LogiCORE IP SPI-4.2 v12.2

User Guide

UG784 July 25, 2012

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/01/2011	1.0	Initial Xilinx release.
06/22/2011	2.0	Updated core to v11.2 and ISE Design Suite to 13.2.
10/19/2011	2.5	Added information about I/O placement for 7 series devices in Sink Core Optional Constraints, page 136 and Source Core Optional Constraints, page 142.
07/25/2012	3.0	Updated core to v12.2. Removed support for ISE Design Suite. Added support for Vivado Design Suite v2012.2.

Table of Contents

Chapter 4: Customizing and Generating the Core

Chapter 5: Designing with the Core

Chapter 6: Constraining the Core

Chapter 7: Special Design Considerations

Appendix H: SPI-4.2 Core Verification

Appendix I: SPI-4.2 Source Interface Timing Budget

Appendix J: Migrating to SPI-4.2 v11.2 and Later

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Licensing the Core

Chapter 3: Core Overview

Chapter 4: Customizing and Generating the Core

Chapter 5: Designing with the Core

Chapter 6: Constraining the Core

Chapter 7: Special Design Considerations

Chapter 8: Example Design

About This Guide

The *LogiCORE IP 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, Customizing and Generating the Core describes how to generate the SPI-4.2 core using the Xilinx Vivado™ Design Suite.
- 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, Example Design describes the files and directories created by the Vivado Design Suite. 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 Vivado Design Suite.
- 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, Debugging contains details about debugging the core.

- Appendix H, SPI-4.2 Core Verification describes the software verification of the SPI-4.2 core.
- Appendix I, SPI-4.2 Source Interface Timing Budget contains timing examples.
- Appendix J, Migrating to SPI-4.2 v11.2 and Later provides details for migrating to the most recent version of the SPI-4.2 core.

Additional Resources

To find additional documentation, see the Xilinx website at:

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:

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
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*`option_name`*] *`design_name`*

Convention	Meaning or Use	Example
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 www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Introduction

The Xilinx LogiCORE ™ IP SPI-4.2 (PL4) 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

For operating system requirements, see the Xilinx Design Tools: Release Notes Guide.

About the Core

The SPI-4.2 core is 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 (XDC) 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

Chapter 2

Licensing the Core

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite. For full access to all core functions in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the SPI-4.2 product page.

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

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®-7, Kintex®-7, or Virtex-7 device. The user logic reads data from the Sink core and writes data into the Source core. An AXI4-Stream FIFO interface is provided for this data access and the FIFO interface facilitates integration within a system. An AXI4-Lite control interface is used to configure the Sink and Source cores in circuit and monitor a suite of status registers.

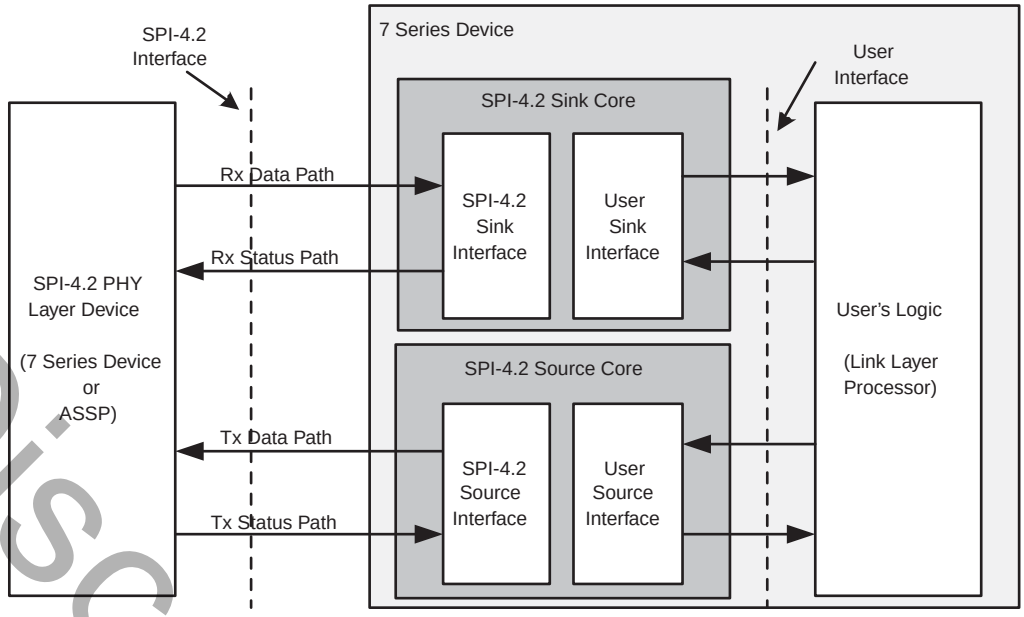

Figure 3-1: **SPI-4.2 Core in a Typical Link Layer Application**

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 an AXI4-stream 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-7 and Kintex-7 FPGAs I/O (embedded SERDES, delay chain, and bitslip module) that runs at data rates exceeding 1 Gbps.
- Provides an AXI4-Stream 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 through an AXI4-Stream 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
- External clocking module to facilitate multiple core implementations
- Enables two modes to access SPI-4.2 flow control data: addressable via AXI4-Lite Control Interface and transparent via AXI4-Stream Status Interface
- Provides an AXI4-Stream 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 four functional modules:

- Sink Data FIFO
- Sink Data Receive
- Sink AXI4-Lite Memory Space
- Sink Status Transmit

The Sink core has the following interfaces:

- Sink SPI-4.2 Interface
- Sink User Interface
  - Sink Control and Status Interface
  - Sink AXI4-Stream FIFO Interface
  - Sink AXI4-Lite Control Interface
  - Sink User 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.

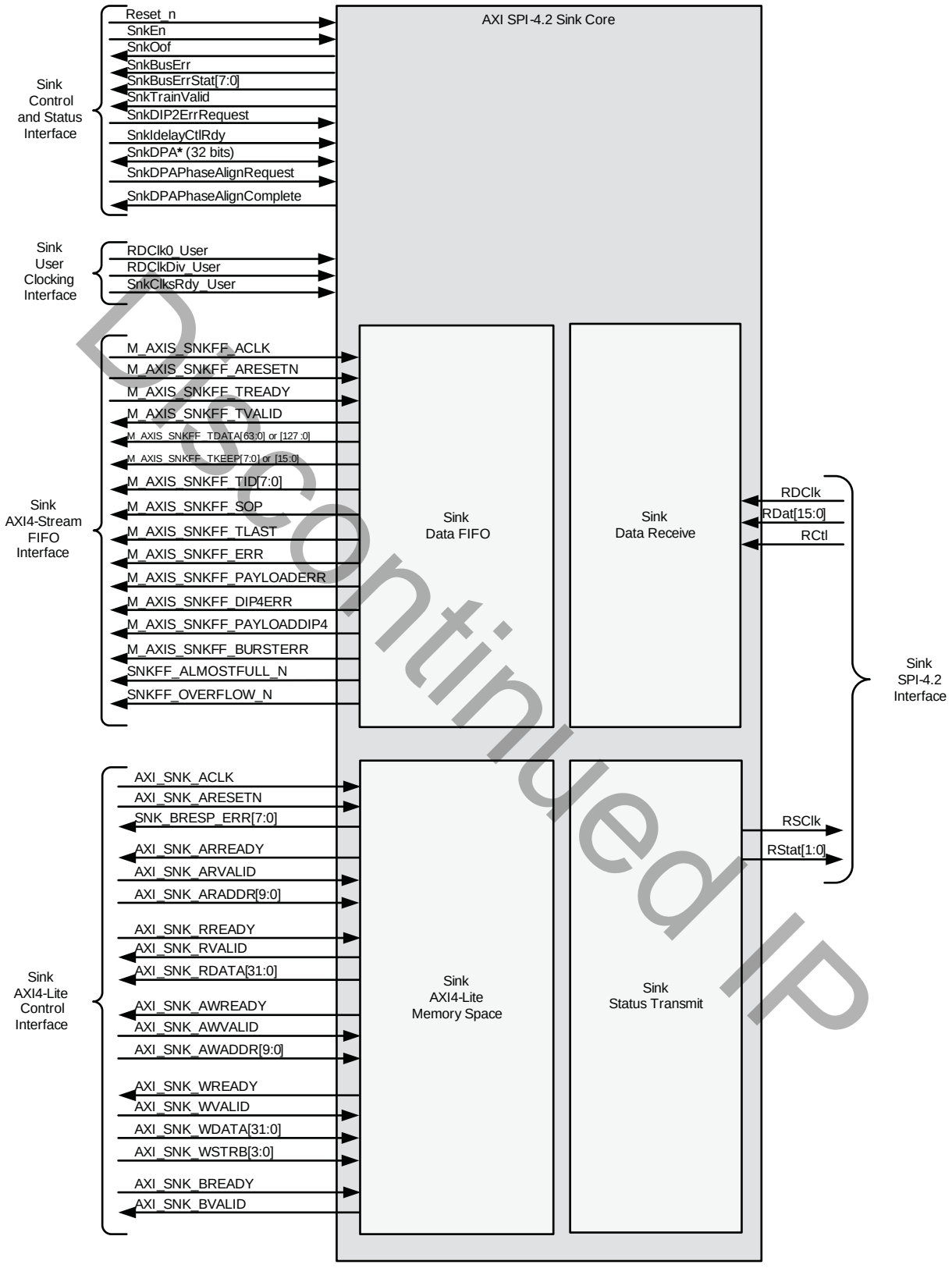

Figure 3-2: **Sink Core Block Diagram**

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 Kintex-7 FPGAs
- Above 500 MHz LVDS DDR I/O (1+ Gbps) in Virtex-7 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 words. 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 (only 8-bits are used, 2-bit status for 4 channels)status 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.
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.
RSClk_P RSClk_N	Output	n/a	**SPI-4.2 Receive Status Clock** (LVDS/LVTTL). 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 parameter RSClkDiv.
RStat_P[1:0] RStat_N[1:0]	Output	RSClk	**SPI-4.2 Receive FIFO Status** (LVDS/LVTTL). FlFO Status Channel flow control interface.

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 up to:

- 312.5 MHz in Virtex-7 FPGAs
- 250 MHz in Kintex-7 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 four smaller interfaces. Each of the four 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.
- **AXI4-Stream FIFO Interface**. The signals of this interface allow access to data received on the SPI-4.2 Interface.
- **AXI4-Lite Control Interface**. The signals of this interface configure the calendar sequence, static configuration parameters, and send flow control information on the SPI-4.2 Interface.
- **User Clocking Interface**. These input signals drive the clocks in the core.

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 are defined in Table 3-2.

Table 3-2: **Sink Control and Status Interface 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. However, the reset signal does not affect the Sink or Source AXI4-Lite Control Interface (the logic and memory space). 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.
SnkEn	Input	RDClkDiv_User	**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.

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

Name	Direction	Clock Domain	Description
SnkIdelayCtlRdy	Input	RDClkDiv_User	**IDELAYCTRL Ready.** Active high signal that indicates when the IDELAY modules are calibrated. 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 connects the SnkIdelayCtlRdy signal.
SnkOof	Output	RDClkDiv_User	**Sink Out-of-Frame**. Active high signal indicating 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	RDClkDiv_User	**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.
SnkBusErrStat[7:0]	Output	RDClkDiv_User	**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 the *SPI-4.2 User Guide* for more information.
SnkTrainValid	Output	RDClkDiv_User	**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 cycles or 5 RDClkDiv_User clock cycles), if the training pattern received is successfully decoded.
SnkDIP2ErrRequest	Input	RDClkDiv_User	**Sink DIP2 Error Request.** This is an active high signal that requests an incorrect DIP-2 to be sent out of the RStat bus. When this signal is asserted, the Sink Status FIFO responds by inverting the next DIP2 value that it transmits.

Sink AXI4-Stream FIFO Interface

The Sink AXI4-Stream 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 AXI4-Stream FIFO Interface Signals**

Name	Direction	Description
M_AXIS_SNKFF_ACLK	Input	**Sink FIFO Clock**. All Sink FIFO Interface signals are synchronous to the rising edge of this clock.
M_AXIS_SNKFF_ARESETN	Input	**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.The reset signal can be asserted asynchronously, but deassertion must be synchronous after the rising edge of ACLK. See the *AMBA® AXI4-Stream Protocol* specification for more information.
M_AXIS_SNKFF_TREADY	Input	**Sink FIFO User Ready Handshake**. Indicates the user is ready to accept streaming data. When both TREADY and TVALID are asserted on the rising edge of M_AXIS_SNKFF_ACLK, one data beat is transferred.
M_AXIS_SNKFF_TVALID	Output	**Sink FIFO Data Valid Handshake.** Indicates the core is presenting valid streaming data. When both TREADY and TVALID are asserted on the rising edge of M_AXIS_SNKFF_ACLK, one data beat is transferred. When asserted (active high), this signal indicates that the information on M_AXIS_SNKFF_TDATA, M_AXIS_SNKFF_TID, M_AXIS_SNKFF_SOP, M_AXIS_SNKFF_TLAST, M_AXIS_SNKFF_BURSTERR, M_AXIS_SNKFF_TKEEP, M_AXIS_SNKFF_ERR, M_AXIS_SNKFF_DIP4ERR, M_AXIS_SNKFF_PAYLOADERR, and M_AXIS_SNKFF_PAYLOADDIP4 are valid.
M_AXIS_SNKFF_TID[7:0]	Output	**Sink FIFO Channel Address**. Channel number associated with the data on M_AXIS_SNKFF_TDATA.
M_AXIS_SNKFF_TDATA[63:0] or M_AXIS_SNKFF_TDATA[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.
M_AXIS_SNKFF_TKEEP[7:0] or M_AXIS_SNKFF_TKEEP[15:0]	Output	**Sink FIFO Data Strobe**. This signal indicates which bytes on the M_AXIS_SNKFF_TDATA bus are valid when the M_AXIS_SNKFF_TLAST signal is asserted (i.e. byte enable). M_AXIS_SNKFF_TKEEP[7:0] is used with a 64-bit interface. M_AXIS_SNKFF_TKEEP[15:0] is used with a 128-bit interface.
M_AXIS_SNKFF_SOP	Output	**Sink FIFO Start of Packet**. When asserted (active high), this signal indicates the start of a packet is being read out of the Sink FIFO.
M_AXIS_SNKFF_TLAST	Output	**Sink FIFO End of Packet (EOP)**. When asserted (active high), this signal indicates that the end of a packet is being read out of the Sink FIFO.
M_AXIS_SNKFF_ERR	Output	**Sink FIFO Error**. When asserted (active high), this signal indicates that the current packet is terminated with an EOP abort condition. This signal is only asserted when M_AXIS_SNKFF_TLAST is asserted.

Table 3-3: **Sink AXI4-Stream FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
M_AXIS_SNKFF_DIP4ERR	Output	**Sink FIFO DIP-4 Error.** When asserted (active high), this signal indicates 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.
M_AXIS_SNKFF_PAYLOADDIP4	Output	**Sink FIFO Payload DIP4 Error.** When asserted (active high), this signal indicates 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.
M_AXIS_SNKFF_BURSTERR	Output	**Sink FIFO Burst Error.** When asserted (active high), this signal indicates that the Sink core has received data that was terminated on a non-credit boundary without an EOP. M_AXIS_SNKFF_BURSTERR 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 M_AXIS_SNKFF_TLAST or M_AXIS_SNKFF_ERR.
M_AXIS_SNKFF_PAYLOADERR	Output	**Sink FIFO Payload Error.** When asserted (active high), this signal indicates 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.
SNKFF_ALMOSTFULL_N	Output	**Sink FIFO Almost Full**. When asserted (active low), this signal indicates that the Sink core is approaching full (as defined by the parameter SnkAFThresAssert), and that immediate action should be taken to prevent overflow.
SNKFF_OVERFLOW_N	Output	**Sink FIFO Overflow.** When asserted (active low), this signal indicates that the Sink core has overflowed and is in an error condition. Data will be lost if SNKFF_OVERFLOW_N is asserted, since no data is written into the FIFO when the overflow signal is asserted.

Sink AXI4-Lite Control Interface (Calendar, Configuration and Status)

The Sink AXI4-Lite Control interface is used to program the calendar memory, status memory, and static configuration memory. This memory space is defined in Figure 3-3.

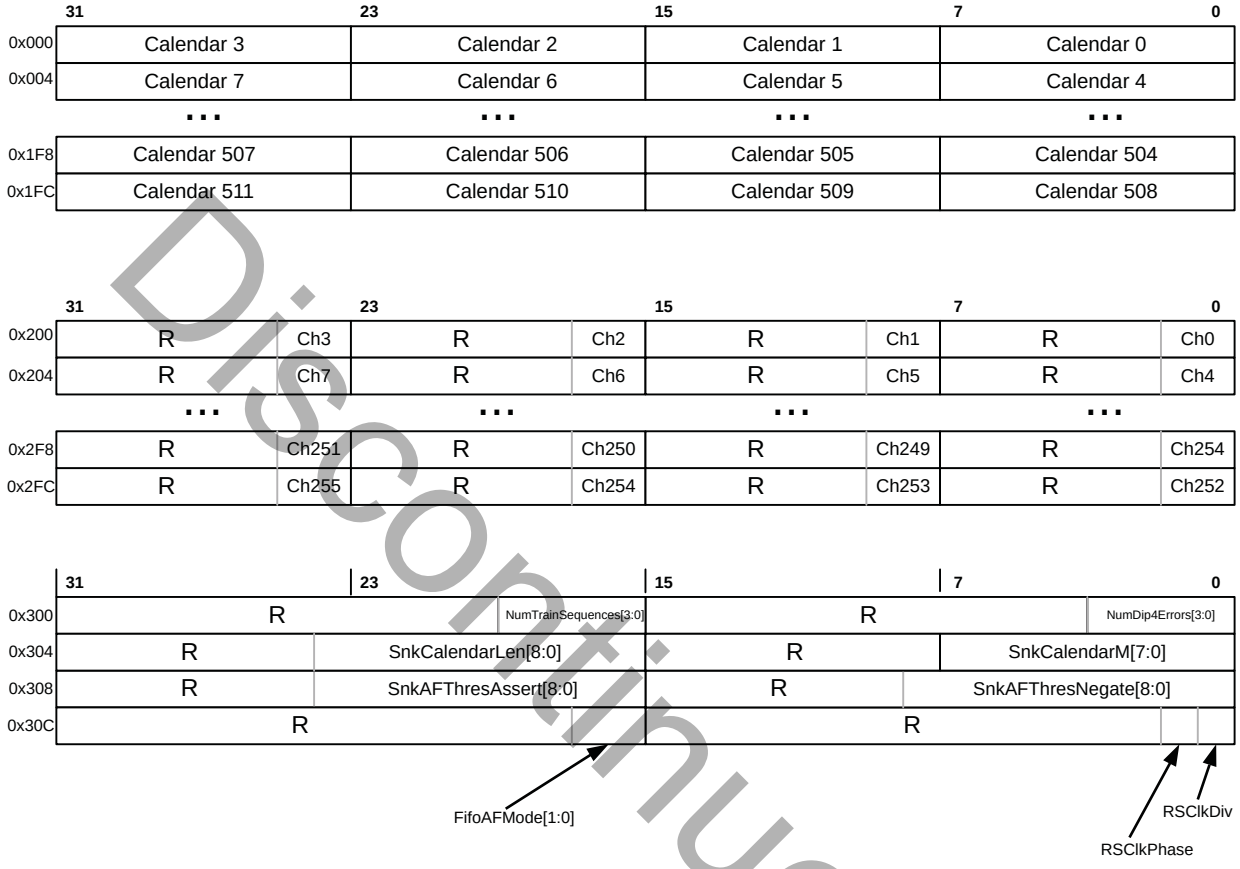

Figure 3-3: **Sink AXI4-Lite Memory Space**

The calendar determines the status channel order and frequency. Through this interface, the user can program the calendar buffer to determine the order and frequency with which the channel status is sent on the SPI-4.2 interface. The Status memory enables the user to send flow control data to the transmitting device. Flow control may be automatically or manually implemented. The static configuration memory enables customization of the core based on individual system requirements. These settings are statically driven inside

the core by writing registers through the Control interface. Table 3-4 describes the Control interface signals, and Table 3-5 defines the static configuration parameters.

Table 3-4: **Sink AXI4-Lite Control Interface Signals**

Name	Direction	Clock Domain	Description
AXI_SNK_ACLK	Input	N/A	**AXI4-Lite Clock**. All Sink AXI4-Lite signals are synchronous to the rising edge of this clock
AXI_SNK_ARESETN	Input	AXI_SNK_ACLK	**AXI4-Lite Reset.** Active-low signal that enables the user to reset the AXI4-Lite interface and all associated logic and memories to chosen GUI settings. The reset signal can be asserted asynchronously, but deassertion must be synchronous after the rising edge of ACLK. See the *AMBA AXI4 Protocol* specification for more information.
Read Address Channel			
AXI_SNK_ARREADY	Output	AXI_SNK_ACLK	**AXI4-Lite Read Address Core Handshake**. Indicates the core is ready to accept a read address. When both ARREADY and ARVALID are asserted on a clock cycle, a one-word read request occurs for the address provided on ARADDR and the core fetches the contents of that address.
AXI_SNK_ARVALID	Input	AXI_SNK_ACLK	**AXI4-Lite Read Address User Handshake**. Indicates the user is presenting a valid read address. When both ARREADY and ARVALID are asserted on a clock cycle, a one-word read request occurs for the address provided on ARADDR and the core fetches the contents of that address
AXI_SNK_ARADDR[9:0]	Input	AXI_SNK_ACLK	**AXI4-Lite Read Address.** Address to be read by the user application.
Read Data Channel			
AXI_SNK_RREADY	Input	AXI_SNK_ACLK	**AXI4-Lite Read Data User Handshake**. Indicates the user is ready to accept read data. When both RREADY and RVALID are asserted on a clock cycle, one data beat is read out of the core. This handshake completes the transaction begun by the ARREADY/ARVALID handshake.
AXI_SNK_RVALID	Output	AXI_SNK_ACLK	**AXI4-Lite Read Data Core Handshake**. Indicates the core is presenting valid read data. When both RREADY and RVALID are asserted on a clock cycle, one data beat is read out of the core. This handshake completes the transaction begun by the ARREADY/ARVALID handshake.
AXI_SNK_RDATA[31:0]	Output	AXI_SNK_ACLK	**AXI4-Lite Read Data.** Data read from address location provided on ARADDR is presented here. Bit 0 is the LSB.
Write Address Channel			
AXI_SNK_AWREADY	Output	AXI_SNK_ACLK	**AXI4-Lite Write Address Core Handshake.** Indicates the core is ready to accept a write address. When both AWREADY and AWVALID are asserted on a clock cycle, one write address beat is accepted by the core. Once both the WREADY/WVALID and AWREADY/AWVALID handshakes occur independently, one write transaction is complete.

Table 3-4: **Sink AXI4-Lite Control Interface Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
AXI_SNK_AWVALID	Input	AXI_SNK_ACLK	**AXI4-Lite Write Address User Handshake.** Indicates the user is presenting a valid write address. When both AWREADY and AWVALID are asserted on a clock cycle, one write address beat is accepted by the core. Once both the WREADY/WVALID and AWREADY/AWVALID handshakes occur independently, one write transaction is complete
AXI_SNK_AWADDR[9:0]	Input	AXI_SNK_ACLK	**AXI4-Lite Write Address**. Address to be written by the user application.
Write Data Channel			
AXI_SNK_WREADY	Output	AXI_SNK_ACLK	**AXI4-Lite Write Data Core Handshake**. Indicates the core is ready to accept write data and strobe. When both WREADY and WVALID are asserted on a clock cycle, one write address beat is accepted by the core. Once both the WREADY/WVALID and AWREADY/AWVALID handshakes occur independently, one write transaction is complete.
AXI_SNK_WVALID	Input	AXI_SNK_ACLK	**AXI4-Lite Write Data User Handshake.** Indicates the user is presenting valid write data and strobe. When both WREADY and WVALID are asserted on a clock cycle, one write address beat is accepted by the core. Once both the WREADY/WVALID and AWREADY/AWVALID handshakes occur independently, one write transaction is complete.
AXI_SNK_WDATA[31:0]	Input	AXI_SNK_ACLK	**AXI4-Lite Write Data**. Data written into the core. Bit 0 is the LSB.
AXI_SNK_WSTRB[3:0]	Input	AXI_SNK_ACLK	**AXI4-Lite Write Strobe**. Byte enable associated with AXI_SNK_WDATA. Bit 0 enables WDATA[7:0], bit 1 enables WDATA[15:8], bit 2 enables WDATA[23:16], and bit 3 enables WDATA[32:24].
Write Response Channel			
AXI_SNK_BREADY	Input	AXI_SNK_ACLK	**AXI4-Lite Write Response User Handshake.** Indicates the user is ready to accept a write response. When both BREADY and BVALID are asserted on a rising edge of AXI_SNK_ACLK, one write response transaction is read out of the core.
AXI_SNK_BVALID	Output	AXI_SNK_ACLK	**AXI4-Lite Write Response Core Handshake.** Indicates the core is presenting a valid write response. When both BREADY and BVALID are asserted on a rising edge of AXI_SNK_ACLK, one write response transaction is read out of the core. Each write transaction will generate a write response, regardless of whether the write succeeded. The Error Bus (below) provides further signaling in regards to errors that occur during write operations.

Table 3-4: **Sink AXI4-Lite Control Interface Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
Sideband Error Bus			
SNK_BRESP_ERR[7:0]	Output	AXI_SNK_ACLK	**AXI4-Lite Error Bus.** Each bit of this bus corresponds to a specific AXI4-Lite Error condition. The error conditions detected are reported as follows: SNK_BRESP_ERR [0]: Write Response counter is full. No further writes or reads will be processed until at least one Write Response is read from the core using BREADY/BVALID. SNK_BRESP_ERR[1]: Write transaction to the Configuration space failed. SNK_BRESP_ERR [2]: Write transaction to the Calendar space has failed. SNK_BRESP_ERR[3]:Read/Write has missed a valid address SNK_BRESP_ERR [4:7]: Reserved bits (tied low).

Sink Static Configuration Parameters

Sink Static Configuration Parameters are memory registers in the core that are written to through the AXI4-Lite Control Interface. The default values of these memory registers are determined through the GUI. The user can customize these registers using the GUI.

If these parameters need to be changed in-circuit, they should be changed when the core is not in operation (disabled and in reset state).

The following steps are recommended when changing static configuration parameters:

1. Disable the sink core (`SnkEn` signal).
2. Assert core reset (`Reset_n` = 0).
3. Change the desired static configuration parameters by using the AXI4-Lite Control interface. See Sink Static Configuration Parameters in Chapter 5.
4. Deassert reset (`Reset_n`=1).
5. Wait at least 10 clock cycles of `RDClkDiv_User` for the sink static configuration parameters 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.

The Sink Static Configuration Parameters are defined in Table 3-5.

Table 3-5: **Sink Static Configuration Parameter Definition**

Name	Range	Description
NumDip4Errors[3:0]	1-15	**Number of DIP-4 Errors**. The Sink interface goes out-of-frame (assert SnkOof) and stops accepting data from the SPI-4.2 bus after receiving NumDip4Errors consecutive DIP-4 errors.
NumTrainSequences[3:0]	1-15	**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.

Table 3-5: **Sink Static Configuration Parameter Definition *(Cont'd)***

Name	Range	Description
SnkCalendar_M[7:0]	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 programmed by the AXI4-Lite interface, 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]	0-511 (effective range 1-512)	**Sink Calendar Length**. The SnkCalendar_Len parameter sets the length of the calendar sequence. The core implements this parameter as a static register programmed by the AXI4-Lite interface, 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.
SnkAFThresNegate[8:0]	SnkAFThresAssert to 508	**Sink Almost Full Threshold Negate**. Defines the minimum number of empty FIFO locations that exist when SNKFF_ALMOSTFULL_N is deasserted. Note that the negate threshold must be greater or equal to the assert threshold (SnkAFThresAssert). When SNKFF_ALMOSTFULL_N is deasserted, the core stops sending flow control (deasserts SNKFF_ALMOSTFULL_N) and resumes transmission of valid FIFO status levels. This indicates to the transmitting device that additional data can be sent.
SnkAFThresAssert[8:0]	1-508	**Sink Almost Full Threshold Assert**. Defines the minimum number of empty FIFO locations that exist when SNKFF_ALMOSTFULL_N is asserted. The assert threshold must be less than or equal to the negate threshold (SnkAFThresNegate). When SNKFF_ALMOSTFULL_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.
RSClkDiv	n/a	**Sink Status Clock Divide**. 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

Table 3-5: **Sink Static Configuration Parameter Definition *(Cont'd)***

Name	Range	Description
RSClkPhase	n/a	**Sink Status Clock Phase**. 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]	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 SNKFF_ALMOSTFULL_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 user's status information (*i.e.*, continues in normal operation). In this case, the user should take immediate action to prevent overflow and loss of data.

