-10: **IDELAY on RDClk for DPA Clock Adjustment (Virtex-6 FPGAs)**

Source Clocking Options for Virtex-5 and Virtex-4 Devices

The Source core supports two clocking options for Virtex-4 and Virtex-5 devices: master clocking and slave clocking. The master clocking configuration provides a complete solution with clock circuitry embedded within the Source core. Slave clocking allows the clocking scheme to be implemented externally to the Source core. This allows you to craft a custom clocking solution or to share the full-rate system clock with multiple Source cores. An example showing the Slave core sharing clock resources between Source cores and a custom clocking solution is illustrated in Figure 7-11.

The master and slave clock configurations are selected in the CORE Generator GUI and are described in detail in the following sections.

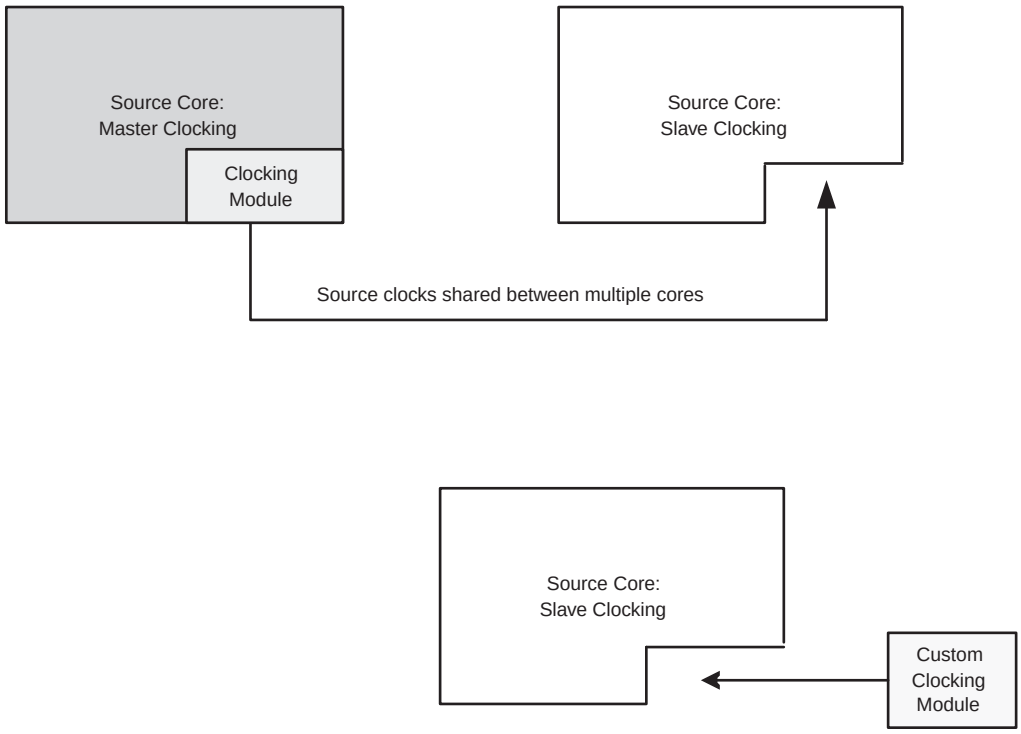


Figure 7-11: **Source Core: Slave Clocking Example**

Master Clocking

Master clocking integrates Source clocking within the Source core. This section provides information about the following configuration options for master clocking.

- Global Clock with DCM
- Global Clock with DCM Standby Logic (Virtex-4 FPGAs only)
- Global Clock with PMCD (Virtex-4 FPGAs only)
- TSClk Global Clock without DCM or PMCD
- Regional Clock

The clock implementations for `SysClk` and `TSClk` are selected in the CORE Generator GUI. Depending on the clocking option you select, different clock resources will be used. Table 7-3 and Table 7-4 provide the clocking resource count for each clocking option

Note: It is recommended that customers use regional clocking to generate SysClk when the Source core is interfacing with a Sink core configured with Dynamic Phase Alignment for a better timing budget. See Appendix H, SPI-4.2 Source Interface Timing Budget for more information.

Table 7-3: **Source Core SysClk Clocking Option Resources**

Clocking Option	BUFR	BUFG	DCM	PMCD
Global Clocking with DCM	0	2	1	0
Global Clocking with DCM Standby Logic (Virtex-4 only)	0	2	1	0

Table 7-3: **Source Core SysClk Clocking Option Resources**

Clocking Option	BUFR	BUFG	DCM	PMCD
Global Clocking with PMCD (Virtex-4 only)	0	2	0	1
Regional Clocking	1	0	0	0

Table 7-4: **Source Core TSClk Clocking Option Resources**

Clocking Option	BUFR	BUFG	DCM	PMCD
Global Clocking with DCM	0	1	1	0
Global Clocking with DCM Standby Logic (Virtex-4 only)	0	1	1	0
Global Clocking with PMCD (Virtex-4 only)	0	1	0	1
Global Clocking without DCM or PMCD	0	1	0	0
Regional Clocking	1	0	0	0

Global Clock with DCM

This option uses DCM and global clock routing to generate and distribute a full-rate clock (`SysClk0_GP`) and a half-rate clock (`SysClkDiv_GP`). This implementation is illustrated in Figure 7-12 and Figure 7-13, showing the data and status clocking. This clocking scheme minimizes jitter using DCM bypassing. Since all global clocks are implemented differentially in the Virtex-5 and Virtex-4 devices, this clocking scheme also minimizes duty-cycle distortion.

To use this implementation in a Virtex-4 design, the following DCM timing parameters must be met to ensure that the Virtex-4 DCMs meet maximum frequency specifications

under all conditions: `TCONFIG`, `DCM_INPUT_CLOCK_STOP`, and `DCM_RESET`. See the *Virtex-4 FPGA Product Specification* and Answer Record 21127 for more information.

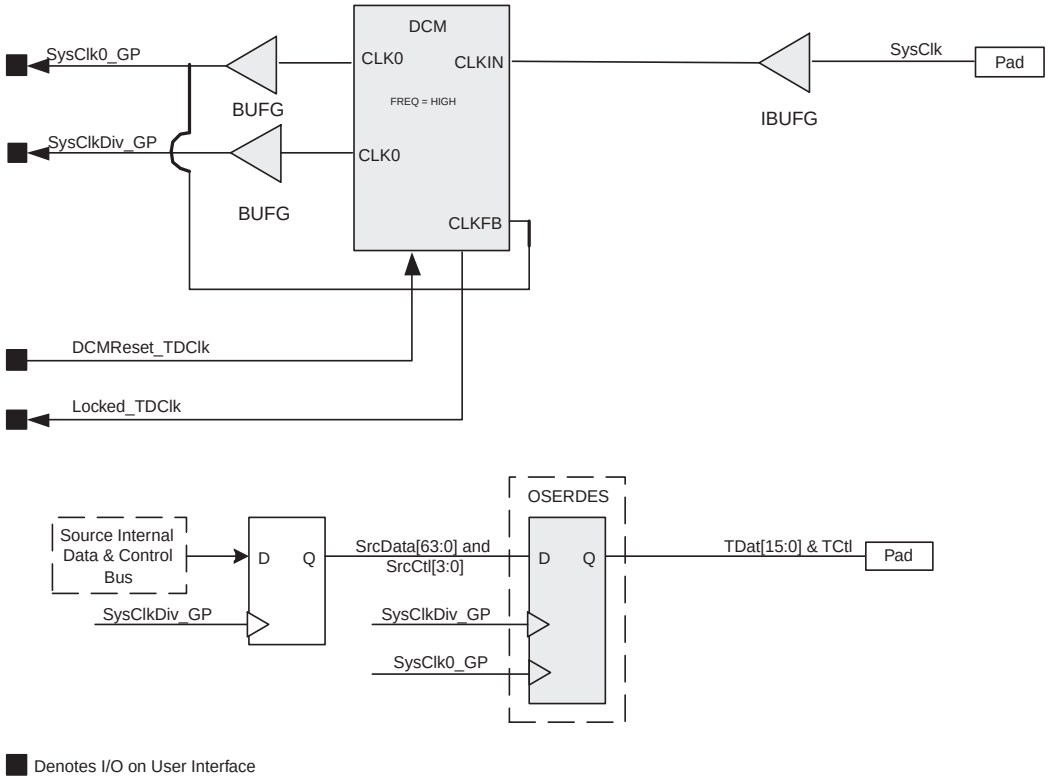


Figure 7-12: **Source Core Master Clocking: SysClk Global Clock with DCM**

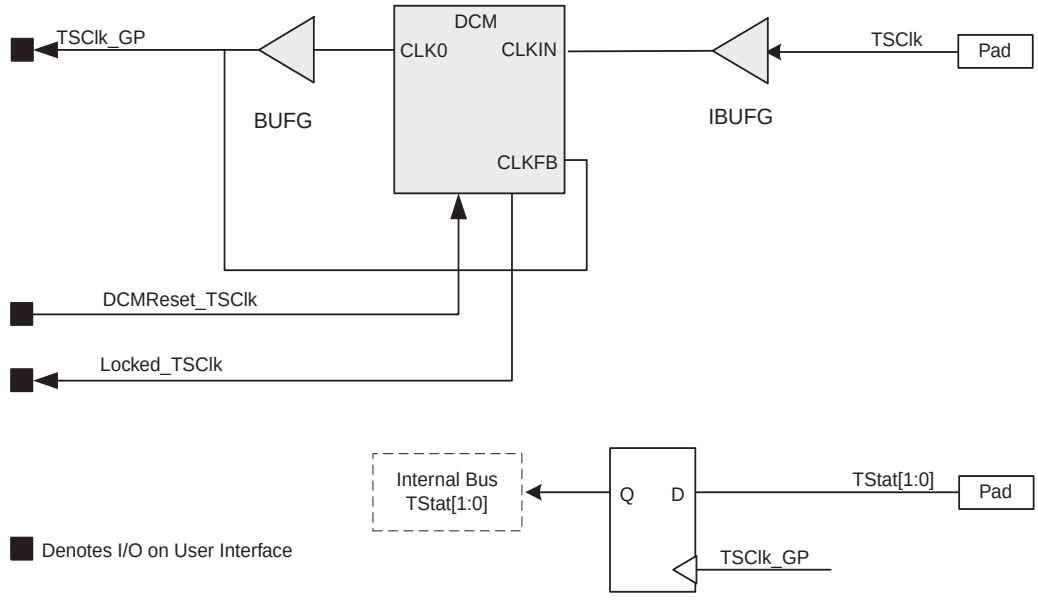


Figure 7-13: **Source Core Master Clocking: TSClk Global Clock with DCM**

Global Clock with DCM Standby Logic (Virtex-4 FPGAs Only)

This option also uses DCM and global clock routing to generate and distribute a full-rate clock (`SysClk0_GP`) and a half-rate clock (`SysClkDiv_GP`) in a Virtex-4 device. Additionally, this option supports long clock stop and reset times. These conditions occur when the DCM timing parameters, `DCM_INPUT_CLOCK_STOP` and `DCM_RESET`, are not met. An additional DCM Standby Logic is used to fulfill these requirements. This configuration is illustrated in Figure 7-14 and Figure 7-15, showing the data and status clocking. The DCM Standby Logic monitors the input clock, and the reset to the DCM. If the input clock or the feedback clock is not toggling for more than 100 ms, or if a reset is asserted, the macro keeps the DCM in continuous calibration mode. When the input clock resumes or the reset is deasserted, the macro will reset the DCM, and continue monitoring.

Note: To simulate a core with this feature, use a post-par (SIMPRIM) simulation model. Simulation with a functional (UNISIM) model that is generated with the core should not be used.

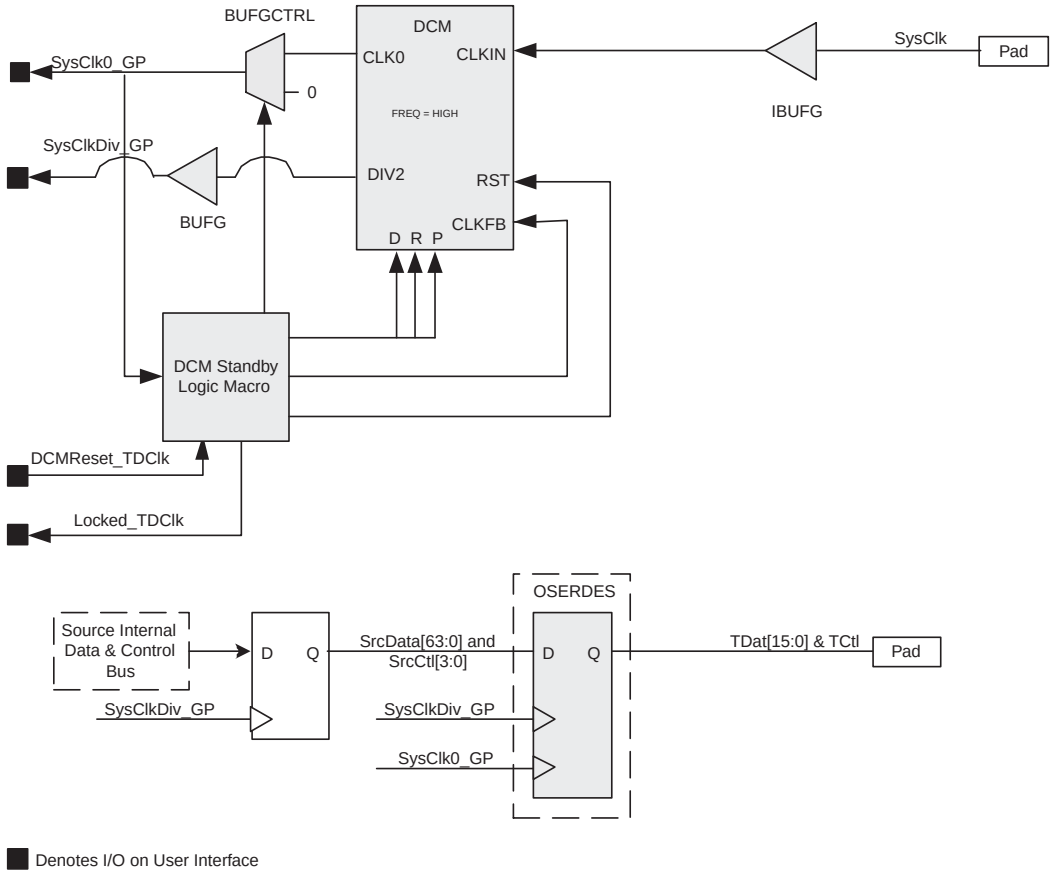


Figure 7-14: **Source Core Master Clocking: Sysclk Global Clock with DCM Standby Logic (Virtex-4 FPGAs Only)**

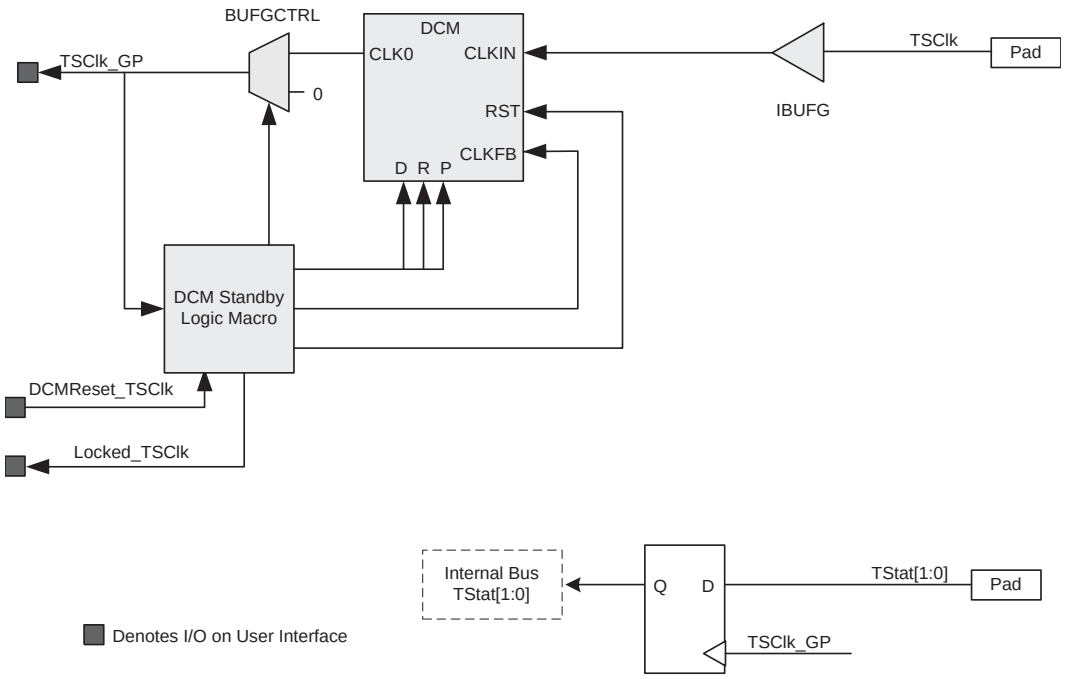


Figure 7-15: **Source Core Master Clocking: TSClk Global Clock with DCM Standby Logic (Virtex-4 FPGAs Only)**

Global Clock with PMCD (Virtex-4 Only)

This implementation is illustrated in Figure 7-16 and Figure 7-17, showing the data and status clocking in a Virtex-4 FPGA device. This clocking scheme uses the PMCD to generate the full-rate clock (`SysClk0_GP`), the half-rate clock (`SysClkDiv_GP`) and the quarter-rate clock (`TSClk_GP`). See the *Virtex-4 FPGA Data Sheet* for the maximum frequency of the PMCD.

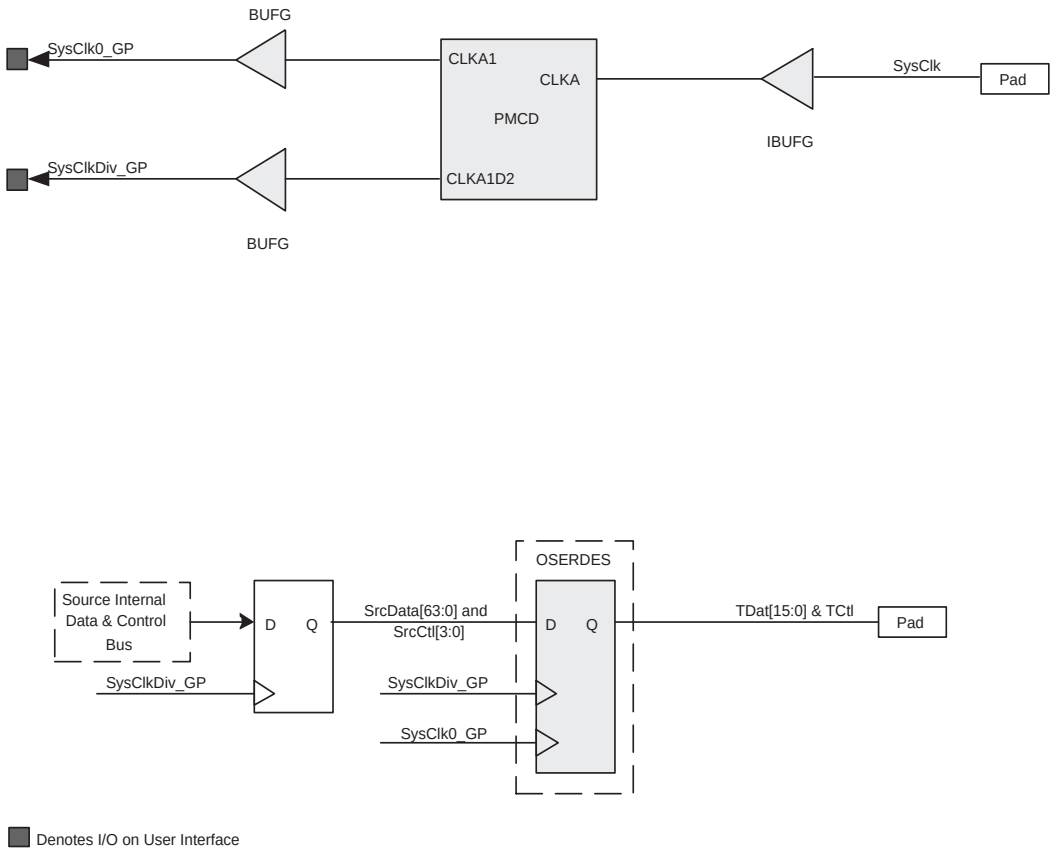


Figure 7-16: **Source Core Master Clocking: Sysclk Global Clock with PMCD (Virtex-4 FPGAs Only)**

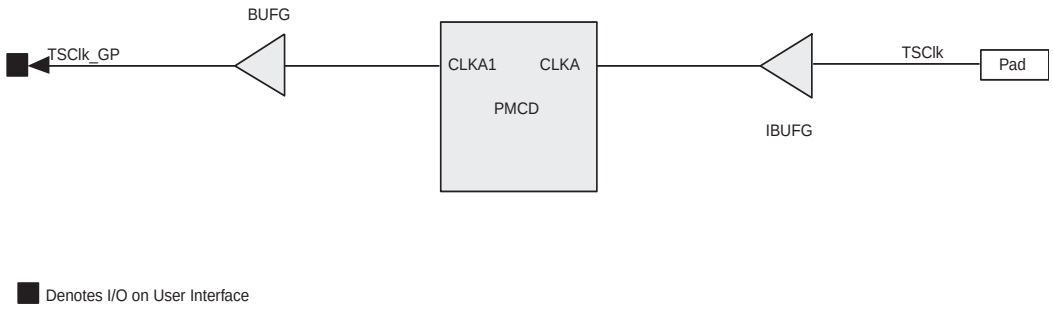


Figure 7-17: **Source Core Master Clocking: TSClk Global Clock with PMCD (Virtex-4 FPGAs Only)**

TSClk Global Clock without DCM or PMCD

This implementation for status checking is illustrated in Figure 7-18. This clocking scheme uses the global clock buffer resources BUFG to generate the quarter-rate clock (*TSClk_GP*).

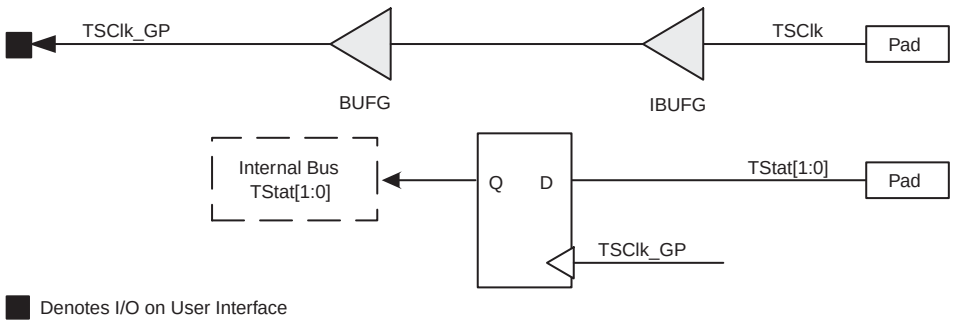


Figure 7-18: **Source Core Master Clocking: TSClk Global Clock Without DCM or PMCD**

Regional Clock

Figure 7-19 and Figure 7-20 illustrate the Regional Clock implementation for data and status clocking. This clocking scheme uses the regional clock buffer resources BUFIO and BUFR to generate the full-rate clock (*SysClk0_GP*), the half-rate clock (*SysClkDiv_GP*) and the quarter-rate clock (*TSClk_GP*).