Sink User Clocking Interface

For the SPI-4.2 AXI core (version 11.1 and onwards), the Sink core's clocking module is not embedded in the core. The clock interface to the core consists of input clock ports required by the internal logic. A list of the Sink clock ports and descriptions is given in Table 3-6.

Table 3-6: **Sink User Clock Signals**

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. This clock is also used to generate RSClk.	110 MHz	MMCM or BUFR Minimum	250+ MHz
SnkClksRdy_User	Input (User Interface)	**Sink Clocks Ready User:** This signal indicates to the sink core that the clocks are stable and ready to use.	NA	NA	NA

Source Core Interfaces

The Source core has five functional modules:

- Source Data FIFO
- Source Data Transmit
- Source Status Logic
- Source AXI-4 Lite Memory Space
- Source Status Receive

The Source core has the following interfaces:

- Source SPI-4.2 Interface
- Source User Interface
  - Source Control and Status Interface
  - Source AXI4-Stream FIFO Interface
  - Source AXI4-Lite Control Interface
  - Source AXI4-Stream Status Interface
  - Source User Clocking Interface

The functional modules and signals which comprise the different interfaces are shown in Figure 3-4 and defined in tables in the following sections.

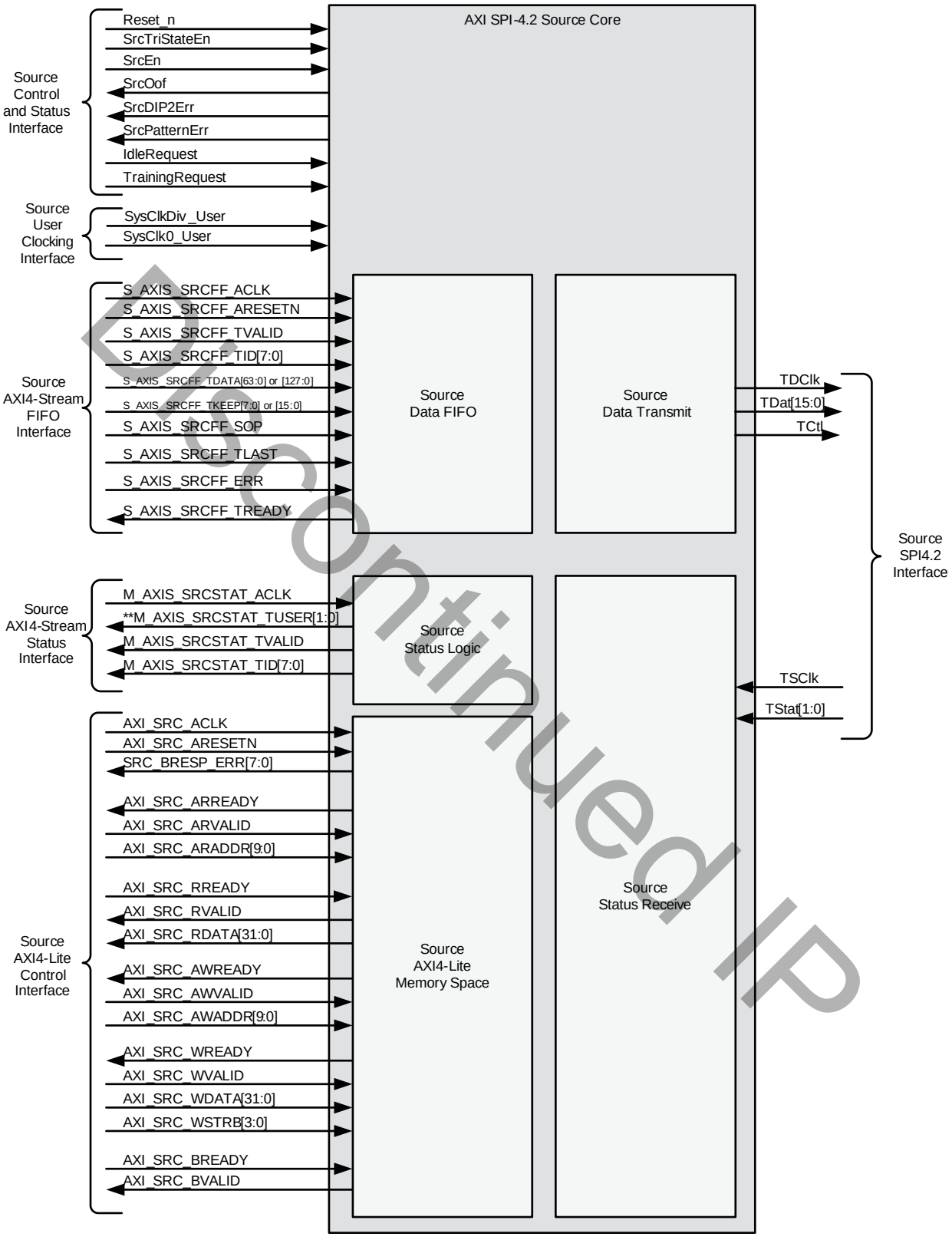

Figure 3-4: **Source Core Block Diagram and I/O Interface Signals**

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:

- 500 MHz LVDS DDR I/O (1 Gbps) in Kintex-7 FPGAs
- Above 500 MHz LVDS DDR I/O (1+ Gbps) in Virtex-7 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 or 128-bit user FIFO 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: AXI4-Stream status interface (transparent mode) or AXI4-Lite Control interface (addressable mode).

The signals on the SPI-4.2 Source Interface are defined in Table 3-7.

Table 3-7: **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_P TSClk_N	Input	n/a	**SPI-4.2 Transmit Status Clock** (LVDS/LVTTL). Source synchronous clock that is received by the Source core with TStat at 1/4 rate (or 1/8 rate) of TDClk.
TStat_P[1:0] TStat_N[1:0]	Input	TSClk	**SPI-4.2 Transmit FIFO Status** (LVDS/LVTTL). FIFO-Status-Channel flow control interface.

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:

- 312.5 MHz in Virtex-7 FPGAs
- 250 MHz in Kintex-7 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**. The signals of this interface apply to the operation of the Sink core.
- **AXI4-Stream FIFO Interface**. The signals of this interface determine the data to be sent on the SPI-4.2 Interface.
- **AXI4-Lite Control Interface**. The signals of this interface configure the calendar sequence, static configuration parameters, and receive flow control information on the SPI-4.2 Interface.
- **AXI4-Stream Status Interface**. The signals of this interface present the latest flow control information received on the SPI-4.2 Interface.
- **User Clocking Interface**. These input signals drive the clocks in the core.

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 3-8 defines the signals of this interface.

Table 3-8: **Source Control and Status Signals**

Name	Direction	Clock Domain	Description
Reset_n	Input	n/a	**Reset_n**. This active low, asynchronous control signal enables the user to restart the entire Source core, and causes the core to go out-of-frame. While Reset_n is asserted, the Source core transmits idles cycles on TDat. However, the Reset signal does not affect the sink or source AXI4-Lite Control Interface (logic and memory space). 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.
SrcEn	Input	SysClkDiv_User	**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, SrcPatternErr and SrcStatFrameErr. When SrcEn is deasserted, the Source core will transmit training patterns on TDat.
SrcOof	Output	SysClkDiv_User	**Source Out-of-Frame.** When this signal is asserted (active high), it indicates 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 parameter NumDip2Matches) are received and SrcEn is asserted.

Table 3-8: **Source Control and Status Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
SrcDIP2Err	Output	M_AXIS_SRCSTAT_ACLK	**Source DIP-2 Parity Error.** When this signal is asserted (active high), it indicates 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	M_AXIS_SRCSTAT_ACLK	**Source Status Frame Error**: When this signal is asserted (active high), it indicates that a non "11" frame word was received after DIP-2 on TStat. This signal is asserted for one clock cycle each time a frame word error is detected.
SrcPatternErr	Output	S_AXIS_SRCFF_ACLK	**Source Data Pattern Error.** When asserted (active high), indicates 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. • All S_AXIS_SRCFF_TKEEP not set to one when S_AXIS_SRCFF_TLAST is deasserted. This signal is asserted for one clock cycle each time an illegal data pattern is written into the Source FIFO.
IdleRequest	Input	SysClkDiv_User	**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 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	SysClkDiv_User	**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.
SrcTriStateEn	Input	SysClkDiv_User	**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	SysClkDiv_User	**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 AXI4-Stream FIFO Interface

The Source AXI4-Stream FIFO Interface signals write data into the FIFO to be transmitted on the SPI-4.2 Interface. Table 3-9 defines the signals of this interface.

Table 3-9: **Sink AXI4-Stream FIFO Interface Signals**

Name	Direction	Clock Domain	Description
S_AXIS_SRCFF_ACLK	Input	n/a	**Source FIFO Clock**. All Source FIFO Interface signals are synchronous to the rising edge of this clock.
S_AXIS_SRCFF_ARESETN	Input	S_AXIS_SRCFF_ACLK	**Source FIFO Reset.** Active low 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. The reset signal can be asserted asynchronously, but deassertion must be synchronous after the rising edge of ACLK. See the *AMBA AXI4-Stream Protocol* specification for more information.
S_AXIS_SRCFF_TREADY	Output	S_AXIS_SRCFF_ACLK	**Source FIFO Core Handshake**. Indicates the core is ready to accept streaming data. When both TREADY and TVALID are asserted on the rising edge of S_AXIS_SRCFF_ACLK, one data beat is transferred into the core. De-assertion of TREADY indicates the Source core has reached the AlmostFull threshold set by the parameter SrcAFThresAssert.
S_AXIS_SRCFF_TVALID	Input	S_AXIS_SRCFF_ACLK	**Source FIFO User Handshake**. Indicates the user is presenting valid streaming data. When both TREADY and TVALID are asserted on the rising edge of S_AXIS_SRCFF_ACLK, one data beat is transferred into the core. When asserted (active high), this signal indicates that the information on S_AXIS_SRCFF_TDATA, S_AXIS_SRCFF_TID, S_AXIS_SRCFF_SOP, S_AXIS_SRCFF_TLAST, S_AXIS_SRCFF_TKEEP, and S_AXIS_SRCFF_ERR are valid.
S_AXIS_SRCFF_TID[7:0]	Input	S_AXIS_SRCFF_ACLK	**Source FIFO Channel Address**. Channel number associated with the data on S_AXIS_SRCFF_TDATA.
S_AXIS_SRCFF_TDATA[63:0] or S_AXIS_SRCFF_TDATA[127:0]	Input	S_AXIS_SRCFF_ACLK	**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.

Table 3-9: **Sink AXI4-Stream FIFO Interface Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
S_AXIS_SRCFF_TKEEP[15:0] or S_AXIS_SRCFF_TKEEP[7:0]	Input	S_AXIS_SRCFF_ACLK	**Source FIFO Strobe.** This signal indicates which bytes on the S_AXIS_SRCFF_TDATA bus are valid when the S_AXIS_SRCFF_TLAST signal is asserted (i.e. byte enable). When S_AXIS_SRCFF_TLAST is deasserted, S_AXIS_SRCFF_TKEEP should always be set to all ones. S_AXIS_SRCFF_TKEEP[7:0] is used with a 64-bit interface. S_AXIS_SRCFF_TKEEP[15:0] is used with a 128-bit interface.
S_AXIS_SRCFF_SOP	Input	S_AXIS_SRCFF_ACLK	**Source FIFO Start of Packet**. When asserted (active high), indicates that the start of a packet is being written into the Source FIFO.
S_AXIS_SRCFF_TLAST	Input	S_AXIS_SRCFF_ACLK	**Source FIFO End of Packet**. When asserted (active high), indicates that the end of a packet is being written into the Source FIFO. May be concurrent with S_AXIS_SRCFF_SOP.
S_AXIS_SRCFF_ERR	Input	S_AXIS_SRCFF_ACLK	**Source FIFO Error**. When asserted (active high) simultaneously with the S_AXIS_SRCFF_TLAST 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. S_AXIS_SRCFF_ERR can be used in combination with S_AXIS_SRCFF_TLAST to insert erroneous DIP-4 values for testing purposes. When S_AXIS_SRCFF_ERR is asserted and S_AXIS_SRCFF_TLAST is not asserted, the core inserts an EOP (1 or 2 bytes depending on the S_AXIS_SRCFF_TKEEP value) with an erroneous DIP-4 value. The erroneous DIP4 value is an inversion of the correctly calculated value.

Source AXI4-Lite Control Interface (Calendar, Configuration and Status)

The Source AXI4-Lite Control interface is used to program the calendar memory, read status memory, and program static configuration memory. This memory space is defined in Figure 3-5.

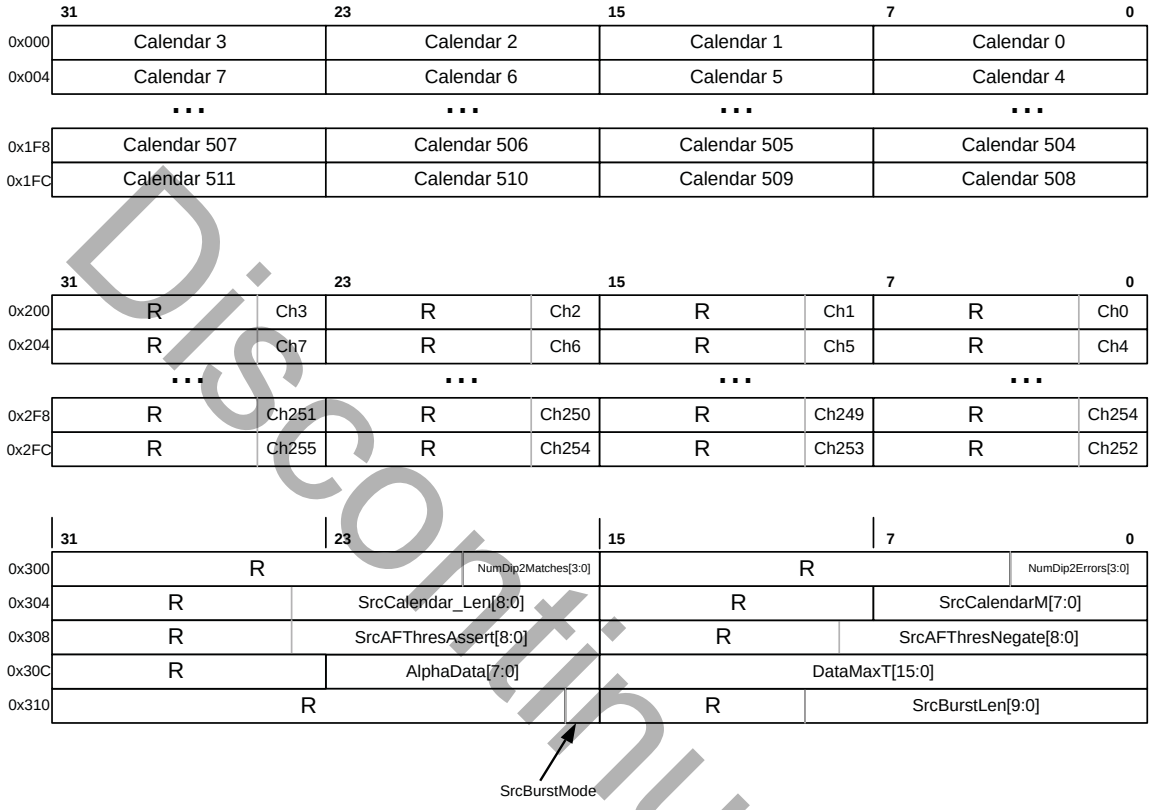

Figure 3-5: **Source AXI4-Lite Memory Space**

The calendar determines the status channel order and frequency. Through this interface, the user can program the calendar buffer to determine the order and frequency with which channel status is received on the SPI-4.2 interface.

Flow control data from the SPI-4.2 Status interface can be received by the user in one of two ways:

- **Addressable Mode** (AXI4-Lite Control Interface): In this mode, the Status information is stored in the AXI4-Lite Control memory space and can be retrieved at the user's convenience.
- **Transparent Mode** (AXI4-Stream Status Interface): In this mode, the Status information is provided real-time as it is received on TStat and is not stored within the core. In this mode, the status memory space as defined in Figure 3-5 is not usable ("Reserved") and reads to it will return all zeroes.

The static configuration memory enables customization of the core based on individual system requirements. These settings are statically driven inside the core by writing

registers through the Control interface. Table 3-10 defines the Control interface signals, and Table 3-11 defines the static configuration parameters.

Table 3-10: **Source AXI4-Lite Control interface Signals**

Name	Direction	Clock Domain	Description
AXI_SRC_ACLK	Input	N/A	**AXI4-Lite Interface Clock**. All Sink AXI4-Lite signals are synchronous to the rising edge of this clock.
AXI_SRC_ARESETN	Input	AXI_SRC_ACLK	**AXI4-Lite Interface Reset.** Active-low signal that enables the user to reset the AXI4-Lite interface and all associated logic and memories to chosen Vivado settings. The reset signal can be asserted asynchronously, but deassertion must be synchronous after the rising edge of ACLK. See the *AMBA AXI4 Protocol* specification for more information.
Read Address Channel			
AXI_SRC_ARREADY	Output	AXI_SRC_ACLK	**AXI4-Lite Read Address Core Handshake**. Indicates the core is ready to accept a read address. When both ARREADY and ARVALID are asserted on a clock cycle, a one-word read request occurs for the address provided on ARADDR and the core fetches the contents of that address.
AXI_SRC_ARVALID	Input	AXI_SRC_ACLK	**AXI4-Lite Read Address User Handshake**. Indicates the user is presenting a valid read address. When both ARREADY and ARVALID are asserted on a clock cycle, a one-word read request occurs for the address provided on ARADDR and the core fetches the contents of that address.
AXI_SRC_ARADDR[9:0]	Input	AXI_SRC_ACLK	**AXI4-Lite Read Address.** Address to be read by the user application.
Read Data Channel			
AXI_SRC_RREADY	Input	AXI_SRC_ACLK	**AXI4-Lite Read Data User Handshake**. Indicates the user is ready to accept read data. When both RREADY and RVALID are asserted on a clock cycle, one data beat is read out of the core. This handshake completes the transaction begun by the ARREADY/ARVALID handshake.
AXI_SRC_RVALID	Output	AXI_SRC_ACLK	**AXI4-Lite Read Data Core Handshake**. Indicates the core is presenting valid read data. When both RREADY and RVALID are asserted on a clock cycle, one data beat is read out of the core. This handshake completes the transaction begun by the ARREADY/ARVALID handshake.
AXI_SRC_RDATA[31:0]	Output	AXI_SRC_ACLK	**AXI4-Lite Read Data.** Data read from address location provided on ARADDR is presented here. Bit 0 is the LSB.
Write Address Channel			
AXI_SRC_AWREADY	Output	AXI_SRC_ACLK	**AXI4-Lite Write Address Core Handshake.** Indicates the core is ready to accept a write address. When both AWREADY and AWVALID are asserted on a clock cycle, one write address beat is accepted by the core. Once both the WREADY/WVALID and AWREADY/AWVALID handshakes occur independently, one write transaction is complete.

Table 3-10: **Source AXI4-Lite Control interface Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
AXI_SRC_AWVALID	Input	AXI_SRC_ACLK	**AXI4-Lite Write Address User Handshake.** Indicates the user is presenting a valid write address. When both AWREADY and AWVALID are asserted on a clock cycle, one write address beat is accepted by the core. Once both the WREADY/WVALID and AWREADY/AWVALID handshakes occur independently, one write transaction is complete.
AXI_SRC_AWADDR[9:0]	Input	AXI_SRC_ACLK	**AXI4-Lite Write Address**. Address to be written by the user application.
Write Data Channel			
AXI_SRC_WREADY	Output	AXI_SRC_ACLK	**AXI4-Lite Write Data Core Handshake**. Indicates the core is ready to accept write data and strobe. When both WREADY and WVALID are asserted on a clock cycle, one write address beat is accepted by the core. Once both the WREADY/WVALID and AWREADY/AWVALID handshakes occur independently, one write transaction is complete
AXI_SRC_WVALID	Input	AXI_SRC_ACLK	**AXI4-Lite Write Data User Handshake.** Indicates the user is presenting valid write data and strobe. When both WREADY and WVALID are asserted on a clock cycle, one write address beat is accepted by the core. Once both the WREADY/WVALID and AWREADY/AWVALID handshakes occur independently, one write transaction is complete.
AXI_SRC_WDATA[31:0]	Input	AXI_SRC_ACLK	**AXI4-Lite Write Data**. Data written into the core. Bit 0 is the LSB.
AXI_SRC_WSTRB[3:0]	Input	AXI_SRC_ACLK	**AXI4-Lite Write Strobe**. Byte enable associated with AXI_SRC_WDATA. Bit 0 enables WDATA[7:0], bit 1 enables WDATA[15:8], bit 2 enables WDATA[23:16], and bit 3 enables WDATA[32:24].
Write Response Channel			
AXI_SRC_BREADY	Input	AXI_SRC_ACLK	**AXI4-Lite Write Response User Handshake.** Indicates the user is ready to accept a write response. When both BREADY and BVALID are asserted on a rising edge of AXI_SRC_ACLK, one write response transaction is read out of the core.
AXI_SRC_BVALID	Output	AXI_SRC_ACLK	**AXI4-Lite Write Response Core Handshake.** Indicates the core is presenting a valid write response. When both BREADY and BVALID are asserted on a rising edge of AXI_SRC_ACLK, one write response transaction is read out of the core. Each write transaction will generate a write response, regardless of whether the write succeeded. The Error Bus (below) provides further signaling in regards to errors that occur during write operations.

Table 3-10: **Source AXI4-Lite Control interface Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
Sideband Error Bus			
SRC_BRESP_ERR[7:0]	Output	AXI_SRC_ACLK	**AXI4-Lite Error Bus.** Each bit of this bus corresponds to a specific AXI4-Lite Error condition. The error conditions detected are reported as follows: SRC_BRESP_ERR [0]: Write Response counter is full. No further writes or reads will be processed until at least one Write Response is read from the core using BREADY/BVALID. • SRC_BRESP_ERR[1]: Write transaction to the Configuration space failed. • SRC_BRESP_ERR [2]: Write transaction to the Calendar space has failed. • SRC_BRESP_ERR[3]: Write transaction to the Status space has failed. • SRC_BRESP_ERR[4]:Read/Write transaction has missed a valid address. This includes all Status memory accesses when using Transparent Status Mode. • SRC_BRESP_ERR [5:7]: Reserved bits (tied low).

Source Static Configuration Parameters

Source Static Configuration Parameters are memory registers in the core that are written to through the AXI4-Lite Control Interface. The default values of these memory registers are determined through the Vivado GUI. The user can customize these registers using the GUI.

If these parameters need to be changed in-circuit, they should be changed when the core is not in operation (disabled and in reset state).

The following steps are recommended when changing static configuration parameters:

1. Disable the source core (`SrcEn` signal).
2. Assert core reset (`Reset_n` = 0).
3. Change the desired static configuration parameters by using the AXI4-Lite control interface. See Source Static Configuration Parameters in Chapter 5.
4. Deassert reset (`Reset_n`=1).
5. Wait at least 10 clock cycles of `SysClkDiv_User` for the source static configuration parameters 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.

The Source Static Configuration Parameters are defined in Table 3-11.

.

Table 3-11: **Source Static Configuration Parameter Definition**

Name	Range	Description
NumDip2Errors[3:0]	1-15	**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]	1-15	**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.
SrcCalendarM[7:0]	0-255 (effective range 1–256)	**Source Calendar Period**. 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 programmed by the AXI4-Lite interface, 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.
SrcCalendarLen[8:0]	0-511 (effective range 1–512)	**Source Calendar Length**. Sets the length of the calendar sequence. The Source core implements this parameter as a static register programmed by the AXI4-Lite interface, 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.
SrcAFThresNegate[8:0]	SrcAFThresAssert to 508	**Source Almost Full Threshold Negate**. Specifies the minimum number of empty FIFO locations to exist in the Source FIFO before the S_AXIS_SRCFF_TREADY is asserted. SrcAFThresNegate is ≥ SrcAFThresAssert.
SrcAFThresAssert[8:0]	If SrcBurstMode = 0: 1-508. If SrcBurstMode = 1 and SrcBurstLen < 256, range is SrcBurstLen-508. If SrcBurstMode = 1 and SrcBurstLen >=256, range is 256-508.	**Source Almost Full Threshold Assert**. Specifies the minimum number of empty FIFO locations to exist in the Source FIFO before S_AXIS_SRCFF_TREADY is de-asserted. SrcAFThresNegate ≥ SrcAFThresAssert.
DataMaxT[15:0]	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.

Table 3-11: **Source Static Configuration Parameter Definition *(Cont'd)***

Name	Range	Description
AlphaData[7:0]	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.
SrcBurstLen[9:0]	If SrcBurstMode = 1, 1-1023 If SrcBurstMode = 0, 1- 256	**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).
SrcBurstMode	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 complete credit of data. When 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.

Source AXI4-Stream Status Interface

The Source AXI4-Stream interface provides flow control information about the SPI-4.2 link.

When the core is used in Addressable Status mode, the interface provides the channel number for the most recently updated SPI-4.2 channel status on `M_AXIS_SRCSTAT_TID`, qualified by the `M_AXIS_SRCSTAT_TVALID` signal. The channel status itself must be accessed via the AXI4-Lite Control interface, with each read providing status for four channels.

When the core is used in Transparent Status mode, the `M_AXIS_SRCSTAT_TUSER[1:0]` bus is added. This port provides the SPI-4.2 status as received on TStat with minimal latency. In this mode, the `M_AXIS_SRCSTAT_TID` bus indicates the channel to which the current status shown on `M_AXIS_SRCSTAT_TUSER` belongs. Both `M_AXIS_SRCSTAT_TID` and `M_AXIS_SRCSTAT_TUSER` are qualified by `M_AXIS_SRCSTAT_TVALID`.

Table 3-12: **Source AXI4-Stream Status Interface Signals**

Name	Direction	Clock Domain	Description
M_AXIS_SRCSTAT_ACLK	Input	N/A	**Status Clock**. All status AXI4-Stream signals are synchronous to the rising edge of this clock. This clock should be derived from TSClk_P/N.
M_AXIS_SRCSTAT_TID[7:0]	Output	M_AXIS_SRCSTAT_ACLK	**Status Channel**. The Source Status Channel is an 8-bit bus containing the channel address that is being updated on M_AXIS_SRCSTAT_TUSER on the current clock cycle.

Table 3-12: **Source AXI4-Stream Status Interface Signals *(Cont'd)***

Name	Direction	Clock Domain	Description
M_AXIS_SRCSTAT_TVALID	Output	M_AXIS_SRCSTAT_ACLK	**Status Valid**. When asserted indicates that M_AXIS_SRCSTAT_TUSER and M_AXIS_SRCSTAT_TID are valid. When the core is processing DIP-2 or frame words, TVALID is deasserted. A transition of TVALID from 0 to 1 indicates that the core has started a new calendar sequence.
M_AXIS_SRCSTAT_TUSER[1:0]	Output	M_AXIS_SRCSTAT_ACLK	**Status (Only Available in Transparent Mode).** Represents the current status received on the SPI-4.2 TStat bus.

Source Clock Interface

For the SPI-4.2 AXI core (version 11.1 and onwards), the Source core's clocking module is external to the core. The clock interface to the Source core consists of input clock ports required by the internal logic. A list of the Source clock ports and their description is given in Table 3-13.

The minimum frequency for all clocks in Table 3-13 is dependent on the minimum frequency of the MMCM and BUFR (depending on which clocking configuration is selected). See the Kintex-7 and Virtex-7 FPGA product specifications 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 `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 a 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 components as close to each other as possible. See Source Core Required Constraints in Chapter 6.

Table 3-13: **Source Clock Signals for Virtex-7 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

XILINX®

Chapter 4

Customizing and Generating the Core

The SPI-4.2 core is a fully configurable implementation of the *OIF-SPI4-02.1* specification. Using the Vivado IP catalog, you can configure the core and customize the delivered files including the example wrapper and XDC files.

Note: After the core is generated, only static configuration parameters can be modified by using the Sink or Source AXI4-Lite Control interface. If other modifications are required, the core must be regenerated with the new options.

Graphical User Interface

The SPI-4.2 Vivado 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 XDC File options.