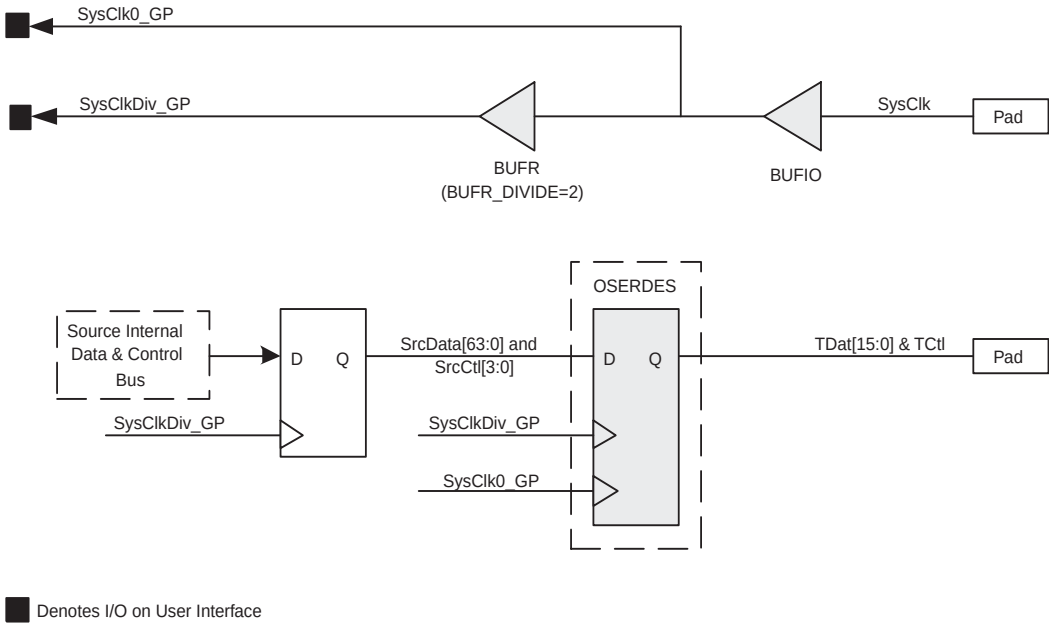


Figure 7-19: **Source Core Master Clocking: SysClk Regional Clock for Virtex-4**

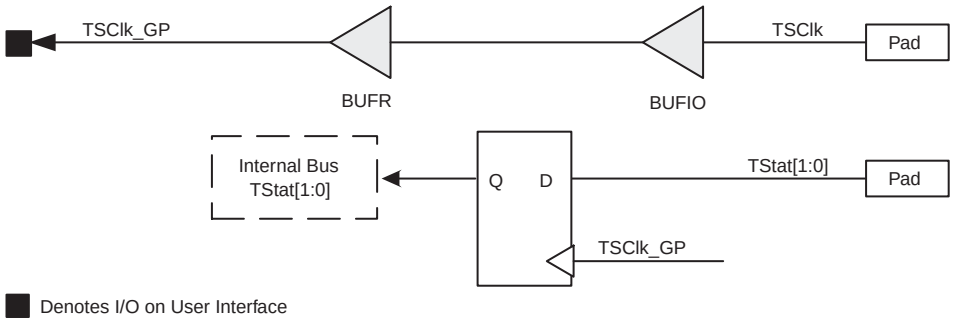


Figure 7-20: **Source Core Master Clocking: TSClk Regional Clock for Virtex-4**

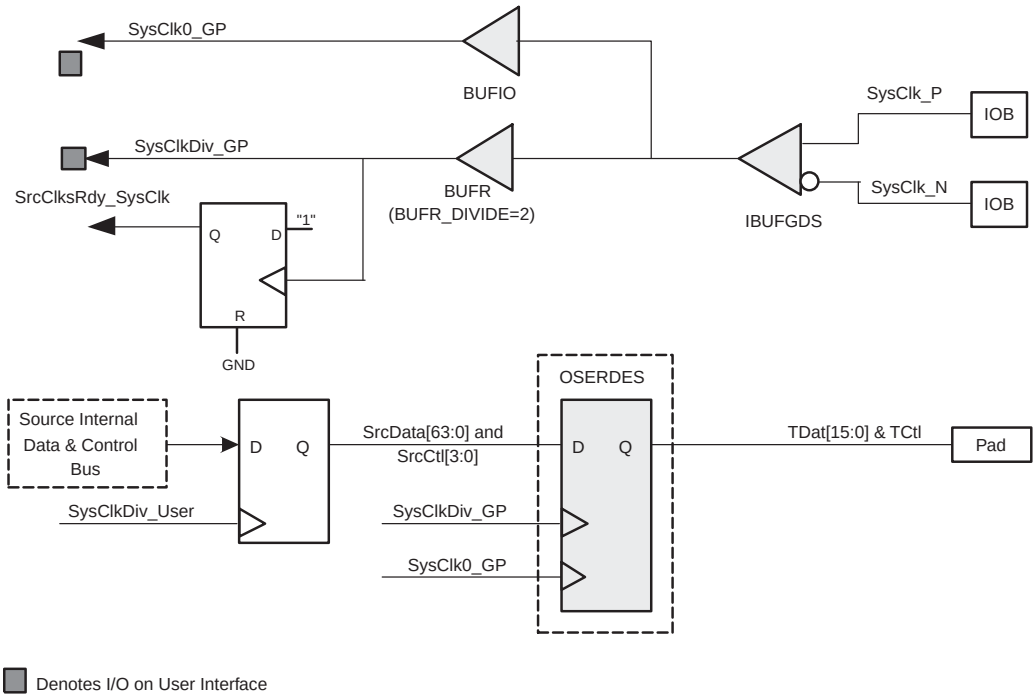


Figure 7-21: **Source Core Master Clocking: SysClk Regional Clock for Virtex-5**

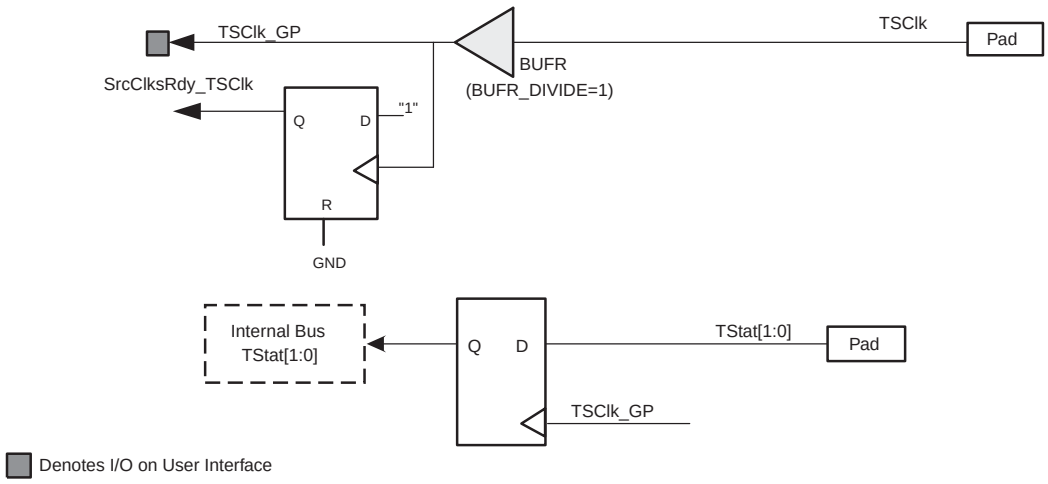


Figure 7-22: **Source Core Master Clocking: TSClk Regional Clock for Virtex-5**

Slave Clocking

The Source slave clocking configuration allows for full-customization of Source core clock implementation. If you have configured multiple SPI-4.2 interfaces in a single device, you can designate one master SPI-4.2 Core to provide clocking for all Source slave-clocking cores. You can implement clocking logic to be external to the Source cores. An example file is provided (`pl4_src_clk.v/.vhd`) to demonstrate the implementation of a clocking module for the Source core.

When using the source slave clocking configuration, you must guarantee that the *`SysClk_P(N)`* source is from a low-jitter off-scope oscillator, or other reliable clock source. The input clocks to the source slave core should not be supplied from the Sink core clocks (for example, *`RDClk_GP`*), as this will introduce significant jitter to the core, and potential functional issues.

An illustration of the Slave clock inputs and the Source slave clocking configuration are shown in Figure 7-23. The inputs are defined in Table 3-19.

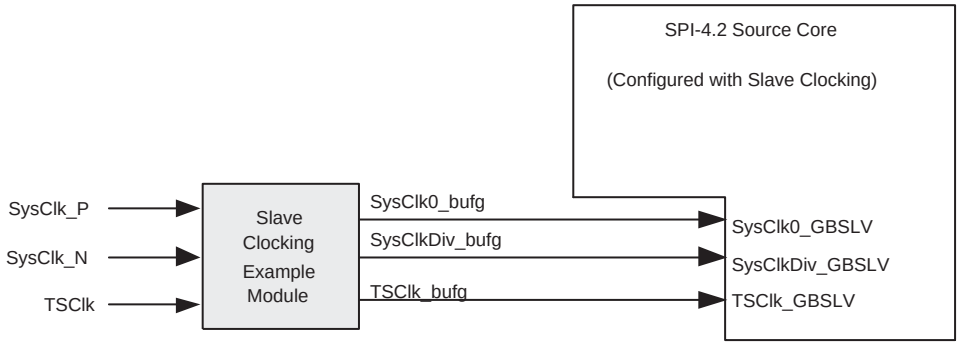


Figure 7-23: **Source Core Slave Clocking**

The clock implementation in the example file provided (`pl4_src_clk.v/.vhd`) uses DCM and global clock routing for the data clock (*`TDClk`*) and status clock (*`TSClk`*). Regional clocking can be implemented by changing the example file

(`pl4_src_clk.v/.vhd`). An illustration of the regional clocking scheme is shown in Figure 7-24.

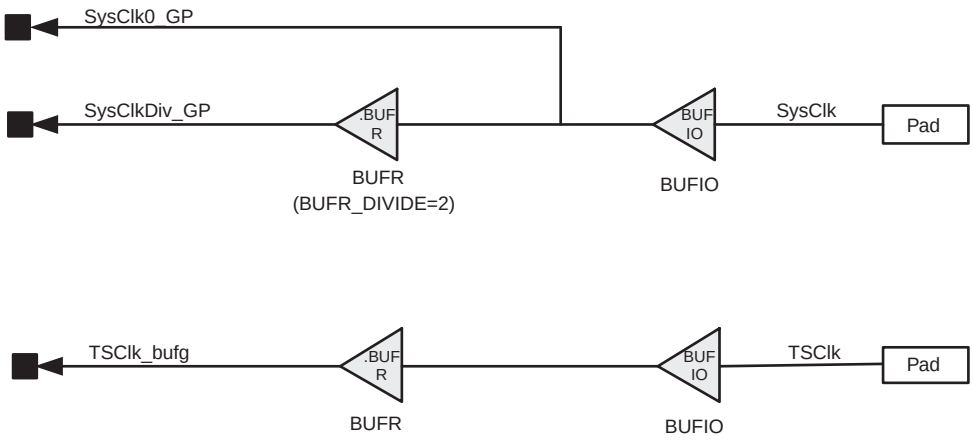


Figure 7-24: **Source Core Slave Regional Clocking**

Source Clocking Options for Virtex-6 Devices

For Virtex-6, the SPI-4.2 solutions deliver clocking circuitry external to the Source core. This is called user clocking mode. This mode allows a customized clocking solution to be created based on individual system requirements or to share the full-rate system clock with multiple source cores. An example showing sharing of clock resources between sources using a custom clocking module is illustrated in Figure 7-25.

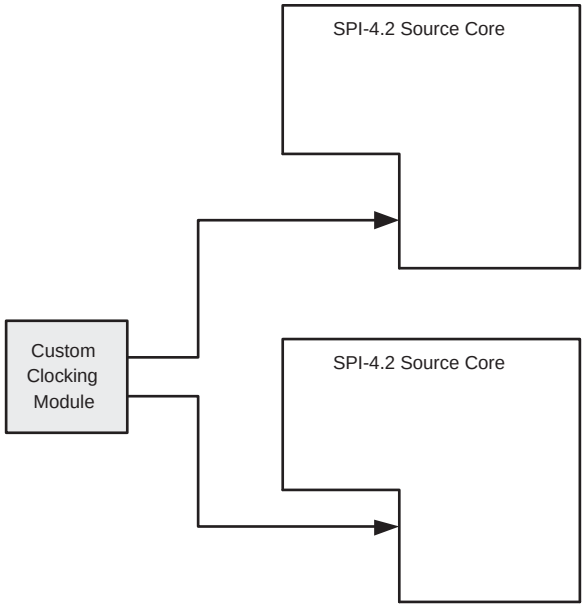


Figure 7-25: **Source Core User Clocking Example for Virtex-6 Devices**

An example file is provided (`pl4_src_clk.[v|vhd]`) to demonstrate the implementation of a clocking module for the Source core. There are two examples of clocking to choose from: global clocking and regional clock. Only regional clocking is

supported for a Source core interfacing with a Sink core configured with static alignment. Depending on the chosen clocking option, different clock resources are used. Table 7-5 and Table 7-6 provide the clocking resource count for each clocking option.

Users should utilize the LogiCORE IP Clocking Wizard to generate the optimum MMCM instantiation for their specific data rate. See the main SPI-4.2 Answer Record for guidance on how to generate MMCM instantiation for the SPI-4.2 core using the Clock Wizard IP. The Clocking Wizard also provides jitter estimates in its summary page. This data can be used to calculate the timing budget of the SPI-4.2 Source Interface. See Appendix H, SPI-4.2 Source Interface Timing Budget for more information.

Note: Use regional clocking to generate SysClk for a better timing budget. See Appendix H, SPI-4.2 Source Interface Timing Budget for more details.

Table 7-5: **Source Core SysClk Clocking Options Resources for Virtex-6 Devices**

Clocking Option	BUFR	BUFIO	BUFG	MMCM
Global Clocking[1]	0	0	4	1
Regional Clocking	1	1	0	0

Table 7-6: **Source Core TSClk Clocking Options Resources for Virtex-6 Devices[a]**

Clocking Option	BUFR	BUFIO	BUFG	MMCM
Global Clocking	0	0	3	1
Regional Clocking	1	0	0	0

a. When Sink core is configured with static alignment, only regional clocking is supported for Source core.

An illustration of the source user clock inputs and the source clock configurations are shown in Figure 7-26. The inputs are defined in Table 3-20, page 56.

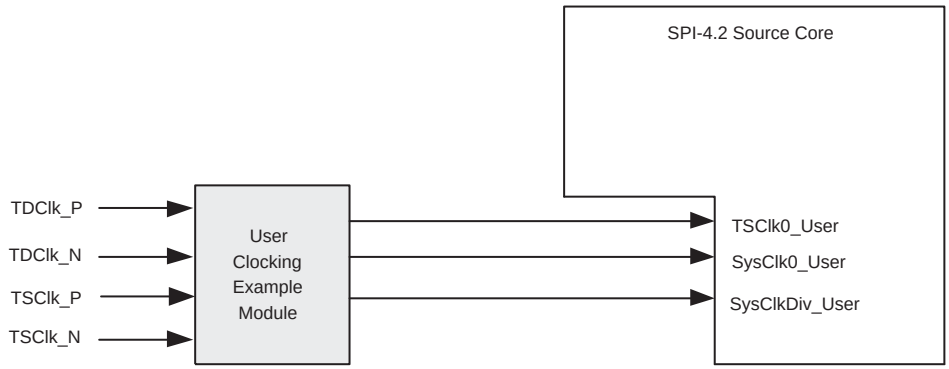


Figure 7-26: **Source Core User Clocking Ports for Virtex-6 Devices**

Global Clocking

This option uses the MMCM and global clock routing to generate and distribute a full-rate clock (SysClk0_User) and a half-rate clock (SysClkDiv_User). The global clock lines are

implemented differentially in Virtex-6 devices. This configuration is illustrated in Figure 7-27 and Figure 7-28 showing data and status clocking.

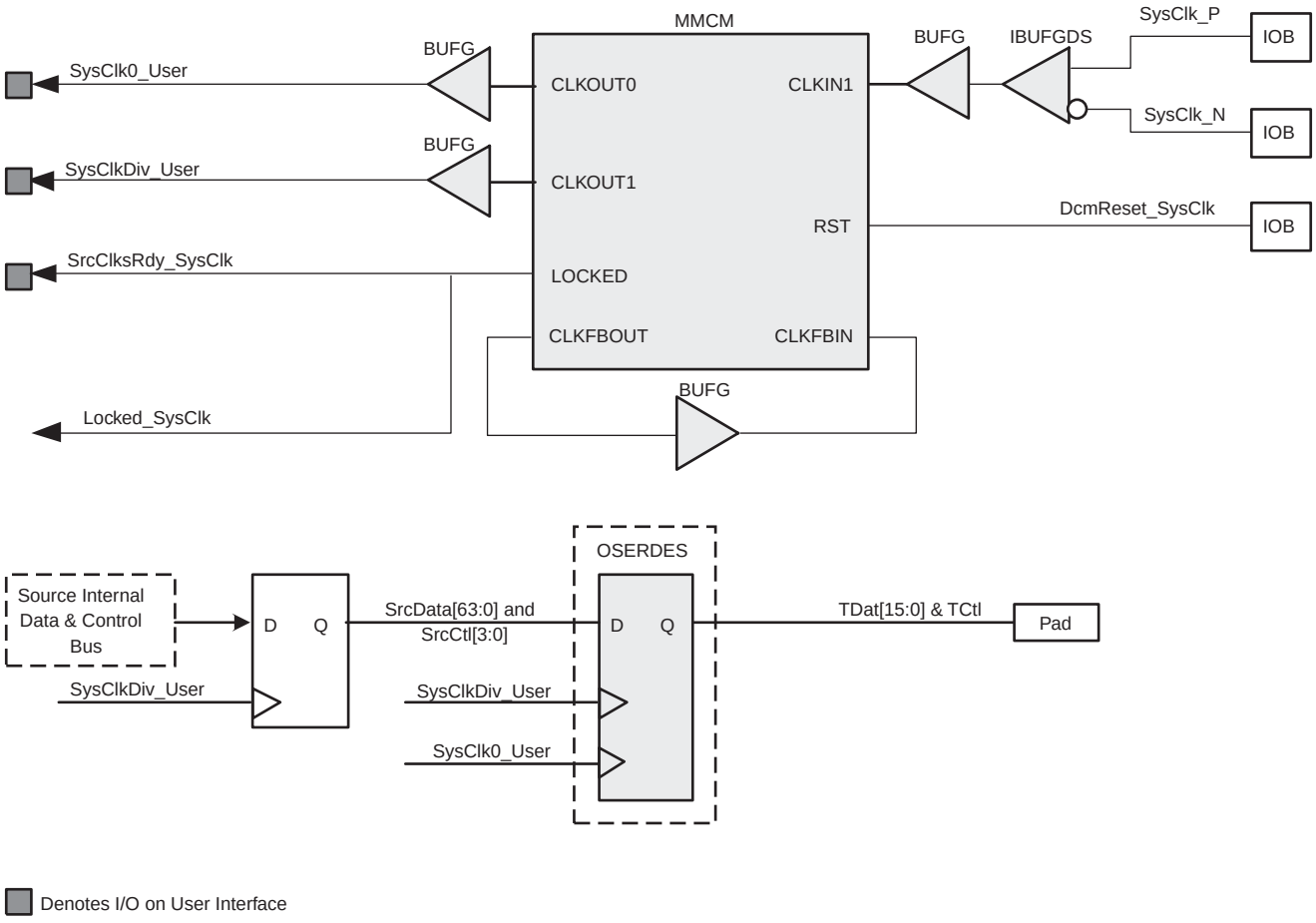


Figure 7-27: **SysClk Global Clocking (Virtex-6 Devices)**

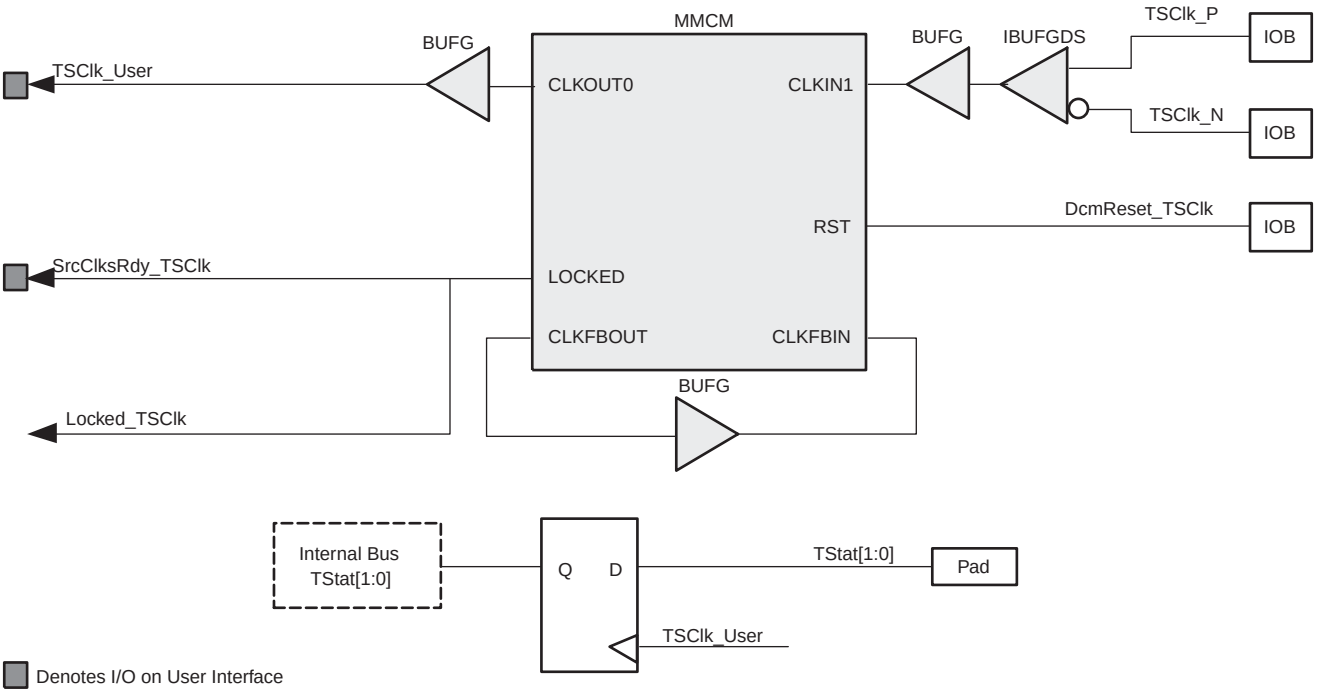


Figure 7-28: **TSClk Global Clocking (Virtex-6 Devices)**

Regional Clocking

Regional clocking uses the regional clock buffer resources BUFIO and BUFR to generate the full-rate clock (SysClk0_User) and half-rate clock (SysClkDiv_User). This configuration is illustrated in Figure 7-29 and Figure 7-30 showing data and status clocking.

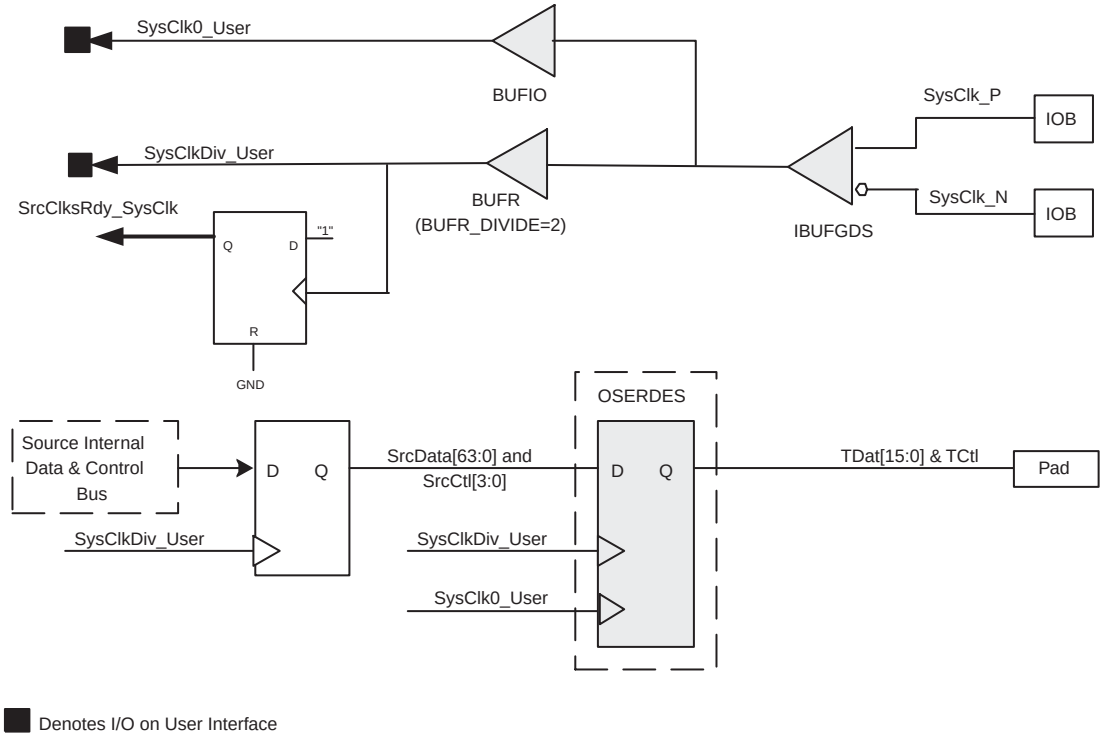


Figure 7-29: **SysClk Regional Clocking**

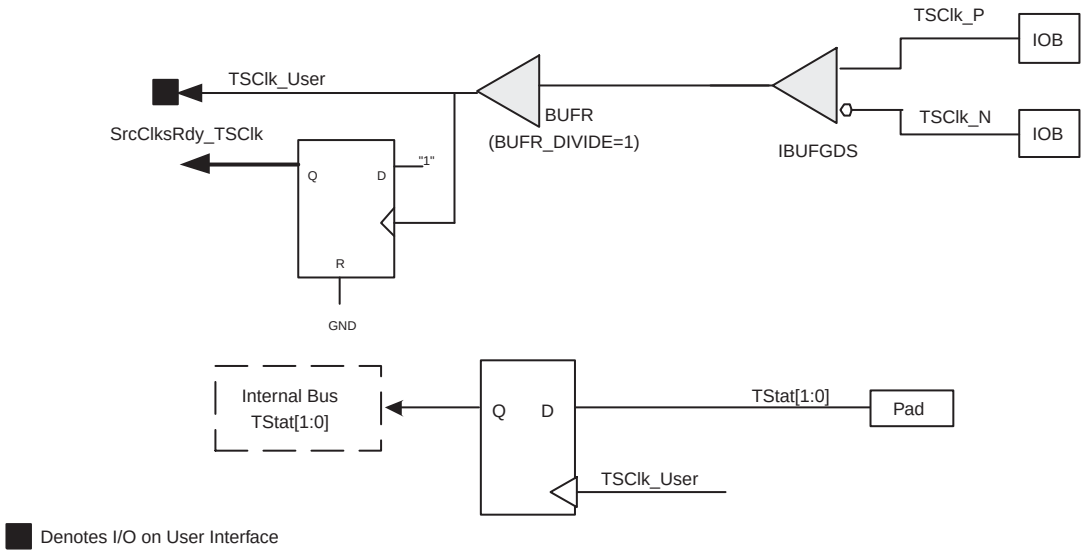


Figure 7-30: **TSClk Regional clocking (Virtex-6 Devices)**

Clocking Guidelines

There are two parameters to consider when choosing the best clocking option for your system:

- Clock frequency
- Available clock resources

Both regional clocking and global clocking with DCM or MMCM can operate up to 1 Gbps. Regional clocking requires fewer clock resources than global clocking with DCM or MMCM; however, it places an area restriction on the logic that uses the regional clock.

Global clocking with PMCD does not add to clock jitter, but has lower operating frequency than regional clocking or global clocking with DCM. For -12 and -11 speed grade, maximum operating frequency is 450 MHz. For -10 speed grade, maximum operating frequency is 400 MHz. For the maximum and minimum frequencies for different clock resources (for example, DCM, PMCD, BUFR), see the *Virtex-4 FPGA Data Sheet*.

Instantiating IDELAYCTRL Modules for Virtex-4 Devices

When the option "Include IDELAYCTRL modules" is not selected, users must instantiate the IDELAYCTRLs in the user design to calibrate the IDELAY elements connected to the Sink SPI-4.2 interface signals: `RDat[15:0]`, `RCtl`, and `RDClk`.

The IDELAYCTRL modules exist in every I/O column in every clock region. The numbers of IDELAYCTRL modules needed in the design depends on the number of banks where `RDat[15:0]`, `RCtl`, and `RDClk` signals are placed. For example, if `RDat[15:0]`, `RCtl`, and `RDClk` are placed in two banks, then two IDELAYCTRL modules are needed.

The following VHDL code segment illustrated how the IDELAYCTRL modules are instantiated and connected in the design:

```
-- Two IDELAYCTRL modules are instantiated, The signals SnkIdelayCtlRst
-- and  SnkIdelayRefClk are user-supplied signals. Ready1 and Ready2 are
-- output signals from the IDELAYCTRL modules.


   sict1 :  IDELAYCTRL
    port map (
    RST     => SnkIdelayCtlRst,
    REFCLK  => SnkIdelayRefClk,
    RDY     => Ready1
    );

   sict2 : IDELAYCTRL
    port map (
    RST     =>  SnkIdelayCtlRst,
    REFCLK  =>  SnkIdelayRefClk
    RDY     =>  Ready2
    );

-- The Ready1 and Ready2 are registered in RDClkDiv_GP clock domain.
-- The RDClkDiv_GP clock signal is the output from the Sink Core.

   rrdy1  : FDR
    port map  (
```

```
            D  =>  Ready1,
            Q  =>  Ready1_int,
            C  =>  RDClkDiv_GP,
            R  =>  SnkIdelayCtlRst
              );

   rrdy2  :  FDR
    port map (
            D   =>  Ready2,
            Q   =>  Ready2_int,
            C   =>  RDClkDiv_GP,
            R   =>  SnkIdelayCtlRst
              );

-- The Ready1_int and Ready2_int are ANDed together to create the signal
-- SnkIdelayCtlRdy which connects to the input SnkIdelayCtlRdy of the -
-- Sink Core

  register_rdy : process (SnkIdelayCtlRst, RDClkDiv_GP)  begin
      if (SnkIdelayCtlRst = '1') then
        Ready_int         <= '0' after TFF;
        SnkIdelayCtlRdy   <= '0' after TFF;
      elsif (RDClkDiv_GP'event and RDClkDiv_GP = '1') then
        Ready_int         <= Ready1_int AND Ready2_int ;
        SnkIdelayCtlRdy   <= Ready_int ;
      end if ;
  end process register_rdy ;

-- SnkIdelayCtlRdy is connected to the input of the Sink Core

    pl4_snk_top0: pl4_snk_top
    port map(
      ....................................

      RDClkDiv_GP        => RDClkDiv_GP,
      SnkIdelayCtlRdy    => SnkIdelayCtlRdy,

    ) ;
```

These IDELAYCTRL modules must be placed using placement constraints in the UCF file to cover all the clock regions where the Sink SPI-4.2 interface signals `RDat[15:0]`, `RCtl`, and `RDClk` are placed. For instance, in the example UCF file:

```
# I/O placement constraints

INST "RCtl*" LOC = "Bank11";
INST "RDat*" LOC = "Bank11";
INST "RDClk" LOC = "Bank3";

# IDELAYCTRLs placement constraints

INST "sict1" LOC = IDELAYCTRL_X0Y3 ;
INST "sict2" LOC = IDELAYCTRL_X1Y4 ;
```

Instantiating IDELAYCTRL Modules for Virtex-6 and Virtex-5 Devices

For Virtex-5 and Virtex-6 SPI-4.2 designs, the IDELAYCTRLs must be instantiated in the top-level core file to calibrate the IODELAY elements connected to the Sink SPI-4.2 interface signals: `RDat[15:0]`, `RCtl`, and `RDClk`.

The IDELAYCTRL modules exist in every I/O column in every clock region. The numbers of IDELAYCTRL modules needed in the design depends on the number of banks where `RDat[15:0]`, `RCtl`, and `RDClk` signals are placed. For example, if `RDat[15:0]`, `RCtl`, and `RDClk` are placed in two banks, then two IDELAYCTRL modules are needed. Starting with ISE v11.1, it is necessary to only instantiate one IDELAYCTRL module. In addition, the duplication and placement of the IDELAYCTRL is done automatically by the ISE software tool. For this flow to work, the IODELAY elements and the IDELAYCTRL for these elements need to be grouped via an IODELAY_GROUP constraint.

For more details about IDELAYCTRL usage and design guidelines, see the appropriate Virtex-5 or Virtex-6 FPGA User Guide.

The following VHDL code segment illustrates how the IDELAYCTRL modules are instantiated and connected in the design:

```
   ------------------------------------------------------------------
   -- IDELAYCTRL instantiation
   ------------------------------------------------------------------


    sict1 :  IDELAYCTRL
     port map (
     RST     => SnkIdelayCtlRst,
     REFCLK  => SnkIdelayRefClk_bufg,
     RDY     => Ready1
     );

   ------------------------------------------------------------------
   --- Ready signals are registered in RDClkDiv_GP clock domain.
   --- The RDClkDiv_GP is the output from the Sink Core
  --- SnkIdelayCtlRdy is connected to the input SnkIdelayCtlRdy of sink
core
   ------------------------------------------------------------------

    rrdy1  : FDR
     port map  (
           D  =>  Ready1,
           Q  =>  Ready1_int,
           C  =>  RDClkDiv_GP,
           R  =>  SnkIdelayCtlRst
             );


   register_rdy : process (SnkIdelayCtlRst, RDClkDiv_GP)
   begin
       if (SnkIdelayCtlRst = '1') then
         Ready_int         <= '0' after TFF;
         SnkIdelayCtlRdy   <= '0' after TFF;
       elsif (RDClkDiv_GP'event and RDClkDiv_GP = '1') then
         Ready_int         <= Ready1_int after TFF;
         SnkIdelayCtlRdy   <= Ready_int after TFF;
```

```
    end if ;
end process register_rdy ;

pl4_snk_top0: pl4_snk_top
port map(
  ....................................

  RDClkDiv_GP         => RDClkDiv_GP,
  SnkIdelayCtlRdy     => SnkIdelayCtlRdy,

  ....................................
);
```

The following UCF segment illustrates how to associate the IDELAYCTRL and its IODELAY elements using constraints:

```
 INST "sict1" IODELAY_GROUP = sink_idelay ;

 INST
"pl4_v9_1_pl4_snk_top0/U0/io0/dpa2/dpa_top0/DATAPAIR*/MASTER_DELAY"
IODELAY_GROUP = sink_idelay ;
 INST
"pl4_v9_1_pl4_snk_top0/U0/io0/dpa2/dpa_top0/DATAPAIR*/SLAVE_DELAY"
IODELAY_GROUP = sink_idelay ;
 INST
"pl4_v9_1_pl4_snk_top0/U0/io0/dpa2/dpa_top0/CTLPAIR*/SLAVE_DELAY"
IODELAY_GROUP = sink_idelay ;
 INST
"pl4_v9_1_pl4_snk_top0/U0/io0/dpa2/dpa_top0/CTLPAIR*/MASTER_DELAY"
IODELAY_GROUP = sink_idelay ;
```

Multiple Core Implementations

Using the Xilinx SPI-4.2 core, a designer can implement multiple SPI-4.2 cores in a single design. This section includes guidelines on how to implement multiple cores.

Instantiating Multiple Cores

With multiple cores, instantiate the modules as separate components in the top-level RTL design, as there are different netlists for each core.

For example, in VHDL:

Sink core:

first_pl4_snk_top0 : pl4_snk_top1

second_pl4_snk_top0 : pl4_snk_top2

Source core:

first_pl4_src_top0 : pl4_src_top1

second_pl4_src_top0 : pl4_src_top2

Instantiation templates for the cores are available in the CORE Generator project directory and have filename extensions of .vho (for VHDL) and .veo (for Verilog).

If the reference clock (*SysClk*) can be shared among different Source cores, generate the Source cores with slave or user clocking to reduce the number of global buffers used in the design. In addition, regional clocking for the SPI-4.2 Source FIFO Status Clocks (*TSClk*) can be implemented using regional clocking to further reduce the number of global clock buffers and DCMs or MMCMs used in the design.

If Source cores with slave or user clocking are used, the separate clocking module (*pl4_src_clk*) needs to be instantiated in the design.

An example clocking module is provided in:

`<comp_name>/example_design/`

The inputs and outputs of the example clock module for Virtex-5 and Virtex-4 devices are:

- **Inputs**: *SysClk* and *TSClk*
- **Outputs**: *Sysclk0_bufg*, *SysClkDiv_bufg*, and *TSClk_bufg*

The inputs and outputs of the example clock module for Virtex-6 are:

- **Inputs**: *SysClk_P/N* and *TSClk_P/N*
- **Outputs**: *Sysclk0_User*, *SysClkDiv_User*, and *TSClk_User*

The outputs of the clocking module, *SysClk0_bufg/user*, and *SysClkDiv_bufg/user* can be used to drive the input clocks of the multiple Source cores instantiated in the design.

For example:

```
first_pl4_src_top0 : pl4_src_top1
port map(
.................
SysClkDiv_GBSLV => SysClkDiv_bufg ,
SysClk0_GSLV => SysClk0_bufg ,
...............
) ;

second_pl4_src_top0 : pl4_src_top2
port map (
..............
SysClkDiv_GBSLV => SysClkDiv_bufg ,
SysClk0_GBSLV => SysClk0_bufg ,
.................
);
```

When instantiating the cores, there are several synthesis attributes that must be included. The cores need to be defined as black boxes for the synthesis tool, and automatic insertion of IBUF or OBUF signals for the SPI-4.2 interface signals must be disabled.

For example, in VHDL and Synplicity:

```
Attribute syn_black_box: boolean ;

Attribute black_box_pad_pin: string ;

Attribute syn_black_box of pl4_snk_top1 : component is true;

Attribute black_box_pad_pin of pl4_snk_top1: component is “RDClk_P,
RDClk_N, RDat_P(15:0), RDat_N(15:0), RCtl_P, RCtl_N” ;

Attribute syn_black_box of pl4_snk_top2 : component is true;
```

```
Attribute black_box_pad_pin of pl4_snk_top2 : component is “RDClk_P,
RDClk_N, RDat_P(15:0), RDat_N(15:0), RCtl_P, RCtl_N”;

Attribute syn_black_box of pl4_src_top1 : component is true ;

Attribute black_box_pad_pin of pl4_src_top1: component is “SysClk_P,
SysClk_N, TDClk_P, TDClk_N, TDat_P(15:0), TDat_N(15), TCtl_P, TCtl_N” ;

Attribute syn_black_box of pl4_src_top2 : component is true ;

Attribute black_blox_pad_pin of pl4_src_top2 : component is “SysClk_P,
SysClk_N, TDClk_P, TDClk_N, TDat_P(15:0), TDat_N(15:0), TCtl_P, TCtl_N”
;
```

Examples of the attributes are available in the delivered example wrapper files:

`<proj>/example_design/*.[v|vhd]`

Generating the Cores

For each core that will be instantiated, unique netlists (with unique component names) must be generated using the Xilinx CORE Generator system. Each .ngc file must also be renamed to match the component names in the top-level file.

For example:

```
pl4_snk_top1.ngc
pl4_snk_top2.ngc
pl4_src_top1.ngc
pl4_src_top2.ngc
```

Creating Top-Level User Constraints File

When instantiating multiple cores, each core is generated separately by the CORE Generator system and includes a separate top-level UCF. Merge the top-level UCF generated for each core to produce a single UCF with all required constraints.

For each core constraint, the instance name in the UCF must be modified to match the instance names in the top-level RTL design. For the timing and I/O pin location constraints, change the names to match the I/O ports declared in the top-level design as shown in the examples below.

- TNMs and TIMESPECs:

```
Net “First_RDClk_P” TNM_NET = “First_RDClk_P”;
TIMESPEC “TS_First_RDClk_P” = PERIOD “First_RDClk_P” 350 Mhz HIGH 50 %;
```

- I/O pins location:

```
INST “First_RDat*” LOC = BANK5”;
INST “First_RCtl” LOC = “BANK5”;
INST “First_RDClk” LOC = “BANK3 “;
INST “First_TDat*” LOC = “BANK9”;
INST “First_TCtl” LOC = “BANK9”;
```

See Chapter 6, Constraining the Core for details on how to place the Area Group, IO Bank, and *IDelayCtrl* components.

Clocking Considerations

If the reference clock (*SysClk*) can be shared between different Source cores, it is recommended that the Source cores with slave or user clocking be used in the design with the external clocking module (pl4_src_clk.v/vhd). In addition, the SPI-4.2 Source Status FIFO Clocks (*TSClk*) can be implemented using regional clock buffer resources to further reduce the number of global clocks and DCMs or MMCMs used in the design.

For Virtex-4 and Virtex-5 devices, the user can also use a single Source core in master clocking mode with global clock option and use the clock outputs (*SysClkDiv_GP* and *SysClk0_GP*) of this core to drive the other Source cores in slave clocking mode.

For Virtex-6 devices, since the core is delivered in user clock clocking (clock module not embedded in the core), the user can use a single Source clocking module with global clock option and use the clock outputs (*SysClkDiv_User* and *SysClk0_User*) of this module to drive the other Source cores.

Virtex-4/Virtex-5 FPGA example of source master clocking module:

```
first_pl4_src_top0 : pl4_src_top1 --- Master clocking mode
port map (
.........
SysClkDiv_GP => SysClkDiv_GP ;
SysClk0_GP => SysClk0_GP;
...........
);

second_pl4_src_top0 : pl4_src_top2 --- Slave clocking mode
port map (
............
SysClkDiv_GBSLV => SysClkDiv_GP;
SysClk0_GBSLV => SysClk0_GP;
............
);
```

Instantiating IDELAYCTRL modules (for Virtex-6 and Virtex-5)

The ISE software handles the replication and placement of the IDELAYCTRL modules automatically if the IDELAYCTRLs have the same refclk and resets. In multiple core implementations, it is typical to have the same reset and refclk signals connected to the IDELAYCTRLs. In this case, only one IDELAYCTRL is needed. The ISE software will duplicate and place the IDELAYCTRLs in the same clock regions as the associated IDELAY elements in the design. See section Instantiating IDELAYCTRL Modules for Virtex-6 and Virtex-5 Devices, page 181 for more details.

If different refclk and reset signals are needed for IDELAYCTRL modules, then each IDELAYCTRL with associated refclk and reset needs to be instantiated in the design. For more details about IDELAYCTRL usage and design guidelines, see the appropriate Virtex-5 or Virtex-6 FPGA User Guide.

XILINX®

Chapter 8

Simulating and Implementing the Core

The SPI-4.2 core is provided as a Xilinx technology-specific netlist and simulation model. The following sections describe how to simulate and implement the SPI-4.2 core in a user design.

Functional Simulation

Functional simulation of the SPI-4.2 core is performed with the provided simulation models (UNISIM models). The simulation models provide cycle-accurate simulations to use to verify your specific application. The SPI -4.2 core has been verified with Mentor Graphics ModelSim PE/SE/EE, Cadence IES, and Synopsys VCS simulators. While other simulation tools can be used to simulate the core, they have not been tested; therefore, functionality cannot be guaranteed. Before attempting functional simulation, perform the following steps to ensure that the simulator environment is properly configured.

1. Compile the Xilinx UniSim libraries (if not already compiled).

   For details, please refer to the current release of the *Synthesis and Simulation Design Guide* available from the Xilinx Documentation page.

2. Compile the simulation model, user application, and user test environment.

   An example functional simulation script is provided with the example design, which compiles the example design and demonstration test bench. You can use this script as an example for creating your test environment. For details about the functional simulation script, see Functional Simulation, page 195.

If the DCM Standby Logic macro is used, the generated simulation (functional) models will not simulate correctly. To simulate a core with this feature, use the timing (or post-par) simulation models.

Generating a Simulation Model

The functional simulation model generated by the SPI-4.2 core is created with the NETGEN tool. Following is the NETGEN command that is used to generate the simulation model for the Sink core:

```
netgen -sim -ofmt vhdl <component_name>_pl4_snk_top.ngc
<component_name>_pl4_snk_top.vhd
```

Generating a Simulation Model with Initialized Calendar

You can program the Sink and Source status calendars in the following ways:

- Using the CORE Generator GUI, initialize the content of the calendar block RAM.
- Using the appropriate calendar programming signals during system operation.

If you program the calendar during system operation, use the provided functional simulation models to verify your design. If you want to initialize the calendar by defining the initial content of the calendar BRAM, use the steps in the section below, Using Calendar Programming Signals.

Using the CORE Generator GUI

When the initial values of the calendar block RAM are defined using a COE file, the CORE Generator system converts the calendar sequence into calendar block RAM constraints in the example UCF. During implementation, the UCF calendar constraints are used to initialize the Sink and Source calendar block RAM content with the desired sequence. However, the functional simulation files must be manually updated to reflect this programming.

The following steps apply only to a Sink or Source gate-level simulation model delivered in the SPI-4.2 release (`<component_name>_pl4_snk_top.vhd or <component_name>_pl4_src_top.vhd,` or similar files). If the complete loopback design is run through NGDBuild, or the complete user design is run through NGDBuild, followed by running netgen, the gate-level netlist will contain the correctly initialized calendar sequence, and no further steps are required.

Using Calendar Programming Signals

1. Generate or modify the top-level UCFs that contain the Sink and Source calendar initialization values.

   An example of a 4-channel Sink core configuration is shown below for the SPI-4.2 core (note that unused entries can either be initialized to 0, or left unused, which will also default the values to 0):

   ```
   INST “pl4_snk_top0/pl4_snk_core0/c1.pl4_snk_cal0/mem.CalRAM/BlockRam”
   INIT_00 =
   0000000000000000000000000000000000000000000000000000000003020100;
   ```

1. Copy the UCF calendar constraints into a temporary UCF using the same name as the SPI-4.2 core edif.

   For example, if the generated Sink edif is `ch4_pl4_snk_top.ngc`, the new UCF should be named `ch4_pl4_snk_top.ucf`. The calendar initialization portion of the `pl4_wrapper.ucf` should then be copied into this new UCF, and the top-level instance name (pl4_snk_top0/ for the Sink Core, pl4_src_top0/ for the Source Core) needs to be removed. For the example above, “pl4_snk_top0/” would be removed so that the file appears as:

   ```
   INST “pl4_snk_core0/c1.pl4_snk_cal0/mem.CalRAM/BlockRam”
   INIT_00 =
   0000000000000000000000000000000000000000000000000000000003020100;
   ```

2. Ensure that the SPI-4.2 core netlist and the corresponding new UCFs are in the same directory, and then run NGDBuild:
   `> ngdbuild ch4_pl4_snk_top`
3. Generate the gate-level simulation netlist by running netgen as follows:
   `> netgen -sim -ofmt vhdl -xon false ch4_pl4_snk_top.ngd`

   The resulting gate-level simulation netlist will contain the calendar sequence load logic. Replace the gate-level netlists (created by the CORE Generator system) that are

located in the `<proj>/<component_name>/test/vhdl` directory with the output from netgen.

Shortened Alignment Simulation Model for Dynamic Phase Alignment

To generate a DPA simulation model with a shortened alignment time, set the Alignment Test Interval to a small value. However, it is not recommended that cores that are generated with a test interval value of less than 128 be used in hardware. For more information, see Dynamic Alignment Implementation Considerations, page 92 and Alignment Test Interval, page 63.

Timing Simulation

Timing simulation of the SPI-4.2 Core is performed on the post-par simulation model after the core and the design are implemented through the Xilinx tools. This simulation will provide not only a cycle-accurate simulation, but also model how the design will operate in hardware. The SPI-4.2 Core has been verified with Mentor Graphics ModelSim PE/SE/EE, Cadence IUS, and Synopsys VCS simulators. While other simulation tools can be used to simulate the core, they have not been tested and functionality cannot be guaranteed. Before attempting timing simulation, use the steps in this section to ensure that the simulator environment is properly configured.

1. Compile the Xilinx SimPrim libraries (if not already compiled).

   For details, please refer to current release of the *Synthesis and Simulation Design Guide* available from the Xilinx Documentation page.

2. Run the design through the Xilinx tool flow.

   An implement script is provided with the example design. You can use this script as an example for creating your environment.

   For details about the implementation script, see Implementing the Example Design, page 195.

3. Compile the post-par simulation model.

   An example timing simulation script is provided with the example design, and may be used as an example for creating the user test environment. For details about the timing simulation script, see Timing Simulation, page 196.

Synthesis

Synthesis of Example Design

Synthesis of the example design is supported by XST and Synplify. While other synthesis tools may be used to synthesize the example design, they have not been tested and functionality can not be guaranteed. For detailed use of the example design, see Chapter 10, Detailed Example Design.

XST

Before synthesizing with XST, be sure that the Xilinx environment is properly configured for use. A sample synthesis script is provided in the implement directory and can be used as an example for synthesizing your design.