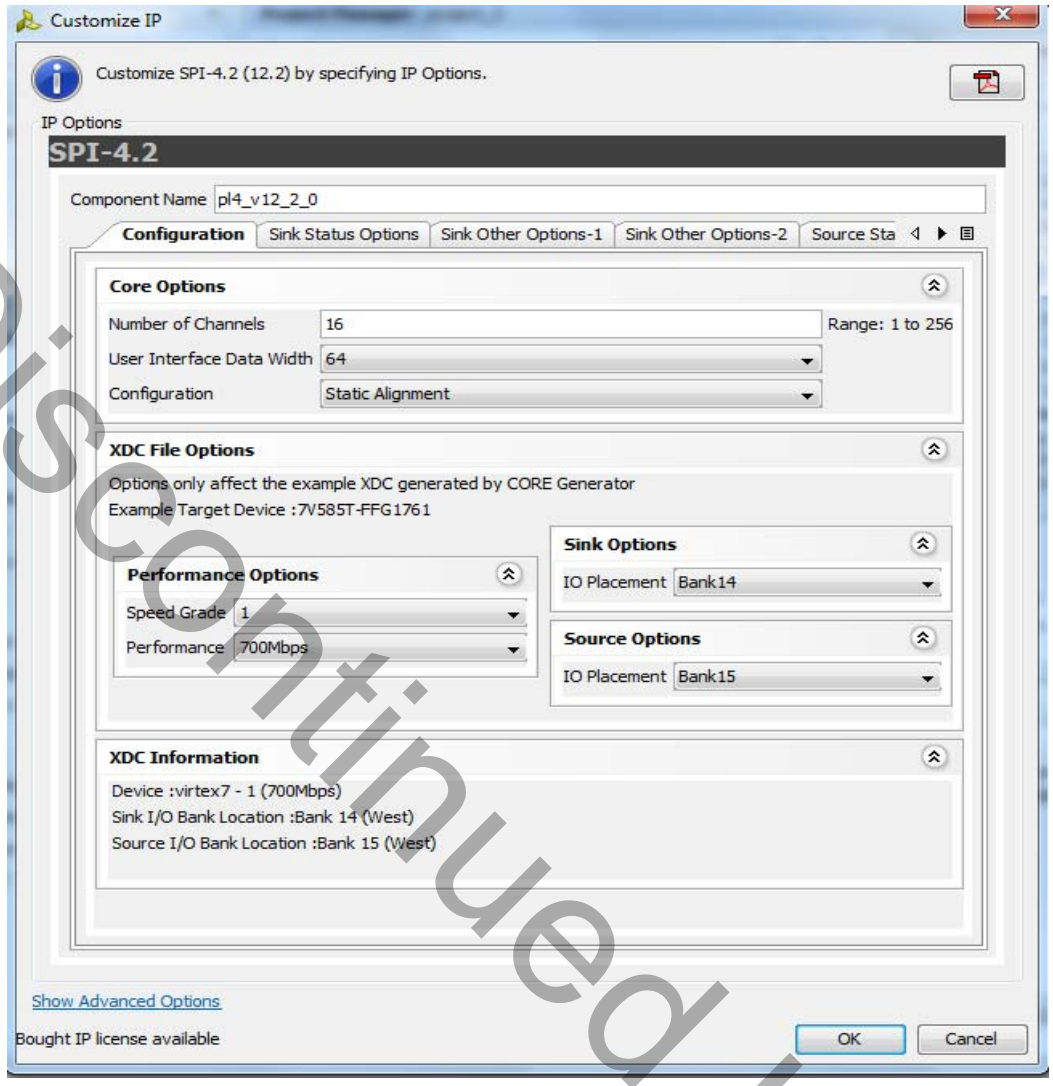

Figure 4-1: **SPI-4.2 Sink and Source Main Customization Screen**

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.

XDC File Options

Selections in the XDC file section of the GUI affect generation of the example XDC file. The XDC file can be edited to change options, or to regenerate the core with the Vivado system to create a new example XDC 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 XDC file.

Sink and Source Options

The I/O placement selection configures the pin-location constraints in the example XDC file. For 7 series devices, all SPI-4.2 interface pins, except LVTTL Status pins, are constrained to High Performance I/Os. These are editable, as described in Chapter 6, Constraining the Core.

XDC Information

This section displays a summary of the contents of the example XDC 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 parameter 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 parameter 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 Sink AXI4-Lite Control 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 parameter `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 parameter `RSClkDiv`; it selects the frequency of *`RSClk`* with respect to RDClk.

Alignment

This is the value of static configuration parameter `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. LVTTL Status I/O is only supported in High Range I/O banks.

Sink Other Options (1) 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 parameter `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 parameter `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 parameter `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 parameter `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

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 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 (2) 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, 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. This feature is only available when the RDClk clock distribution is set to regional buffer.

Specify Initial Tap Setting

Found under Enable DPA Clock Adjustment. This option allows you to pick the initial tap setting for the inserted IDELAY. The default value is 16. This value should be changed only after consulting Xilinx.

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 parameter `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 parameter `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 mode. For more information, see the Source Flow Control, page 114.

Status I/O

This option controls whether the Source core status I/O in the generated wrapper file uses LVDS or LVTTL I/O. LVTTL Status I/O is only supported in High Range I/O banks.

Synchronization

This option defines the synchronization parameters of the core.

Number of DIP2 Matches

This is the value of static configuration parameter `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 parameter `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 (1) 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 parameter `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 parameter `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 parameter `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 parameter `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 125.

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 parameter `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 parameter `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 (2) Screen

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

Clocking

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 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.

- `M_AXIS_SNKFF_TREADY`
- `S_AXIS_SRCFF_TVALID`

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 Vivado IP catalog. Modifications made not using the Vivado IP catalog 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 and AXI4-Lite reset.

   To reset the core and I/O logic, Reset_n must be asserted. To reset the AXI4-Lite interface (logic and associated memory), `AXI_SNK_ARESETN` (Sink core) and `AXI_SRC_ARESETN` (Source core) must be asserted. The FIFO resets, `M_AXIS_SNKFF_ARESETN` and `S_AXIS_SRCFF_ARESETN`, may also be asserted but are not required. The reset signals for each portion of the core must remain asserted until the clocks are ready for use.

   - Required Resets: `Reset_n`, `AXI_SNK_ARESETN`, `AXI_SRC_ARESETN`
   - Optional Resets: `M_AXIS_SNKFF_ARESETN`, `S_AXIS_SRCFF_ARESETN`
2. Disable the cores (`SrcEn = 0, SnkEn = 0`).
3. Assert and release the MMCM resets.

   This step is only applicable if `RDClk0_User`, `RDClkDiv_User`, `AXI_SNK_ACLK`, `M_AXIS_SNKFF_ACLK`, `SysClkDiv_User`, `SysClk0_User`, `AXI_SRC_ACLK`, or `S_AXIS_SRCFF_ACLK` are distributed with global clocking using an MMCM. If regional clocking is selected for all clocks, this step can be skipped. If one or more MMCMs are used, reset each MMCM in the core while the core is in reset. Reset the MMCM by asserting the MMCM reset signal (`Rst`). Once the MMCM reset is pulsed, wait for the assertion of the MMCM locked signal (`Locked`). 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 core uses IDELAYs (Dynamic Alignment core, or Static Alignment with Regional Clocking), reset the IDELAYCTRL blocks associated with the Sink core. The example design provided with the core uses the `SnkIdelayCtlRst` input at the <core_name>_top.v(vhd) level to perform this reset.

   The reset to the IDELAYCTRL should be pulsed after FPGA configuration and once the reference clock input of the IDELAYCTRL has stabilized, to ensure proper IDELAY operation.

5. Wait until all clocks are ready for use.
6. De-assert AXI4-Lite resets (`AXI_SRC_ARESETN` and `AXI_SNK_ARESETN`). De-assert AXI4-Stream resets (`S_AXIS_SRCFF_ARESETN` and `M_AXIS_SNKFF_ARESETN`) if they were asserted.
7. Use the AXI4-Lite interface to overwrite the static configuration values provided by Vivado, if desired.

Note: No AXI4-Lite write operations should occur before the fourth clock edge after the SrcEn/SnkEn or Reset_n signal toggles. Any write operations attempted prior to this clock edge will cause an error on the sideband error bus (SRC_BRESP_ERR[2:1] or SNK_BRESP_ERR[2:1]).

8. Deassert core reset (`Reset_n`).
9. Initialize the Status Calendar.

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

   a. Initialize calendar with a default value.

      Using the Vivado IP catalog, load an initialization file with the calendar contents. This negates the need to use the AXI4-Lite interface to initialize the calendar. See Chapter 4, Customizing and Generating the Core for more information.

   b. Programming calendar after reset.

      Use the AXI4-Lite interface to program the calendar contents. Note that any operations on the AXI4-Lite interface must wait until the fourth clock edge after any changes on `SnkEn/SrcEn` or `Reset_n` for proper operation. See Sink Calendar Memory Initialization, page 83 and Source Calendar Memory Initialization, page 118 sections for more information.

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

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 8-bit status bus to a 2-bit status word.

The following sections explain how the Sink core interfaces operate. See Sink Core Interfaces, page 27 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 on the AXI4-Stream FIFO 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, 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 AXI4-Stream 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 M_AXIS_SNKFF_SOP and M_AXIS_SNKFF_TLAST for channel 1 are asserted.

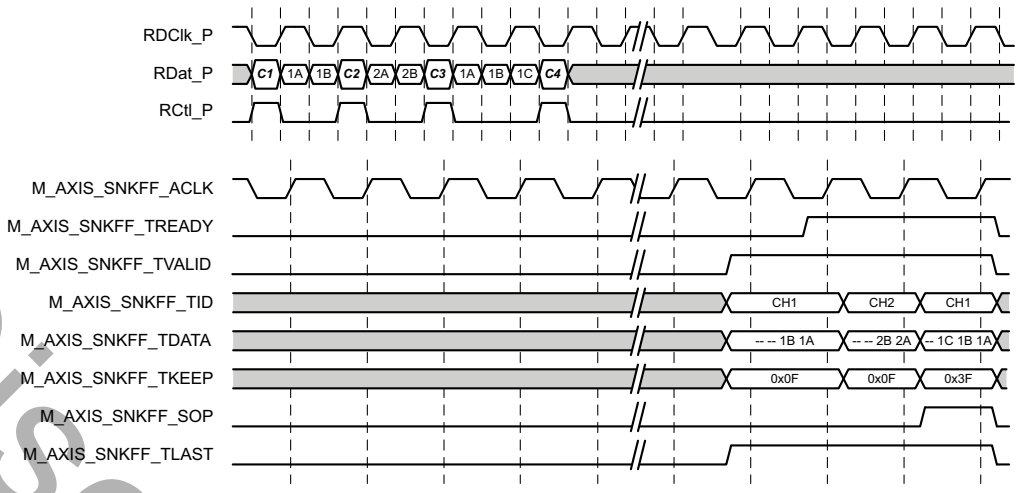

Figure 5-1: **SPI-4.2 Interface to User Interface**

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 M_AXIS_SNKFF_SOP and M_AXIS_SNKFF_TLAST 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 `M_AXIS_SNKFF_SOP` and `M_AXIS_SNKFF_TLAST` are both asserted.

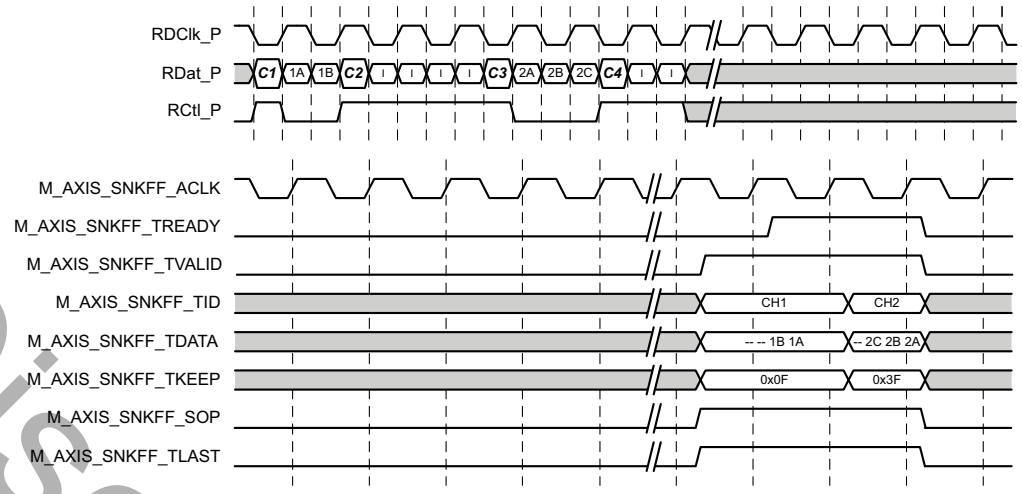

Figure 5-2: **Sink Data Path - Short Packet Transfers with Minimum SOP Spacing Enforced**

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 AXI4-Stream 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 right justified and swizzled when written into the FIFO: RDAT[15:8] is presented on M_AXIS_SNKFF_TDATA[7:0] and RDAT[7:0] is presented on M_AXIS_SNKFF_TDATA[15:8]. 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**

SPI-4.2 Sink Data				AXI4-Stream FIFO Interface (from Sink FIFO) M_AXIS_SNKFF_*		
Data Received on the SPI-4.2 Interface (RDat [15:0])		RCtl	RDClk cycle	*TDATA[63:0])	*ACLK cycle	Control Bits Read
SOP	b:[1001.0000.0010.pppp]	1	1	N/A	n	N/A
Word 0 (P0)	0xF1E2	0	2			
Word 1 (P1)	0xD3C4	0	3			
Word 2 (P2)	0xB5A6	0	4			

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

SPI-4.2 Sink Data				AXI4-Stream FIFO Interface (from Sink FIFO) M_AXIS_SNKFF_*		
Data Received on the SPI-4.2 Interface (RDat [15:0])		RCtl	RDClk cycle	*TDATA[63:0])	*ACLK cycle	Control Bits Read
Word 3 (P3)	0xF9E8	0	5	P3.P2.P1.P0 = E8F9.A6B5.C4D3.E2F1	n + 1	*SOP = 1 *TLAST = 0 *TKEEP= 0xFF *ERR = 0 *TID = 00000010
Word 4 (P4)	0x1F2E	0	6			
Word 5 (P5)	0x3D4C	0	7			
Word 6 (P6)	0x5B6A	0	8			
Word 7 (P7)	0x9F8E	0	9	P7.P6.P5.P4 = 8E9F.6A5B.4C3D.2E1F	n + 2	*SOP= 0 *TLAST = 0 *TKEEP= 0xFF *ERR = 0 *TID = 00000010
Word 8 (P8)	0xABCD	0	10			
Word 9 (P9)	0x1200	0	11			
EOP/ Valid bytes	b:[0110.aaaa.aaaa.pppp]	1	12			
				P9.P8 = 0000.0000.0012.CDAB	n + 3	*SOP= 0 *TLAST = 1 *TKEEP = 0x07 *ERR = 0 *TID = 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	M_AXIS_SNKFF_SOP, M_AXIS_SNKFF_TID[7:0] <== RDat[11:4]
New Burst (address change)	RDat[15] = 1, RDat[12] = 0	M_AXIS_SNKFF_TID[7:0] <== RDat[11:4]

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
End of Packet (EOP, 2 bytes valid)	RDat[14:13] = 10	M_AXIS_SNKFF_TLAST, M_AXIS_SNKFF_TKEEP[7:0] When RDat[14:13] = 10: TKEEP = 0xFF if data bits 63-0 have valid data TKEEP = 0x3F if data bits 47-0 have valid data TKEEP = 0x0F if data bits 31-0 have valid data TKEEP = 0x03 if data bits 15-0 have valid data
End of Packet (EOP, 1 bytes valid)	RDat[14:13] = 11	M_AXIS_SNKFF_TLAST, M_AXIS_SNKFF_TKEEP[7:0] When RDat[14:13] = 11: TKEEP = 0x7F if data bits 55-0 have valid data TKEEP = 0x1F if data bits 39-0 have valid data TKEEP = 0x07 if data bits 23-0 have valid data TKEEP = 0x01if data bits 7-0 have valid data
End of Packet (EOP Abort, error condition)	RDat[14:13] = 01	M_AXIS_SNKFF_ERR & M_AXIS_SNKFF_TLAST

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	M_AXIS_SNKFF_SOP, M_AXIS_SNKFF_TID[7:0] <== RDat[11:4]
New Burst (address change)	RDat[15] = 1, RDat[12] = 0	M_AXIS_SNKFF_TID[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	M_AXIS_SNKFF_TLAST, M_AXIS_SNKFF_TKEEP[7:0] When RDat[14:13] = 10: TKEEP = 0xFFFF if data bits 127-0 have valid data TKEEP = 0x3FFF if data bits 111-0 have valid data TKEEP =0x0FFF if data bits 95-0 have valid data TKEEP = 0x03FF if data bits 79-0 have valid data TKEEP =0x00FF if data bits 63-0 have valid data TKEEP = 0x003F if data bits 47-0 have valid data TKEEP = 0x000F if data bits 31-0 have valid data TKEEP = 0x0003 if data bits 15-0 have valid data
End of Packet (EOP, 1 byte valid)	RDat[14:13] = 11	M_AXIS_SNKFF_TLAST, M_AXIS_SNKFF_TKEEP[7:0] When RDat[14:13] = 11: TKEEP = 0x7FFF if data bits 119-0 have valid data TKEEP = 0x1FFF if data bits 103-0 have valid data TKEEP = 0x07FF if data bits 87-0 have valid data TKEEP = 0x01FF if data bits 71-0 have valid data TKEEP = 0x007F if data bits 55-0 have valid data TKEEP = 0x001F if data bits 39-0 have valid data TKEEP = 0x0007 if data bits 23-0 have valid data TKEEP =0x0001 if data bits 7-0 have valid data
End of Packet (EOP Abort, error condition)	RDat[14:13] = 01	M_AXIS_SNKFF_ERR & M_AXIS_SNKFF_TLAST

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 69). With a 64-bit and 128-bit data interface, this user interface can operate up to:

- 312.5 MHz in Virtex®-7 FPGAs
- 250 MHz in Kintex-7 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 Sink user interface has three major signal groups:

- **Control and Status Signals**. Apply to the operation of the entire Sink core.
- **AXI4-Stream FIFO Signals**. Allow access to data received on the SPI-4.2 Interface.
- **AXI4-Lite Control Signals**. Used to configure the calendar sequence, static configuration parameters, and 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, page 30.

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 `M_AXIS_SNKFF_TDATA`. These signals are triggered concurrently with the `SnkBusErr` 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_User` clock cycle.

- **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.

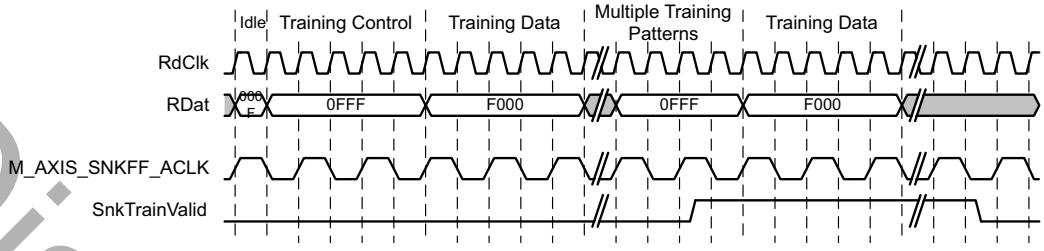

Figure 5-3: **Sink Training Valid Status**

- **Reset_n.** Use to restart the entire Sink core with the exclusion of the Sink AXI4-Lite Control Interface, logic and memory space. It causes the interface to go out-of-frame. When `Reset_n` is deasserted, the Sink core initiates the synchronization startup sequence.

Sink AXI4-Stream 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 32. Waveforms illustrating the handshaking and FIFO status signals are shown in Figure 5-4. The Sink FIFO Interface signals are synchronous to `M_AXIS_SNKFF_ACLK`, and the FIFO is 510 words deep. For a 64-bit interface, a FIFO word is 1 credit wide and for 128-bit interface, a FIFO word is 2 credits wide.

Sink FIFO Data Valid Handshake

Figure 5-4 shows the Data Valid handshake (`M_AXIS_SNKFF_TVALID`) signal. As shown in the waveform, the valid flag is asserted when there is a data available to be read out from the FIFO. The FIFO logic pushes out the valid data as soon as its available to be read from the FIFO regardless of whether the corresponding user logic is ready to read it. Although a valid word is pushed out on the interface whenever it's available, to read out this word, the User Ready Handshake (`M_AXIS_SNKFF_TREADY`) signal needs to be asserted for one cycle to complete the handshake.

The Sink FIFO Data Valid is not an actual indicator of the status of the Sink FIFO. This flag only indicates whether data is available to be read out from the read pipeline of the FIFO interface. Hence, it is possible that a number of data have been written into the FIFO, but

the Sink FIFO Data Valid flag is de-asserted. Data written into the FIFO may not be ready to be read out until a number of cycles later.

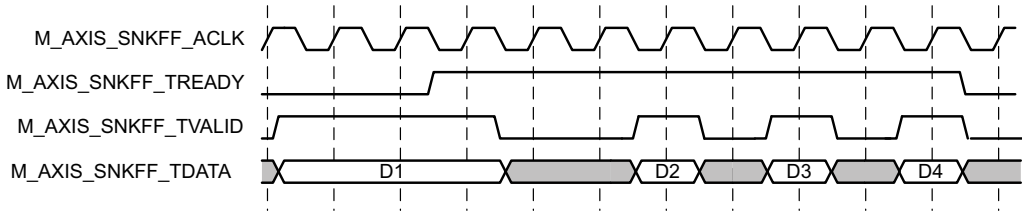

Figure 5-4: **Sink FIFO TVALID/TREADY Handshake**

Sink Almost Full

The behavior of the Sink Almost Full flag (`SNKFF_ALMOSTFULL_N`) is dependent on the static configuration parameters `SnkAFThresAssert` and `SnkAFThresNegate`. When the `SNKFF_ALMOSTFULL_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 `SNKFF_ALMOSTFULL_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 (`SNKFF_OVERFLOW_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 `SNKFF_ALMOSTFULL_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 `M_AXIS_SNKFF_TVALID` output is not actually an indicator of an absence of any data in the Sink FIFO. Rather, this signal indicates that the Sink (user) FIFO interface does, or does not have a valid data to be read out. Consequently it is possible for the data valid handshake to be deasserted 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 the deassertions of the valid flag, 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 `SNKFF_ALMOSTFULL_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 bursts will cause the Sink FIFO to fill up to a point where `SNKFF_ALMOSTFULL_N` may be asserted even though the Valid flag is indicating data not ready on the read interface. This occurs because new bursts 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 than 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 `M_AXIS_SNKFF_TVALID` to deassert regardless of the presence or absence of additional (incomplete) data bursts in the Sink FIFO. The `SNKFF_ALMOSTFULL_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 AXI4-Stream FIFO Reset

The assertion of the reset signal clears the FIFO (and the associated data path logic) while remaining in-frame. When `M_AXIS_SNKFF_ARESETN` is deasserted, the Sink data path will not write data into the FIFO until a packet with a valid SOP is received. The AXI4-Stream FIFO Reset signal needs to be asserted at the minimum for 3 `M_AXIS_SNKFF_ACLK` cycles.

Sink AXI4-Lite Control Interface

The Sink AXI4-Lite Control interface allows the user to program the calendar memory, status memory and static configuration memory as defined in Figure 3-3, page 34. A summary of the Sink AXI4-Lite Control interface and definition is provided in Table 3-4,

page 35.

Writing and updating the static configuration memory will define the characteristics and behavior of the core. Writing to the status memory updates the flow control data to be sent on the SPI-4.2 Interface. The channel order and frequency that status is sent is determined by the content of the calendar memory. 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 5-8 illustrates how calendar information determines the order and frequency that FIFO channel status information is transmitted on *RStat*.

The Sink AXI-4 Lite Control interface is broken into several groups: Read Address Channel, Read Data Channel, Write Address Channel, Write Data Channel, Write Response Channel and Sideband Error bus. With the exception of the Sideband error bus, all other signal groups require a handshake between their respective ready and valid signals to complete before an operation is performed. A read operation will first require the read address handshake to complete before the data is available on AXI_SNK_RDATA and AXI_SNK_RVALID. To read out the data, AXI_SNK_RREADY needs to be asserted to complete the read data handshake. For a write operation to occur, both the address and data handshake need to be completed. This will then generate a response on AXI_SNK_BVALID that must be read out by asserting AXI_SNK_BREADY. The write response is generated regardless of whether the write operation is successful. Note that this interface will only allow six outstanding write responses before the write response counter is full. Once the write response counter is full, no operations can be performed (write or read) on the AXI4-Lite Control Interface, and SNK_BRESP_ERR[0] will assert. No read or write operations can occur until at least one write response is drained from the core by asserting AXI_SNK_BREADY. Figure 5-5 shows the behavior of the write data, address and response channel.

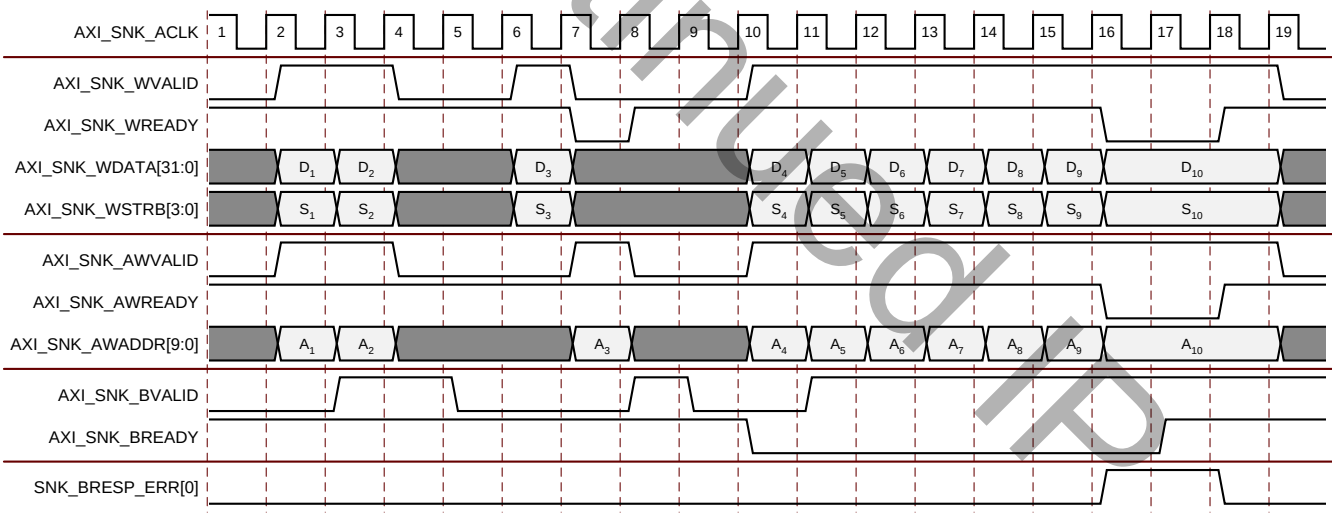

Figure 5-5: **Write Response Behavior of the Sink AXI4-Lite Control Interface**

If an error has occurred when executing the write or read operation, a bit on SNK_BRESP_ERR[7:0] will assert. If you access any address that is beyond the valid Sink AXI4-Lite Memory Space, for example address space 0x310 to 0x3FC, SNK_BRESP_ERR[3] will assert. Reads of the reserved bits/bytes within the valid address space will return all zeros. Writes to the reserved bits/bytes within the valid address space will generate a write response but will not have any affect.

Note: No AXI4-Lite write operations should occur before the fourth clock edge after the SnkEn or Reset_n signal toggles. Any write operations attempted prior to this clock edge will cause an error on the sideband error bus (SNK_BRESP_ERR[2:1]).

Although the read and write operations require separate channels, the two operations cannot be performed simultaneously. However, if a read and write request occur simultaneously, the write operation will have precedence over the read in the Sink AXI4-Lite Control Interface. This precedence enables continuous write operations to status memory to update flow control information.

As long as there are no outstanding write operations, read operations may be performed. Figure 5-6 gives an example of read operations.

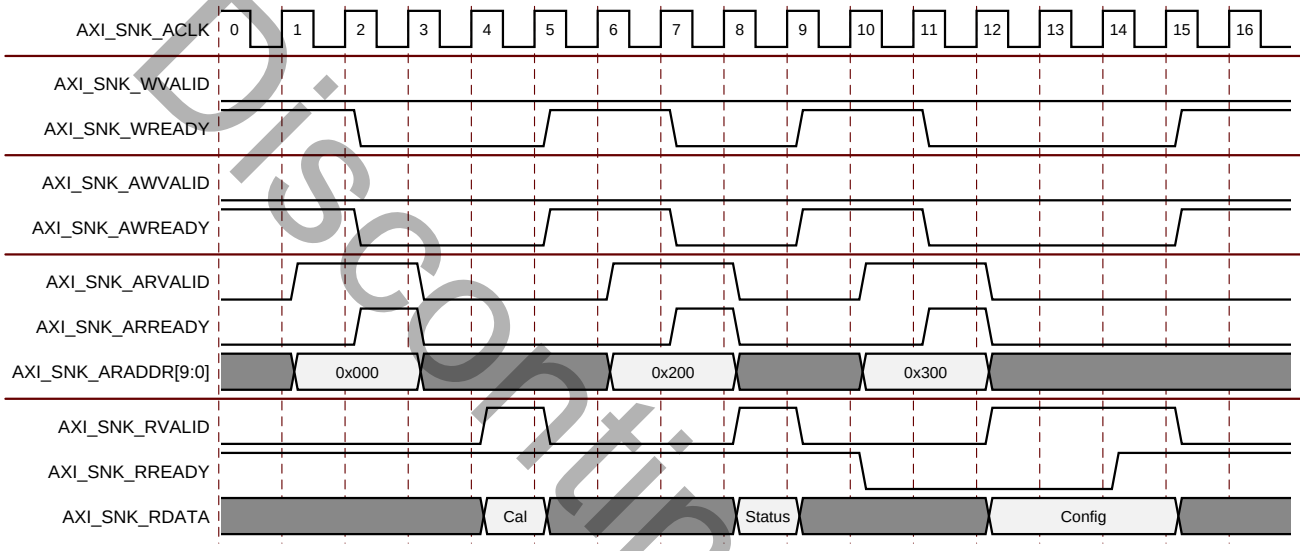

Figure 5-6: **Read Operation on the Sink AXI4-Lite Control Interface**

The Sink AXI4-Lite Control interface also has a reset signal AXI_SNK_ARESETN. This signal effects all the logic, data path and memory space associated with the interface. This signal requires a minimal pulse of two AXI_SNK_ACLK cycles. Both the calendar and static configuration memory space will revert to their default value when reset. These default values are chosen through loading a COE file (calendar memory space) and selecting values (static configuration parameters) in the Vivado IP catalog. See Chapter 4, Customizing and Generating the Core. The default value for all status channels is 10 or satisfied.

Figure 5-7 demonstrates the behavior of the AXI-4 Lite Control interface signals when reset occurs.

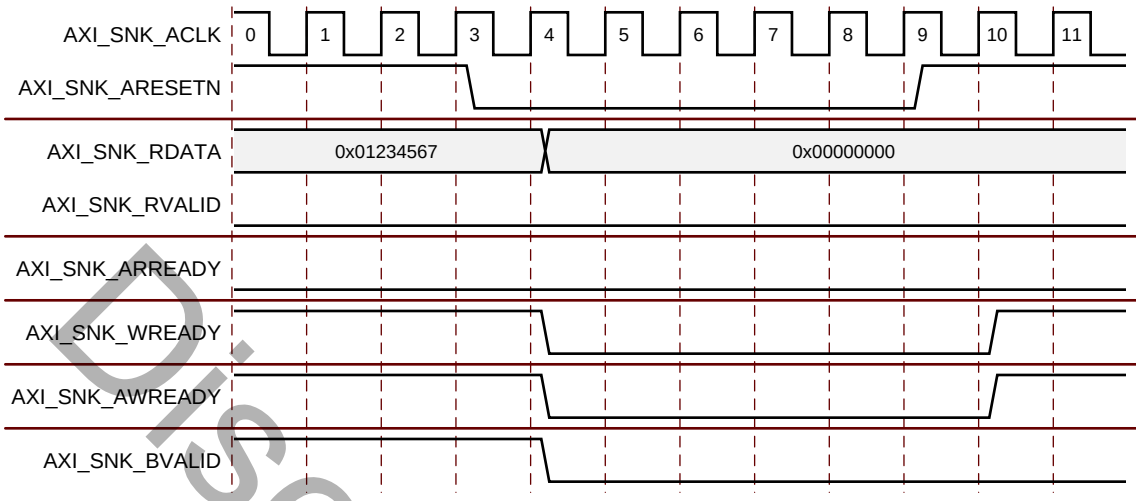

Figure 5-7: **Sink AXI Lite Control Interface Reset Behavior**

A detailed description on how to access and program the different memory spaces using the AXI4-Lite Control interface is described in this section. Although it is possible to read the memory spaces when the interface is not in reset state, there may be restrictions on when writing is permitted to each memory space.

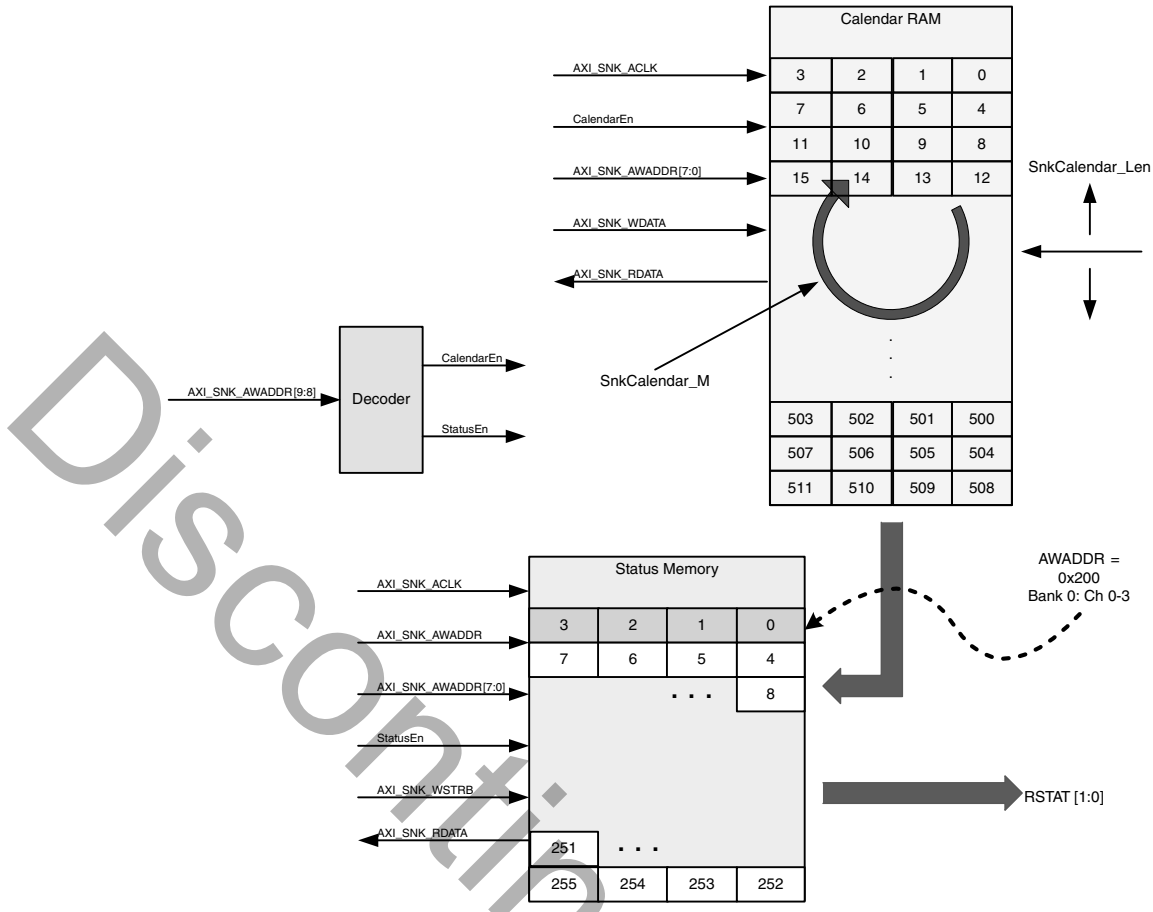

Figure 5-8: **Calendar Memory and Status Memory Block Diagram**

Sink Calendar Memory Initialization

There are two ways to initialize the Sink Calendar. One way is to load a COE file in the Vivado IP catalog, which puts the contents of the COE file into the XDC file (see Chapter 4, Customizing and 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 Sink AXI4-Lite Control interface to write the calendar sequence to memory. `AXI_SNK_AWADDR` is used to indicate the location in the calendar buffer, `AXI_SNK_WDATA` is used to indicate the channel number that should be written into that location , and `AXI_SNK_WSTRB` is used to indicate which of the possible four calendar entries should be written. When outputting `RStat`, the status for the channels written to `AXI_SNK_AWADDR`=0x000 are output first, followed by `AXI_SNK_AWADDR`=0x004, etc., until the end of the Calendar is reached, as defined by `SnkCalendar_Len`.

Based on Figure 3-3, page 34 of the Sink AXI4-Lite Memory Space, the valid address range for the calendar memory is from 0x000 to 0x1FC. Note that the calendar memory cannot be written to when the core is enabled (`SnkEn`=1). If this rule is violated and the write

operation to the calendar space has failed, SNK_BRESP_ERR[2] will assert. See Figure 5-9 for an example.

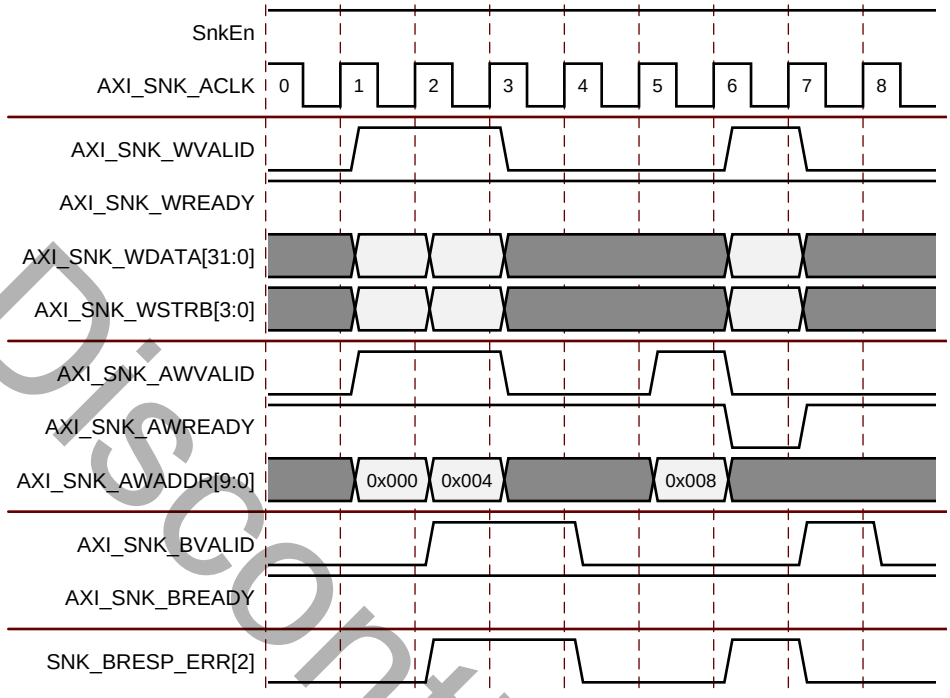

Figure 5-9: **Unsuccessful Calendar Initialization**

The waveform in Figure 5-10 illustrates the programming of the Sink Calendar. In this example, SnkCalendar_Len is set to eight and SnkCalendar_M is set to zero (indicating that the calendar length is nine, 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:

1. Status for channel 3
2. Status for channel 0
3. Status for channel 1
4. Status for channel 2
5. Status for channel 1
6. Status for channel 2
7. Status for channel 3
8. Status for channel 0
9. Status for channel 0

To verify what has been programmed into the calendar buffer, read the contents using the AXI4-Lite Read Data/Address Channels (AXI_SNK_R* and AXI_SNK_AR*).

See Appendix F, SPI-4.2 Calendar Programming for more examples of programming the SPI-4.2 Calendar.

Note: For a 1-channel system, it is not necessary to program the Calendar. By default, all locations are set to zero.

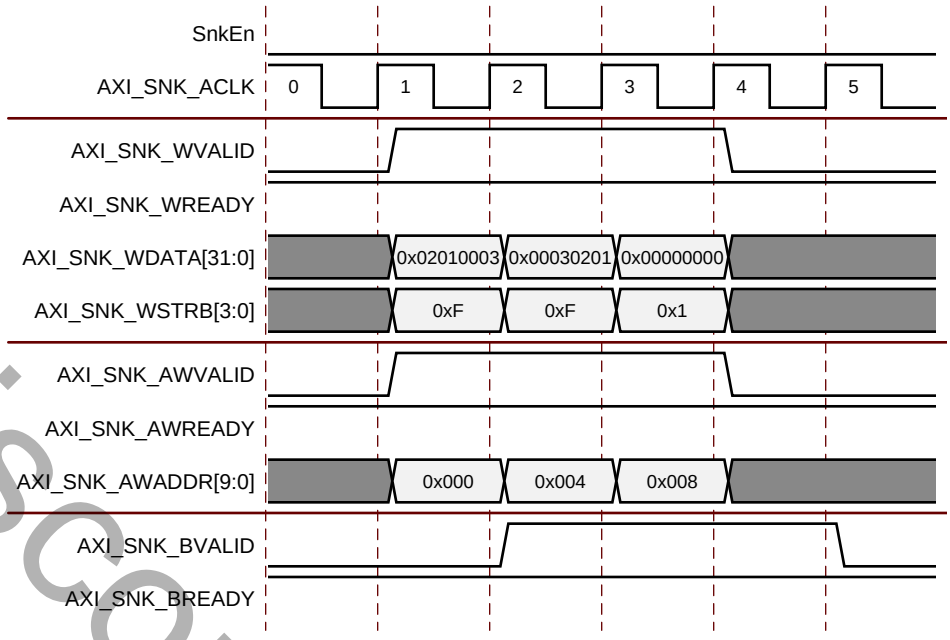

Figure 5-10: **Sink Calendar Initialization**

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 5-11. 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 AXI4-Lite Control interface has a 32-bit data bus for all channel configurations; for example, whether the core is configured for 4, 128, or 256 channels. This interface allows you to write the FIFO Status Channel data for 4 channels at a time (each channel uses 2 bits of each byte). There are 10 address lines (AXI_SNK_AWADDR[9:0]) for selecting which 4 channels you are accessing. Based on Figure Figure 3-3, page 34 of the Sink AXI4-Lite Memory Space, the valid address range for the flow control status memory is from 0x200 to 0x2FC. The status memory can be written to regardless of the core's state.

The latency between the user interface and SPI-4.2 Interface for the Sink Addressable Status Path is one `AXI_SNK_ACLK` cycle for the AXI4-Lite interface operation to occur, plus six `RSClk` cycles.

Configure the Sink core to write status for four channels per clock cycle. Use the `AXI_SNK_AWADDR` bus to select the four channels. The core supports configurations of 1-256 channels.

The four channels of Channel Status are addressed as follows:

`Bank 0: AXI_SNK_AWADDR[9:0]`=0x200 for channels 3 to 0

`Bank 1: AXI_SNK_AWADDR[9:0]`=0x204 for channels 7 to 4

`Bank 2: AXI_SNK_AWADDR[9:0]`=0x208 for channels 11 to 8

`Bank 3: AXI_SNK_AWADDR[9:0]`=0x20C for channels 15 to 12

...

`Bank 62: AXI_SNK_AWADDR[9:0]`=0x2F8 for channels 251 to 248

`Bank 63: AXI_SNK_AWADDR[9:0]`=0x2FC for channels 255 to 252

The status is mapped to the 32-bit bus as follows:

For Bank0: `AXI_SNK_AWADDR[9:0]`=0x200

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

`AXI_SNK_WDATA[8:7]` => Channel 1

`AXI_SNK_WDATA[16:15]` => Channel 2

`AXI_SNK_WDATA[24:23]` => Channel 3

Note: The status values default to "Satisfied" when the AXI4-Lite interface is reset.

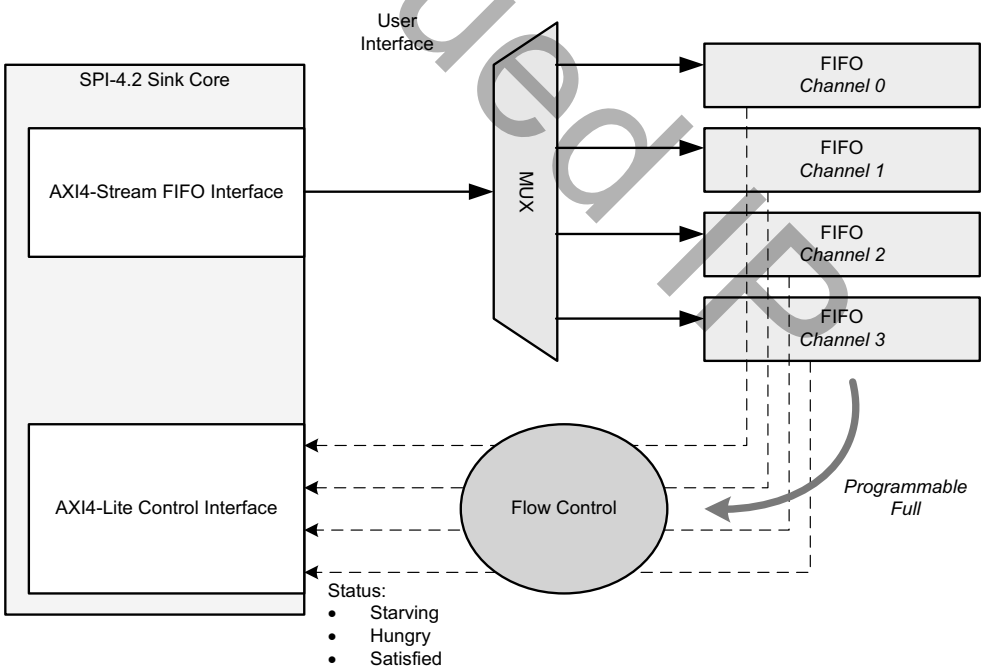

Figure 5-11: **Typical Flow Control Implementation for 4-Channel System**

Sink AXI4-Lite Control Interface: Status Example 1

Figure 5-12 illustrates writing to the Sink AXI4-Lite Control interface for a 4-channel SPI-4.2 Sink core. In this example, the strobe (AXI_SNK_WSTRB) functionality is utilized to only update the channel statuses that have changed. Since only the first four channels require status information, AXI_SNK_AWADDR is always driven to 0x200.

Each of the Sink AXI4-Lite Write Address and Write Data Channel's handshake signals (AXI_SNK_AWVALID/AWREADY and AXI_SNK_WVALID/WREADY) must be asserted simultaneously for one clock cycle for the status address and data to be updated in status memory. The status written in this example is shown in Table 5-4. The status data on AXI_SNK_WDATA[31:0] is represented in hexadecimal.

Table 5-4: **Status Example 1: Status Written via AXI4-Lite Interface**

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

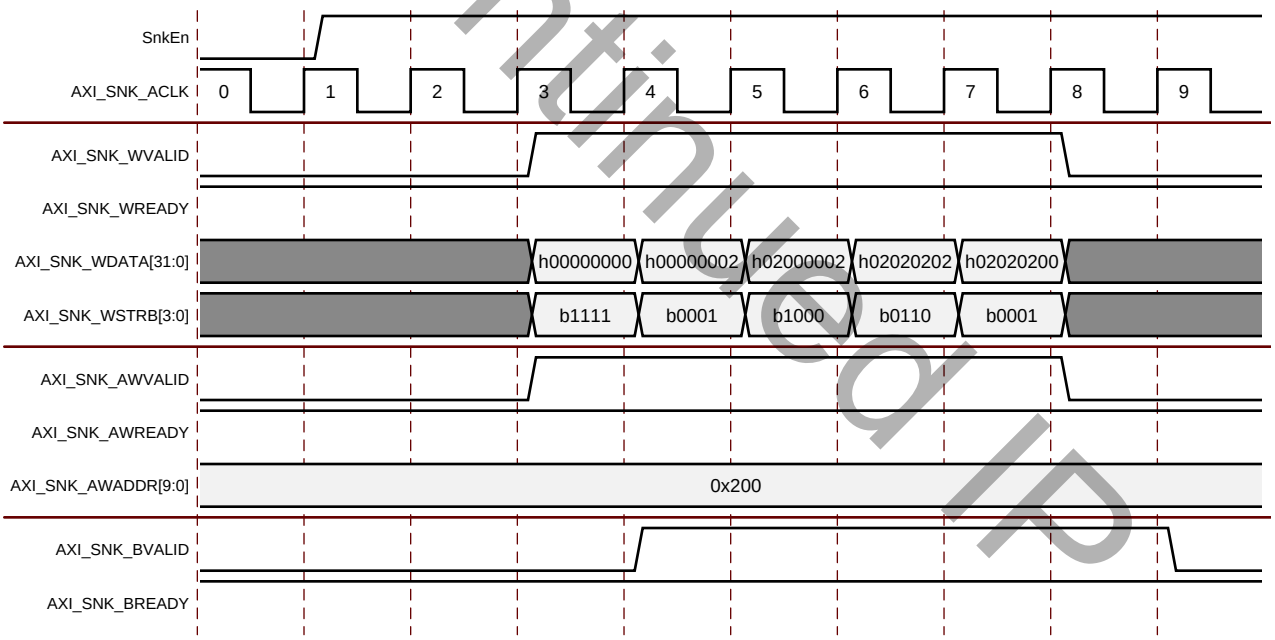

Figure 5-12: **Sink Status FIFO Write Example 1: 4-Channel Configuration**

Sink AXI4-Lite Control Interface: Status Example 2

Figure 5-13 illustrates writing to the Status FIFO Interface for a 64-channel SPI-4.2 Sink core. Address the following sixteen banks, depending on the status of the channel that is being updated, to write status for 64 channels:

- Bank 0: AXI_SNK_WRADDR[9:0]= 0x200, for channels 3 to 0

- `Bank 1: AXI_SNK_AWADDR[9:0]`= 0x204, for channels 7 to 4
- `Bank 2: AXI_SNK_AWADDR[9:0]`= 0x208, for channels 15 to 8
- `Bank 3: AXI_SNK_AWADDR[9:0]`= 0x20C, for channels 19 to 16

...

- `Bank 14: AXI_SNK_AWADDR[9:0]`= 0x2F8, for channels 59 to 56
- `Bank 15: AXI_SNK_AWADDR[9:0]`= 0x2FC, for channels 63 to 60

The mask (`AXI_SNK_WSTRB[3:0]`) is used to update only the channels for which FIFO status has changed. The status written is shown in Table 5-5.

Table 5-5: **Status Example 2: Status Written via AXI4-Lite Interface**

Clock Cycle	Status Address	Status Mask	Starving Status	Satisfied Status
4	Bank 0	1111	CH 0-3	none
5	Bank 0	0001	CH 1-3	CH 0
6	Bank 1	1000	none	CH 4-7
7	Bank 2	1111	CH 8-11	none
8	Bank 3	1111	CH 12-15	none
9	Bank 0	1000	CH 1, 2	CH 0, 3

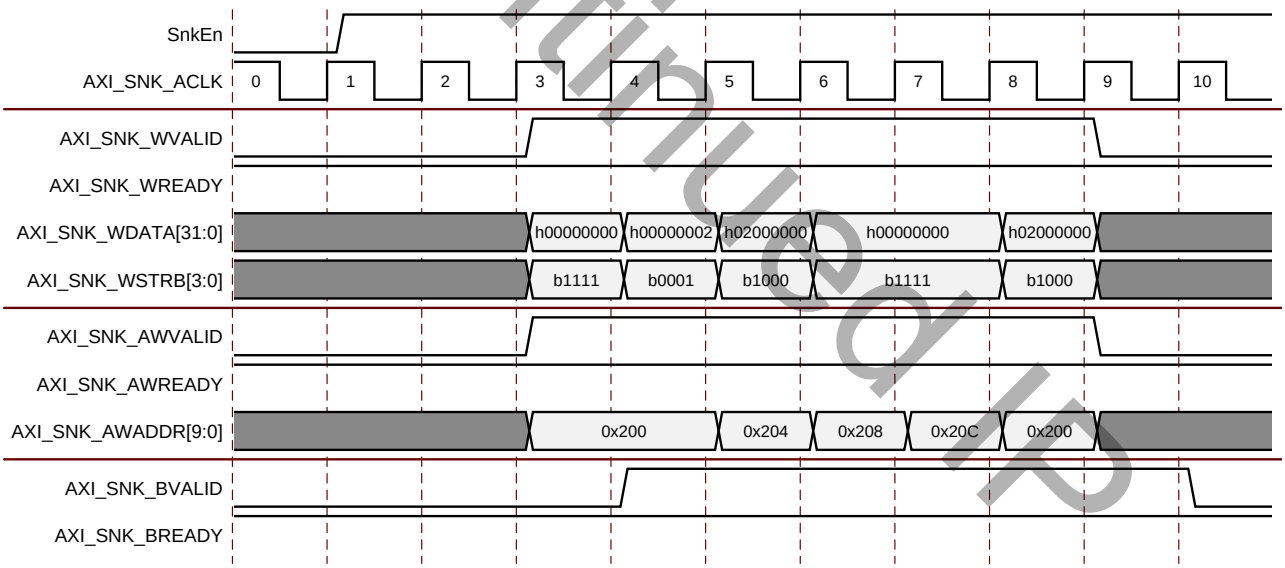

Figure 5-13: **Sink Status FIFO Write Example 2: 64-Channel Configuration**

Sink AXI4-Lite Control Interface: Status Example 3

This example illustrates status received on the user interface and written to the SPI-4.2 bus. Figure 5-14 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: 00, 00, 00, 00
- Sequence 2: 10, 00, 00, 00
- Sequence 3: 10, 00, 00, 10
- Sequence 4: 10, 10, 10, 10

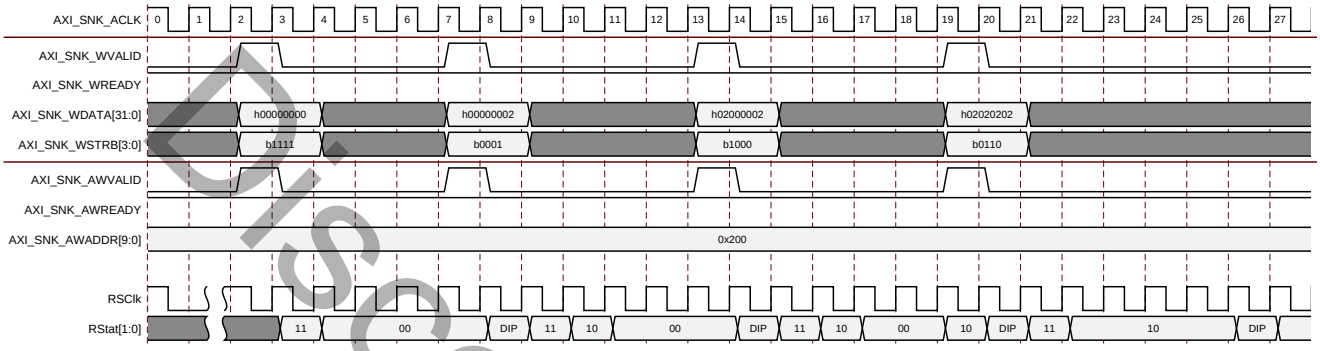

Figure 5-14: **Sink Status Path - User Interface to SPI-4.2 Interface**

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 Parameters

There are two ways to initialize the Sink Static configuration parameters. First is by setting the GUI parameters in the Vivado IP catalog, which initializes the registers that make up the static configuration parameter memory space. For more information, see Chapter 4, Customizing and Generating the Core. The other method is to initialize the static configuration parameters in-circuit at start-up.

For a full list of Sink static configuration parameters, see Table 3-5, page 37.

Initializing the Static Configuration Parameters In-Circuit

The Sink static configuration parameters are statically driven inside the core by writing registers through the AXI4-Lite Control interface. All Sink Static configuration parameters can 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 parameters:

1. Disable the sink core (`SnkEn` signal).
2. Assert core reset (`Reset_n` = 0).
3. Change the desired static configuration parameters by using the AXI4-Lite Control interface to write to the appropriate address space.
4. Deassert reset (`Reset_n`=1).
5. Wait at least 10 clock cycles of `RDClkDiv_User` for the sink static configuration parameters 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.