1. Create an XST project file or open the ISE GUI.

2. Add the necessary user source files to the project file or ISE GUI.
3. If creating a project file, also add the unisim_comp.v[hd] file located in the `<Xilinx Install Path>/[vhdl | verilog]/src/ise/` directory.

   This file is included automatically when using the ISE GUI.
4. Synthesize the user application by using one of the following:
   - Using the Project Navigator ISE environment: double-click Synthesize-XST in the Processes for Source screen. Set the HDL language to VHDL or Verilog, the results directory and the part being used.
   - Using a command line mode: at the prompt, start an XST shell session by typing **`xst`** at the prompt and pressing enter. Synthesize the design by calling the XST run command to process the files in the project file.

     For more assistance in customizing the synthesis of a design, see the XST section of the Xilinx development tools manual, at www.xilinx.com/documentation.

Synplify

Before synthesizing with Synplify, ensure that the Synplify environment is properly configured. A sample synthesis script is provided in the implement directory and can be used as an example for synthesizing your design.

1. Create a Synplify project file.
2. Add the necessary user source files to the project file.
3. Select target device and speed grade.
4. Synthesize the user application.

Xilinx Tool Flow

This section provides an overview of the Xilinx tool flow and discusses how to implement the SPI-4.2 Core and the design with the Xilinx implementation tools. Detailed information about Xilinx tools can be found in the *Xilinx Development System Reference Guide*.

Before executing the Xilinx tool flow, generate a design netlist where the SPI-4.2 Core is instantiated. All required constraints must be set in the UCF. See Chapter 6, Constraining the Core for more information.

Example Design Script

An implementation script is provided with the example design to execute all the commands described below. This script can be used as an example of how to run the Xilinx tools with the SPI-4.2 core. For details about the example design, see Chapter 10, Detailed Example Design.

NGDBuild

Run ngdbuild to translate and merge the various source files of a design into a single NGD design database.

Ngdbuild command example

ngdbuild `<component_name>_top`

The output of ngdbuild will be `component_name_top.ngd`.

Mapping the Design

To map the logic gates of the user's design (previously written to an NGD file by ngdbuild) into the CLBs and IOBs of the physical device, the map command must be executed. The map command writes out this physical design to an NCD file.

Map command example

map -o mapped.ncd component_name_top.ngd

The map command outputs a `mapped.ncd` and `mapped.pcf`.

Place and Route

The par command (par) must be executed to place and route design logic components (mapped physical logic cells) contained within a NCD file. These are based on the layout and timing requirements specified within the physical constraints file (PCF).

par command example

par mapped.ncd routed.ncd

The par command outputs `routed.ncd` file that contains the placed and routed design.

Static Timing Analysis

To evaluate timing closure on a design and create a timing report file (TWR) derived from static timing analysis of the physical design file (NCD), the `trce` command must be executed. The analysis is typically based on constraints included in the optional physical constraints file (PCF).

trce command example

trce -e 10 routed.ncd mapped.pcf -o routed.twr

The `trce` command outputs a routed.twr file that performs a timing analysis of the placed and routed design, based on user constraints.

Timing Simulation

After your design is functionally correct and meets all timing constraints, Xilinx recommends that you perform a back-annotated timing simulation to verify that the entire design functions correctly before you test the design with hardware. The `netgen` command generates a post-par simulation model, which includes all timing information.

netgen command example

```
netgen -sim -ofmt <vhdl | verilog> routed.ncd
```

The netgen command outputs routed.v[hd] and routed.sdf files, which allow the user to run timing simulation.

Generating a Bitstream

To create the configuration (BIT) file based on the contents of a physical implementation file (NCD), execute the bitgen command. The BIT file defines the behavior of the programmed FPGA.

bitgen command example

```
bitgen -w routed routed mapped.pcf
```

Note: Use caution when setting required bitgen options, including selecting the startup clock. See the *Development System Reference Guide* for details.

XILINX®

Chapter 9

Quick Start Example Design

The quick start steps provide information to quickly generate a SPI-4.2 core, run the design through implementation with the Xilinx tools, and simulate the example design using the provided demonstration test bench. For more detailed information about this example design, see Chapter 10, Detailed Example Design.

Overview

The SPI-4.2 example design consists of the following:

- SPI-4.2 Sink and Source core netlists
- SPI-4.2 Sink and Source core simulation models
- Example HDL wrapper (which instantiates the cores and example design)
- Customizable demonstration test bench to simulate the example design

Generating the Core

To generate a SPI-4.2 core with default values using the Xilinx CORE Generator system, do the following:

1. Start the CORE Generator system.

   For help starting and using the CORE Generator system, see the *Xilinx CORE Generator Guide*, available from the ISE documentation.
2. Choose File > New Project.
3. Type a directory name. For this example design, use the directory name *design*.
4. Set the following project options:
   - Part Options
     - From Target Architecture, select either a Virtex®-6, Virtex-5, or Virtex-4 device.

     Note: If an unsupported silicon family is selected, the SPI-4.2 core will not appear in the taxonomy tree.

     Note: The Device, Package and Speed Grade selected in the Part Options tab have no effect on the generated core. The core is delivered with an example UCF targeting either the Virtex-6 6VLX75T-FF784, Virtex-6 -1L 6VLX75TL-FF784, Virtex-5 5V1X50-FF676, or Virtex-4 4VLX25-FF668 device.
   - Generation Options
     - For Design Entry, select either VHDL or Verilog.
     - For Vendor, select Synplicity or Other (for XST).

5. After creating the project, locate the directory containing the SPI-4.2 core in the taxonomy tree; it appears under Communications & Networking > Telecommunications > SPI-4.2.
6. Double-click the core to bring up the customization GUI.
7. In the Component Name field, enter a name for the core instance. (In this example, the name *quickstart* is used.)
8. After selecting the desired features and parameters from the GUI screens, click Generate.

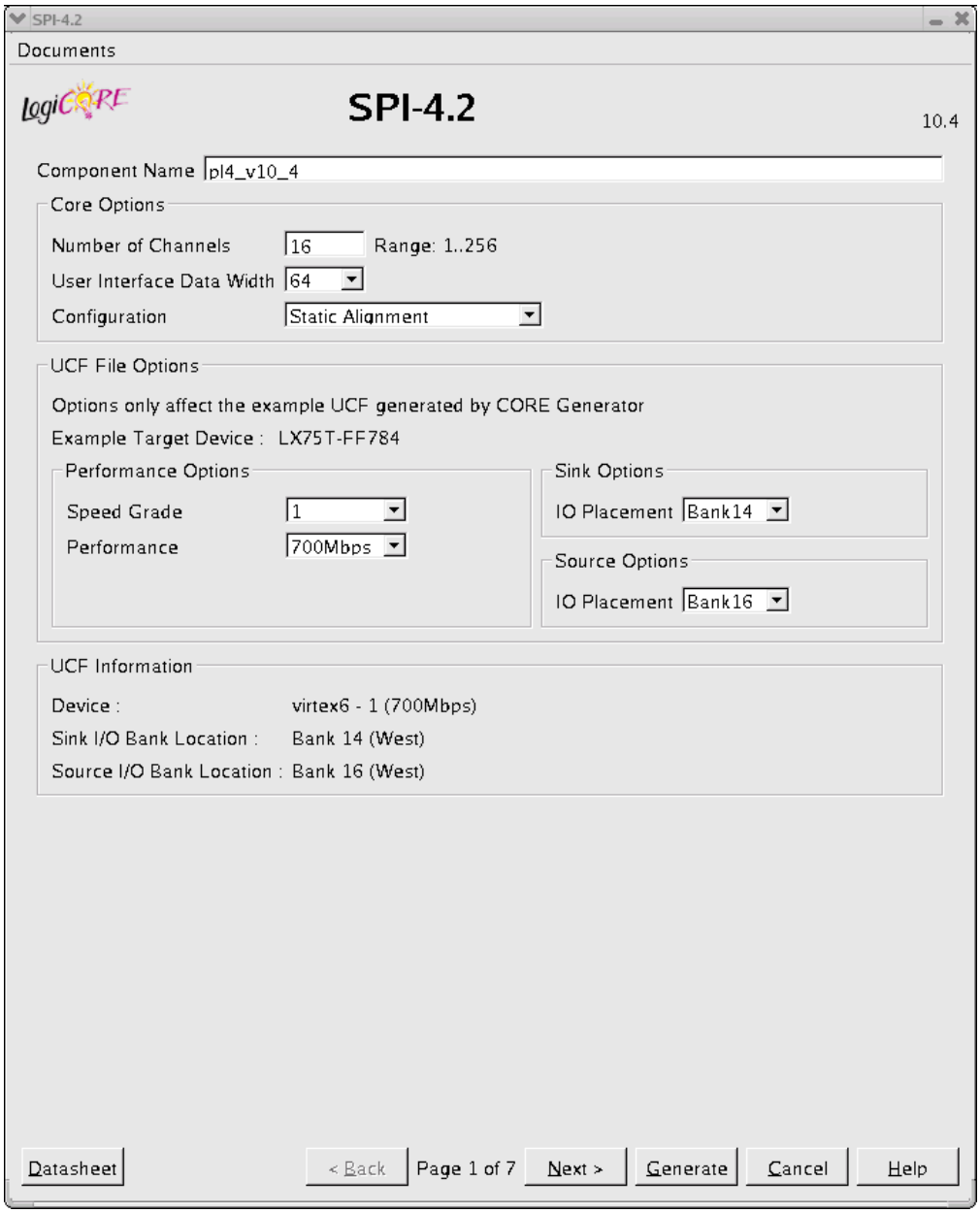


Figure 9-1: **Core Customization GUI Main Window**

The cores and supporting files, including the example design, are generated in the project directory. For detailed information and an illustration of the example design files and directories produced, see Directory and File Contents in Chapter 4.

Implementing the Example Design

After generating a core with a Full System Hardware Evaluation or Full license, the netlists and the example design can be processed by the Xilinx implementation tools. The generated output files include scripts to assist you in running the Xilinx tools.

To implement the SPI-4.2 example design, open a command prompt or terminal window and type the following commands:

For Windows

```
ms-dos> cd <proj>\<quickstart>\implement
ms-dos> implement.bat
```

For Linux

```
% cd <proj>/<quickstart>/implement
% ./implement.sh
```

These commands execute a script that synthesizes, builds, maps, and place-and-routes the example design. The script then generates a post-par simulation model for use in timing simulation. The resulting files are placed in the results directory.

Running the Simulation

Using the provided example design, you can quickly simulate and observe the behavior of the SPI-4.2 core. There are two different simulation types, functional and timing. The simulation models provided are either in VHDL or Verilog, depending on the CORE Generator Design Entry project option selected by the user.

Setting up for Simulation

The Xilinx UniSim and SimPrim libraries must be mapped into the simulator. If the UniSim or SimPrim libraries are not set for the test environment, please refer to the Synthesis and Simulation Design Guide, available from the ISE documentation.

Functional Simulation

Instructions for running a functional simulation of the SPI-4.2 core using either VHDL or Verilog are given below. Functional simulation models are provided when the core is generated. Note that implementing the core before simulating the functional models is not required. If a configuration file (referenced in the CORE Generator GUI as the COE file) was used to program the calendar, special steps are required to include the calendar sequence in the simulation. See Chapter 5, Designing with the Core for details on including the calendar initialization values in simulation.

To run a VHDL or Verilog functional simulation of the example design using ModelSim:

1. Set the current directory to:

   <quickstart>/simulation/functional/
2. Launch the ModelSim simulator.
3. Launch the simulation script:

   ```
   modelsim> do simulate_mti.do
   ```

To run a VHDL or Verilog functional simulation of the example design using NCSIM:

1. Set the current directory to:

<*quickstart*>/simulation/functional/

2. Execute the simulation script:

```
% simulate_ncsim.sh
ms-dos> simulate_ncsim.bat
```

To run a Verilog functional simulation of the example design using VCS:

1. Set the current directory to:

   <*quickstart*>/simulation/functional/

2. Execute the simulation script:

```
% simulate_vcs.sh
```

The simulation script compiles the functional simulation models, the loopback and the demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

Timing Simulation

Timing simulation is available only with purchase of the core (Full license) or with access to the Full System Hardware Evaluation license. With a Simulation Only Evaluation license the core cannot be run through the implementation tools, which is required for timing based simulation.

Instructions for running a timing simulation of the SPI-4.2 core using either VHDL or Verilog are given below. A timing simulation model is generated when the core is run through the Xilinx tools using the implement script. Calendar information specified in a COE file is included in the timing simulation netlist.

To run a VHDL or Verilog timing simulation of the example design using ModelSim:

1. Set the current directory to:

   <*quickstart*>/simulation/timing/

2. Launch the ModelSim simulator.
3. Launch the simulation script:

```
modelsim> do simulate_mti.do
```

To run a VHDL or Verilog timing simulation of the example design using NCSIM:

1. Set the current directory to:

   <quickstart>/simulation/timing/

2. Execute the simulation script:

```
ms-dos> simulate_ncsim.bat
% simulate_ncsim.sh
```

To run a Verilog timing simulation of the example design using VCS:

1. Set the current directory to:

   <*quickstart*>/simulation/timing/

2. Execute the simulation script:

```
% simulate_vcs.sh
```

The simulation script compiles the timing simulation model and the demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

Chapter 10

Detailed Example Design

This chapter provides detailed information about the example design, including a description of files and the directory structure generated by the Xilinx CORE Generator, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

- **<project directory>**
  Top-level project directory; name is user-defined
  - <project directory>/<component name>
    Core release notes file
    - <component name>/doc
      Product documentation
    - <component name>/example design
      Verilog and VHDL design files
    - <component name>/implement
      Implementation script files
      - implement/results
        Results directory created after implementation scripts are run; contains implement script results.
    - <component name>/simulation
      Simulation scripts
      - simulation/functional
        Functional simulation files
      - simulation/timing
        Timing simulation files

Directory and File Contents

The SPI-4.2 core directories and their associated files are defined in the following sections.

<project directory>

The project directory contains all the CORE Generator project files.

Table 10-1: **Project Directory**

Name	Description
`<project_dir>`	
`<component_name>_pl4_snk_top.ngc` `<component_name>_pl4_src_top.ngc`	Top-level netlists.
`<component_name>_pl4_snk_top.v[hd]` `<component_name>_pl4_src_top.v[hd]`	Verilog and VHDL simulation models.
`<component_name>.xco`	CORE Generator project-specific option file; can be used as an input to the CORE Generator.
`<component_name>_flist.txt`	List of files delivered with the core.
`<component_name>_pl4_snk_top.{vho|veo}` `<component_name>_pl4_src_top.{vho|veo}`	VHDL and Verilog instantiation templates.

Back to Top

<project directory>/<component name>

The <component name> directory contains the release notes file provided with the core, which may include last-minute changes and updates.

Table 10-2: **Component Name Directory**

Name	Description
<project_dir>/<component_name>	
`spi4_2_readme.txt`	Core release notes text file.

Back to Top

<component name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 10-3: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
`spi4_2_ds209.pdf`	SPI-4.2 Data Sheet
`spi4_2_ug153.pdf`	SPI-4.2 User Guide

Back to Top

<component name>/example design

The example design directory contains the example design files provided with the core.

Table 10-4: **Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
`<component_name>_top.ucf`	User constraints file (UCF) provides example constraints necessary for processing the core using Xilinx implementation tools. This file can be modified to meet individual system requirements. The example UCF contains timing and placement constraints for both Sink and Source cores.
`<component_name>_top.v[hd]`	VHDL or Verilog wrapper file for the example design; it instantiates the Sink and Source cores and the loopback module. This is the top-level synthesis file for the example design.
`pl4_fifo_loopback.v[hd]`	Top-level loopback file used in the example design; it instantiates the loopback read and write modules.
`pl4_fifo_loopback_read.v[hd]`	Loopback read module used in the example design; it interfaces to the SPI-4.2 Sink core.
`pl4_fifo_loopback_write.v[hd]`	Loopback write module used in the example design; it interfaces to the SPI-4.2 Source core.
`pl4_src_clk.v[hd]`	Example clocking module used in the example design when the Source core is configured for slave clocking (Virtex-4 or Virtex-5 devices) or user clocking (Virtex-6 devices).
`pl4_snk_clk.v[hd]`	Example clocking module used in the example design when the Sink core is configured for user clocking (Virtex-6 devices only).
`virtex4.v`	Module instantiation for Virtex-4 primitives.
`virtex5.v`	Module instantiation for Virtex-5 primitives.

Back to Top

<component name>/implement

The implement directory contains the core implementation script files.

Table 10-5: **Implement Directory**

Name	Description
`<project_dir>/<component_name>/implement`	
`implement.{sh|bat}`	Windows (.bat) or Linux (.sh) script that processes the example design through the Xilinx tool flow.
`xst.prj`	XST project file for the example design; it lists all of the source files to be synthesized. It is only available when the CORE Generator vendor project option is set to "Other."
`xst.scr`	XST script file for the example design that is used to synthesize the core, and it is called from implement.{sh|bat}. It is only available when the CORE Generator vendor project option is set to "Other."
`synplify.prj`	Synplicity project file for the example design; it lists all of the source files to be synthesized. It is only available when the CORE Generator vendor project option is set to "Synplicity."

Back to Top

implement/results

The results directory is created by the implement script, after which the implement script results are placed in the results directory.

Table 10-6: **Results Directory**

Name	Description
<project_dir>/<component_name>/implement/results	
Implement script result files.	

Back to Top

<component name>/simulation

The simulation directory contains the necessary files to test a VHDL or Verilog example design with the demonstration test bench.

Table 10-7: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
data_file.dat	Data file containing the data to be sent across the SPI-4.2 Interface
pl4_clk_gen.v[hd]	Demo Test bench Clock Generator
pl4_data_monitor.v[hd]	Demo Test bench Data Monitor
pl4_demo_testbench.v[hd]	Demo Test bench Top Level Module
pl4_procedures.v[hd]	Demo Test bench Procedures Module
pl4_startup.v[hd]	Demo Test bench DCM/MMCM Startup and Calendar Loader Module
pl4_status_monitor.v[hd]	Demo Test bench Status Monitor
pl4_stimulus.v[hd]	Demo Test bench Data and Status Stimulus Module
pl4_testcase.v[hd] pl4_testcase_pkg.v[hd]	Controls the operation of the demonstration test bench and can be user-modified.
snk_calendar.dat	Data file containing the calendar information for the Sink interface
src_calendar.dat	Data file containing the calendar information for the Source interface
[glbl.v]	Asserts initial global reset pulse (Verilog only)

Back to Top

simulation/functional

The functional directory contains functional simulation scripts provided with the core.

Table 10-8: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
`simulate_mti.do`	ModelSim macro file that compiles the functional netlist, loopback HDL, and demo HDL source. The script also loads and runs the simulation for 8 µs.
`wave_mti.do`	ModelSim macro file that opens a wave window and adds key signals to the wave viewer. The wave_mti.do file is called by the simulate_mti.do macro file.
`simulate_ncsim.sh` `simulate_ncsim.bat`	Shell scripts that compile the functional netlist and loopback HDL source. The script also launches NCSIM and runs the simulation for 8 µs.
`wave_ncsim.sv`	NCSIM macro file that opens a wave window and adds key signals to the wave viewer. The wave_ncsim.sv file is called by the simulate_ncsim.sh or simulate_ncsim.bat file.
`simulate_vcs.sh (verilog only)`	Shell script that compiles the functional netlist and example design. The script also runs the functional simulation using VCS.
`vcs_session.tcl (verilog only)`	VCS tcl script that opens a wave window. This macro is called by the simulate_vcs.sh script.
`ucli_commands.key (verilog only)`	VCS command file. This file is called by the simulate_vcs.sh script.

Back to Top

simulation/timing

The timing directory contains timing simulation scripts provided with the core.

Table 10-9: **Timing Directory**

Name	Description
<project_dir>/<component_name>/simulation/timing	
`simulate_mti.do`	ModelSim macro file that compiles the post-par timing netlist and demo HDL source. The script also loads and runs the simulation for 12 µs. The implement script must first be run to generate the post-par timing simulation model. Simulation can only be run after the timing simulation model is generated.

Table 10-9: **Timing Directory *(Cont'd)***

Name	Description
`wave_mti.do`	ModelSim macro file that opens a wave window and adds key signals to the wave viewer. The wave_mti.do file is called by the simulate_mti.do macro file.
`simulate_ncsim.sh` `simulate_ncsim.bat`	Shell scripts that compile the post-par timing netlist and loopback HDL source. The script also launches NCSIM and runs the simulation for 12 µs.
`wave_ncsim.sv`	A NCSIM macro file that opens a wave window and adds key signals to the wave viewer. The wave_ncsim.sv file is called by the simulate_ncsim.sh or simulate_ncsim.bat file.
`simulate_vcs.sh (verilog only)`	Shell script that compiles the post-par timing netlist and example design. The script also runs the timing simulation using VCS.
`vcs_session.tcl (verilog only)`	VCS tcl script that opens a wave window. This macro is called by the simulate_vcs.sh script.
`ucli_commands.key (verilog only)`	VCS command file. This file is called by the simulate_vcs.sh script.

Back to Top

Implementation and Simulation Scripts

The implementation script is either a shell script or a batch file that runs the example design through the Xilinx tool flow. The scripts are located in the following directory:

```
<proj_dir>/<component_name>/implement/
```

The implementation scripts are parameterized based on the Design Entry Tool and Design Entry Language CORE Generator project options. If either of these project options are changed, the core must be regenerated to create the appropriate implementation scripts.

If the core was generated with the Full System Hardware Evaluation or the Full license, the implementation script is present and performs the following steps:

1. Synthesizes the example design using the selected synthesis tool (XST or Synplify).
2. Runs `ngdbuild` to consolidate the core netlists, wrapper netlist, and constraints file into the common database.
3. Runs `map` to perform technology specific mapping of the design.
4. Runs `par` to perform place and route of the design.
5. Runs `trce` to perform static timing analysis of the routed design.
6. Runs `bitgen` to generate a bitstream for download to the target FPGA.
7. Runs `netgen` to generate a post-par simulation model for use in timing simulation.

Simulation Script Details

The simulation scripts for ModelSim, NCSIM, and VCS that simulate the demonstration test bench are located in one of the following directories:

```
<proj_dir>/<component_name>/simulation/{functional | timing }/
```

For functional simulation, the simulation script performs the following tasks:

1. Compiles the simulation models provided with the core.
2. Compiles the loopback example design.
3. Compiles the wrapper file, which instantiates the cores and the loopback.
4. Compiles the demonstration test bench.
5. Starts a simulation of the demonstration test bench.
6. Opens the waveform viewer and adds key signals (`wave_mti.do|wave_ncsim.sv|vcs_session.tcl`).
7. Runs the simulation.

For timing simulation, the simulation script performs the following tasks:

1. Compiles the post-par design example, which includes the cores and the loopback.
2. Compiles the demonstration test bench.
3. Starts a simulation of the demonstration test bench.
4. Opens the waveform viewer and adds key signals (`wave_mti.do|wave_ncsim.sv|vcs_session.tcl`).
5. Runs the simulation.

Example Design Configuration

In the example design, a Loopback Module is connected to the user interface of the SPI-4.2 core. Typically, the user interface would be connected directly to the design. The SPI-4.2 Interface, which is the interface defined by the *OIF-SPI4-02.1* specification, typically

connects to a SPI-4.2 PHY layer device or network processor. Figure 10-1 shows the example design modules architecture and interfaces to the SPI-4.2 core.