Note that the static configuration parameters can only be written to when the core is disabled and in reset state. If this is not the case, the write transaction will fail and the signal `SNK_BRESP_ERR[1]` will assert. See Figure 5-15 for an example of a failed write operation on static configuration memory.

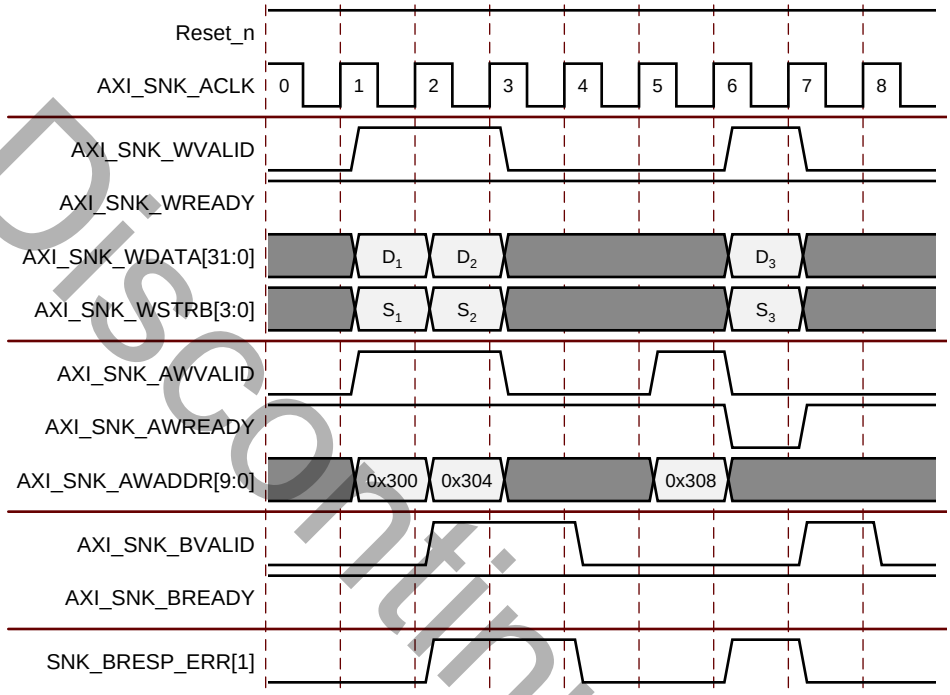

Figure 5-15: **Failed Write Operation on Static Configuration Memory**

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 parameter Sink FIFO in Almost Full Mode (`FifoAFMode[1:0]`). Figure 5-16 through Figure 5-18 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 `SNKFF_ALMOSTFULL_N` is deasserted and the Sink core

transitions back to in-frame. This is illustrated in Figure 5-16.

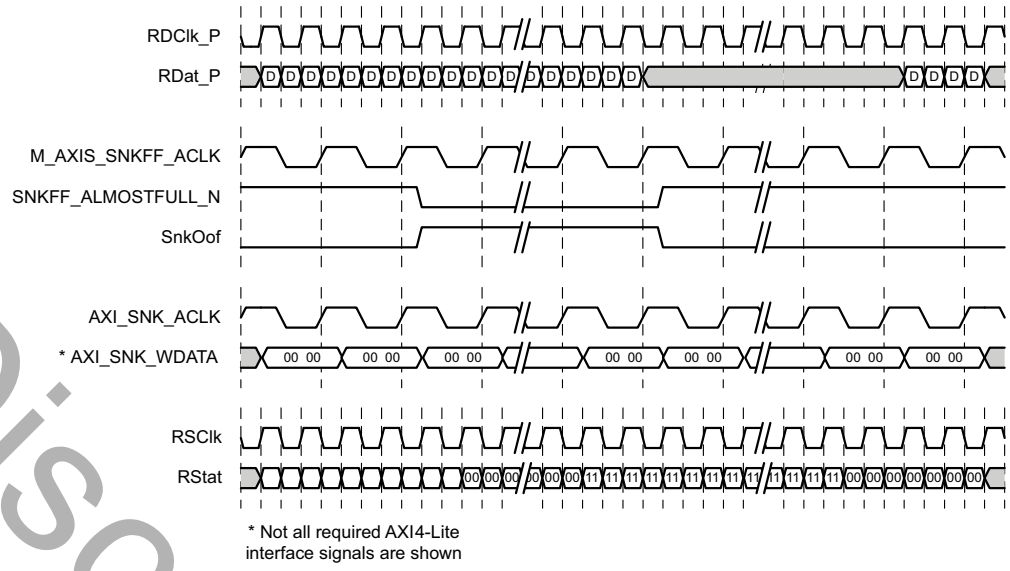

Figure 5-16: **FIFO Almost Full Mode "00**

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 `SNKFF_ALMOSTFULL_N` is deasserted. This is illustrated in Figure 5-17.

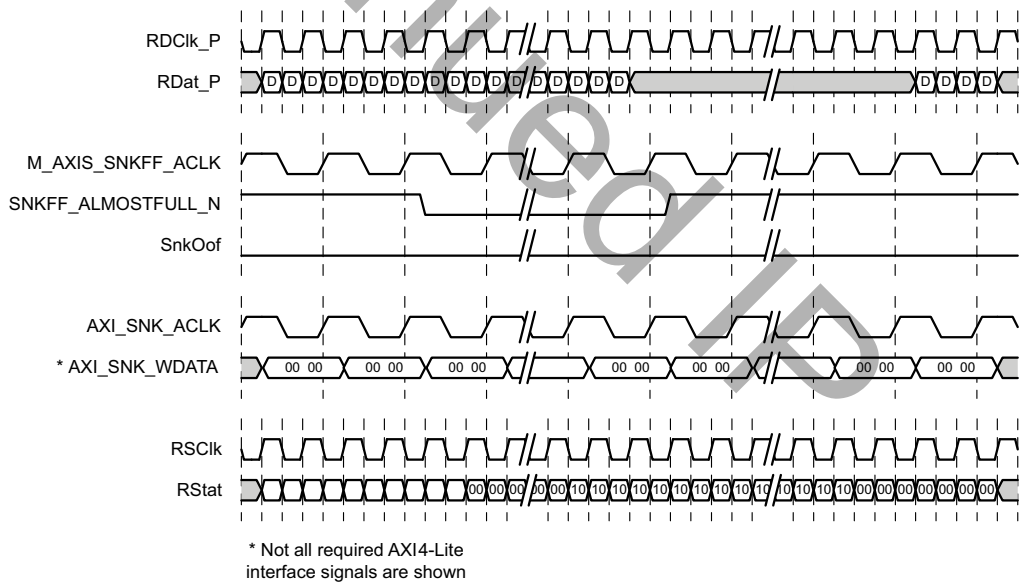

Figure 5-17: **FIFO Almost Full Mode "01"**

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 5-18). In this case, take immediate action to prevent FIFO overflow and loss of data.

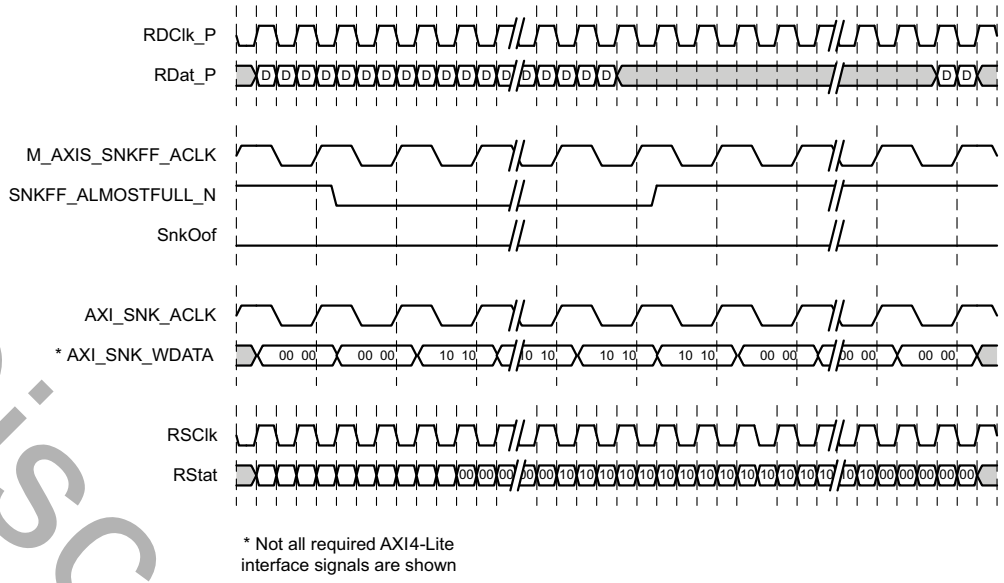

Figure 5-18: **FIFO Almost Full Mode "10" or "11"**

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-7 and Kintex-7 FPGA designs.

The IDELAY feature in Virtex-7 and Kintex-7 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

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 311 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).

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 5-6. 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 `SnkDPAPhaseAlignRequest` 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 `SnkDPAPhaseAlignRequest` is pulsed.

If `SnkDPAPhaseAlignRequest` is activated when the Sink logic is not receiving a SPI-4.2 training sequence or continuous training patterns (but the SPI-4.2 interface signals are toggling), the logic may fail to achieve the correct alignment. There is a remote possibility that `SnkDPAPhaseAlignComplete` could be asserted without being properly aligned.

If `SnkDPAPhaseAlignRequest` is activated when the Sink logic is only sending idles and none of the SPI-4.2 inputs are toggling (for example, the transmitting PHY is not powered), the DPA logic will not assert `SnkDPAPhaseAlignComplete` and SnkDPAFailed signal will assert.

If alignment is not achieved upon the completion of the alignment sequence, the `SnkDPAFailed` signal will assert. Cores configured with auto-retry will immediately initiate a new alignment sequence at this point and continue to attempt alignment unless `SnkDPAPhaseAlignRequest` is held high. The "auto-retry" features is only applicable after `SnkDPAPhaseAlignRequest` 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 `SnkDPAPhaseAlignRequest`.

For cores not configured with auto-retry, `SnkDPAFailed` is asserted at the end of each unsuccessful alignment attempt and will de-assert the next time `SnkDPAPhaseAlignRequest` is asserted. Note all DPA logic operates on `RDClk` input and if `RDClk` is not toggling, `SnkDPAPhaseAlignRequest` transitions will not initiate alignment.

Every time `SnkDPAPhaseAlignRequest` 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 `SnkDPAPhaseAlignRequest`. This behavior of `SnkDPAPhaseAlignRequest` enables the user to force to go out of frame (asserting `SnkDPAPhaseAlignRequest`), wait for the corresponding Transmit device to respond

with training patterns (hold `SnkDPAPhaseAlignRequest` high) and restart alignment when training patterns are received (deassert `SnkDPAPhaseAlignRequest`).

Table 5-6: **Dynamic Phase Alignment SIgnals**

Signal Name	Direction	Clock Domain	Description
SnkDPAPhaseAlignComplete	Output	RDClkDiv_User	**Phase Alignment Complete.** Active high signal that indicates phase alignment is complete.
SnkDPAPhaseAlignRequest	Input	RDClkDiv_User	**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	RDClkDiv_User	**Phase Alignment Failed**. Active high signal that indicates phase alignment has failed at the end of the alignment sequence.
SnkDPARamAddr [5:0]	Output	RDClkDiv_User	**Phase Alignment RAM Address.** Bus indicating the ISERDES tap value that corresponds to the data on SnkDPARamData.
SnkDPARamData [16:0]	Output	RDClkDiv_User	**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_User	**Phase Alignment RAM Valid**. Active high signal indicating the information on SnkDPARamData and SnkDPARamAddr is valid.
SnkCDPAHalt (optional)	Input	RDClkDiv_User	**Phase Alignment DPA Halt**. Active high signal that enables the user to halt the pointer adjustment of continuous DPA operation.
SnkDPADiagWin (optional)	Input	RDClkDiv_User	**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 the 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_User	**Phase Alignment DPA Address Reset.** Active high signal that clears the SnkDPARamAddr counter.
SnkDPAAddrEn (optional)	Input	RDClkDiv_User	**Phase Alignment DPA Address Enable.** Active high signal that enables the SnkDPARamAddr counter.
SnkDPAClkDlyRst (optional)	Output	RDClkDiv_User	**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.

Table 5-6: **Dynamic Phase Alignment SIgnals *(Cont'd)***

Signal Name	Direction	Clock Domain	Description
SnkDPAClkDlyCe (optional)	Output	RDClkDiv_User	**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.
SnkDPAClkDlyInc (optional)	Output	RDClkDiv_User	**Phase Alignment IDELAY Increment**. Active high signal used to increment the delay value 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.

All bits of the SPI-4.2 bus are 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 Vivado IP catalog 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 encouraged 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 standard 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 `SnkDPAPhaseAlignRequest` to `SnkDPAPhaseAlignComplete`). See Table 5-6 for more information. Additional debugging information is also available when invoking the DPA status monitoring feature. See DPA Status Monitoring, page 96.

The DPA also has the following advanced 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 Ports for 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 16) or at least as many IDELAY taps as correspond to 1 UI.

Continuous Alignment Considerations

This option is an enhancement that may be 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 SnkDPAPhaseAlignRequest after it has achieved alignment as indicated by the assertion of SnkDPAPhaseAlignComplete 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 is particularly 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 SnkDPAPhaseAlignComplete is asserted. SnkBusErrStat reverts back to its normal function as soon as SnkDPAPhaseAlignComplete 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 SnkDPADiagWin is pulsed, the DPA logic traverses the 32-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

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 Kintex-7 and Virtex-7 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 SnkDPAPhaseAlignRequest and SnkDPAPhaseAlignComplete signals are present but not used for the Static Alignment core. SnkDPAPhaseAlignRequest should be tied to a constant zero, and SnkDPAPhaseAlignComplete should be ignored.

Alignment Implementation Considerations for Global Clocking Configuration

The core supports legacy static alignment, which uses the MMCM to phase shift RDClk. The ability of the MMCM to shift the internal clock in small increments enables RDClk to be shifted relative to the sampled data. For statically-aligned systems, the MMCM output clock phase offset is a critical part of the system. The static alignment solution, using the 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 MMCM does not compensate for deviations in delay between bits.

The optimal MMCM setting (CLKOUT0_PHASE) needs to be determined 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 MMCM CLKOUT0_PHASE 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 MMCM CLKOUT0_PHASE value that will be effective across all hardware platforms. Xilinx also does not recommend that you attempt to determine the CLKOUT0_PHASE 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 CLKOUT0_PHASE 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. For information about finding the ideal phase shift value for your system, see the Xilinx SPI-4.2 Answer Record 16112.

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

Alignment Considerations for Regional Clocking Configuration

Static alignment can be performed using the IDELAY function when regional clock distribution is used. The ability of the IDELAY function to delay its input by small increments (78 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.

The optimal delay in the IDELAY function (IODELAY primitive) needs to be determined 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 `pl4_snk_clk.v[hd]` 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 311MHz to 500+ MHz `RDClk` frequency. The Static Alignment feature operates on systems ranging from the minimum MMCM frequency for the architecture to 350 MHz.

In terms of resources, the Dynamic Phase Alignment logic and its features add approximately 300 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 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 closely matched.

Synchronization and Startup

Once the Sink core has been initialized (Initializing the SPI-4.2 Core, page 68), the Sink core must be synchronized before data and status can be received and transmitted. Figure 5-19

shows a state machine diagram illustrating a Sink core startup sequence and error condition processing.

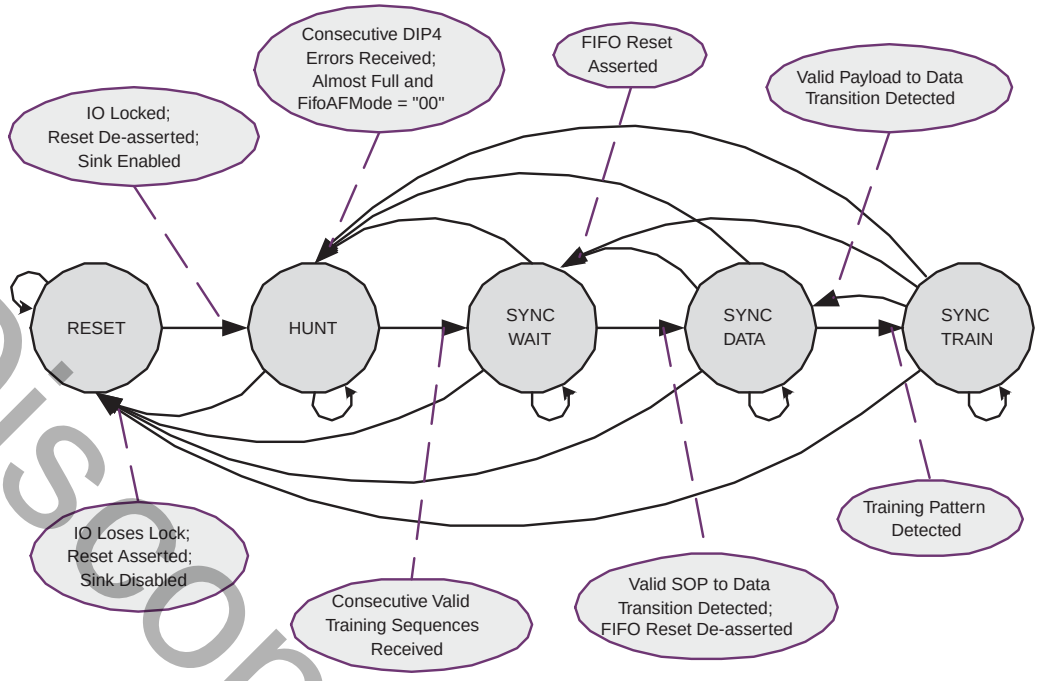

Figure 5-19: **Sink Startup Sequence State Machine**

Reset

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

- `All input clocks(RDClk0_User, RDClkDiv_User, M_AXIS_SRCSTAT_ACLK`, and others) are stable and `RDClksRdy_User` = 1
- `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 `SnkDPAPhaseAlignRequest` for two user clock cycles.
    - Wait for `SnkDPAPhaseAlignComplete` 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 static configuration 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.

- `M_AXIS_SNKFF_ARESETN` 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 (`SnkDPAPhaseAlignComplete` asserted).
- `Reset_n` is deasserted.
- `M_AXIS_SNKFF_ARESETN` is deasserted.
- `SnkEn` is asserted.
- `SnkOof` is deasserted (`NumTrainSequences` consecutive training patterns received).
- First valid SOP-to-data transition detected (after `SnkOof` or `M_AXIS_SNKFF_ARESETN` 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`.

- `SnkDPAPhaseAlignComplete` is deasserted.
  - If the I/O loses lock (`SnkDPAPhaseAlignComplete` = 0) the data written to the FIFO is corrupt and is immediately terminated.
  - `SnkDPAPhaseAlignRequest` 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.
- SNKFF_ALMOSTFULL_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 M_AXIS_SNKFF_ARESETN 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 SOPs 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 SNKFF_OVERFLOW_N asserting when SNKFF_ALMOSTFULL_N has not asserted. Any time that SNKFF_OVERFLOW_N is asserted, the integrity of the data in the FIFO is compromised and the FIFO must be reset.

Figure 5-20 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.

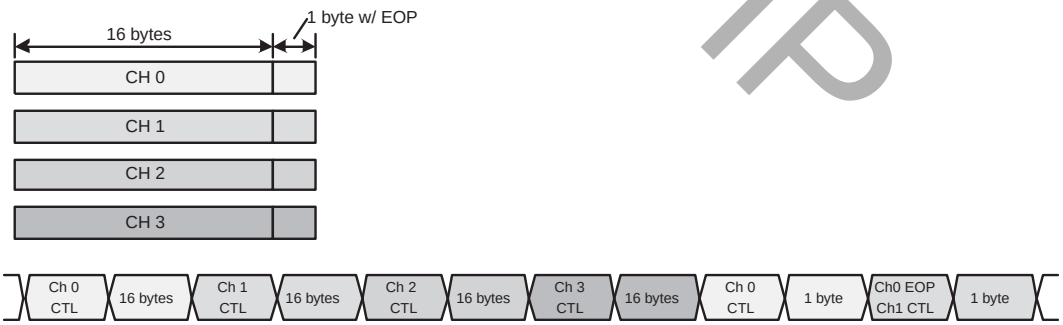

Figure 5-20: **Short Packet Support**

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 M_AXIS_SNKFF_BURSTERR. M_AXIS_SNKFF_BURSTERR 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 M_AXIS_SNKFF_TLAST or M_AXIS_SNKFF_ERR.

EOP Abort Handling

When an EOP abort is received, the Sink core asserts the output flags M_AXIS_SNKFF_TLAST and M_AXIS_SNKFF_ERR when the packet is terminated. In this case, the Sink core does not assert M_AXIS_SNKFF_BURSTERR.

Sink SPI-4.2 Bus Error and Sink Bus Error Status

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 M_AXIS_SNKFF_TDATA. 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 (SnkOof asserted). See DPA Status Monitoring, page 96. This feature is available if the

Enable DPA Status Monitoring, page 61 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 5-21.

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 5-21 illustrates this behavior.

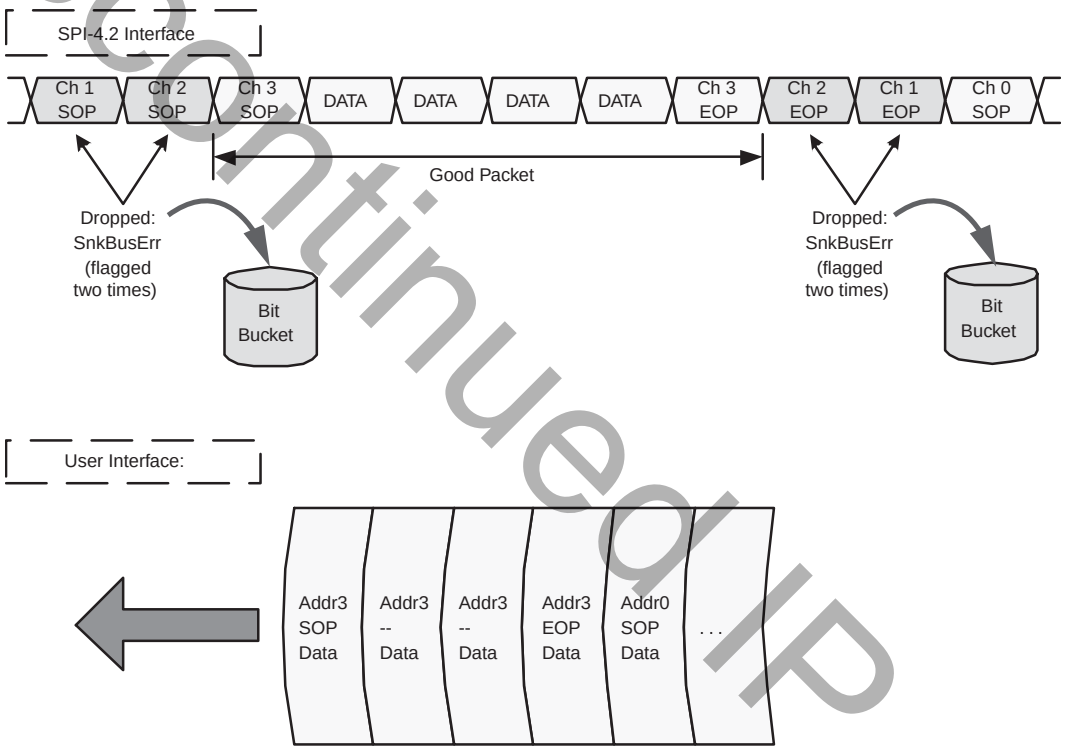

Figure 5-21: **Sequential Payload Control Word Example**

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 M_AXIS_SNKFF_DIP4ERR flag. Figure 5-22 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 M_AXIS_SNKFF_PAYLOADDIP4 flag. If the payload control word was also the end-of-burst control word for the previous packet, then

M_AXIS_SNKFF_DIP4ERR 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 M_AXIS_SNKFF_DIP4ERR, and each packet that was started with a DIP-4 error on M_AXIS_SNKFF_PAYLOADDIP4. This is illustrated in Figure 5-23 where packet 1 is flagged with a M_AXIS_SNKFF_DIP4ERR and packet 2 is flagged with M_AXIS_SNKFF_PAYLOADDIP4. Both DIP-4 errors are asserted at the end of the burst or packet.

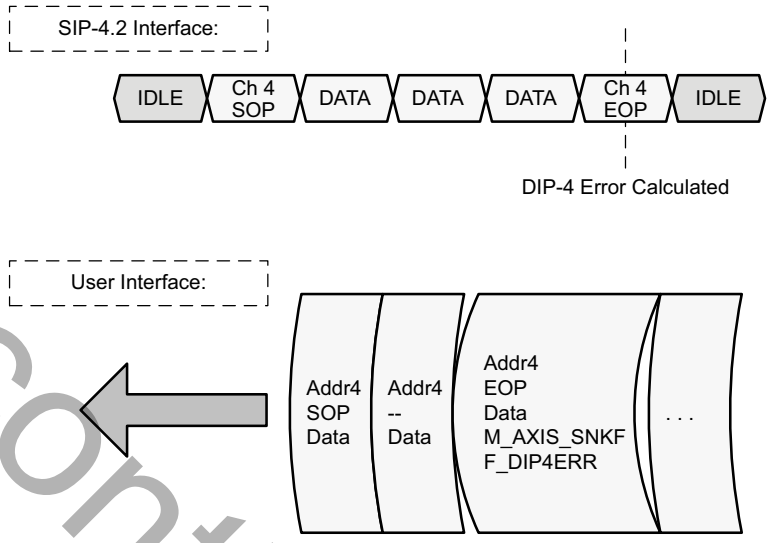

Figure 5-22: **Example of Error Flag M_AXIS_SNKFF_DIP4ERR**

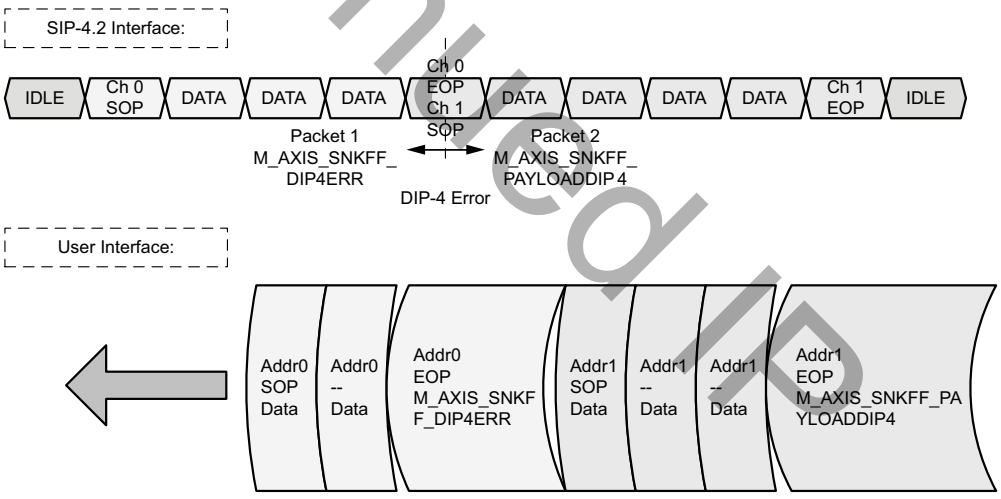

Figure 5-23: **Example of Error Flag M_AXIS_SNKFF_DIP4ERR and M_AXIS_SNKFF_PAYLOADDIP4**

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 M_AXIS_SNKFF_PAYLOADERR 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 5-24.