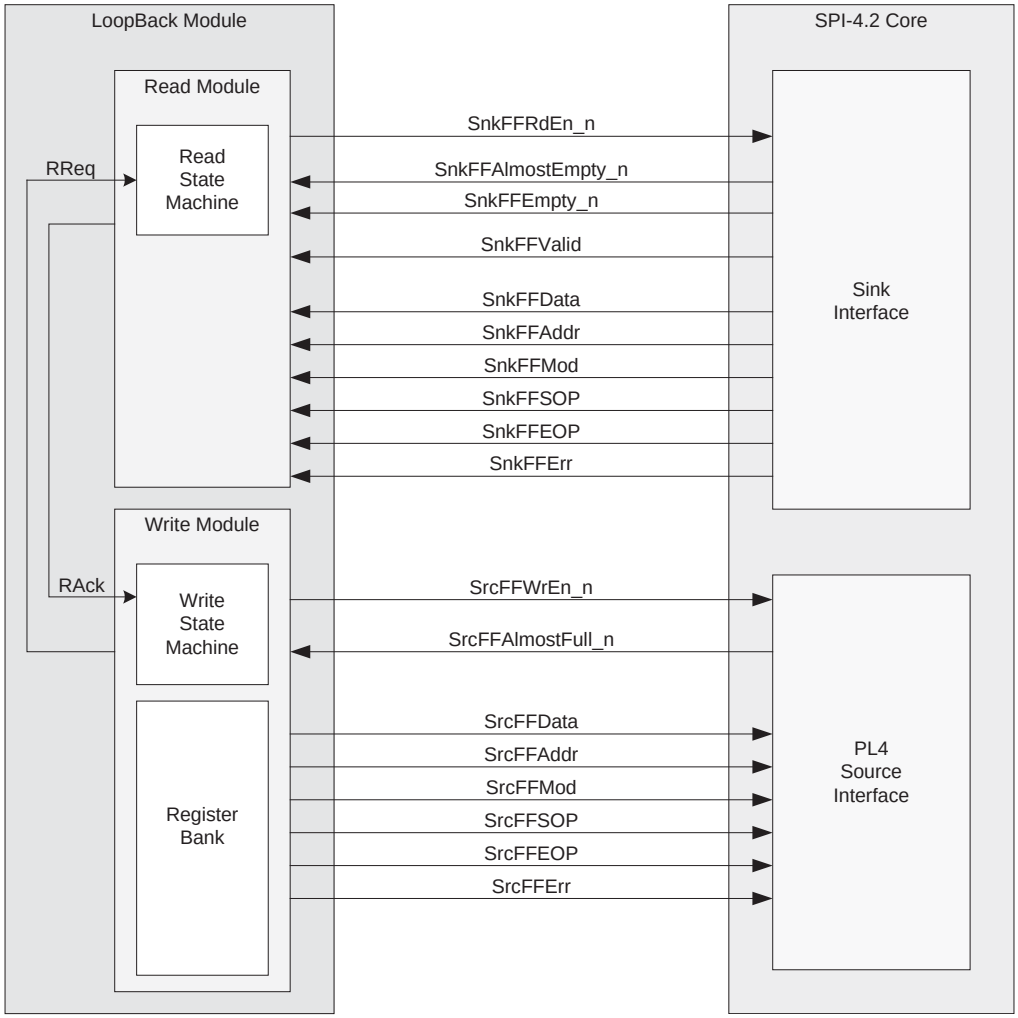


Figure 10-1: **Example Design Configuration**

Loopback Module

The Loopback Module connects to the user interface of the SPI-4.2 Sink and Source cores. There is a Read Module that accesses packet data from the Sink FIFO and a Write Module that transfers data into the Source FIFO. The Read Module polls the status signals `SnkFFEmpty_n` and `SnkFFAlmostEmpty_n` to determine whether it can perform a read from the Sink FIFO. The Write Module polls `SrcFFAlmostFull_n` to determine whether it can transfer data into the Source FIFO.

Basic Loopback Operation

When the Almost Full flag (`SrcFFAlmostFull_n`) is deasserted, the Write Module asserts a read request (`RReq`) that is sent to the Read Module. When a read request is received, the Read Module verifies that the FIFO is not empty and initiates a read from the Sink FIFO. On the next cycle, the data appears on `SnkFFData`, and `SnkFFValid` is asserted. `SnkFFValid` drives the `SrcFFWrEn_n` signal directly, which enables the writing of data into the Source FIFO. The transfer of data continues until the Source FIFO becomes

almost full or the Sink FIFO becomes empty. If the Source FIFO becomes almost full, all outstanding data is written into the Source FIFO and the transfer of data between the FIFOs is halted.

Demonstration Test Bench

The demonstration test bench emulates a PHY device by generating and receiving packet data across the SPI-4.2 interface. The interface between the demonstration test bench and the SPI-4.2 core is illustrated in Figure 10-2.

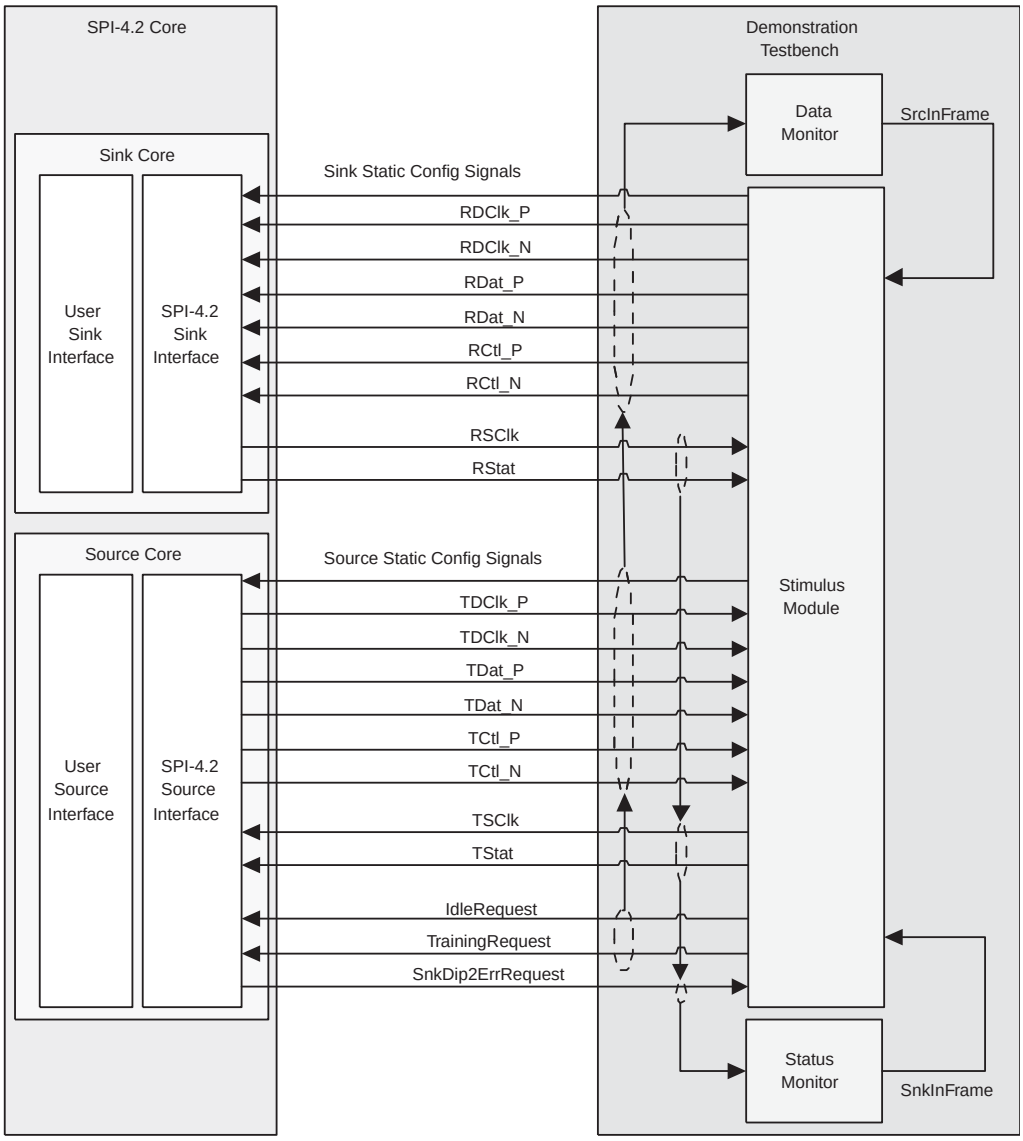


Figure 10-2: **Demonstration Test Bench Connections**

The modules for sending data and status are described in Customizing the Demonstration Test Bench later in this section. As described below and shown in Figure 10-3, the demonstration test bench consists of the following modules:

- Clock Generator
- Startup
- Stimulus
- Data Monitor
- Status Monitor
- Testcase

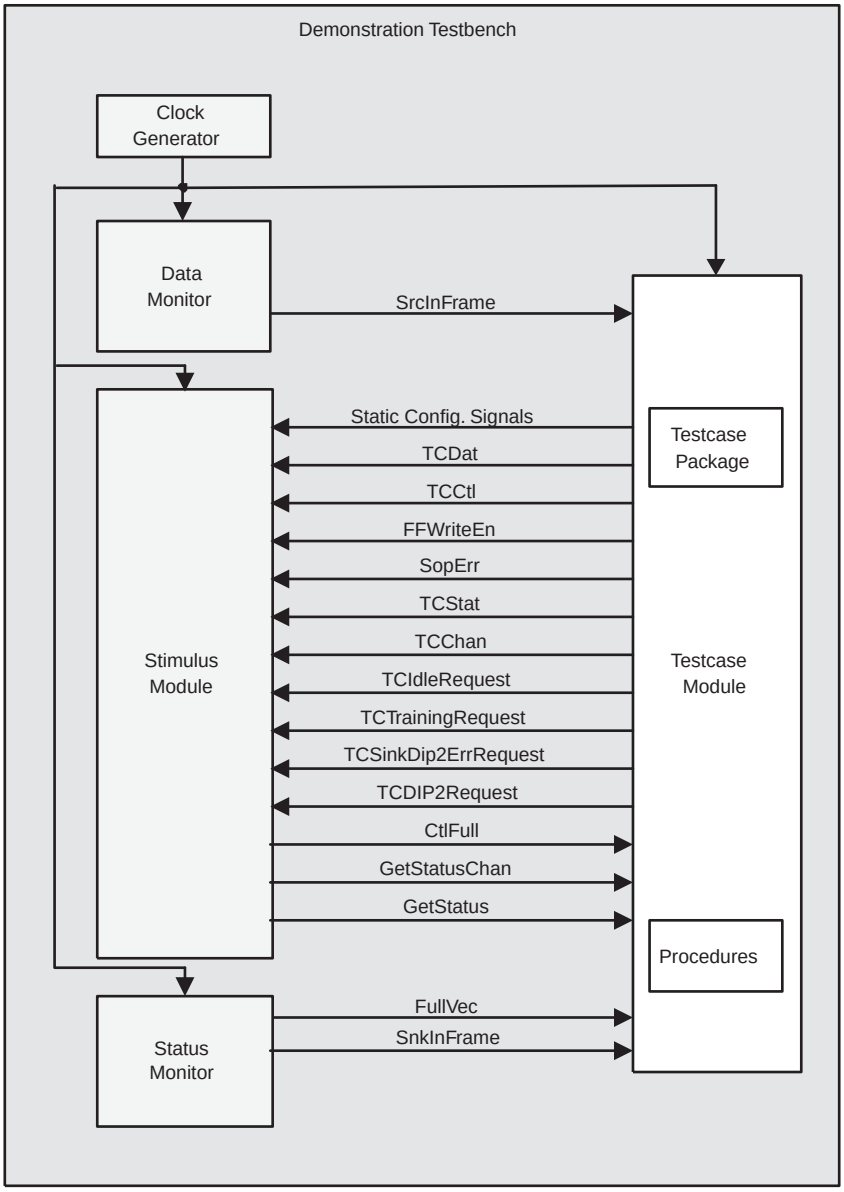


Figure 10-3: **Test Bench Modules**

Clock Generator

The Clock Generator creates all of the clocks that are used in the Design Example, including `SysClk`, `RDClk2x`, `UserClk`, `TSClk`, and `SnkIdelayRefClk`. These clocks are described in more detail in Table 10-10.

Startup Module

The Startup Module contains three functions: DCM/MMCM setup, calendar loading, and Dynamic Phase Alignment (DPA) Initialization. These functions are described in detail in the following sections.

DCM/MMCM Startup

The DCM/MMCM Startup is a state machine that ensures that the DCMs/MMCMs are reset in the appropriate order. If they are not reset appropriately, the DCMs/MMCMs will not lock. The Startup Module first asserts `DCMReset_TDClk`. Once `Locked_TDClk` is asserted, it resets `DCMReset_RDClk`. Then it waits for `Locked_RDClk` before asserting `DCMReset_TSClk`. After `Locked_TSClk` is asserted, the state machine waits until the `SnkClksRdy` and `SrcClksRdy` signals are asserted. The `Reset_n` signal is deasserted only after this occurs. All operations are performed in the `SysClk` domain.

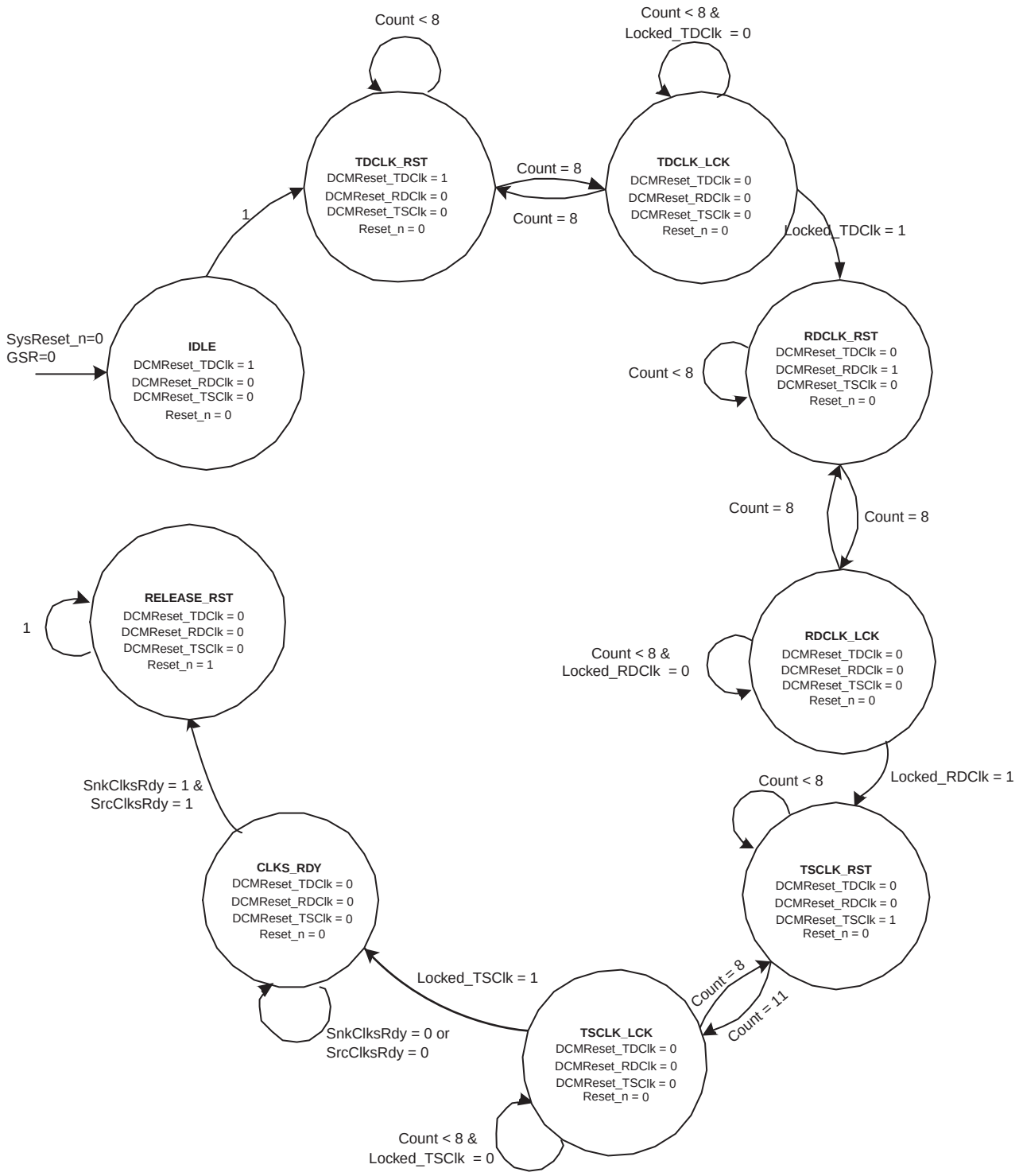


Figure 10-4: **Startup State Diagram**

Figure 10-4 illustrates the nine states for this machine.

- **IDLE** Initial state after system reset (GSR); `DCMReset_TDClk` is asserted.
- **TDCLK_RST** Holds `DCMReset_TDClk` for 8 cycles then releases it.

- **TDCLK_LCK** Waits for the `Locked_TDClk` signal.
- **RDCLK_RST** Holds `DCMReset_RDClk` for 8 cycles then releases it
- **RDCLK_LCK** Waits for the `Locked_RDClk` signal.
- **TSCLK_RST** Holds `DCMReset_TSClk` for 11 cycles then releases it.
- **TSCLK_LCK** Waits for the `Locked_TSClk` signal.
- **CLKS_RDY** Waits for `SnkClksRdy` and `SrcClksRdy` signals.
- **RELEASE_RST** Releases `Reset_n`.

The DCM/MMCM reset duration in this sequence is used for general simulation only. In hardware, the user must follow the DCM/MMCM reset duration as specified in the appropriate data sheet:

- DS302, *Virtex-4 FPGA Data Sheet: DC and Switching Characteristics*
- DS202, *Virtex-5 FPGA Data Sheet: DC and Switching Characteristics*
- DS152, *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics*

Calendar Loader

The second function of the Startup module is the logic to load the calendars. The demonstration test bench reads the Sink calendar sequence and the Source calendar sequence from two different files and loads this information into the calendars of the Sink and Source cores and into the Stimulus module. It also loads the calendar into the Status Monitor so that it can identify which channel is receiving status. The calendar sequences can be modified (see Calendar Sequence Files (Sink and Source), page 216).

DPA Initialization

The third function of the Startup module is to initialize the Dynamic Phase Alignment section of the Sink core. It is present in the module only if Dynamic Alignment is selected in the CORE Generator system. It simply asserts the PhaseAlignRequest signal to the Sink core for two cycles of `UserClk` once the core is out of reset.

Once `PhaseAlignRequest` is asserted, the dynamic alignment algorithm needs some time before completing its alignment and asserting `PhaseAlignComplete`. This value is dependent on the frequency of `RDClk` and when `PhaseAlignRequest` is asserted.

Stimulus Module

While the testcase and procedures modules are used to generate data and status, the stimulus module is used to actually send this data to the SPI-4.2 core. The stimulus module either transmits data and status generated by the testcase module, or it directly transmits training or idle data and framing status. In addition to sending status and data, the stimulus module drives the static configuration signals defined in the testcase module. The behavior of the stimulus module can be modified with the constants defined in the testcase package.

The Stimulus module also performs the following operations:

- Sends training or framing if the core is out of frame
- Inserts periodic training on `RDat`
- Ensures minimum SOP spacing is met
- Calculates DIP2 and DIP4 values

- Drives Source core request signals
- Merges SOP and EOP control words

The Stimulus module has two status inputs: `SnkInFrame` and `SrcInFrame`. If `SnkInFrame` is deasserted, the stimulus module sends training patterns over `RDat` until `SnkInFrame` is asserted. If `SrcInFrame` is deasserted, the stimulus module sends framing over `TStat` until `SrcInFrame` is asserted.

Procedures Module

The procedures module is a package of functions instantiated in the testcase module to simplify sending data and status to the stimulus module. Using these functions, you can create any desired sequence of data or status. The method by which functions are called varies among languages, and is described in the appendices.

The following functions are supported in the procedures module:

- **send_packet** Used to transmit an entire packet of data. This procedure will always send an SOP control word before the burst of data and an EOP control word following the data burst.
- **send_user_data** Used to transmit a burst of data. The presence of an SOP control word (before the burst of data) and an EOP control word (following the data burst) can be specified. The EOP can optionally specify an abort (ERR).
- **send_idles** Used to send idle cycles.
- **send_training** Used to send training patterns.
- **sop_spacing** Used to send erred data by sending two SOP words in less than eight cycles. This function limits the number of cycles between the two SOPs to less than seven. This ensures that an SOP spacing error occurs.
- **reset** Used to reset the interface to the stimulus module. Should be called at the beginning of any testcase.
- **send_status** Used to change the status (on `TStat`) for a particular channel.
- **get_status** Used to check the status of a specific channel.

Data Monitor

The data monitor is responsible for verifying that data sent from the demonstration test bench is the same as the data received from the core. This is accomplished by monitoring the `RDat` and `RCtl` signals that are input into the Sink core, and comparing them to the `TCtl` and `TDat` signals output from the Source core. This is a simple comparison as long as the data being sent does not violate the *OIF-SPI4-02.1* specification. If the specification is violated, the SPI-4.2 core modifies the data to enforce compliance, and the data monitor accounts for the modification before comparing `TDat` to `RDat`. In addition to the data, the monitor also verifies DIP4, SOP spacing, IDLE request, Training request, `DATA_MAX_T`, and `ALPHA_DATA` compliance. Changes in the testcase can create situations that cause the data monitor to output warning messages. For more information on output warning messages, see Appendix A, Messages and Warnings.

Status Monitor

The status monitor inspects the RStat bus. In addition to verifying correct values for channel status, it compiles the current status for each channel into the vector FullVec. FullVec is used by the testcase module when the CHECK_RSTAT constant is set to stall data on RDat when the targeted channel is full. See Table 10-11 for more information about the FullVec vector.

The status monitor also calculates the DIP2 value for RStat and compares it with what is actually received. If there is an error, it looks at the signal SnkDIP2ErrRequest to see if it was asserted and the error is expected.

Lastly, the signal SnkInFrame is created in the status monitor by inverting SnkOof. This signal is used by the stimulus module to send training. See Appendix A, Messages and Warnings.

Customizing the Demonstration Test Bench

The demonstration test bench can be used with default settings or customized to observe the behavior of the SPI-4.2 core for different configurations.

The demonstration test bench can be programmed to transmit a range of stimuli by modifying TSCLK_LCK.

- Testcase Package: contains constants used by the testcase module
- Testcase Module: generates data and status
- Sink Calendar Sequence: contains the channel order for the Sink core status
- Source Calendar Sequence: contains the channel order for the Source core status

The following sections describe each module, including customization methods and resulting behavior. The module descriptions are applicable to both VHDL and Verilog designs. Language-specific details for VHDL are provided in Appendix B, VHDL Details Language-specific details and source code showing how to further randomize input to the SPI-4.2 core for Verilog are provided in Appendix C, Verilog Details.

Testcase Package

The testcase package contains a list of constants that define the ways that the cores and demonstration test bench operate. Some of these are user-defined and can be modified, while others are defined when the core is generated. Table 10-10 provides test bench constants that can be modified. These constants are modified by regenerating the core in the CORE Generator system.

Table 10-10: **Testcase Package User-Defined Constants**

Name	Constant Type	Default Value (Range)	Description
SNK_CAL_DATA	String	snk_calendar.dat <filename>	Contains the name of the file with the Sink calendar sequence to be programmed.
SRC_CAL_DATA	String	src_calendar.dat <filename>	Contains the name of the file with the Source calendar sequence to be programmed.
SNK_ALPHA_DATA	Integer	3 <0 - 255>	Sets the number of repetitions of the 20-word training pattern sent to the Sink core (0 means don't send periodic training).

Table 10-10: **Testcase Package User-Defined Constants *(Cont'd)***

Name	Constant Type	Default Value (Range)	Description
SNK_DATA_MAX_T	Integer	4000 <0-65535>	Sets the number of cycles between training patterns sent to the Sink core (0 means don't send periodic training).
MERGE_PAYLOAD	Integer	0 <0 or 1>	Before data is sent on RDat, the demonstration test bench can either merge an EOP and SOP control word into one payload control word, or it can leave them as two separate control words. **1**: Merge EOP and SOP is enabled. **0**: Merge EOP and SOP is disabled.
CHECK_RSTAT	Integer	0 <0 or 1>	The demonstration test bench can operate in two modes with respect to the incoming status signal RStat. It either ignores the value on RStat or checks the value on RStat. **0**: Ignore the value on RStat. The test bench continues to send data on RDat regardless of the status of the current channel. **1**: Check the value on RStat. The test bench checks the status of the current channel before sending data to it. If the channel is satisfied (RStat = '10'), then the test bench does not send the packet of data and instead tries to send the next packet. The test bench sends the packet if the channel is starving or hungry (RStat = '01' or '00').
DATA_TYPE	Integer	1 <0, 1, 2>	Three types of data can be generated on RDat. The first type simply increments the data on each channel (e.g. sends 0, 1, 2 to channel 0, sends 0, 1, 2 to channel 1, then sends 3, 4, 5 to channel 0). The second sends randomized data on RDat. The last type sends data read from the file <TEST_DATA_FILE>. **0**: Send incremental data **1**: Send random data **2:** Send data read from file
TEST_DATA_FILE	String	data_file.dat <filename>	Contains the name of the file to be read if DATA_TYPE = 2
RANDOM_SEED	Integer (Verilog)	5431 <any 32-bit integer value>	Initial seed for the random number generator. To get different results between two runs of a random test bench, the seed must be changed. If the seed is not changed between runs, then every random number is the same as the previous run.
	std_logic_vector(31 downto 0) (VHDL)	x"1537" <any 32-bit vector>	

Table 10-10: **Testcase Package User-Defined Constants *(Cont'd)***

Name	Constant Type	Default Value (Range)	Description
DATA_NUM_TRAIN_SEQ	Integer	3 <0 - 255>	Sets the number of complete training patterns that the demonstration test bench has to receive on TDat (upon startup) before it stops sending framing sequences on TStat. Once this happens, the demonstration test bench begins sending valid status.
TDCLK_PERIOD	Time	2.86 ns <time>	Sets the period of the SysClk signal, which is used by the Source core to generate TDClk. Value must be greater than or equal to 2.00 ns (≤ 500 MHz).
RDCLK_PERIOD	Time	2.86 ns <time>	Sets the period of the RDClk signal and the half-period of the RDClk2x signal. Value must be greater than or equal to 2.00 ns (≤ 500 MHz).
USERCLK_PERIOD	Time	5.71 ns <time>	Sets the period of the UserClk, used for the loopback interface to the cores and programming of the calendars. Value must be greater than or equal to 4.00 ns (≤ 250 MHz).
TFF	Time	500 ps <time>	Clock-to-out time used by logic in the demonstration test bench

Testcase Module

The testcase module generates data and sends it to the stimulus module, which in turn transmits data to the Sink core and status to the Source core. The following data is created in the testcase module:

- Static configuration signals
- SPI-4.2 and demonstration test bench requests
- Source core status and Sink core data

Figure 10-2 shows the interface between the testcase and stimulus modules.

The static configuration signals are set when the SPI-4.2 core is generated; these signals can also be modified in circuit. The description of these signals can be found in Chapter 5, Designing with the Core.

The status and data generation is simplified by instantiating the procedures module and calling the functions contained in the module. This allows the testcase module to be completely asynchronous, as all of the clocking is done in the procedures module.

Table 10-11 contains a list of common useful test case signals and descriptions.

Table 10-11: **Useful Testcase Signals**

Name	Description
FullVec	An array of bits indicating the last status received on RStat for each channel. For each channel, the corresponding bit is set (1) if the status received was '10' - satisfied, and cleared (0) if the status was '01' - hungry or '00' - starving.
NumLinks	The number of channels for which the core was configured.

Table 10-11: **Useful Testcase Signals**

Reset_n	Reset signal to the Sink and Source core (active low).
SnkEn	Enable signal to the Sink core.
SnkFifoReset_n	FIFO Reset signal to the Sink core (active low).
SnkInFrame	Asserted when the Sink core is in frame (as interpreted by the status monitor).
SnkOof	Out-of-Frame signal from the Sink core.
SrcEn	Enable signal to the Source core.
SrcFifoReset_n	FIFO Reset signal to the Source core (active low).
SrcInFrame	Asserted when the Source core is in frame (as interpreted by the data monitor).
SrcOof	Out-of-Frame signal from the Source core.

There are five request signals that can be asserted in the testcase module. The first four signals interface to the stimulus module (see Figure 10-2, page 206). The fifth is encapsulated with the generated data sent to the stimulus module. Table 10-12 details request signals.

Table 10-12: **Testcase Module Request Signals**

Name	Function
TCIdleRequest	Drives the IdleRequest input to the Source core, which results in idles begin transmitted on TDat.
TCTrainingRequest	Drives the TrainingRequest input to the Source core, which causes training to be sent on TDat.
TCSnkDip2ErrRequest	Drives the SnkDip2ErrRequest input to the Sink core, which results in DIP2 errors on RStat.
TCDIP2Request	When asserted (active high), causes DIP2 errors to be transmitted on TStat.
TCDIP4Request	When asserted (active high), causes DIP4 errors to be transmitted on RDat.

In addition to the request signals described above, the test case module has control over the Sink and Source cores with the `SnkEn, SrcEn, SnkFifoReset_n,` and `SrcFifoReset_n` signals. Descriptions of these signals can be found in Chapter 3, Core Overview.

The Source core status is also generated in the test case module using functions contained in the procedures module. Using the function `send_status`, you can specify a channel and the status for that channel. This sends the status and the channel to the stimulus module for transmission to the core. The stimulus module ensures that the status is sent in the correct location of the calendar sequence.

Calendar Sequence Files (Sink and Source)

The `snk_calendar.dat` and `src_calendar.dat` files are used to define the order that status is sent on the SPI-4.2 Interface. The number of lines in a file is equal to the length of the calendar sequence (`SnkCalendar_Len + 1` and `SrcCalendar_Len +1`). Each line of the file represents an 8-bit calendar entry in hexadecimal format. For example, a calendar with a length of five and a sequence of <channel 0, channel 1, channel 0, channel 2, channel 3> can be generated by the following format:

```
00
01
00
02
03
```

File names are defined in the test case package, and can be changed if desired.

XILINX®

Appendix A

Messages and Warnings

Data and Status Monitor Warnings

The Data and Status monitors continuously check data sent to and received from the demonstration test bench. There are several common warnings that occur when the Testcase module is modified. The warnings are listed and described below.

TDat Warning: Source is segmenting packets <*simulation time*>

This warning means that the Source core is sending payload resumes in the middle of sending a burst. This is acceptable operation if `SrcBurstMode` = 0. If `SrcBurstMode` = 1, this should only occur if the maximum burst length is reached (as defined by `SrcBurstLen`).

RStat Info: Sink is out of frame. Expect TDat mismatches <*simulation time*>

This indicates that the Sink core went out of frame during operation. Unless training or idles are being sent on `RDat` when this occurs, there will be data errors on TDat. This is because what is being sent in on `RDat` is no longer being transferred to TDat.

RStat Info: Expected DIP2 mismatch received: `SnkDip2ErrReqFlag` = 1 <*simulation time*>

This indicates that a `DIP2` error was detected on `RStat`. It is only a note and not an error because `SnkDip2ErrReq` was asserted, which means that a `DIP2` error is expected.

RDat Warning: Protocol Violation #4. Idle follows data on a non-credit boundary <*simulation time*>

This indicates that the SPI-4.2 protocol was violated when data was sent from the demonstration test bench. The most likely cause is that `send_user_data` was used to send data without an EOPS, which ended on a non-credit boundary, then an idle was sent using `send_idles`.

RDat Warning: Protocol Violation

Any RDat protocol violation occurred because of incorrectly formatted data transmitted from the Testcase Module (that is, they are user-created).

TDat/TCtl Error: Data mismatch

** TCtl Error: Control Mismatch : Expected : '1', Received '0'.

** TDat Error: Data Mismatch #4. Expected x0040, Received x0041.

These error messages might occur when the test bench continues sending data when the Sink core asserts the Almost Full flag, causing the Sink FIFO to overflow.

Test Bench: Test bench may hang up

** The test bench may hang up with the following message: 'Source core sent its first data. 9105500 ps...'

This issue may occur when the last packet sent by the test bench is not terminated by an EOP (even when it should). This causes this packet to be stuck in the source core.

Timing Simulation Warning and Error Messages

There are several common simulation warnings and error messages when timing simulation is run on the example design. These warnings and messages are described in this appendix.

TDat Error: Data Mismatch # 4. Expected 000f, Received 000x. 339280 ps

The data mismatch results from the data going to unknown "x" state. To prevent "x" from propagating in your simulation, add the "+no_notifier" option to the `vsim` command when using ModelSim Simulator (MTI). If you are using other simulators, consult the manufacturer documentation for possible ways to turn off "x" propagation.

SETUP, HOLD, RECOVERY violation on /X_FF

These violations might come from either the Sink core or Source Core, and they originated from register elements that are transiting between two clock domains. These timing violations can be safely ignored.

When running simulation on a SPI-4.2 Sink Core with Global Clocking and DPA Clock Adjustment option, the signal `Locked_RDClk` (from RDClk DCM) might get deasserted after `PhaseAlignRequest` is asserted. When the `PhaseAlignRequest` has been asserted, the IDELAY goes through the reset process and the clock stops toggling momentarily. This might cause the lock signal from the DCM to get deasserted in simulation (this does not occur in hardware testing). `Locked_RDClk` should be ignored after the `PhaseAlignRequest` has been asserted in simulation.

Memory Collision Error

The "Memory Collision" error occurs occasionally in both functional and timing simulations because the calendar block is trying to read out values at the same time that you are writing them in; however, this is not a problem because you are only supposed to write the calendar when the core is disabled.

** Error: */X_ISERDES SETUP Low VIOLATION ON D WITH RESPECT TO CLK;

** Error
pl4_demo_testbench.pl4_wrapper0.\core_pl4_snk_top0/U0/io0/dpa2/dpa_top0/DATAPAIR0/SLAVE .xsetuphold_chk $setup

** Error
pl4_demo_testbench.pl4_wrapper0.\core_pl4_snk_top0/U0/io0/dpa2/dpa_top0/CTLPAIR/SLAVE .xsetuphold_chk $setup

This error message may occur for some dynamic alignment cores during the data alignment of the Sink core (PhaseAlignComplete=0) and before the core goes in frame. During alignment, the Sink core is looking for the data eye. In this process, the data bus is delayed by different amounts relative to the clock, causing setup violations. So, it is safe to ignore this error while PhaseAlignComplete=0. Once the alignment is complete (PhaseAlignComplete =1), this error should not occur for non-continuous dynamic alignment cores.

** Error pl4_demo_testbench.pl4_wrapper0.\core_pl4_snk_top0/U0/io0/chan17/U1 .xsetuphold_chk $setup(CLK 437466 ps, D 437411 ps, 866 ps)

This error message may occur for some static alignment cores. The phase shift setting provided with the SPI-4.2 core in the constraints file is only a place-holder and may not be appropriate for all designs. See Chapter 5, Designing with the Core for more information.

** Error: Error : input CLKIN1 of X_MMCM_ADV is stopped. Reset is required for X_MMCM_ADV whether stopped input clock comes back or not.

Time: 526250 ps Iteration: 2 Instance:
/pl4_demo_testbench/pl4_wrapper0/pl4_src_clk0_mmcm1

This error message will occur when the internal SysClk and TSClk is generated using MMCM (Virtex-6 devices). This error can be ignored because this MMCM will reset and then subsequently achieve lock after this error message occurs.

Timing Closure

2 constraints not met (from *.par report)

There are a few instances where designs with a 128-bit user interface fail to meet timing when the user interface clock (`SnkFFClk`, `SrcFFClk`) runs at the same frequency as the internal core clock (`RDClkDiv_GP`, `RDClkDiv_User`, `SysClkDiv_GP`, `SysClkDiv_GBSLV`) as defined by the example UCF constraint. The 128-bit user interface is not intended to run at the same rate as the internal core clock because the internal data path of the core is 64 bits wide. If the user interface clock frequency is reduced to three-quarters of the internal core clock frequency, timing closure will be achieved. This enables the user to maintain an efficient data throughput while writing data into the core faster than the core is able process it.

XILINX®

Appendix B

VHDL Details

Procedures Module

The procedures module is a package of functions instantiated in the testcase module to simplify the sending of data and status to the Stimulus module. By using these functions, the user can create any desired sequence of data or status. All functions are called from the Testcase module using the following format:

```
Format: <function name>(<IOBus>, <inputs>)
Example: send_packet(ProBusR, 0, 40): A 40-byte long packet is sent on
channel 0.
```

The procedures module handles all clocking for the Testcase module. For an example of how these procedures are used, see the default file (`pl4_testcase.vhd`) provided with the core.

All functions in the VHDL procedures module use a passed-in record to inspect and modify the state of the interface with the Stimulus module. There are two such record types defined in the procedures module: `ProceduresRDClkBusType` (PBr) and `ProceduresTSClkBusType` (PBt). For a usage example, see the provided testcase file (`pl4_testcase.vhd`).

The tables in this section describe supported functions included in the procedures module.

reset (`PBr`) and **reset** (`PBt`) procedures are used to initialize the PBr and PBt records. They must be called at the beginning of every testcase.

The **send_packet** procedure is used to transmit an entire packet of data. This procedure always sends a SOP control word before the burst of data and an EOP control word following the data burst. The EOPs (bits 14:13 of the control word following the burst) are automatically calculated from the number of bytes sent.

Table B-1: **send_packet (PBr, addr, bytes) Inputs**

Name	Range	Description
ADDR	0 to 255	Channel on which the packet should be sent.
BYTES	1 to 255	Number of bytes to send on the selected channel.

The **send_user_data** procedure is used to transmit a burst of data. The presence of a SOP control word (before the burst of data) and an EOP control word (following the data burst), can be specified. The EOPs (bits 14:13 of the control word following the burst) are

automatically calculated from the number of bytes sent. ERR has a higher priority than EOP; if EOP and ERR are both '1', the EOPs for the burst is an EOP abort = '01'.

Table B-2: **send_user_data (PBr, SOP, EOP, Err, Addr, bytes) Inputs**

Name	Range	Description
SOP	0 or 1	Defines if the packet should begin with a SOP.
EOP	0 or 1	Defines if the packet should be terminated with an EOP.
ERR	0 or 1	Defines if the packet should be terminated with an EOP abort.
ADDR	0 to 255	Channel on which the packet should be sent.
BYTES	1 to 255	Number of bytes to send on the selected channel.

The **send_idles** procedure is used to send idle control words.

Table B-3: **send_idles (PBr, cycles) Inputs**

Name	Range	Description
CYCLES	0 to 511	Number of idle control words to send on RDat.

The **send_training** procedure is used to send training patterns.

Table B-4: **send_training (PBr, patterns) Inputs**

Name	Range	Description
PATTERNS	0 to 255	Number of training patterns to send.

The **sop_spacing** procedure is used to send errored data by sending two SOPs in less than eight cycles. This function limits the number of cycles between the two SOPs to less than seven. This ensures that a SOP spacing error occurs.

Table B-5: **sop_spacing (PBr, Bytes1, Err1, Addr1, EOP2, Err2, Addr2, Bytes2, num_cycles) Inputs**

Name	Range	Description
BYTES1	0 to 10	The number of bytes to send in the first burst. This is limited to 10 bytes to ensure SOP spacing is violated.
ERR1	0 or 1	Defines if the first packet should be terminated with an EOP abort. If set to 0 the EOPs will be calculated from BYTES1.
ADDR1	0 to 255	Channel on which the first packet should be sent.
EOP2	0 or 1	Defines if the second packet should be terminated with an EOP.
ERR2	0 or 1	Defines if the second packet should be terminated with an EOP abort. If set to 0 the EOPs will be calculated from Bytes1.
ADDR2	0 to 255	Channel on which the second packet should be sent.

Table B-5: **sop_spacing (PBr, Bytes1, Err1, Addr1, EOP2, Err2, Addr2, Bytes2, num_cycles) Inputs *(Cont'd)***

Name	Range	Description
BYTES2	1 to 255	The number of bytes to send in the second burst.
NUM_CYCLES	0 to [5 - roundup (BYTES1/2)]	The number of idle cycles between the first and second burst.

The **send_status** procedure is used to change the status for a particular channel.

Table B-6: **send_status (PBt, channel, value) Inputs**

Name	Range	Description
CHANNEL	0 to 255	Defines the channel for which status is updated.
VALUE	00,01,10,11	Defines the new status value to assign to the selected channel.

The **get_status** procedure is called to check status of a specific channel. It will cause the status value of that channel to be returned to the testcase.

Table B-7: **get_status (PBt, channel) Inputs**

Name	Range	Description
CHANNEL	0 to 255	Defines the channel for which status is read.

XILINX®

Appendix C

Verilog Details

Procedures Module

The procedures module is a package of functions instantiated in the Testcase module to simplify sending data and status to the Stimulus module. Use these functions to create any desired sequence of data or status. All functions are called from the Testcase module using the following format:

```
Format: tasks.<function name>(<inputs>)
Example: tasks.send_packet(0,40): A 40-byte long packet is sent on
channel 0.
```

The procedures module handles all clocking for the Testcase module. For an example of how these procedures are used, see the default file (`pl4_testcase.v`) provided with the core.

The tables in this section describe the supported functions included in the procedures module.

The reset procedure is used to reset the interface to the Stimulus Module. This procedure should be called at the beginning of any testcase.

The **send_packet** procedure is used to transmit an entire packet of data. This procedure will always send a SOP control word before the burst of data and an EOP control word following the data burst. The EOPS (bits 14:13 of the control word following the burst) are automatically calculated from the number of bytes sent.

Table C-1: **send_packet (Addr, bytes) Inputs**

Name	Range	Description
ADDR	0 to 255	Channel on which the packet should be sent.
BYTES	1 to 255	Number of bytes to send on the selected channel.

The `send_user_data` procedure is used to transmit a burst of data. The presence of a SOP control word (before the burst of data) and an EOP control word (following the data burst), can be specified. The EOPS (bits 14:13 of the control word following the burst) are automatically calculated from the number of bytes sent. ERR has a higher priority than EOP; if EOP and ERR are both '1', the EOPs for the burst is an EOP abort = '01.'

Table C-2: **send_user_data (SOP, EOP, Err, Addr, bytes) Inputs**

Name	Range	Description
SOP	0 or 1	Defines if the packet should begin with an SOP.
EOP	0 or 1	Defines if the packet should be terminated with an EOP.
ERR	0 or 1	Defines if the packet should be terminated with an EOP abort.
ADDR	0 to 255	Channel on which the packet should be sent.
BYTES	1 to 255	Number of bytes to send on the selected channel.

The `send_idles` procedure is used to send idle control words.

Table C-3: **send_idles (cycles) Inputs**

Name	Range	Description
CYCLES	0 to 511	Number of idle control words to send on RDat.

The `send_training` procedure is used to send training patterns.

Table C-4: **send_training (patterns) Inputs**

Name	Range	Description
PATTERNS	0 to 255	Number of training patterns to send.

The `sop_spacing` procedure is used to send erred data by sending two SOPs in less than eight cycles. This function limits the number of cycles between the two SOPs to less than seven. This ensures that a SOP spacing error occurs.

Table C-5: **sop_spacing (Bytes1, Err1, Addr1, EOP2, Err2, Addr2, Bytes2, num_cycles) Inputs**

Name	Range	Description
BYTES1	0 to 10	The number of bytes to send in the first burst. This is limited to 10 bytes to ensure SOP spacing is violated.
ERR1	0 or 1	Defines if the first packet should be terminated with an EOP abort. If set to 0 the EOPs will be calculated from BYTES1.
ADDR1	0 to 255	Channel on which the first packet should be sent.
EOP2	0 or 1	Defines if the second packet should be terminated with an EOP.
ERR2	0 or 1	Defines if the second packet should be terminated with an EOP abort. If set to 0 the EOPs will be calculated from Bytes1.

Table C-5: **sop_spacing (Bytes1, Err1, Addr1, EOP2, Err2, Addr2, Bytes2, num_cycles) Inputs *(Cont'd)***

Name	Range	Description
ADDR2	0 to 255	Channel on which the second packet should be sent.
BYTES2	1 to 255	The number of bytes to send in the second burst.
NUM_CYCLES	0 to [5 - roundup (BYTES1/2)]	The number of idle cycles between the first and second burst.

The `send_status` procedure is used to change the status for a particular channel.

Table C-6: **send_status (channel, value) Inputs**

Name	Range	Description
CHANNEL	0 to 255	Defines the channel whose status will be updated.
VALUE	00,01,10,11	Defines the new status value to assign to the selected channel.

The `get_status` procedure is called to check status of a specific channel. It will cause the status value of that channel to be returned to the Testcase.

Table C-7: **get_status (channel) Inputs**

Input	Range	Description
CHANNEL	0 to 255	Defines the channel whose status will be read.

Random Testcase Sample Code

The following code is an example that can be inserted into the `pl4_testcase.v` file to send randomized data to the Sink core. It should replace the default code used to send data. In addition to sending randomized data, it also randomly asserts each request signal.

```
wait (Reset_n == 1);
@ (posedge RDClk2x);

//*****************************************************************
******
// Sends out randomized data, idles, or training.
// It also randomly toggles TCIdleRequest, TCTrainingRequest,
// TCDIP4Request, TCDIP2Request, and TCSnkDip2ErrRequest
//*****************************************************************
******
forever
begin
  RandTask = {$random(`RANDOM_SEED + $time)} % 4;
  RandIdleRequest = {$random(`RANDOM_SEED + $random(`RANDOM_SEED +
$time))} % 100;
  RandTrainingRequest = {$random(`RANDOM_SEED + $time)} % 100;
  RandDIP4Request = {$random(`RANDOM_SEED + $time +
$random(`RANDOM_SEED))} % 100;
  RandDIP2Request = {$random($random(`RANDOM_SEED) + $time)} % 100;
  RandSnkDip2ErrRequest = {$random(`RANDOM_SEED + $random($time))} %
100;
```

```
//Randomly set TCIdleRequest to 1
if ((RandIdleRequest == 0) || (TCIdleRequest == 1))
begin
  if (TCIdleRequest == 1)
  begin
    if (IdleRequestCnt > 0)
    begin
      IdleRequestCnt <= IdleRequestCnt - 1'b1;
      TCIdleRequest <= 1'b1;
    end
    else
    begin
      IdleRequestCnt <= 'b0;
      TCIdleRequest <= 1'b0;
    end
  end
  else
  begin
    TCIdleRequest <= 1'b1;
    IdleRequestCnt <= {$random(`RANDOM_SEED + $time)} % 9;
  end
end

//Randomly set TCTrainingRequest to 1
if ((RandTrainingRequest == 0) || (TCTrainingRequest == 1))
begin
  if (TCTrainingRequest == 1)
  begin
    if (TrainingRequestCnt > 0)
    begin
      TrainingRequestCnt <= TrainingRequestCnt - 1'b1;
      TCTrainingRequest <= 1'b1;
    end
    else
    begin
      TrainingRequestCnt <= 'b0;
      TCTrainingRequest <= 1'b0;
    end
  end
  else
  begin
    TCTrainingRequest <= 1'b1;
    TrainingRequestCnt <= {$random(`RANDOM_SEED + $time)} % 9;
  end
end

//Randomly set TCDIP4Request to 1
if ((RandDIP4Request == 0) || (TCDIP4Request == 1))
begin
  if (TCDIP4Request == 1)
  begin
    if (DIP4RequestCnt > 0)
    begin
      DIP4RequestCnt <= DIP4RequestCnt - 1'b1;
      TCDIP4Request <= 1'b1;
    end
    else
    begin
```

```
      DIP4RequestCnt <= 'b0;
      TCDIP4Request <= 1'b0;
    end
  end
  else
  begin
    TCDIP4Request <= 1'b1;
    DIP4RequestCnt <= {$random(`RANDOM_SEED + $time)} % 9;
  end
end

//Randomly set TCDIP2Request to 1
if ((RandDIP2Request == 0) || (TCDIP2Request == 1))
begin
  if (TCDIP2Request == 1)
  begin
    if (DIP2RequestCnt > 0)
    begin
      DIP2RequestCnt <= DIP2RequestCnt - 1'b1;
      TCDIP2Request <= 1'b1;
    end
    else
    begin
      DIP2RequestCnt <= 'b0;
      TCDIP2Request <= 1'b0;
    end
  end
  else
  begin
    TCDIP2Request <= 1'b1;
    DIP2RequestCnt <= {$random(`RANDOM_SEED + $time)} % 9;
  end
end

//Randomly set TCSnkDip2ErrRequest to 1
if ((RandSnkDip2ErrRequest == 0) || (TCSnkDip2ErrRequest == 1))
begin
  if (TCSnkDip2ErrRequest == 1)
  begin
    if (SnkDip2ErrRequestCnt > 0)
    begin
      SnkDip2ErrRequestCnt <= SnkDip2ErrRequestCnt - 1'b1;
      TCSnkDip2ErrRequest <= 1'b1;
    end
    else
    begin
      SnkDip2ErrRequestCnt <= 'b0;
      TCSnkDip2ErrRequest <= 1'b0;
    end
  end
  else
  begin
    TCSnkDip2ErrRequest <= 1'b1;
    SnkDip2ErrRequestCnt <= {$random(`RANDOM_SEED + $time)} % 9;
  end
end

//Sends a random sized packet to a random channel
if (RandTask == 0)
```

```
  begin
    tasks.send_packet({$random(`RANDOM_SEED + $time)} % (`NUM_CHANNELS
- 1), ($random(`RANDOM_SEED + $time) % 255) + 1'b1);
  end
  //Sends a random sized packet to a random channel. Also SOP, EOP, and
  //Err are randomized
  else if (RandTask == 1)
  begin
    tasks.send_user_data({$random(`RANDOM_SEED + $time)} % 2,
{$random(`RANDOM_SEED + $time + $random(`RANDOM_SEED))} % 2,
{$random(`RANDOM_SEED + $time + $random(`RANDOM_SEED + $time))} % 2,
{$random(`RANDOM_SEED + $time)} % (`NUM_CHANNELS - 1),
($random(`RANDOM_SEED + $time) % 255) + 1'b1);
  end
  //Sends a random number of idles to the Sink Core
  else if (RandTask == 2)
  begin
    tasks.send_idles(({$random(`RANDOM_SEED + $time)} % 10) + 1);
  end
  //Sends a random number of training patterns to the sink core
  else if (RandTask == 3)
  begin
    tasks.send_training(({$random(`RANDOM_SEED + $time)} % 10) + 1);
  end
  else
  begin
    @ (posedge RDClk2x);
    $display("Out of Range: %0d", $time);
  end
end
```

XILINX®

Appendix D

SPI-4.2 File Descriptions

This appendix lists the files generated by the Xilinx CORE Generator system.