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

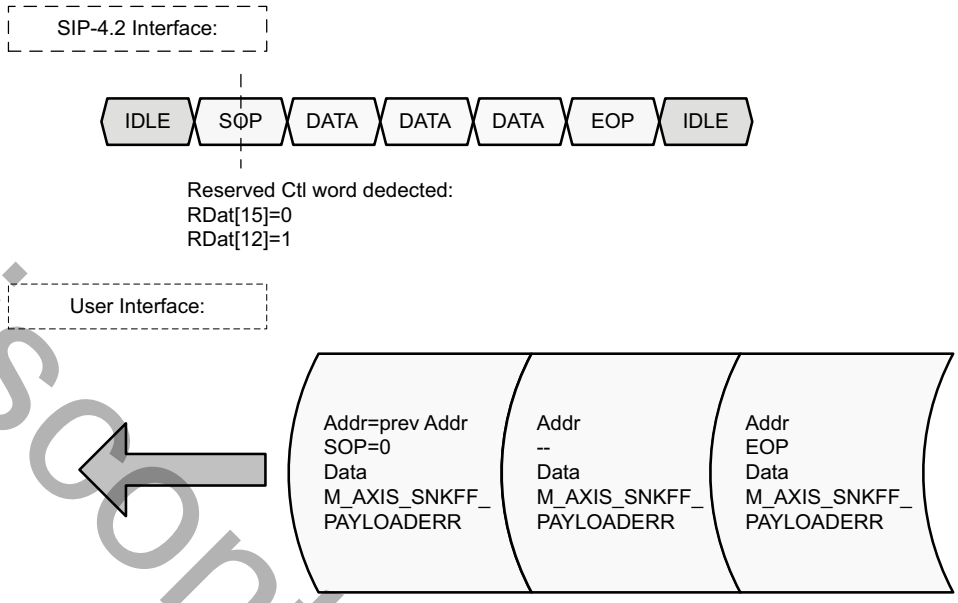

Figure 5-24: **Example of Error Flag M_AXIS_SNKFF_PAYLOADERR**

Source Core

Basic Operation

The Source core receives 64- or 128-bit data on the user interface and converts this to 16-bit data which is transferred across the SPI-4.2 interface. It also receives flow control information for the SPI-4.2 interface and processes it into 8-bit (4 channel) or 2-bit (1 channel) status words, depending on the enabled status FIFO interface (accessed on the user interface).

The following sections explain how the Source core operates. See Source Core Interfaces, page 40 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 800 Mbps SPI-4.2 data rate and a 64-bit user interface can write data into the Source core at 200 MHz. If a 128-bit user interface is used, data can be written into the Source core at 100 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 5-25. 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).

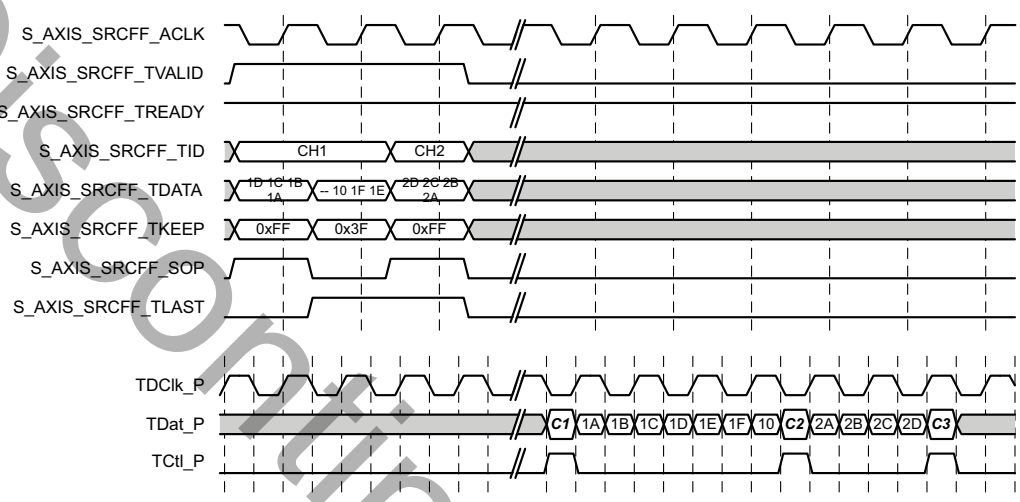

Figure 5-25: **Source Data Path: User Interface to SPI-4.2 Interface**

Source Data Path: Example 2

Figure 5-26 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 5-27.

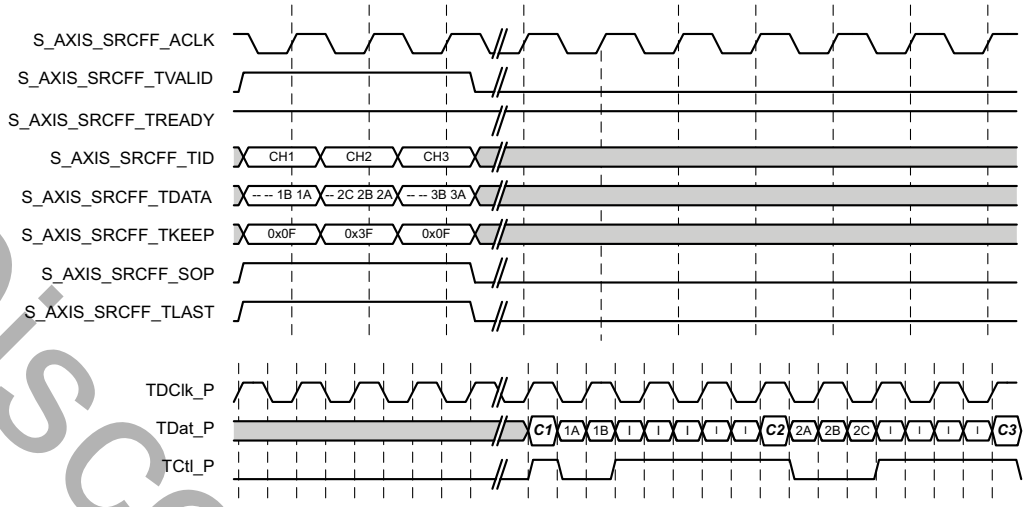

Figure 5-26: **Source Data Path - Minimum SOP Spacing Enforced**

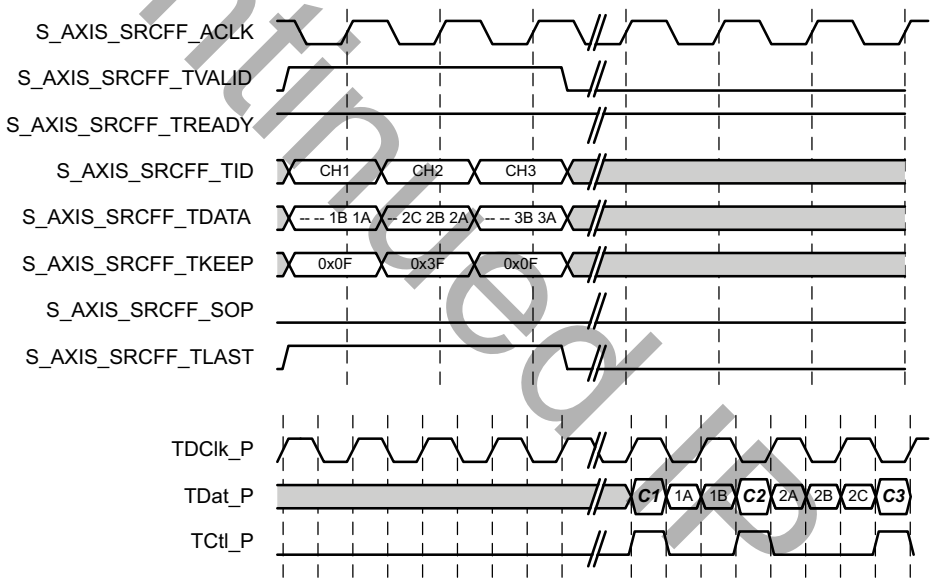

Figure 5-27: **Source Data Path - Short Packet Transfers**

The Source core formats the data to be written onto the SPI-4.2 bus (`TDat`). Table 5-7 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 preceding 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 5-8) and for a 64-bit user interface (Table 5-9).

Table 5-7: **Example of Formatting Source FIFO Data for a 64-bit User Interface**

Source AXI4-Stream Interface (to Source FIFO) S_AXIS_SRCFF_*			SPI-4.2 Source Data			
Data Written to the Source FIFO (*TDATA[63:0])	***ACLK cycle**	**FIFO Control Bit**	**Data Transmitted on the SPI-4.2 Interface (TDat [15:0])**		**TCtl**	**TDClk cycle**
[8E9F.A6B5.C4D3.E2F1]	1	*SOP = 1 *TLAST = 0 *TKEEP = 0xFF *TID = 0000.0010 *ERR = 0	N/A	N/A	N/A	n
			SOP	b:[1001.0000.0010.pppp]	1	n+1
			Word 0 (P0)	0xF1E2	0	n+2
			Word 1 (P1)	0xD3C4	0	n+3
[E8F9.6A5B.4C3D.2E1F]	2	*SOP= 0 *TLAST = 0 *TKEEP = 0xFF *TID = 0000.0010 *ERR = 0	Word 2 (P2)	0xB5A6	0	n+4
			Word 3 (P3)	0x9F8E	0	n+5
			Word 4 (P4)	0x1F2E	0	n+6
			Word 5 (P5)	0x3D4C	0	n+7
[0000.0000.0012.CDAB]	3	*SOP= 0 *TLAST=1 *TKEEP = 0x03 *TID = 0000.0010 *ERR = 0	Word 6 (P6)	0x5B6A	0	n+8
			Word 7 (P7)	0xF9E8	0	n+9
			Word 8 (P8)	0xABCD	0	n+10
			Word 9 (P9)	0x1200	0	n+11
	4		TLAST/ TKEEP	b:[0110.0000.0010.pppp]	1	n+12

Table 5-8: **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)	S_AXIS_SRCFF_SOP, S_AXIS_SRCFF_TID[7:0]	TDat[15] =1, TDat[12]=1, TDat[11:4] <== S_AXIS_SRCFF_TID[7:0]
New Burst (address change without SOP)	S_AXIS_SRCFF_TID[7:0]	TDat[15] = 1, TDat[12] = 0, TDat[11:4] <== S_AXIS_SRCFF_TID[7:0]
End of Packet (EOP, even bytes valid)	S_AXIS_SRCFF_TLAST, S_AXIS_SRCFF_TKEEP[15:0] When TDat[14:13] = 10: TKEEP = 0xFFFF if data bits 127-0 have valid data TKEEP = 0x3FFF if data bits 111-0 have valid data TKEEP = 0x0FFF if data bits 95-0 have valid data TKEEP = 0x03FF if data bits 79-0 have valid data TKEEP = 0x00FF if data bits 63-0 have valid data TKEEP = 0x003F if data bits 47-0 have valid data TKEEP = 0x000F if data bits 31-0 have valid data TKEEP = 0x0003 if data bits 15-0 have valid data	TDat[14:13] = 10
End of Packet (EOP, odd bytes valid)	S_AXIS_SRCFF_TLAST & S_AXIS_SRCFF_TID[15:0] When TDat[14:13] = 11: TKEEP = 0x7FFF if data bits 119-0 have valid data TKEEP = 0x1FFF if data bits 103-0 have valid data TKEEP = 0x07FF if data bits 87-0 have valid data TKEEP = 0x01FF if data bits 71-0 have valid data TKEEP = ox007F if data bits 55-0 have valid data TKEEP = 0x001F if data bits 39-0 have valid data TKEEP = 0x0007 if data bits 23-0 have valid data TKEEP = 0x0001 if data bits 7-0 have valid data	TDat[14:13] = 11
End of Packet (EOP, abort, error condition)	S_AXIS_SRCFF_ERR, S_AXIS_SRCFF_TLAST, S_AXIS_SRCFF_TKEEP[15:0]	TDat[14:13] = 01

Table 5-9: **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)	S_AXIS_SRCFF_SOP, S_AXIS_SRCFF_TID[7:0]	TDat[15] =1, TDat[12]=1, TDat[11:4] <== S_AXIS_SRCFF_TID[7:0]
New Burst (address change without SOP)	S_AXIS_SRCFF_TID[7:0]	TDat[15] = 1, TDat[12] = 0, TDat[11:4] <== S_AXIS_SRCFF_TID[7:0]

Table 5-9: **SPI-4.2 Control Word Mapping to 64-bit User Interface *(Cont'd)***

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)	S_AXIS_SRCFF_TLAST, S_AXIS_SRCFF_TKEEP[7:0] When TDat[14:13] = 10: TKEEP = 0xFF if data bits 63-0 have valid data TKEEP = 0x3F if data bits 47-0 have valid data TKEEP = 0x0F if data bits 31-0 have valid data TKEEP = 0x03 if data bits 15-0 have valid data	TDat[14:13] = 10
End of Packet (EOP, odd bytes valid)	S_AXIS_SRCFF_TLAST & S_AXIS_SRCFF_TKEEP[7:0] When TDat[14:13] = 11: TKEEP = 0x7F if data bits 55-0 have valid data TKEEP = 0x1F if data bits 39-0 have valid data TKEEP = 0x07 if data bits 23-0 have valid data TKEEP = 0x01 if data bits 7-0 have valid data	TDat[14:13] = 11
End of Packet (EOP, abort, error condition)	S_AXIS_SRCFF_ERR, S_AXIS_SRCFF_TLAST, S_AXIS_SRCFF_TKEEP[7:0]	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 3-8, page 43) 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 parameter `AlphaData[7:0]` (see Table 3-11, page 51) 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 parameter `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 3-8, page 43) 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 `S_AXIS_SRCFF_ERR` signal. When `S_AXIS_SRCFF_ERR` is asserted and `S_AXIS_SRCFF_TLAST` 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 5-29 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:

- 250 MHz in Kintex-7 FPGAs
- 312.5 MHz in Virtex-7 FPGAs

The Source user interface has four major signal groups:

- **Control and Status Signals**. Apply to the operation of the entire Source core.
- **AXI4-Stream FIFO Signals**. Writes data to the FIFO to be transmitted on the SPI-4.2 Interface.
- **AXI4-Stream Status Interface.** Provides flow control information about the SPI-4.2 link, and (in Transparent Status mode) provides status received on the SPI-4.2 Interface.
- **AXI4-Lite Control Signals**. Used to configure the calendar sequence, static configuration parameters, and (in Addressable Status mode) 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 3-8, page 43.

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 S_AXIS_SRCFF_TKEEP signal contains a zero without an EOP: In this case an EOP abort will not be asserted. When this occurs, the Source core will ignore the S_AXIS_SRCFF_TKEEP value and send the data word with TKEEP set to all ones.

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

- **Source Status Frame Error** (SrcStatFrameErr) 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 signal Reset_n provides a core-wide reset capability with the exclusion of the Source AXI4-Lite Control Interface, logic and memory space. Reset_n is used to restart the entire Source core, clear the data FIFO, and cause the interface to go out-of-frame. When Reset_n is deasserted, the Source core will initiate the synchronization startup sequence.

Source AXI4-Stream FIFO Interface Signals

The Source AXI4-Stream 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 3-9, page 45.

The Source FIFO Interface signals are synchronous to S_AXIS_SRCFF_ACLK, 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.

S_AXIS_SRCFF_ARESETN is used to clear the FIFO (and the associated data path logic) while remaining in-frame. When S_AXIS_SRCFF_ARESETN is deasserted, the Source core will send idle cycles until the user writes data into the FIFO. S_AXIS_SRCFF_ARESETN 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.

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 5-28 and Figure 5-29. 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 114.

Source FIFO Almost Full

Figure 5-28 shows the Almost Full response of the Source FIFO. The behavior of the Source FIFO Ready signal (`S_AXIS_SRCFF_TREADY`) is dependent on the static configuration parameters `SrcAFThresAssert` and `SrcAFThresNegate`. When `S_AXIS_SRCFF_TREADY` is de-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 `S_AXIS_SRCFF_TREADY` is re-asserted. Note that since `S_AXIS_SRCFF_TREADY` de-asserts when the FIFO is at the almost full threshold, thus not allowing further data to be written in, it is not possible to overflow the FIFO.

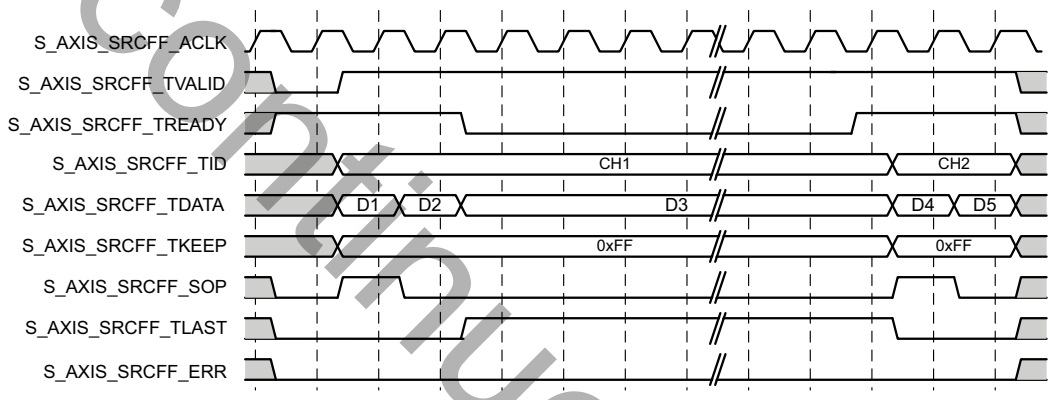

Figure 5-28: **Source FIFO Almost-full Condition**

Writing to the Source FIFO

The user may pause a transfer on a credit (16 bytes) boundary, as illustrated Figure 5-29. 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.

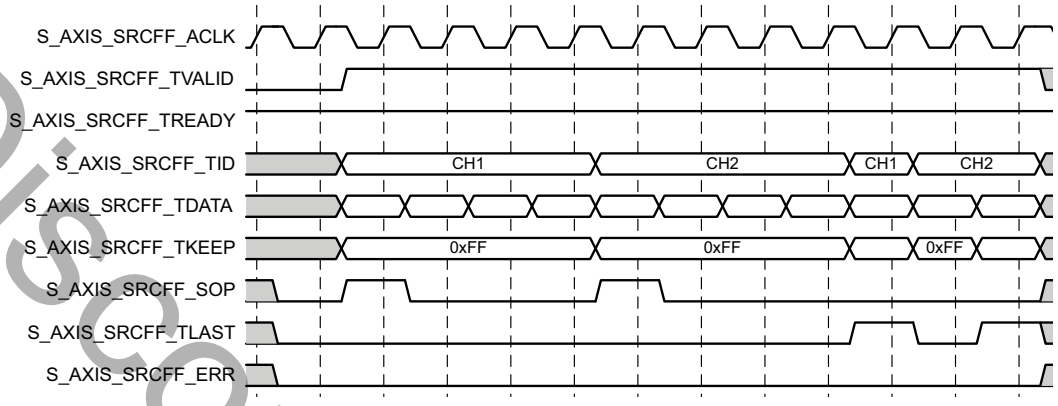

Figure 5-29: **Writing to the Source FIFO**

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 `S_AXIS_SRCFF_ERR` signal. When a `S_AXIS_SRCFF_TLAST` flag is asserted, `S_AXIS_SRCFF_ERR` is used to indicate that the current packet contains an error and causes the core to transmit an EOP abort with the packet. When `S_AXIS_SRCFF_TLAST` is not asserted, the assertion of `S_AXIS_SRCFF_ERR` causes the core to force the insertion of an EOP (1 byte or 2 bytes depending on `S_AXIS_SRCFF_TKEEP`) 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 `S_AXIS_SRCFF_SOP`.

Source AXI4-Stream FIFO Reset

The assertion of the reset signal clears the FIFO (and the associated data path logic) while remaining in-frame. The source core will insert an EOP abort if it was in the middle of processing a burst of data and start sending idles. When `M_AXIS_SRCFF_ARESETN` is deasserted, the Source data path will start sending data once it received a first complete burst of data. The AXI4-Stream FIFO Reset signal needs to be asserted at the minimum for three `S_AXIS_SRCFF_ACLK` cycles.

Source Flow Control

The source core receives flow control information for the SPI-4.2 interfaces (`TStat_P/N`, `TSClk`) and it can presented to the user in one of two ways:

- **Addressable Mode** (Source AXI4-Lite Control interface): In this mode, the Status information is stored in the AXI4-Lite Control memory space and can be retrieved at

the user's convenience. In this mode, users can retrieve an 8-bit (4 channel) status words in each read.

- **Transparent Mode** (Sink AXI4-Stream Status interface): In this mode, the Status information is provided real-time (with minimum latency) as it is received on TStat and is not stored within the core. This means the status memory space as defined in Figure 3-5, page 47 is not usable ("Reserved") in this mode and reads to it will return all zeroes.

For both these modes, the Source AXI4-Stream Status interface provides information on the channel number for the most recently updated SPI-4.2 channel status. The following sections will describe the different interfaces available in each of these modes.

Source AXI4-Stream Status Interface

The Source AXI4-Stream Status Interface provides flow control (status) information using the signal set described in Table 3-12, page 52.

When the core is used in Addressable Status mode, the interface provides the channel number for the most recently updated SPI-4.2 channel status on `M_AXIS_SRCSTAT_TID`, qualified by the `M_AXIS_SRCSTAT_TVALID` signal. The channel status itself must be accessed via the AXI4-Lite Control interface, with each read providing status for four channels.

When the core is used in Transparent Status mode, the `M_AXIS_SRCSTAT_TUSER[1:0]` bus is added. This port provides the SPI-4.2 status as received on TStat with minimal latency. In this mode, the `M_AXIS_SRCSTAT_TID` bus indicates the channel to which the current status shown on `M_AXIS_SRCSTAT_TUSER` belongs. Both `M_AXIS_SRCSTAT_TID` and `M_AXIS_SRCSTAT_TUSER` are qualified by `M_AXIS_SRCSTAT_TVALID.`

Behavior of the AXI4-Stream Status interface when the Source core is in-frame is shown below in Figure 5-30. The figure shows a 16-channel Source core programmed with a round-robin calendar sequence, where SrcCalendarLen = 15 and SrcCalendarM = 0.

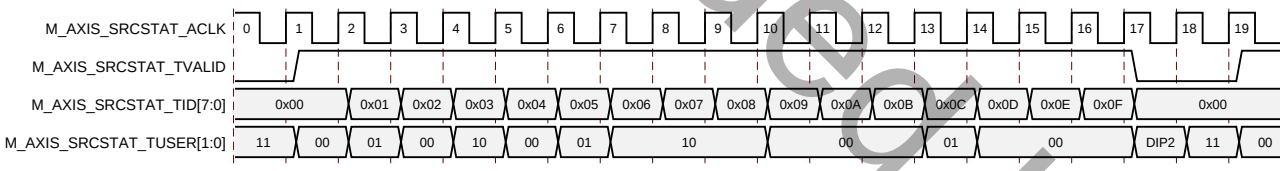

Figure 5-30: **Source AXI4-Stream Status Interface In-frame Behavior**

Source AXI4-Lite Control Interface

The Source AXI4-Lite Control interface allows the user to program the calendar memory, read status memory (in addressable mode), and program static configuration memory. This memory space is defined in Figure 3-5, page 47. A summary of the AXI4-Lite Control interface and definition is provided in Table 3-4, page 35.

Writing and updating the static configuration memory will define characteristics and behavior of the core. When the Source core is in Addressable Status mode, reading from the status memory provides access to the flow control data received on the SPI-4.2 Interface. A two-bit register is provided for each location in the calendar to store the channel status information (hungry=01, starving=00, satisfied=10).The channel order and frequency that status is received and updated is determined by the content of the calendar memory.

The Source AXI-4 Lite Control interface is divided into several groups: Read Address Channel, Read Data Channel, Write Address Channel, Write Data Channel, Write Response Channel and Sideband Error bus. With the exception of the Sideband error bus, all other signal groups require a handshake between their respective ready and valid signals to complete before an operation is performed. A read operation will first require the read address handshake (`AXI_SRC_ARVALID` and `AXI_SRC_ARREADY`) to complete before the data is available on `AXI_SRC_RDATA` and `AXI_SRC_RVALID`. To read out the data, `AXI_SRC_RREADY` needs to be asserted to complete the read data handshake. For a write operation to occur, both the address and data handshake must be completed. This will then generate a response on `AXI_SRC_BVALID` that must be read out by asserting `AXI_SRC_BREADY`. The write response is generated regardless of whether the write operation is successful. Note that this interface will only allow 6 outstanding write responses before the write response counter is full. Once the write response counter is full, no operations can be performed (write or read) on the AXI4-Lite Control Interface, and `SRC_BRESP_ERR[0]` will assert. No read or write operations can occur until at least one write response is drained from the core by asserting `AXI_SRC_BREADY`. See Figure 5-31 for the behavior of the write data, address and response channel.

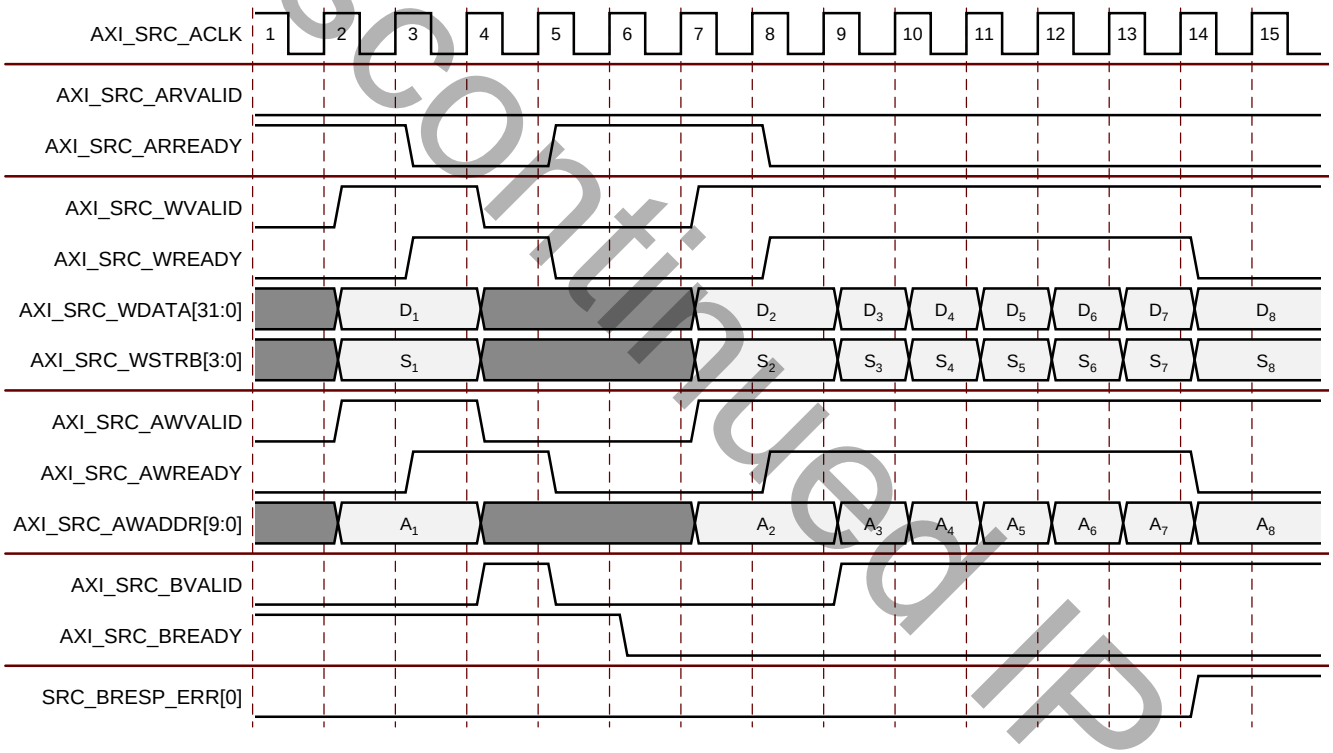

Figure 5-31: **Write Response Behavior of the Sink AXI Lite Control Interface**

If an error has occurred when executing the write or read operation, a bit on `SRC_BRESP_ERR[7:0]` will assert. If any address is accessed that is beyond the valid Source AXI4-Lite Memory Space, for example address space 0x314 to 0x3FC, `SRC_BRESP_ERR[3]` will assert. Reads to the reserved bits/bytes within the valid address space will return all zeroes. Writes to the reserved bits/bytes within the valid address space will generate a write response but will not have any affect.

Note: No AXI4-Lite write operations should occur before the fourth clock edge after the SrcEn or Reset_n signal toggles. Any write operations attempted prior to this clock edge will cause an error on the sideband error bus (SRC_BRESP_ERR[2:1]).

Although the read and write operations use separate channels, the two operations cannot be performed simultaneously. However, if a single read and write request occur simultaneously, the read operation will have precedence over the write in the Source AXI4-Lite Control Interface. However, continuous read or write operations take precedence over a single write or read request. Hence, if a read request occurs when the interface is already performing a write operation and another write operation is also requested, the write operation will take precedence. The read operation will only be executed once there are no outstanding write operations requested.

Figure 5-32 gives an example of consecutive read operations. Note that a read operation on a status or configuration memory takes one clock cycle while a read operation on a calendar memory takes two clock cycles before valid data is available. A continuous read to the status memory will provide valid data every other clock cycle.

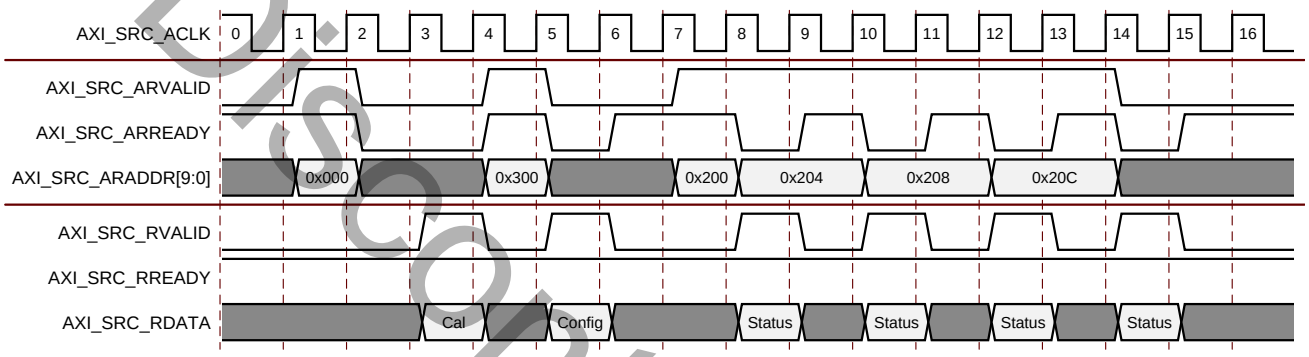

Figure 5-32: **Read Operation on the Source AXI4-Lite Control Interface**

The Source AXI4-Lite Control interface also has a reset signal `AXI_SRC_ARESETN`. This signal effects all the logic, data path and memory space associated with the interface. This signal requires a minimal pulse of two `AXI_SRC_ACLK` cycles. Both the calendar and static configuration memory space will revert to their default value when reset. These default values are chosen via loading a COE file (calendar memory space) and selecting values (static configuration parameters) in the Vivado IP catalog. See Chapter 4, Customizing and Generating the Core. The default value for all status channels is "10" or satisfied when in addressable mode.

Figure 5-33 demonstrates the behavior of the AXI-4 Lite Control interface signals when reset occurs.

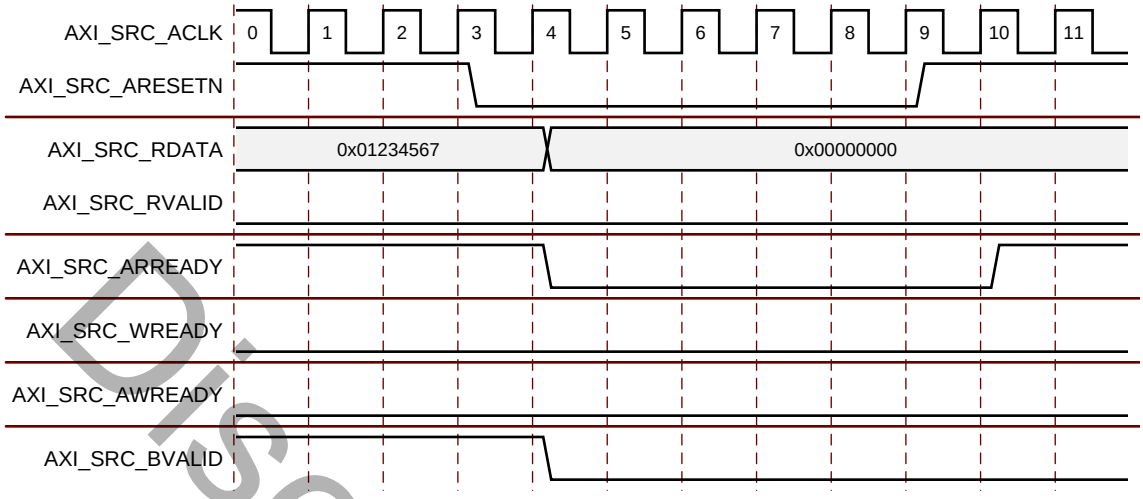

Figure 5-33: **Source AXI4-Lite Control Interface Reset Behavior**

A detailed description on how to access and program the different memory spaces using the AXI4-Lite Control interface will be illustrated in each subsection that follows. Note that although reading of the memory space is possible when the interface is not in reset state, there may be restrictions on when writing is permitted to each memory space.

Source Calendar Memory Initialization

There are two ways to initialize the Source calendar. First is by loading the COE file in the Vivado IP catalog, which loads the Calendar contents into the XDC file. For more information, see Chapter 4, Customizing and Generating the Core. The other method is to initialize the calendar in-circuit at startup.

Initializing the Calendar In-Circuit

At startup, the Source Calendar buffer can be programmed by first deasserting Source Enable (`SrcEn`), then using the Source AXI4-Lite interface to write the calendar sequence to memory. `AXI_SRC_AWADDR` is used to indicate the location in the calendar buffer, `AXI_SRC_WDATA` is used to indicate the channel number that should be written into that location, and `AXI_SRC_WSTRB` is used to indicate which of the possible four calendar entries should be written. This programming defines the sequence in which the status for each channel will be received. It is programmed identically to the device to which the Source core transmits data.

Based on Figure 3-5, page 47, the valid address range for the calendar memory is from 0x000 to 0x1FC. Note that the calendar memory cannot be written to when the core is enabled (`SrcEn`=1). If this rule is violated and the write operation to the calendar space has failed, `SRC_BRESP_ERR[2]` will assert.

The waveform in Figure 5-34 illustrates the programming of the Source Calendar. In this example, `SrcCalendar_Len` is set to eight and `SrcCalendar_M` is set to zero (indicating that the calendar length is nine, 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 1
- Status for channel 2
- Status for channel 3
- Status for channel 0
- Status for channel 0

To verify what has been programmed into the calendar buffer, read the contents using the AXI4-Lite Read Data/Address Channels (`AXI_SRC_R* and AXI_SRC_AR*`).

See Appendix F, SPI-4.2 Calendar Programming for more examples of programming the SPI-4.2 Calendar.

Note: For a one channel system, it is not necessary to program the calendar since by default all locations are set to zero.

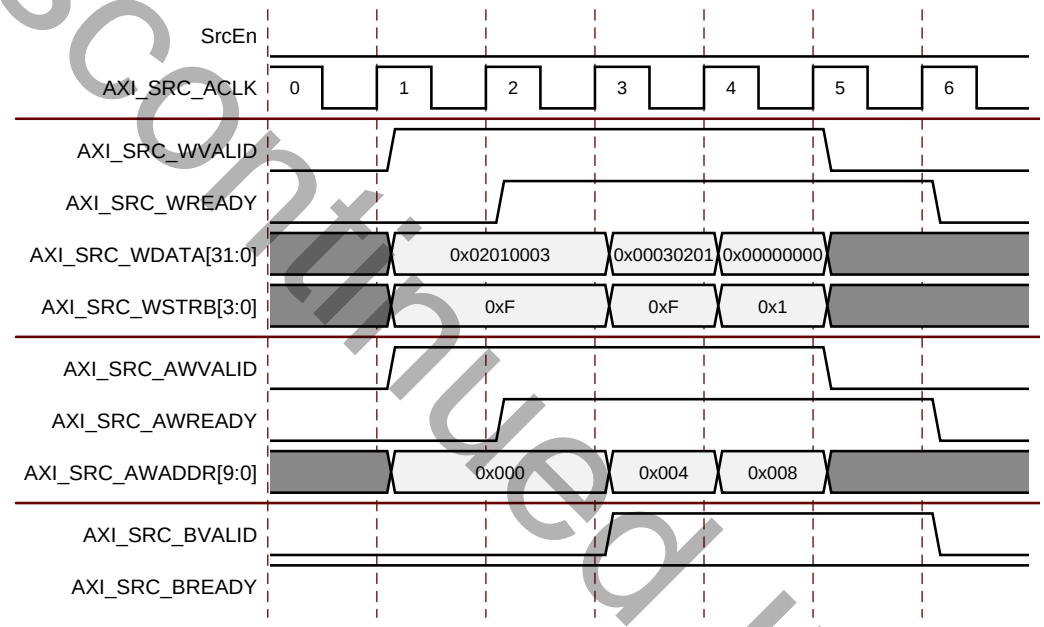

Figure 5-34: **Source Calendar Initialization**

Source Flow Control: Addressable Status Mode

The AXI4-Lite Control interface has a 32-bit bus for all channel configurations; for example, whether the core is configured for 4, 128, or 256 channels. This interface allows you to read the FIFO Status data for 4 channels at a time (uses two bits of each byte for each channel). There are 10 address lines (`AXI_SRC_ARADDR[9:0]`) for selecting which 4 channels you are accessing. Based on Figure 3-5, page 47, the valid address range for the

flow control status memory is from 0x200 to 0x2FC. A block diagram of how the Addressable Status memory processes the received SPI-4.2 Status is shown in Figure 5-35.

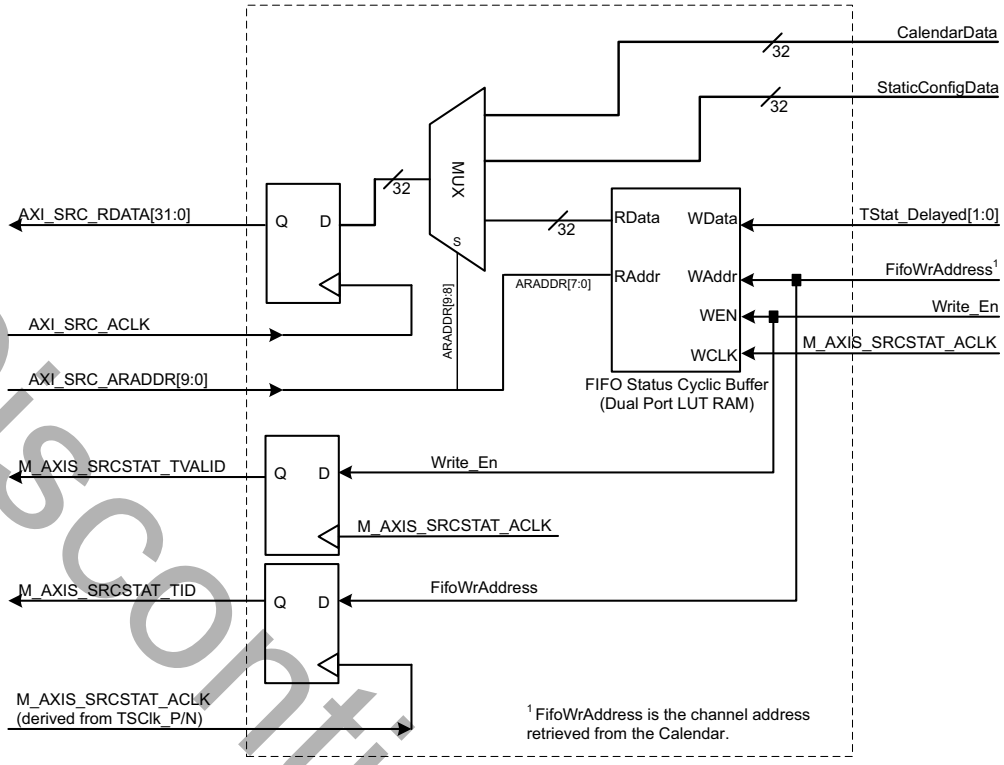

Figure 5-35: **Addressable Status Memory**

The minimum latency between the SPI-4.2 Status Interface and the AXI4-Lite Control interface for the Addressable Status Mode is nine `M_AXIS_SRCSTAT_ACLK` (derived from `TSClk_P/N`) cycles and one `AXI_SRC_ACLK` cycle for the AXI4-Lite read operation.

The four channels of Channel Status are addressed as follows:

`Bank 0: AXI_SRC_ARADDR[9:0]`=0x200 for channels 3 to 0

`Bank 1: AXI_SRC_ARADDR[9:0]`=0x204 for channels 7 to 4

`Bank 2: AXI_SRC_ARADDR[9:0]`=0x208 for channels 11 to 8

`Bank 3: AXI_SRC_ARADDR[9:0]`=0x20C for channels 15 to 12

...

`Bank 62: AXI_SRC_ARADDR[9:0]`=0x2F8 for channels 251 to 248

`Bank 63: AXI_SRC_ARADDR[9:0]`=0x2FC for channels 255 to 252

The status is mapped to the 32-bit bus as follows:

For Bank0: `AXI_SRC_ARADDR[9:0]`=0x200

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

`AXI_SRC_RDATA[8:7]` => Channel 1

`AXI_SRC_RDATA[16:15]` => Channel 2

`AXI_SRC_RDATA[24:23]` => Channel 3

Note: For the Source core using Addressable Status mode, the status memory is read-only--writes to the status memory will have no effect and will result in an error indicated on SRC_BRESP_ERR[3].

A typical user flow-control design is shown in Figure 5-36. 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.

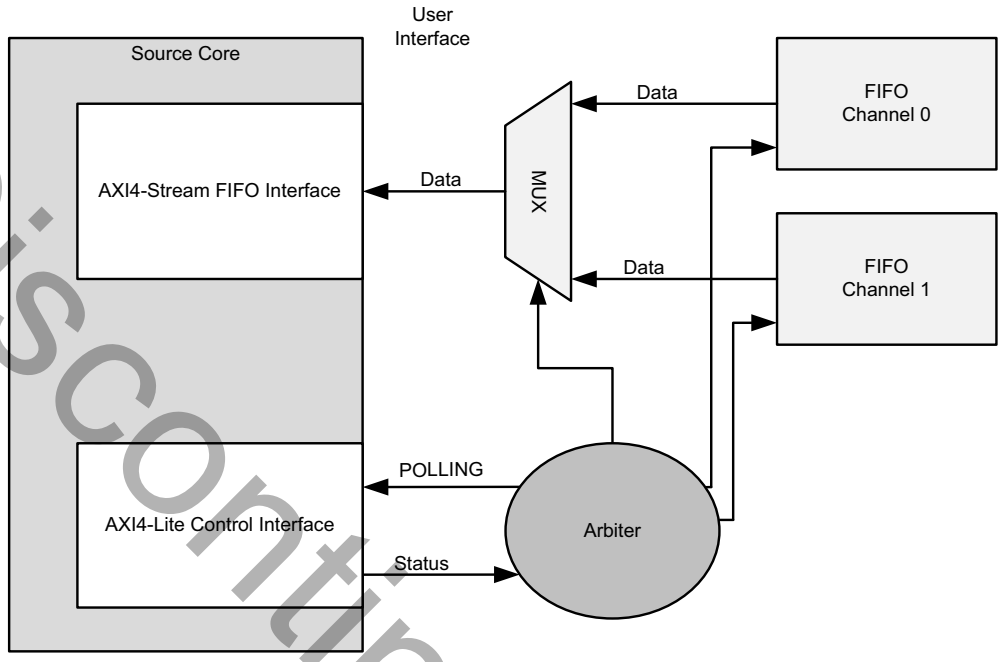

Figure 5-36: **Typical User Design Example**

The operation of the AXI4-Lite interface for reading Addressable Status is explained using two examples described below and illustrated in Figures 5-37 and 5-38

Addressable Status Mode: Example 1

Example 1 illustrates reading the Status Memory for a 4-channel SPI-4.2 Source core as shown in Figure 5-37. As there are only 4 channels, the Source AXI4-Lite Read Address bus (`AXI_SRC_ARADDR[9:0]`) will always be a value of 0x200 when reading status information. The completion of the Read Address Channel handshake (`AXI_SRC_ARREADY/ARVALID`) causes the core to fetch the contents of the memory at the address provided on `AXI_SRC_ARADDR[9:0]`. The interface will respond by placing the requested data on `AXI_SRC_RDATA` and asserting `AXI_SRC_RVALID` on the next clock edge. The user logic accepts the data by asserting the `AXI_SRC_RREADY` signal to complete the Read Data Channel handshake. This way, the status information for 4 channels can be read every other clock cycle.

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.

Table 5-10: **Status Example 1: Status Read via AXI4-Lite Interface**

Clock Cycle	Starving Status	Satisfied Status
2-3	CH 0-3	none
4-5	CH 0-3	none

Table 5-10: **Status Example 1: Status Read via AXI4-Lite Interface**

Clock Cycle	Starving Status	Satisfied Status
6-7	CH 1-3	CH 0
8-9	CH 2-3	CH 0-1
10-11	CH 3	CH 0-2

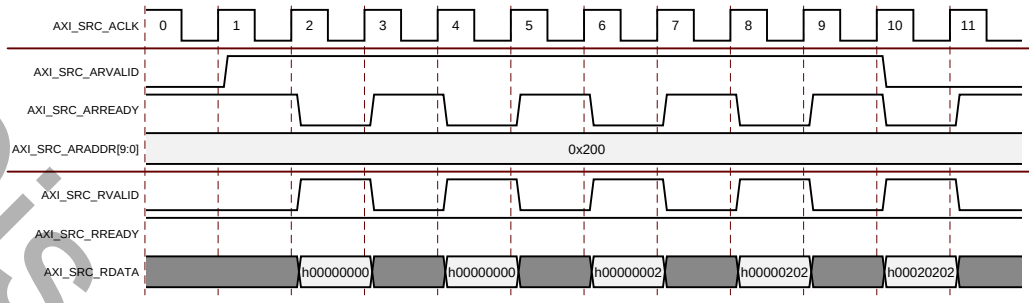

Figure 5-37: **Addressable Status FIFO Read: 4-Channel Configuration**

Addressable Status Mode: Example 2

Example 2 illustrates reading the Status Memory for a 256-channel SPI-4.2 Source core and is shown in Figure 5-42. To read the status for 256 channels, you must address all 64 banks depending on which channel status is being read.

- Bank 0: `AXI_SRC_ARADDR[9:0]` = 0x200, for channels 3 to 0
- Bank 1: `AXI_SRC_ARADDR[9:0]` = 0x204, for channels 7 to 4
- Bank 2: `AXI_SRC_ARADDR[9:0]` = 0x208, for channels 11 to 8
- Bank 3: `AXI_SRC_ARADDR[9:0]` = 0x20C, for channels 15 to 12
- ...
- Bank 63: `AXI_SRC_ARADDR[9:0]` = 0x2FC, for channels 255 to 252

The status read in the example shown in Figure 5-38 is summarized in the table below.

Table 5-11: **Status Example 2: Status Read via AXI4-Lite Interface**

Clock Cycle	Status Address	Starving Status	Satisfied Status
2-3	Bank 63	CH 252-255	None
4-5	Bank 0	CH 0-3	None
6-7	Bank 63	CH 253-255	CH 252
8-9	Bank 0	CH 0-1	CH 2-3
10-11	Bank 63	CH 252	CH 253-255

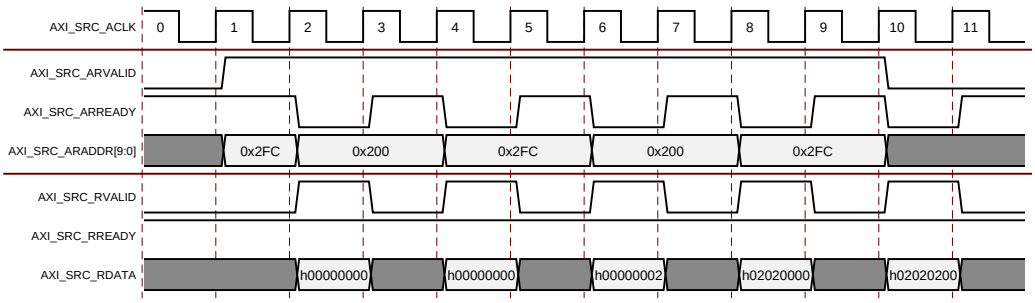

Figure 5-38: **Addressable Status FIFO Read: 256-Channel Configuration**

The Source AXI4-Stream Status Interface signal `M_AXIS_SRCSTAT_TID[7:0]` indicates which channel status was last updated on the SPI-4.2 Interface and is qualified by the AXI4-Stream Status Interface Valid signal (`M_AXIS_SRCSTAT_TVALID`=1). The `M_AXIS_SRCSTAT_TVALID` signal enables `M_AXIS_SRCSTAT_TID[7:0]`to be encoded, and the valid signal is disabled when the core processes a received DIP-2 or framing word.

While the `M_AXIS_SRCSTAT_TID` bus can be used as an indicator for which channel's status was most recently updated, the recommended and simplest method of reading status from the AXI4-Lite Control interface is a round-robin approach that reads the status for each bank of channels at least once per Calendar sequence. This is the method used in this example.

Source Flow Control: Transparent Status Mode

When using the Transparent Status mode the status is not read via the AXI4-Lite Control interface, but rather is provided via an additional 2-bit signal (`M_AXIS_SRCSTAT_TUSER`) in the Source AXI4-Stream Status Interface. For this mode, you are presented with the current status received on the SPI-4.2 Interface. The 2-bit status provided on `M_AXIS_SRCSTAT_TUSER[1:0]` is presented alongside a corresponding channel address (`M_AXIS_SRCSTAT_TID[7:0]`) and is qualified with the valid signal `M_AXIS_SRCSTAT_TVALID`. Unlike the Addressable Mode, the Transparent mode does not store the received status in memory. This means it is not possible to access the status of a specific channel, but the status will come in real-time as it is received by the Source core.

Note: For the Source core in Transparent Status mode, the Status memory portion of the AXI4-Lite memory space does not exist. Reads will return all zeroes and indicate an error on SRC_BRESP_ERR[4], and writes will have no effect and result in an error indicated on SRC_BRESP_ERR[4].

Figure 5-39 is a block diagram of how the Transparent Mode core processes the received SPI-4.2 FIFO Status. The minimum latency between the SPI-4.2 Status Interface and the AXI4-Stream Status interface for the Transparent Status Mode is five `M_AXIS_SRCSTAT_ACLK` cycles.

Figure 5-40 illustrates the output of the Transparent Mode AXI4-Stream Status Interface for a 256-channel configuration. On each clock cycle, the status (`M_AXIS_SRCSTAT_TUSER[1:0]`) and its corresponding channel (`M_AXIS_SRCSTAT_TID[7:0]`) is presented. The Source Status and channel address are only valid when `M_AXIS_SRCSTAT_TVALID` is asserted (equal to one). When the `M_AXIS_SRCSTAT_TVALID` signal is deasserted (equal to zero), the interface is receiving

DIP-2 or framing and the data on `M_AXIS_SRCSTAT_TUSER[1:0]` is not valid. All Source AXI4-Stream Status Interface signals are synchronous to `M_AXIS_SRCSTAT_ACLK`.

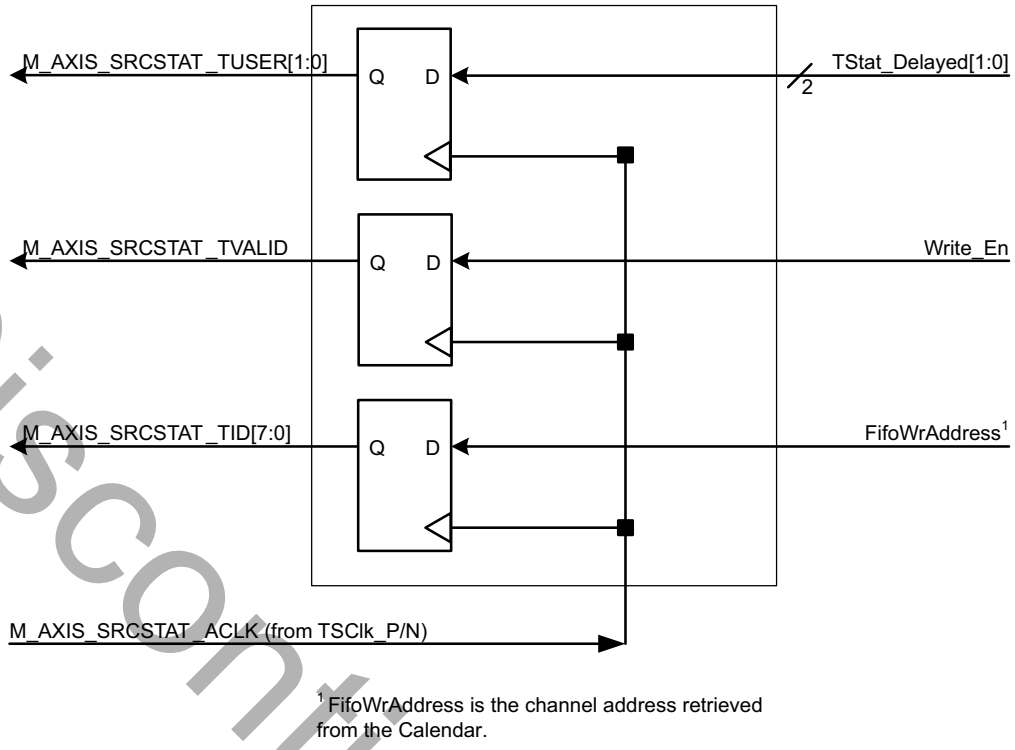

Figure 5-39: **Transparent AXI-4 Stream Status Interface Block Diagram**

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

Table 5-12: **Transparent mode: Status Read via AXI4-Stream Status Interface**

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

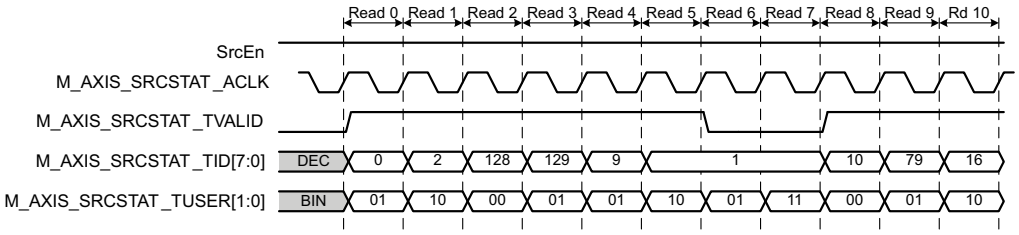

Figure 5-40: **Transparent Mode: 256-channel Configuration**

Source Static Configuration Parameters

There are two ways to initialize the Source Static configuration parameters. First is by setting the GUI parameters in the Vivado IP catalog, which initializes the registers that make up the static configuration parameter memory space. For more information, see Chapter 4, Customizing and Generating the Core. The other method is to initialize the static configuration parameters in-circuit at start-up.

For a full list of Source static configuration parameters, see Table 3-11, page 51.

Initializing the Static Configuration Parameters In-Circuit

The Source static configuration parameters are statically driven inside the core by writing registers through the AXI4-Lite Control interface. All Source Static configuration parameters can 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 parameters:

1. Disable the source core (`SrcEn` signal).
2. Assert core reset (`Reset_n` = 0).
3. Change the desired static configuration parameters by using the AXI4-Lite Control interface to write to the appropriate address space.
4. Deassert reset (`Reset_n`=1).
5. Wait at least 10 clock cycles of `SysClkDiv_User` for the source static configuration parameters 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.

Note that the static configuration parameters can only be written to when the core is disabled and in reset state. If this is not the case, the write transaction will fail and the signal `SRC_BRESP_ERR[1]` will assert.

Source Burst Mode

Source Burst Mode (`SrcBurstMode`) is a static configuration parameter 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 (`S_AXIS_SRCFF_TLAST`.)

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 are written than SrcBurstLen, then that data will be written out in bursts the size of

SrcBurstLen, but only after `S_AXIS_SRCFF_VALID` has been de-asserted. No data is sent out until you have written more credits then the setting in SrcBurstLen.

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 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 parameter `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. The Source only transmits the packet after 2 credits of data are written into the FIFO.

Figure 5-41 illustrates the behavior of the Source core when `SrcBurstMode`=1.

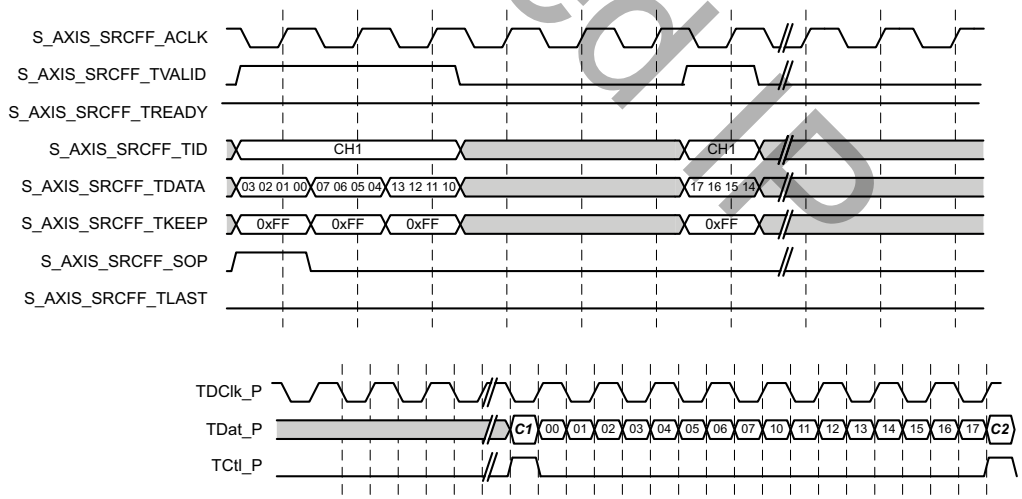

Figure 5-41: **Example Of Source Burst Mode = 1**

Synchronization and Startup

Once the Source core has been initialized (Initializing the SPI-4.2 Core, page 68), the Source core must be synchronized before data and status can be received and transmitted. Figure 5-42 is a state-machine diagram illustrating the Source core startup and error condition processing.