Name	Description
CORE Generator Project Files (<proj_dir>)	
<component_name>_pl4_snk_top.ngc*	Sink core netlist
<component_name>_pl4_src_top.ngc*	Source core netlist
<component_name>_pl4_snk_top.v[hd]	Sink core functional VHDL or Verilog simulation model
<component_name>_pl4_src_top.v[hd]	Source core functional VHDL or Verilog simulation model
<component_name>_pl4_snk_top.v{ho|eo}	Sink core VHDL or Verilog instantiation template
<component_name>_pl4_src_top.v{ho|eo}	Source core VHDL or Verilog instantiation template
<component_name>.xco	CORE Generator project specific option file
<component_name>_flist.txt	
Release Notes (<proj_dir>/<component_name>)	
spi4_2_readme.txt	SPI-4.2 Release Notes (text file)
Documentation (<proj_dir>/<component_name>/doc)	
spi4_2_ds209.pdf	SPI-4.2 Core Data Sheet (PDF)
spi4_2_ug153.pdf	SPI-4.2 User Guide (PDF)
Example Design Files (<proj_dir>/<component_name>/example_design)	
<component_name>_top.ucf*	SPI-4.2 Core Wrapper Design Example Constraints File
<component_name>_top.v[hd]*	Core VHDL or Verilog Top Level Wrapper File
[<component_name>_snk_top.v]	Verilog component declaration for the Sink core
[<component_name>_src_top.v]	Verilog component declaration for the Source core
pl4_fifo_loopback.v[hd]	FIFO Loopback VHDL or Verilog Top-level File
pl4_fifo_loopback_read.v[hd]	FIFO Loopback VHDL or Verilog Read Module
pl4_fifo_loopback_write.v[hd]	FIFO Loopback VHDL or Verilog Write Module
virtex4.v	Module instantiation for Virtex®-4 primitives.
virtex5.v	Module instantiation for Virtex-5 primitives.

Name	Description
Implementation Script Files (<proj_dir>/<component_name>/implement)	
implement.sh	UNIX Implementation Script File
implement.bat	DOS Implementation Script File
xst.prj	XST Synthesis Project File
xst.scr	XST Synthesis Script File
synplify.prj	Synplicity Synthesis Project File
Demonstration Testbench Simulation Files (<proj_dir>/<component_name>/simulation)	
data_file.dat	Data file containing the data to be sent across the SPI-4.2 Interface
pl4_clk_gen.v[hd]	Demo Testbench Clock Generator
pl4_data_monitor.v[hd]	Demo Testbench Data Monitor
pl4_demo_testbench.v[hd]	Demo Testbench Top Level Module
pl4_procedures.v[hd]	Demo Testbench Procedures Module
pl4_src_clk.v[hd]	Example clocking module used when source core is configured for slave clocking (Virtex-4 or Virtex-5 devices) or user clocking (Virtex-6 devices).
pl4_snk_clk.v[hd]	Example clocking module used when the Sink core is configured for user clocking (Virtex-6 devices only).
pl4_startup.v[hd]	Demo Testbench DCM or MMCM Startup and Calendar Loader Module
pl4_status_monitor.v[hd]	Demo Testbench Status Monitor
pl4_stimulus.v[hd]	Demo Testbench Data and Status Stimulus Module
pl4_testcase.v[hd]	Demo Testbench Testcase Module
pl4_testcase_pkg.v[hd]	Demo Testbench Package File
snk_calendar.dat	Data file containing the calendar information for the Sink interface
src_calendar.dat	Data file containing the calendar information for the Source interface

Functional Simulation Scripts (<proj_dir>/component_name>/simulation/functional)	
simulate_mti.do	ModelSim script for compiling the SPI-4.2 core functional simulation netlist and demo testbench files; this script also loads and runs the simulation.
wave_mti.do	ModelSim script for loading a pre-formatted waveform
simulate_ncsim.sh	UNIX shell script for compiling the SPI-4.2 core functional simulation netlist and demo testbench files for Incisive Enterprise Simulator (IES); this script also loads and runs the simulation.

Name	Description
simulate_ncsim.bat	DOS shell script for compiling the SPI-4.2 core functional simulation netlist and demo testbench files; this script also loads and runs the simulation.
wave_ncsim.sv	IES script for loading a preformatted waveform.
simulate_vcs.sh (Verilog only)	Shell script that compiles the functional netlist and example design. The script also runs the functional simulation using VCS.
vcs_session.tcl (Verilog only)	VCS tcl script that opens a wave window. This macro is called by the simulate_vcs.sh script.
ucli_commands.key (Verilog only)	VCS command file. This file is called by the simulate_vcs.sh script.
Timing Simulation Scripts (<proj_dir>/component_name>/simulation/timing)	
simulate_mti.do	ModelSim script for compiling the SPI-4.2 core back-annotated timing netlist and demo testbench files; this script also loads and runs the simulation.
wave_mti.do	ModelSim script for loading a pre-formatted waveform
simulate_ncsim.sh	UNIX shell script for compiling the SPI-4.2 core back-annotated timing netlist and demo testbench files for IES; this script also loads and runs the simulation.
simulate_ncsim.bat	DOS shell script for compiling the SPI-4.2 core back-annotated timing netlist and demo testbench files; this script also loads and runs the simulation.
wave_ncsim.sv	IES script for loading a preformatted waveform.
simulate_vcs.sh (Verilog only)	Shell script that compiles the post-par timing netlist and example design. The script also runs the timing simulation using VCS.
vcs_session.tcl (Verilog only)	VCS tcl script that opens a wave window. This macro is called by the simulate_vcs.sh script.
ucli_commands.key (Verilog only)	VCS command file. This file is called by the simulate_vcs.sh script.

* Indicates that the file is required for implementation.

XILINX®

Appendix E

SPI-4.2 Control Word

This appendix defines the SPI-4.2 control word format as shown in Table E-1. This format is reproduced from the *OIF-SPI4-02.1* specification, Table 6.2.

Table E-1: **SPI-4.2 Control Word Format**

Bit Position	Label	Description
15	Type	Control Word Type Set to either of the following: 1: payload control word 0: idle or training control word
14:13	EOPS	End-of-Packet Status Set to one of the following values below according to the status of the immediately preceding payload transfer 00: Not an EOP 01: EOP Abort 10: EOP Normal termination, 2 bytes valid 11: EOP Normal termination, 1 byte valid EOPS is valid in the first control word following a burst transfer. It is ignored and set to “00” otherwise.
12	SOP	Start-of-Packet Set to “1” if the payload transfer immediately following the control word corresponds to the start of a packet. Set to “0” otherwise. Set to “0” in all idle and training control words.
11:4	ADDRESS	Port Address 8-bit port address of the payload transfer immediately following the control word. None of the addresses are reserved (all 256 are available for payload transfer). Set to zeroes in all idle control words. Set to ones in all training control words.
3:0	DIP-4	4-bit Diagonal Interleaved Parity 4-bit odd parity computed over the current control word and the immediately preceding data words (if any) following the last control word.

XILINX®

Appendix F

SPI-4.2 Calendar Programming

This appendix lists examples that describe how to program calendars for the Source Status FIFO and Sink Status FIFO of the SPI-4.2 core.

Overview

In a typical application, the calendars for the Source FIFO status and Sink FIFO status will be programmed identically. In this case, you may choose to combine the RX and TX calendar input signals (clocks, write enable, address, and data) and drive them from the same Source. This allows initialization of the RX and TX calendars simultaneously.

In the SPI-4.2 core, the notion of calendar replaces the polling/packet (or cell) available functionality in previous POS-PHY and UTOPIA specifications. In these preceding standards, the link or ATM layer polls the channels and the physical layer responds with a "packet available" or "cell available" status. In SPI-4.2, the polling is replaced by FIFO status reporting of each channel in a specific order that is controlled by the calendar. In this implementation, as illustrated in the examples below, the calendar is inserted as a table containing channel numbers that is initialized at power-up.

Example 1

In a channelized OC-192 with 192 STS-1 channels, all channels have equal bandwidth and should report their status with equal frequency. In this case, the Calendar Length is 192 (Calendar_Len=191) and the Calendar entries are: 0, 1, 2, …, 191.

Example 2

In a channelized OC-192 with three STS-48 channels (0, 1, and 2) and 4 STS-12 channels (3, 4, 5, and 6), the three STS-48 channels have four times the bandwidth of the 4 STS-12 channels. Therefore, the 3 high-speed channels should report their status 4 times as frequently as the low-speed channels in one Calendar cycle. In this case, the Calendar Length is 16 (Calendar_Len=15) and the Calendar entries are: 0, 1, 2, 3, 0, 1, 2, 4, 0, 1, 2, 5, 0, 1, 2, 6.

Once the Calendar is programmed, the user circuitry updates FIFO status in the dual-port RAM in the Sink block and the SPI-4.2 core sends the updated status information in the order programmed in the calendar. Likewise, in the Source block, the SPI-4.2 core receives the FIFO status information according to the order programmed in the calendar and writes the status in the dual-port RAM to be read by the user circuitry.

Example 3

In a OC-192c application, 1 channel requires the complete SPI-4.2 bandwidth. In this case, the calendar length can be set to 1 (*Calendar_Len*=0). The calendar does not have to be programmed on startup, as it will initialize to all zeros on power-up.

XILINX®

Appendix G

SPI-4.2 Core Verification

Extensive software testing with an internally developed test suite is performed for each SPI-4.2 release. Using our in-house verification environment, the SPI-4.2 Core was tested in RTL, post-ngdbuild, and timing simulation. When using the in-house verification environment, the SPI-4.2 core was tested in three stages:

- Functional (RTL) verification
- Gate-level (post ngdbuild back-annotation HDL) verification
- Gate-level with back-annotated Timing (with SDF file) verification targeting the following device/frequency combinations:
  - Virtex®-4 FPGAs -10 devices up to 800 Mbps on the SPI-4.2 Interface and 225 MHz on the User Interface (SrcFFClk and SnkFFClk clocks).
  - Virtex-4 FPGAs -11 devices up to 900 Mbps on the SPI-4.2 Interface and 250 MHz on the User Interface (SrcFFClk and SnkFFClk clocks).
  - Virtex-5 FPGAs -1 devices up to 1 Gbps on the SPI-4.2 Interface and 250 MHz on the User Interface (SrcFFClk and SnkFFClk clocks).
  - Virtex-5 FPGAs -2 devices up to 1+ Gbps on the SPI-4.2 Interface and 312 MHz on the User Interface (SrcFFClk and SnkFFClk clocks).
  - Virtex-6 FPGAs -1, -2, and -3 devices up to 1+ Gbps on the SPI-4.2 Interface and 250+ MHz on the User Interface (SrcFFClk and SnkFFClk clocks).

For each of these stages, each feature of the SPI-4.2 core was fully verified. The following are examples of stimulus used to verify the features:

Verification of Valid Data

- SPI-4.2 bus traffic that contains short packets that were smaller than one credit (16 bytes)
- SPI-4.2 bus traffic that contains long packets that were larger than one credit (16 bytes)
- SPI-4.2 bus traffic of a constant packet size
- SPI-4.2 bus traffic of a variable packet size
- SPI-4.2 bus traffic that consisted of a single burst or packet
- SPI-4.2 bus traffic that caused the SPI-4.2 Sink FIFO to be constantly almost full
- Backend data traffic that caused the SPI-4.2 Source FIFO to be constantly almost full
- SPI-4.2 bus traffic that caused the Sink FIFO to overflow
- Backend data traffic that caused the Source FIFO to overflow

Verification of Invalid Data

- SPI-4.2 bus traffic that contained incorrect DIP-4 values
- SPI-4.2 status traffic that contained incorrect DIP-2 values
- SPI-4.2 status traffic that indicated that the receiving end of the SPI-4.2 Source block was out-of-frame
- SPI-4.2 bus traffic that violated SOP spacing and had incorrect control word formats
- SPI-4.2 bus traffic that contained data bursts that were not preceded by payload control words
- SPI-4.2 bus traffic that terminated on non-credit boundaries with no EOP.
- SPI-4.2 bus traffic that contained reserved control words
- Backend data traffic that contained no EOP with non-zero MODs.

The behavior of the core was fully verified for a range of core configurations.

Verification for Range of Clock Frequencies

- SPI-4.2 bus clock: 300 MHz to above 500 MHz
- `SrcFFClk` and `SnkFFClk`: 150 MHz to 350 MHz
- `SnkStatClk` and `SrcStatClk`: 100 MHz to 200 MHz
- `SrcCalClk` and `SnkCalClk`: 100 MHz to 200 MHz

In addition to extensive simulation, Xilinx has verified core operation in silicon on the Virtex-6 FPGA ML623, Virtex-5 FPGA ML550, and Virtex-4 FPGA ML450 networking interface platforms. The core was verified at 500+ MHz DDR (equivalent to approximately 16+ Gbps aggregate data rate on the SPI-4.2 Interface and 250+ MHz on the User Interface) for the Virtex architectures.

The ML450 hardware platform that is used for internal testing connects the Source core to the Sink core on a single Virtex-4 device, with 16 inches of cabling for static alignment and skewed cabling for dynamic alignment. The skewed cable introduces seven inches of skew between every adjacent bit. Pseudo-random data is generated internally by an LFSR (x^16 + x^12 + x^3 + x^1 + x^0) and transmitted by the Source core. The data received by the Sink core is then compared against the transmitted data. The SPI-4.2 solution has been tested on this platform with various packet and burst sizes. Additionally, the continuous DPA solution has also been tested on this platform by dynamically skewing the incoming RDClk. This is done by inserting an IDELAY in the RDClk path and changing the tap value counter up and down to emulate a shifting clock. The ML550, ML623 and SP623 hardware boards are also configured similarly to the ML450 when testing the SPI-4.2 core.

Local field offices have access to this hardware platform and an on-site demonstration of SPI-4.2 operating on the ML450 and ML550 platforms can be arranged. Contact your local Xilinx sales representative or field applications engineer for more information.

Appendix H

SPI-4.2 Source Interface Timing Budget

This appendix contains examples on how to create and analyze a SPI-4.2 source interface timing budget. By doing this analysis, users can calculate how much system margin can be allocated for PCB design and system noise. In addition, the analysis can help verify that the design meets the OIF_SPI-4.02.1 specification. Analysis is provided for Sink cores configured as Static or Dynamic as well as different clocking schemes. If trade-offs between clocking schemes are significant, these instances are highlighted.

The timing budgets are based on the Xilinx SPI-4.2 IP and the OIF-SPI-4-02.1 specification requirements. The requirements for the Xilinx SPI-4.2 IP is detailed in the Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3 and Source Clock Interface for Virtex-6 Devices in Chapter 3.

The following figure shows the system-level interface and reference point calculations presented in the OIF specification. These reference points will be referred to in the timing budget calculation.

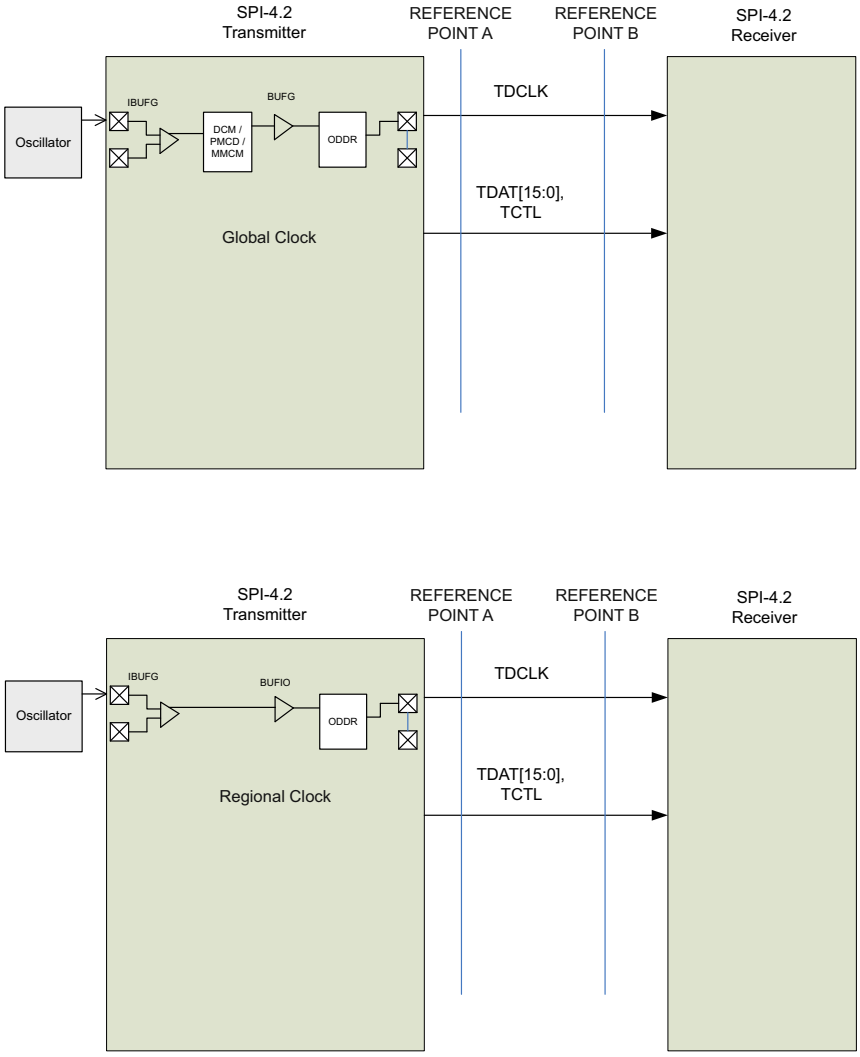


Figure H-1: **OIF Specification Reference Points**

Virtex-4 Interface Timing

There are three clocking schemes available for Virtex-4 devices: global clocking with DCM, global clocking with PMCD, and regional clocking. This section calculates the SPI-4.2 source interface timing budget for these clocking schemes when the Sink core is configured as either static and dynamic alignment.

Sink Core (Static Alignment)

Static alignment can be used for data rates less than or equal to 700 Mbps. Table 6.8a of the OIF specification provides the required timing values at reference point A (TX pins), and worst-case cumulative skew and jitter at reference point B (RX pins) of 790 ps.

The examples below calculate the total jitter and skew at both reference points A and B for a data rate of 700 Mbps. These examples are based on a Source core design targeted at XC4VLX200-FF1514 with -10 speed grade.

Global Clock with DCM

Table H-1: **Sink/Static - Global Clock with DCM: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
FPGA Output Clock DCD(2)	150 ps
FPGA Output Clock Jitter(3)	100 ps
FPGA Output Data Jitter(4)	100 ps
FPGA Global Clock Tree Skew(5)	48 ps
FPGA Package Skew(6)	58 ps
Calculations	
TX Total Skew + Jitter at Reference Point A(7)	*506 ps*
TX Total Skew + Jitter as % of UI at Reference Point A	*0.35 UI*
TX Total Clock DCD	*5.2%*

Table H-2: **Sink/Static - Global Clock with DCM: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Calculations	
OIF Worst-case Skew + Jitter(8)	*790 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Skew and Jitter at Point B - TX Total Jitter at Point A = 790 ps - 506ps	*284 ps*

Notes:

1. Low jitter reference clock requirement in Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3.
2. CLKOUT_DUTY_CYCLE_DLL parameter in DS112, *Virtex-4 Family Overview Data Sheet*.
3. CLKOUT_PER_JITT_0 parameter in DS112, *Virtex-4 Family Overview Data Sheet*.
4. Data Jitter reflects clock jitter and assumes worst case uncorrelated.

5. The FPGA Global Clock Tree Skew is extracted from PAR report (.par file) after the design is placed and routed. In the "Generating Clock Report" section of the PAR report, a table lists the net skew numbers for the clock nets in the design. The clock net used for this calculation is clock net (`*sysclk0_gp`) that drives the output OSERDES and ODDR of the SPI-4.2 interface output signals: `TDat[15:0]`, `TCtl`, and `TDClk`. For best results, all the listed SPI-4.2 interface output should be placed in the same bank.
6. The FPGA Package Skew is calculated using these steps. First run the command **`partgen -v xc4vlx200ff1513`** (use the appropriate device/package) to generate a file that contains the trace lengths for all the pins in the device/package combination. Use this file to find the trace lengths for all the pins assigned to SPI-4.2 interface output signals: `TDat[15:0]`, `TCtl`, and `TDClk`. The difference between the largest trace length and smallest trace length is the package skew. Then multiply this package skew number with 150 ps per inch (industry standard average propagation delay) to get the final FPGA Package Skew number in ps.
7. TX Total Skew + Jitter = 1 + 2 + 3 + 4 + 5 + 6. Assumes worst case linear addition of skew and jitter components.
8. G_max parameter in Table 6.8a of OIF Standard.

Global Clock with PMCD

Table H-3: **Sink/Static - Global Clock with PMCD: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
Input Reference Clock (SYSCLK) DCD(2)	58 ps
FPGA Output Clock DCD(3)	150 ps
FPGA Output Clock Jitter(4)	0 ps
FPGA Output Data Jitter(5)	50 ps
FPGA Global Clock Tree Skew(6)	48 ps
FPGA Package Skew(7)	58 ps
Calculations	
TX Total Skew + Jitter at Reference Point A(8)	*384 ps*
TX Total Skew + Jitter as % of UI at Reference Point A	*0.27 UI*
TX Total Clock DCD	*7%*

Table H-4: **Sink/Static - Global Clock with PMCD: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps

Table H-4: **Sink/Static - Global Clock with PMCD: Reference Point B (RX pins)**

Variable	Measurement
Calculations	
OIF Worst-case Skew + Jitter(9)	*790 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Skew and Jitter at Point B - TX Total Jitter at Point A = 790 ps - 384 ps =	*406 ps*

Notes:

1. Low jitter reference clock requirement in Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3.
2. For a PMCD based design, the input reference clock duty cycle requirement is 48/52 or tighter.
3. TDCD_CLK parameter in DS112, *Virtex-4 Family Overview Data Sheet*.
4. PMCD does not introduce jitter; so the output clock jitter is the same as the input reference clock jitter. The 0 ps notes that the PMCD does not add jitter to the clock (Table 48, Note 1 of DS112, *Virtex-4 Family Overview Data Sheet*).
5. Data Jitter reflects input reference clock jitter and assumes worst case uncorrelated.
6. The FPGA Global Clock Tree Skew is extracted from PAR report (.par file) after the design is placed and routed. In the "Generating Clock Report" section of the PAR report, a table lists the net skew numbers for the clock nets in the design. The clock net used for this calculation is clock net (`*sysclk0_gp`) that drives the output OSERDES and ODDR of the SPI-4.2 interface output signals: `TDat[15:0]`, `TCtl`, and `TDClk`. For best results, all the listed SPI-4.2 interface output should be placed in the same bank.
7. The FPGA Package Skew is calculated using these steps. First run the command **`partgen -v xc4vlx200ff1513`** (use the appropriate device/package) to generate a file that contains the trace lengths for all the pins in the device/package combination. Use this file to find the trace lengths for all the pins assigned to SPI-4.2 interface output signals: `TDat[15:0]`, `TCtl`, and `TDClk`. The difference between the largest trace length and smallest trace length is the package skew. Then multiply this package skew number with 150 ps per inch (industry standard average propagation delay) to get the final FPGA Package Skew number in ps.
8. TX Total Skew + Jitter = 1+2+3+4+5+6. Assumes worst case linear addition of skew and jitter components.
9. G_max parameter in Table 6.8a of OIF Standard.

Regional Clock

Table H-5: **Sink/Static - Regional Clock: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps

Table H-5: **Sink/Static - Regional Clock: Reference Point A (TX pins)**

Variable	Measurement
Input Reference Clock (SYSCLK) DCD(2)	58 ps
FPGA Output Clock DCD(3)	100 ps
FPGA Output Clock Jitter(4)	0 ps
FPGA Output Data Jitter(5)	50 ps
FPGA I/O Clock Tree Skew(6)	50 ps
FPGA Package Skew(7)	180 ps
Calculations	
TX Total Skew + Jitter at Reference Point A(8)	*488 ps*
TX Total Skew + Jitter as % of UI at Reference Point A	*0.34 UI*
TX Total Clock DCD	*6%*

Table H-6: **Sink/Static - Regional Clock: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Calculations	
OIF Worst-case Skew + Jitter(9)	*790 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Skew and Jitter at Point B - TX Total Jitter at Point A = 790 ps - 488 ps	*302 ps*

Notes:

1. Low jitter reference clock requirement in Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3.
2. For Regional Clock based design, the input reference clock duty cycle requirement is 48/52 or tighter.
3. TDCD_BUFIO parameter in DS112, *Virtex-4 Family Overview Data Sheet*.
4. BUFIO does not introduce jitter; so the output clock jitter is the same as the input reference clock jitter. The 0 ps note that the BUFIO does not add jitter to the clock.
5. Data Jitter reflects input reference clock jitter and assumes worst case uncorrelated.
6. TCKSKEW parameter in DS112, *Virtex-4 Family Overview Data Sheet* (XC4VLX200, -10 speed grade). This parameter is device and speed grade dependent.
7. TPKGSKEW parameter in DS112, *Virtex-4 Family Overview Data Sheet* (XC4VLX200-FF1513). This parameter is device and package dependent.
8. TX Total Skew + Jitter = 1 + 2 + 3 + 4 + 5 + 6. Assumes worst case linear addition of skew and jitter components.
9. G_max parameter in Table 6.8a of OIF Standard.

Sink Core (Dynamic Alignment)

Dynamic alignment must be used for data rates more than 700 Mbps. Table 6.8c of the OIF specification gives the required timing values at reference point A (TX pins), and Table 6.8d gives the total jitter allowed at reference point B (RX pins) which is 0.57 UI (total jitter = clock jitter + data jitter). The Xilinx SPI-4.2 Sink core works correctly with this total jitter at the RX pins.

These examples calculate the total jitter at both reference points A and B for a data rate of 800 Mbps. These examples are based on a Source core design targeted to an XC4VLX200-FF1514 with -10 speed grade. Comparing the examples below, the regional clock design has the smallest duty cycle distortion and total jitter.

Global Clock with DCM

Table H-7: **Sink/Dynamic - Global Clock with DCM: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
FPGA Output Clock DCD(2)	150 ps
FPGA Output Clock Jitter(3)	100 ps
FPGA Output Data Jitter(4)	100 ps
Calculations	
TX Total Jitter at Reference Point A(5)	*400 ps*
TX Total Jitter as % of UI at Reference Point A	*0.32 UI*
TX Total Clock DCD	*6%*

Table H-8: **Sink/Dynamic - Global Clock with DCM: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
OIF Clock Jitter(6)	0.13 UI
OIF Data Jitter(7)	0.44 UI
Calculations	
OIF Total Jitter	*0.57 UI*
OIF Total Jitter (ps)	*712.50 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Jitter at Point B - TX Total Jitter at Point A = 712.50 ps - 400 ps	*312.50 ps*

Notes:

1. Low jitter reference clock requirement in Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3.
2. CLKOUT_DUTY_CYCLE_DLL parameter in DS112, *Virtex-4 Family Overview Data Sheet*.
3. CLKOUT_PER_JITT_0 parameter in DS112, *Virtex-4 Family Overview Data Sheet*.
4. Data Jitter reflects clock jitter and assumes worst case uncorrelated.
5. TX Total Jitter = 1 + 2 + 3 + 4 . Assumes worst case linear addition of jitter components.
6. J_clkB parameter in Table 6.8d of OIF Standard.
7. J_datB parameter in Table 6.8d of OIF Standard.

Global Clock with PMCD

Table H-9: **Sink/Dynamic - Global Clock with PMCD: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
Input Reference Clock (SYSCLK) DCD(2)	50 ps
FPGA Output Clock DCD(3)	150 ps
FPGA Output Clock Jitter(4)	0 ps
FPGA Output Data Jitter(5)	50 ps
Calculations	
TX Total Jitter at Reference Point A(6)	*300 ps*
TX Total Jitter as % of UI at Reference Point A	*0.24 UI*
TX Total Clock DCD	*8%*

Table H-10: **Sink/Dynamic - Global Clock with PMCD: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
OIF Clock Jitter(7)	0.13 UI
OIF Data Jitter(8)	0.44 UI
Calculations	
OIF Total Jitter	*0.57 UI*

Table H-10: **Sink/Dynamic - Global Clock with PMCD: Reference Point B (RX pins)**

Variable	Measurement
OIF Total Jitter (ps)	*712.50 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Jitter at Point B - TX Total Jitter at Point A = 712.50 ps - 300 ps	*412.50 ps*

Notes:

1. Low jitter reference clock requirement in Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3.
2. For PMCD based design, the input reference clock duty cycle requirement is 48/52 or tighter.
3. TDCD_CLK parameter in DS112, *Virtex-4 Family Overview Data Sheet*.
4. PMCD does not introduce jitter, so the output clock jitter is the same as the input reference clock jitter. The 0 ps note that the PMCD does not add jitter to the clock. (Table 48, Note 1 of the DS112, *Virtex-4 Family Overview Data Sheet*).
5. Data Jitter reflects input reference clock jitter and assumes worst case uncorrelated.
6. TX Total Jitter = 1 + 2 + 3 + 4 + 5. Assumes worst case linear addition of jitter components.
7. J_clkB parameter in Table 6.8d of OIF Standard.
8. J_datB parameter in Table 6.8d of OIF Standard

Regional Clock

Table H-11: **Sink/Dynamic - Regional Clock: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
Input Reference Clock (SYSCLK) DCD(2)	50 ps
FPGA Output Clock DCD(3)	100 ps
FPGA Output Clock Jitter(4)	0 ps
FPGA Output Data Jitter(5)	50 ps
Calculations	
TX Total Jitter at Reference Point A(6)	*250 ps*
TX Total Jitter as % of UI at Reference Point A	*0.20 UI*
TX Total Clock DCD	*6%*

Table H-12: **Sink/Dynamic - Regional Clock: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
OIF Clock Jitter(7)	0.13 UI
OIF Data Jitter(8)	0.44 UI
Calculations	
OIF Total Jitter	*0.57 UI*
OIF Total Jitter (ps)	*712.50 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Jitter at Point B - TX Total Jitter at Point A = 712.50 ps - 250 ps	*462.50 ps*

Notes:

1. Low jitter reference clock requirement in Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3.
2. For Regional Clock based design, the input reference clock duty cycle requirement is 48/52 or tighter.
3. TDCD_BUFIO parameter in DS112, *Virtex-4 Family Overview Data Sheet*.
4. BUFIO does not introduce jitter, so the output clock jitter is the same as the input reference clock jitter. The 0 ps note that the BUFIO does not add jitter to the clock.
5. Data Jitter reflects input reference clock jitter and assumes worst case uncorrelated.
6. TX Total Jitter = 1 + 2 + 3 + 4 + 5. Assumes worst case linear addition of jitter components.
7. J_clkB parameter in Table 6.8d of OIF Standard.
8. J_datB parameter in Table 6.8d of OIF Standard.

Virtex-5 Interface Timing

There are two clocking schemes available for Virtex-5 devices: global clocking with DCM and regional clocking. This section calculates the SPI-4.2 source interface timing budget for these clocking schemes when the Sink core is configured as either static and dynamic alignment.

Sink Core (Static Alignment)

Static alignment can be used for data rates less than or equal to 700 Mbps. Table 6.8a of the OIF spec provides the required timing values at Reference Point A (TX pins), and worst-case cumulative skew and jitter at Reference Point B (RX pins) of 790 ps.

The examples in this section calculate the total jitter and skew at both reference points A and B for a data rate of 700 Mbps. These examples are based on a Source Core design targeted to an XC5VLX330-FF1760 with -1 speed grade. Comparing the examples below, the regional clock design has the smallest duty cycle distortion and total jitter.

Global Clock with DCM

Table H-13: **Sink/Static - Global Clock with DCM: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
FPGA Output Clock DCD(2)	180 ps
FPGA Output Clock Jitter(3)	120 ps
FPGA Output Data Jitter(4)	120 ps
FPGA Global Clock Tree Skew(5)	18 ps
FPGA Package Skew(6)	40 ps
Calculations	
TX Total Skew + Jitter at Reference Point A(7)	*528 ps*
TX Total Skew + Jitter as % of UI at Reference Point A	*0.37 UI*
TX Total Clock DCD	*6%*

Table H-14: **Sink/Static - Global Clock with DCM: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Calculations	
OIF Worst-case Skew + Jitter(8)	*790 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Skew and Jitter at Point B - TX Total Jitter at Point A = 790 ps - 528 ps	*262 ps*

Notes:

1. Low jitter reference clock requirement in Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3.
2. TDUTY_CYC_DLL parameter in DS100, *Virtex-5 Family Overview Data Sheet* (-1 speed grade).
3. TPERJITT_0 parameter in DS100, *Virtex-5 Family Overview Data Sheet*.
4. Data Jitter reflects clock jitter and assumes worst case uncorrelated.
5. The FPGA Global Clock Tree Skew is extracted from PAR report (.par File) after the design is placed and routed. In the "Generating Clock Report" section of the PAR report, a table lists the net skew numbers for the clock nets in the design. The clock net used for this calculation is clock net (*sysclk0_gp) that drives the output OSERDES and

ODDR of the SPI-4.2 interface output signals: TDat[15:0], TCtl, and TDClk. For best result, all the listed SPI-4.2 interface output should be placed in the same bank.

6. The FPGA Package Skew is calculated using these steps. First run the command partgen -v xc5vlx330ff1760 (use the correct device/package in your design) to generate a file that contains the trace lengths for all the pins in the device/package combination. Use this file to find the trace lengths for all the pins assigned to SPI-4.2 interface output signals: TDat[15:0], TCtl, TDClk. The difference between the largest trace length and smallest trace length is the package skew. Then multiply this package skew number with 150 ps per inch (industry standard propagation delay) to get the final FPGA Package Skew number is ps.
7. TX Total Skew + Jitter = 1 + 2 + 3 + 4 + 5 + 6. Assumes worst case linear addition of skew and jitter components.
8. G_max parameter in Table 6.8a of OIF Standard.

Regional Clock

Table H-15: **Sink/Static - Regional Clock: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
Input Reference Clock (SYSCLK) DCD(2)	58 ps
FPGA Output Clock DCD(3)	100 ps
FPGA Output Clock Jitter(4)	0 ps
FPGA Output Data Jitter(5)	50 ps
FPGA I/O Clock Tree Skew(6)	80 ps
FPGA Package Skew(7)	177 ps
Calculations	
TX Total Skew + Jitter at Reference Point A(8)	*515 ps*
TX Total Skew + Jitter as % of UI at Reference Point A	*0.36 UI*
TX Total Clock DCD	*6%*

Table H-16: **Sink/Static - Regional Clock: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Calculations	

Table H-16: **Sink/Static - Regional Clock: Reference Point B (RX pins)**

Variable	Measurement
OIF Worst-case Skew + Jitter(9)	*790 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Skew and Jitter at Point B - TX Total Jitter at Point A = 790 ps - 515 ps	*275 ps*

Notes:

1. Low jitter reference clock requirement in Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3.
2. For Regional Clock based design, the input reference clock duty cycle requirement is 48/52 or tighter.
3. TDCD_BUFIO parameter in DS100, *Virtex-5 Family Overview Data Sheet* (-1 speed grade).
4. BUFIO does not introduce jitter, so the output clock jitter is the same as input reference clock jitter. The 0 ps note that the BUFIO does not add jitter to the clock.
5. Data Jitter reflects input reference clock jitter and assumes worst case worst case uncorrelated.
6. TBUFIOSKEW parameter in DS100, *Virtex-5 Family Overview Data Sheet* (-1 speed grade).
7. TPKGSKEW parameter in DS100, *Virtex-5 Family Overview Data Sheet* (XC5VFX330-FF1760). This parameter is device and package dependent.
8. TX Total Skew + Jitter = 1 + 2 + 3 + 4 + 5 + 6 + 7. Assumes worst case linear addition of skew and jitter components.
9. G_max parameter in Table 6.8a of OIF Standard.

Sink Core (Dynamic Alignment)

Dynamic alignment must be used for data rates more than 700 Mbps. Table 6.8c of the OIF specification gives the required timing values at reference point A (TX pins), and Table 6.8d gives the total jitter allowed at reference point B (RX pins) which is O.57 UI (total jitter = clock jitter + data jitter). The Xilinx SPI-4.2 Sink Core works correctly with this total jitter at the RX pins.

The examples in this section calculate the total jitter at both reference points A and B for a data rate of 800 Mbps. These examples are based on a Source Core design targeted to an XC5VLX330-FF1760 with -1 speed grade. Comparing these examples, the regional clock design has the smallest duty cycle distortion and total jitter.

Global Clock with DCM

Table H-17: **Sink/Dynamic - Global Clock with DCM: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps

Table H-17: **Sink/Dynamic - Global Clock with DCM: Reference Point A (TX pins)**

Variable	Measurement
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
FPGA Output Clock DCD(2)	180 ps
FPGA Output Clock Jitter(3)	120 ps
FPGA Output Data Jitter(4)	120 ps
Calculations	
TX Total Jitter at Reference Point A(5)	*470 ps*
TX Total Jitter as % of UI at Reference Point A	*0.38 UI*
TX Total Clock DCD	*7%*

Table H-18: **Sink/Dynamic - Global Clock with DCM: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
OIF Clock Jitter(6)	0.13 UI
OIF Data Jitter(7)	0.44 UI
Calculations	
OIF Total Jitter	*0.57 UI*
OIF Total Jitter (ps)	*712.50 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Jitter at Point B - TX Total Jitter at Point A = 712.50 ps - 470 ps	*242.50 ps*

Notes:

1. Low jitter reference clock requirement in Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3.
2. TDUTY_CYC_DLL parameter in DS100, *Virtex-5 Family Overview Data Sheet* (for -1 speed grade).
3. TPERJITT_0 parameter in DS100, *Virtex-5 Family Overview Data Sheet*.
4. Data Jitter reflects clock jitter and assumes worst case uncorrelated.
5. TX Total Jitter = 1 + 2 + 3 + 4 . Assumes worst case linear addition of jitter components.
6. J_clkB parameter in Table 6.8d of OIF Standard.
7. J_datB parameter in Table 6.8d of OIF Standard.

Regional Clock

Table H-19: **Sink/Dynamic - Regional Clock: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
Input Reference Clock (SYSCLK) DCD(2)	50 ps
FPGA Output Clock DCD(3)	100 ps
FPGA Output Clock Jitter(4)	0 ps
FPGA Output Data Jitter(5)	50 ps
Calculations	
TX Total Jitter at Reference Point A(6)	*250 ps*
TX Total Jitter as % of UI at Reference Point A	*0.20 UI*
TX Total Clock DCD	*6%*

Table H-20: **Sink/Dynamic - Regional Clock: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
OIF Clock Jitter(7)	0.13 UI
OIF Data Jitter(8)	0.44 UI
Calculations	
OIF Total Jitter	*0.57 UI*
OIF Total Jitter (ps)	*712.50 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Jitter at Point B - TX Total Jitter at Point A = 712.50 ps - 250 ps	*462.50 ps*

Notes:

1. Low jitter reference clock requirement in Source Clocking Interface for Virtex-5 and Virtex-4 in Chapter 3.
2. For Regional Clock based design, the input reference clock duty cycle requirement is 48/52 or tighter.
3. TDCD_BUFIO parameter in DS100, *Virtex-5 Family Overview Data Sheet*.
4. BUFIO does not introduce jitter, so the output clock jitter is the same as the input reference clock jitter. The 0 ps note that the BUFIO does not add jitter to the clock.

5. Data Jitter reflects input reference clock jitter and assumes worst case uncorrelated.
6. TX Total Jitter = 1 + 2 + 3 + 4 + 5. Assumes worst case linear addition of jitter components.
7. J_clkB parameter in Table 6.8d of OIF Standard.
8. J_datB parameter in Table 6.8d of OIF Standard.

Virtex-6 Interface Timing

There are two clocking schemes available for Virtex-6 devices: global clocking with MMCM and regional clocking. This section calculates the SPI-4.2 source interface timing budget for these clocking schemes when the Sink core is configured as either static and dynamic alignment.

Sink Core (Static Alignment)

Static alignment can be used for data rates less than or equal to 700 Mbps. Table 6.8a of the OIF specification provides the required timing values at reference point A (TX pins), and worst-case cumulative skew and jitter at reference point B (RX pins) of 790 ps. For Virtex-6 devices, only regional clocking is supported in this configuration.

These examples calculate the total jitter and skew at both reference points A and B for a data rate of 700Mpbs. These examples are based on a source core design targeted to an XC6VHX380T-FF1924 with -1 speed grade.

Regional Clock

Table H-21: **Sink/Static - Regional Clock: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
Input Reference Clock (SYSCLK) DCD(2)	58 ps
FPGA Output Clock DCD(3)	80 ps
FPGA Output Clock Jitter(4)	0 ps
FPGA Output Data Jitter(5)	50 ps
FPGA I/O Clock Tree Skew(6)	30 ps
FPGA Package Skew(7)	220 ps
Calculations	
TX Total Skew + Jitter at Reference Point A(8)	*478 ps*
TX Total Skew + Jitter as % of UI at Reference Point A	*0.33 UI*
TX Total Clock DCD	*5.6%*

Table H-22: **Sink/Static - Regional Clock: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	350 MHz
UI (ps)	1430 ps
Calculations	
OIF Worst-case Skew + Jitter(9)	*790 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Skew and Jitter at Point B - TX Total Jitter at Point A = 790 ps - 478 ps	*312 ps*

Notes:

1. Low jitter reference clock requirement in Source Clock Interface for Virtex-6 Devices in Chapter 3.
2. For Regional Clock based design, the input reference clock duty cycle requirement is 48/52 or tighter.
3. TDCD_BUFIO parameter in DS150, *Virtex-6 Family Overview Data Sheet* (-1 speed grade).
4. BUFIO does not introduce jitter, so the output clock jitter is the same as input reference clock jitter. The 0 ps note that the BUFIO does not add jitter to the clock.
5. Data Jitter reflects input reference clock jitter and assumes worst case uncorrelated.
6. TBUFIOSKEW parameter in DS150, *Virtex-6 Family Overview Data Sheet* (-1 speed grade).
7. TPKGSKEW parameter in DS150, *Virtex-6 Family Overview Data Sheet* (XC6VHX380T-FF1924).
8. This parameter is device and package dependent.
9. TX Total Skew + Jitter = 1 + 2 + 3 + 4 + 5 + 6 + 7. Assumes worst case linear addition of skew and jitter components.
10. G_max parameter in Table 6.8a of OIF Standard.

Sink Core (Dynamic Alignment)

Dynamic alignment must be used for data rates more than 700 Mbps. Table 6.8c of the OIF specification gives the required timing values at reference point A (TX pins), and Table 6.8d gives the total jitter allowed at reference point B (RX pins) which is O.57 UI (total jitter = clock jitter + data jitter). The Xilinx SPI-4.2 Sink core works correctly with this total jitter at the RX pins.

These examples calculate the total jitter at both reference points A and B for a data rate of 800 Mbps. These examples are based on a Source core design targeted to an XC5VLX330-FF1760 with -1 speed grade. Comparing the examples below, the regional clock design has the smallest duty cycle distortion and total jitter.

Global Clock with MMCM

Table H-23: **Sink/Dynamic - Global Clock with MMCM: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
FPGA Output Clock DCD(2)	200 ps
FPGA Output Clock Jitter(3)	90 ps
FPGA Output Data Jitter(4)	90 ps
Calculations	
TX Total Jitter at Reference Point A(5)	*380 ps*
TX Total Jitter as % of UI at Reference Point A	*0.30 UI*
TX Total Clock DCD	*8%*

Table H-24: **Sink/Dynamic - Global Clock with MMCM: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
OIF Clock Jitter(6)	0.13 UI
OIF Data Jitter(7)	0.44 UI
Calculations	
OIF Total Jitter	*0.57 UI*
OIF Total Jitter (ps)	*712.50 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Jitter at Point B - TX Total Jitter at Point A = 712.50 ps - 380 ps	*332.50 ps*

Notes:

1. Low jitter reference clock requirement in Source Clock Interface for Virtex-6 Devices in Chapter 3.
2. TOUTDUTY parameter in DS150, *Virtex-6 Family Overview Data Sheet* (-1 speed grade).
3. TOUTJITTER parameter. Calculated using Clock Wizard (Table 63, Note 3 of the DS150, *Virtex-6 Family Overview Data Sheet*).
4. Data Jitter reflects clock jitter and assumes worst case uncorrelated.

5. TX Total Jitter = 2 + 3 + 4 . Assumes worst case linear addition of jitter components. The input reference clock jitter is not included in the calculation because it is filtered out by MMCM.
6. J_clkB parameter in Table 6.8d of OIF Standard.
7. J_datB parameter in Table 6.8d of OIF Standard.

Regional Clock

Table H-25: **Sink/Dynamic - Regional Clock: Reference Point A (TX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
Input Reference Clock (SYSCLK) Jitter(1)	50 ps
Input Reference Clock (SYSCLK) DCD(2)	50 ps
FPGA Output Clock DCD(3)	80 ps
FPGA Output Clock Jitter(4)	0 ps
FPGA Output Data Jitter(5)	50 ps
Calculations	
TX Total Jitter at Reference Point A(6)	*230 ps*
TX Total Jitter as % of UI at Reference Point A	*0.18 UI*
TX Total Clock DCD	*5.2%*

Table H-26: **Sink/Dynamic - Regional Clock: Reference Point B (RX pins)**

Variable	Measurement
Output Clock Frequency	400 MHz
UI (ps)	1250 ps
OIF Clock Jitter(7)	0.13 UI
OIF Data Jitter(8)	0.44 UI
Calculations	
OIF Total Jitter	*0.57 UI*
OIF Total Jitter (ps)	*712.50 ps*
Remaining System Margin (for PCB design, System Noise, and others) = OIF Total Jitter at Point B - TX Total Jitter at Point A = 712.50 ps - 230 ps	*482.50 ps*

Notes:

1. Low jitter reference clock requirement in Source Clock Interface for Virtex-6 Devices in Chapter 3.

2. For Regional Clock based design, the input reference clock duty cycle requirement is 48/52 or tighter.
3. TDCD_BUFIO parameter in DS150, *Virtex-6 Family Overview Data Sheet*.
4. BUFIO does not introduce jitter, so the output clock jitter is the same as the input reference clock jitter. The 0 ps note that the BUFIO does not add jitter to the clock.
5. Data Jitter reflects input reference clock jitter and assumes worst case uncorrelated.
6. TX Total Jitter = 1 + 2 + 3 + 4 + 5. Assumes worst case linear addition of jitter components.
7. J_clkB parameter in Table 6.8d of OIF Standard.
8. J_datB parameter in Table 6.8d of OIF Standard.