RESET

The Source core remains in the reset state until all input clocks are stable (`SysClkDiv_User, SysClk0_User, etc`) 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.

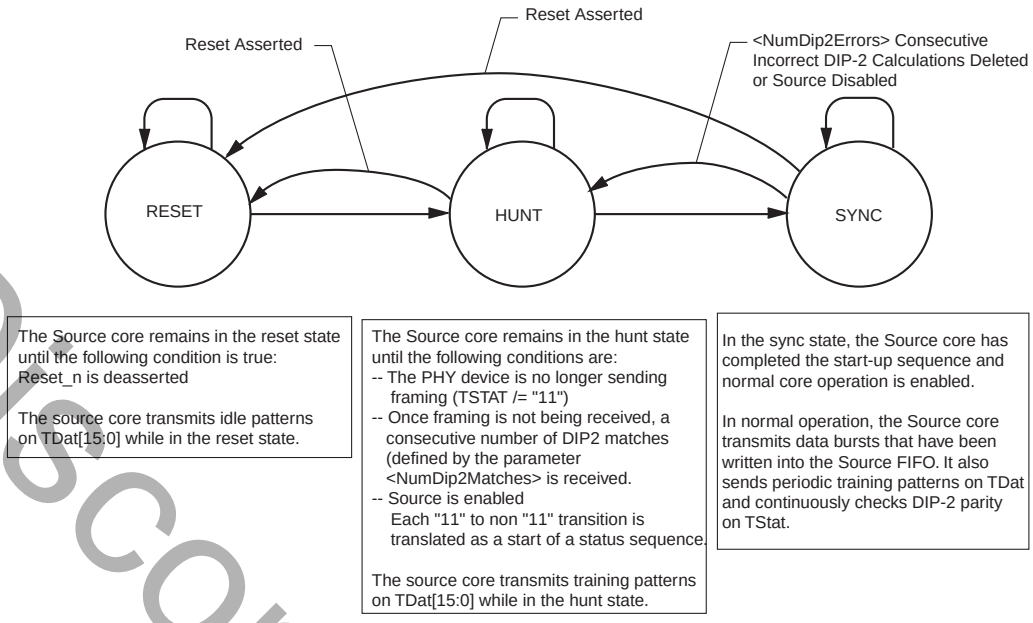

Figure 5-42: **Source Startup Sequence State Machine**

Error Handling

This section describes how the Source 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 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 the user status interface.

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: If S_AXIS_SRCFF_ARESETN 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 S_AXIS_SRCFF_TKEEP signal contains a zero without an EOP**. In this case, an EOP abort will not be inserted. When this occurs, the Source core will ignore the `S_AXIS_SRCFF_TKEEP` value and send the data word with TKEEP set to all ones.

  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 zeroes, the Source core sets unused bytes to zero (as required by the SPI-4.2 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 (XDC).

In this chapter, `<core_instance_name>` indicates the instance name used to instantiate the core 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 XDC the cores are instantiated in a top-level wrapper file as `<component_name>_core`. In this context, `<component_name>` is the name given by the user in the Vivado Design Suite SPI-4.2 GUI.

Overview

The Kintex™-7 and Virtex®-7 FPGAs with 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 resources further extend 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 XDC 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 XDC or specific constraints, see the *Xilinx Libraries Guide* and/or *Development System Reference Guide*.

SPI-4.2 Core Constraints

Sink Core Required Constraints

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.

Timespecs for Clocks

These constraints specify the frequency and duty-cycle of the clock signals. 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.

```
create_clock -period 2.857 -name RDClk_P -add [get_ports RDClk_P]

set_multicycle_path -from [get_cells -of_objects [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/EnRSClk_
int]] -to [get_cells -of_objects [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/EnRSClk_
int]] 2
```

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.

```
create_clock -period 11.425 -name AxiLiteClk -add [get_ports
AxiLiteClk]

create_clock -period 5.714 -name LoopbackClk -add [get_ports
LoopbackClk]
```

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

```
create_clock -add -name AXI_SNK_ACLK -period  11.425 [get_ports
AXI_SNK_ACLK]

create_clock -add -name M_AXIS_SNKFF_ACLK -period 5.714 [get_ports
M_AXIS_SNKFF_ACLK]
```

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.

```
set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/*core0/queue0/FifoRes
et_snkclk]] -to [get_cells -hierarchical * -filter {LIB_CELL == FDPE
||LIB_CELL == FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
```

```
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/*core0/queue0/FifoRes
et_snkffclk]] -to [get_cells -hierarchical * -filter {LIB_CELL == FDPE
||LIB_CELL == FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/*core0/pre0/FifoReset
_snkclk]] -to [get_cells -hierarchical * -filter {LIB_CELL == FDPE
||LIB_CELL == FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/clkdomain0/snk_clk_rs
t]] -to [get_cells -hierarchical * -filter {LIB_CELL == FDPE ||LIB_CELL
== FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/clkdomain0/snk_ff_clk
_rst]] -to [get_cells -hierarchical * -filter {LIB_CELL == FDPE
||LIB_CELL == FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/clkdomain0/snk_alite_
clk_aliterst] ] -to [get_cells -hierarchical * -filter {LIB_CELL ==
FDPE ||LIB_CELL == FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/Reset_snkclk]] -
to [get_cells -hierarchical * -filter {LIB_CELL == FDPE ||LIB_CELL ==
FDCE }] 5.00
```

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 XDC file. It is dependent on the input frequency:

```
set_property CLKIN1_PERIOD 2.85 [get_cells pl4_snk_clk0/mmcm0]
```

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.

```
set_property  CLKOUT0_PHASE 90 [get_cells pl4_snk_clk0/mmcm0]
```

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.

```
set_property IDELAY_VALUE  10 [get_cells pl4_snk_clk0/rdclk_idel]
```

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 Vivado IP catalog are 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 XC7V585T-FFG1761 or XC7K70T-FBG676)

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

MMCM and BUFG placement

The MMCM and 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 XDC that dictate this, this is a requirement for the Xilinx SPI-4.2 solution.

I/O Placement

In the SPI-4.2 core, place the SPI-4.2 I/Os according to need. There are no restrictions for placing the I/Os in the bank options provided in the GUI. For 7 series devices, the SPI-4.2 interface pins (except LVTTL status I/O) are placed by default in High Performance I/O banks. To change this placement to High Range I/O banks, see the Sink Core Optional Constraints, page 136. The placement of the I/Os can be defined using I/O pin lock constraints.

An example of how to define I/O pin lock constraints is provided in the example design XDC file:

```
set_property LOC AJ36 [get_ports {RDat_P[0]}]
set_property LOC AJ37 [get_ports {RDat_N[0]}]
set_property LOC AP36 [get_ports {RDat_P[1]}]
set_property LOC AP37 [get_ports {RDat_N[1]}]
set_property LOC AK37 [get_ports {RDat_P[2]}]
set_property LOC AL37 [get_ports {RDat_N[2]}]

. . .
```

All the SPI-4.2 I/Os do not need to be in a single bank as given in the example XDC. 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.

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 XDC:

```
set_property LOC AK34 [get_ports RDClk_P]
set_property LOC AL34 [get_ports RDClk_N]
```

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 XDC:

```
set_property LOC AK34 [get_ports RDClk_P]
set_property LOC AL34 [get_ports RDClk_N]
```


IDELAYCTRL Module Placement

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, page 163 for the HDL examples. The following XDC segment illustrates how to associate the IDELAYCTRL and its IODELAYelements using constraints:

```
set_property IODELAY_GROUP sink_idelay [get_cells sict1]

set_property IODELAY_GROUP sink_idelay [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*DA
TAPAIR*/*MASTER_DELAY]

set_property IODELAY_GROUP sink_idelay [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*DA
TAPAIR*/*SLAVE_DELAY]

set_property IODELAY_GROUP sink_idelay [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*CT
LPAIR*/*SLAVE_DELAY]

set_property IODELAY_GROUP sink_idelay [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*CT
LPAIR*/*MASTER_DELAY]
```

If using Static Alignment configuration, or Dynamic Alignment configuration with the "DPA Clock Adjustment" feature, the following constraint is also required:

```
set_property IODELAY_GROUP sink_idelay [get_cells
pl4_snk_clk0/rdclk_idel]
```

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.

```
set_property IOB TRUE [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/rstat1_f
f]
```

```
set_property IOB TRUE [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/rstat0_f
f]

set_property IOB TRUE [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/rsclk_ff
]
```

Sink Core Optional Constraints

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:

```
set_property IOSTANDARD  LVDS_25_DCI [get_cells RDat_P(*)]
set_property IOSTANDARD  LVDS_25_DCI [get_cells RDat_N(*)]
set_property IOSTANDARD  LVDS_25_DCI [get_cells RCtl_P]
set_property IOSTANDARD  LVDS_25_DCI [get_cells RCtl_N]
set_property IOSTANDARD  LVDS_25_DCI [get_cells RDClk_P]
set_property IOSTANDARD  LVDS_25_DCI [get_cells RDClk_N]
```

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 Alignment configurations:

```
set_property DIFF_TERM TRUE [get_cells pl4_snk_clk0/rdclk_ibuf0]

set_property DIFF_TERM TRUE [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/StaticAlign.buffe
r_data/*BUFIN*]
```

For dynamic alignment configuration, uncomment the following:

```
set_property DIFF_TERM TRUE [get_cells pl4_snk_clk0/rdclk_ibuf0]

set_property DIFF_TERM TRUE [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*DA
TAPAIR*/*BUFIN*.INBUF*]

set_property DIFF_TERM TRUE [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*CT
LPAIR*/*BUFIN*.INBUF*]
```

To use the 7-Series FPGAs High Range (HR) I/O for `RDClk`, `RDat`, and `RCtl` pins, update any applicable IOSTANDARD generics on I/O buffer instantiations in the user/example design from `LVDS` to `LVDS_25` (see DS823, *LogiCORE IP SPI-4.2 Data Shee*t for supported HR I/O performance). For the example design, these are located in `pl4_snk_clk.v[hd]` and `<core_name>_top.v[hd]`. The `pl4_snk_clk.v[hd]` file contains the IBUFGDS instantiation for `RDClk`, and the `<core_name>_top.v[hd]` file contains the OBUFTDS instantiations for `RSClk` and `RStat[1:0]`. In addition to targeting High Range I/O pins using LOC constraints, add the following constraints to the XDC file to change the I/O standard for the 16 `RDat` pins and the `RCtl` pin. For the dynamic alignment Sink core:

```
set_property IOSTANDARD LVDS_25 [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*DA
TAPAIR*/*BUFIN*.INBUF*]

set_property IOSTANDARD LVDS_25 [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*CT
LPAIR*/*BUFIN*.INBUF*]
```

For the static alignment Sink Core:

```
#set_property IOSTANDARD LVDS_25 [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/StaticAlign.buffe
r_data/*BUFIN*]
```

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 7 series devices, the following constraints are used to place the Sink core in a single clock region, but do not constrain the Status and AXI4-Lite Control logic. This eases timing closure when LVTTL status I/O is selected:

```
create_pblock AG_pl4_snk

add_cells_to_pblock AG_pl4_snk [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/*core0/*]

resize_pblock [get_pblocks AG_pl4_snk] -add
{CLOCKREGION_X0Y3:CLOCKREGION_X0Y3}
```

Timing Ignores Constraints

Since the Sink core internal static configuration signals are driven statically from a register, apply false path attributes to the static configuration signals to create proper timing ignore paths.

To apply the false path attribute to these paths, include the following constraints in the design:

```
set_false_path -through [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/CfgRam/N
umDip4Errors_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/CfgRam/N
umTrainSequences_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/CfgRam/S
nkCalendarM_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/CfgRam/S
nkCalendarLen_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/CfgRam/S
nkAFThresAssert_i*]
```

```
set_false_path -through [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/CfgRam/S
nkAFThresNegate_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/CfgRam/F
ifoAFMode_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/CfgRam/R
SClkPhase*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/axlite/almem/CfgRam/R
SClkDiv_i*]
```

Modifying HIGH_PERFORMANCE_MODE attribute for IODELAYE1/ IODELAYE2 to Improve Power Consumption

The SPI-4.2 core netlist generated by the Vivado IP catalog sets the HIGH_PERFORMANCE_MODE attribute on the IODELAYE2 elements to "TRUE". If targeting a performance of less than 700 Mbps, this attribute can be changed to "FALSE" in the XDC file. This will allow the IODELAY to consume less power.

The following constraints are used to change the HIGH_PERFORMANCE_MODE attribute:

```
set_property HIGH_PERFORMANCE_MODE FALSE [get_cells
pl4_snk_clk0/rdclk_idel]

set_property HIGH_PERFORMANCE_MODE FALSE [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*DA
TAPAIR*/*MASTER_DELAY]

set_property HIGH_PERFORMANCE_MODE FALSE [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*DA
TAPAIR*/*SLAVE_DELAY]

set_property HIGH_PERFORMANCE_MODE FALSE [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*CT
LPAIR*/*SLAVE_DELAY]

set_property HIGH_PERFORMANCE_MODE FALSE [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/io0/*dpa/dpa_top0/*CT
LPAIR*/*MASTER_DELAY]
```

Initializing the Sink Calendar Sequence Using Constraints

The Sink core Calendar memory can be initialized through the Sink AXI4-Lite Control interface Sink Calendar Memory Initialization in Chapter 5, or optionally via XDC constraints. The following is an example of two XDC constraints that initialize the first 64 calendar locations to a round-robin sequence for channels 0-15:

```
set_property INIT_OO
0F0E0D0C0B0A09080706050403020100F0E0D0C0B0A09080706050403020100
[get_cells
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CalRam/cr
am/ramb_bl.ramb36_dp_bl.ram36_bl]
```

```
set_property INIT_01
0F0E0D0C0B0A09080706050403020100F0E0D0C0B0A09080706050403020100
[get_cells
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CalRam/cr
am/ramb_bl.ramb36_dp_bl.ram36_bl]
```

Source Core Required Constraints

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.

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.

```
create_clock -add -name SysClk_P -period 2.857 [get_ports SysClk_P]

set_input_jitter SysClk_P  0.200

create_clock -add -name TSClk_P -period   11.425 [get_ports TSClk_P]
```

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

```
create_clock -add -name AXI_SRC_ACLK -period 11.425  [get_ports
AXI_SRC_ACLK]

create_clock -add -name S_AXIS_SRCFF_ACLK -period 5.714 [get_ports
S_AXIS_SRCFF_ACLK]

set_input_jitter M_AXIS_SRCFF_ACLK 0.300

create_clock -add -name M_AXIS_SRCSTAT_ACLK -period 11.425 [get_ports
M_AXIS_SRCSTAT_ACLK]
```

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.

```
set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/io0/SrcReset*]] -to
[get_cells -hierarchical * -filter {LIB_CELL == FDPE ||LIB_CELL == FDCE
}] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
```

```
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/rst0/SrcClk_Reset*]]
-to [get_cells -hierarchical * -filter {LIB_CELL == FDPE ||LIB_CELL ==
FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/rst0/SrcFFClk_Reset*]
] -to [get_cells -hierarchical * -filter {LIB_CELL == FDPE ||LIB_CELL ==
FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/rst0/TSClk_Reset*]] -
to [get_cells -hierarchical * -filter {LIB_CELL == FDPE ||LIB_CELL ==
FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/rst0/SrcCLK_FifoReset
_sync ]] -to [get_cells -hierarchical * -filter {LIB_CELL == FDPE
||LIB_CELL == FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/rst0/SrcFFCLK_FifoRes
et_sync ]] -to [get_cells -hierarchical * -filter {LIB_CELL == FDPE
||LIB_CELL == FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/SrcClk_data_Reset]] -
to [get_cells -hierarchical * -filter {LIB_CELL == FDPE ||LIB_CELL ==
FDCE }] 5.00

set_max_delay -from [all_fanin -startpoints_only -flat -only_cells
[get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/SrcClk_u1_Reset]] -to
[get_cells -hierarchical * -filter {LIB_CELL == FDPE ||LIB_CELL == FDCE
}] 5.00
```

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 by the Vivado IP catalog 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 example XC7V585T-FFG1761 or XC7K70T-FBG676).

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

MMCM and BUFG placement

The MMCM and BUFG(s) it drives need to be placed in the same top or bottom half of the chip. This is a requirement for the Xilinx SPI-4.2 solution, and the following constraints from the example XDC show this placement:

```
set_property LOC MMCME2_ADV_X0Y2 [get_cells pl4_src_clk0/mmcm0]

set_property LOC BUFGCTRL_X0Y3 [get_cells pl4_src_clk0/sysclk0_bufg0]
```

BUFR and BUFIO placement

To reduce duty cycle distortion with a core using Regional clocking, the BUFR and BUFIO for SysClk should be placed as close as possible to the elements driven by these clock buffers (i.e. the SPI-4.2 TDClk, TDat, and TCtl pins). The following constraints from the example XDC show this placement:

```
set_property LOC BUFIO_X0Y17 [get_cells pl4_src_clk0/tdclk0_bufio]

set_property LOC BUFR_X0Y17  [get_cells pl4_src_clk0/srcclk0_bufr0]
```

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. For 7 series devices, the SPI-4.2 interface pins (except LVTTL status I/O) are placed by default in High Performance I/O banks. To change this placement to High Range I/O banks, see the Source Core Optional Constraints, page 142. The placement of the I/Os can be defined using I/O pin lock constraints.

An example of how to define I/O pin lock constraints is provided in the example design XDC file:

```
set_property LOC AM41 [get_ports {TDat_P[0]}]

set_property LOC AM42 [get_ports {TDat_N[0]}]

set_property LOC AR38 [get_ports {TDat_P[1]}]

set_property LOC AR39 [get_ports {TDat_N[1]}]

set_property LOC AN40 [get_ports {TDat_P[2]}]

set_property LOC AN41 [get_ports {TDat_N[2]}]

...
```

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 I, SPI-4.2 Source Interface Timing Budget. This appendix illustrates how package and clock skew will affect the source interface timing budget.

When using 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 or BUFR 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. The XDCs generated by the Vivado IP catalog provide examples of this.

You can also place SysClk and TSClk using the two constraints above. 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 Source core logic. For instance, in the example XDC, a pin-lock constraint is used to specify I/O placement:

```
set_property LOC AV40 [get_ports SysClk_P]

set_property LOC AW40 [get_ports SysClk_N]
```

If global clocking is chosen, SysClk must be placed on a pin that is connected to a global clock buffer. In the example XDC, a pin-lock constraint is used to specify I/O placement:

```
set_property LOC AV40 [get_ports SysClk_P]

set_property LOC AW40 [get_ports SysClk_N]
```

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.

```
set_property IOB TRUE [get_cells
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/tstat1_ff
]

set_property IOB TRUE [get_cells
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/tstat0_ff
]
```

Source Core Optional Constraints

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 XDC. Uncomment them to include them in your design.

```
set_property CLKOUT0_PHASE 180 [get_cells pl4_src_clk0/mmcm1]
```

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:

```
set_property IOSTANDARD  LVDS_25_DCI [get_cells SysClk_P]
```

```
set_property IOSTANDARD  LVDS_25_DCI [get_cells SysClk_N]
```

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:

```
set_property DIFF_TERM TRUE [get_cells pl4_src_clk0/sysclk_ibufg0]
```

To use the 7-Series FPGAs High Range (HR) I/O for SysClk, TDClk, TDat, and TCtl pins, update any applicable IOSTANDARD generics on I/O buffer instantiations in the user/example design from LVDS to LVDS_25 (see DS823, *LogiCORE IP SPI-4.2 Data Sheet* for supported HR I/O performance). For the example design, these are located in pl4_src_clk.v[hd] and <core_name>_top.v[hd]. The pl4_src_clk.v[hd] file contains the IBUFGDS instantiations for SysClk and TSClk, and the <core_name>_top.v[hd] file contains the IBUFDS instantiations for TStat[1:0]. In addition to targeting High Range I/O pins using LOC constraints, add the following constraints to the XDC file to change the I/O standard for the 16 TDat pins and the TCtl pin:

```
set_property IOSTANDARD  LVDS_25 [get_cells
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/io0/src_ddr*/*ddr*_bu
f]

set_property IOSTANDARD  LVDS_25 [get_cells
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/io0/src_ctl*/*ddr*_bu
f]
```

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.

The following constraints are used to place the Source core in a single clock region, but do not constrain the Status and AXI4-Lite Control logic. This eases timing closure when LVTTL status I/O is selected:

```
create_pblock AG_pl4_src

add_cells_to_pblock AG_pl4_src [get_cells
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/core0/*]

resize_pblock [get_pblocks AG_pl4_src] -add
{CLOCKREGION_X0Y4:CLOCKREGION_X0Y4}
```

Timing Ignore Constraints

Since the Source core internal static configuration signals are driven statically from a register, apply false path attributes to the static configuration signals to create proper timing ignore paths.

To apply the false path attributes to these paths, include the following constraints in the design:

```
set_false_path -through [get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CfgRam/Nu
mDip2Matches_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CfgRam/Sr
cCalendarM_i*]
```

```
set_false_path -through [get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CfgRam/Sr
cCalendarLen_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CfgRam/Sr
cAFThresAssert_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CfgRam/Sr
cAFThresNegate_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CfgRam/Al
phaData_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CfgRam/Da
taMaxT_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CfgRam/Sr
cBurstMode_i*]

set_false_path -through [get_nets
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/CfgRam/Sr
cBurstLen_i*]
```

Initializing the Source Calendar Sequence Using Constraints

The Source core Calendar memory can be initialized through the Source AXI4-Lite Control interface Source Calendar Memory Initialization in Chapter 5, or optionally via XDC constraints. The following is an example of two XDC constraints that initialize the first 64 calendar locations to a round-robin sequence for channels 0-15:

```
set_property INIT_OO
0F0E0D0C0B0A09080706050403020100F0E0D0C0B0A090807060504030201000
[get_cells
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/cram/cram
/ramb_bl.ramb36_dp_bl.ram36_bl]

set_property INIT_01
0F0E0D0C0B0A09080706050403020100F0E0D0C0B0A090807060504030201000
[get_cells
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/alite/almem/cram/cram
/ramb_bl.ramb36_dp_bl.ram36_bl]
```

User Constraints

In certain cases, additional constraints may need to be added to cover other logic implemented in the design. While the XDC 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 XDC 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 XC7V585T Virtex-7 FPGA part. This is achieved by modifying the example XDC. The static alignment example shows the migration of the Sink and Source cores to the west region of the part (banks 14 and 15, respectively). The dynamic phase alignment example shows the migration of the Sink and Source cores to the west region of the part (banks 14 and 15, 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: MMCM, regional and global buffers

Some typical region selections within a device are:

- Source Core: One clock region on the same side of the device, east or west.
- Sink Core (static): One clock region on the same side of the device.
- Sink Core (dynamic): One clock region on the same side of the device.

The east side is the side of the device with numbered I/O banks: 36 and 37. The west side is the side of the device with numbered I/O banks: 14 and 15.

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 XDC must be modified. While modifying the constraints, ensure that changes are within the specifications described by the Sink and Source core required constraints. The XDC modifications are provided in this section.

Note: Using optional constraints is at user discretion.

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 constrained to pins that coincide with the clock regions of the Sink core.

In the following example, Bank 14 must contain at least 17 LVDS I/O pairs (not all pin constraints shown):

```
set_property LOC AJ36 [get_ports {RDat_P[0]}]

set_property LOC AJ37 [get_ports {RDat_N[0]}]
```

```
set_property LOC AP36 [get_ports {RDat_P[1]}]

set_property LOC AP37 [get_ports {RDat_N[1]}]

set_property LOC AK37 [get_ports {RDat_P[2]}]

set_property LOC AL37 [get_ports {RDat_N[2]}]

...
```

Specify pin placement for `RDClk` I/O. See Placement Constraints, page 134 for more information about placement constraints. For example:

```
set_property LOC AK34 [get_ports RDClk_P]

set_property LOC AL34 [get_ports RDClk_N]
```

Specify an area group constraint if necessary. In the XDC, area group `AG_pl4_snk` is the clock region on the same side of the device. For example:

```
create_pblock AG_pl4_snk

add_cells_to_pblock AG_pl4_snk [get_cells
<core_instance_name>/U0/pl4_snk_top/*pl4_snk_top/*core0/*]

resize_pblock [get_pblocks AG_pl4_snk] -add
{CLOCKREGION_X0Y3:CLOCKREGION_X0Y3}
```

Source Core

Specify pin placement for `SysClk` I/O. See Placement Constraints, page 140 for detailed information. For example:

```
set_property LOC AV40 [get_ports SysClk_P]

set_property LOC AW40 [get_ports SysClk_N]
```

The Source core clock element placement constraints must also be moved accordingly. For SysClk Global Clocking:

```
set_property LOC MMCME2_ADV_X0Y2 [get_cells pl4_src_clk0/mmcm0]

set_property LOC BUFGCTRL_X0Y3 [get_cells pl4_src_clk0/sysclk0_bufg0]
```

Or, for SysClk Regional Clocking:

```
set_property LOC BUFIO_X0Y17 [get_cells pl4_src_clk0/tdclk0_bufio]

set_property LOC BUFR_X0Y17  [get_cells pl4_src_clk0/srcclk0_bufr0]
```

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. In the following example, Bank 15 contains at least 18 LVDS I/O pairs:

```
set_property LOC AM41 [get_ports {TDat_P[0]}]

set_property LOC AM42 [get_ports {TDat_N[0]}]

set_property LOC AR38 [get_ports {TDat_P[1]}]

set_property LOC AR39 [get_ports {TDat_N[1]}]

set_property LOC AN40 [get_ports {TDat_P[2]}]

set_property LOC AN41 [get_ports {TDat_N[2]}]
```

Specify pin placement for "TSClk" I/O. See Placement Constraints, page 140. For example (regional clocking, LVDS):

```
set_property LOC AU39 [get_ports TSClk_P]

set_property LOC AV39 [get_ports TSClk_N]
```

Specify an area group constraint if necessary. In the XDC, area group "AG_pl4_src" is the clock region on the same side of the device.

```
create_pblock AG_pl4_src

add_cells_to_pblock AG_pl4_src [get_cells
<core_instance_name>/U0/pl4_src_top/*pl4_src_top/core0/*]

resize_pblock [get_pblocks AG_pl4_src] -add
{CLOCKREGION_X0Y4:CLOCKREGION_X0Y4}
```

Special Consideration for Dynamic Phase Alignment

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 modified XDC, the I/O placement could be defined as:

```
set_property LOC AJ36 [get_ports {RDat_P[0]}]

set_property LOC AJ37 [get_ports {RDat_N[0]}]

set_property LOC AP36 [get_ports {RDat_P[1]}]

set_property LOC AP37 [get_ports {RDat_N[1]}]

set_property LOC AK37 [get_ports {RDat_P[2]}]

set_property LOC AL37 [get_ports {RDat_N[2]}]

...
```

These pin-locks are all within Bank 14, which is in clock region X0Y3.

The I/O constraints for the Source core need to be locked to the I/O bank within the clock regions on the Source core. This is recommended for regional clocking. For example, in the modified XDC, the I/O placements could be defined as:

```
set_property LOC AM41 [get_ports {TDat_P[0]}]

set_property LOC AM42 [get_ports {TDat_N[0]}]

set_property LOC AR38 [get_ports {TDat_P[1]}]
```

```
set_property LOC AR39 [get_ports {TDat_N[1]}]
set_property LOC AN40 [get_ports {TDat_P[2]}]
set_property LOC AN41 [get_ports {TDat_N[2]}]
...
```

These pin-locks are all within Bank15, which is in clock region X0Y4.

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
- Source Clocking Options
- Clocking Guidelines
- Instantiating IDELAYCTRL Modules

Sink Clocking Options

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 clocking. Depending on the chosen clocking option, different clock resources are used. Table 7-1 provides the clocking resource count for each clocking option.

Use the LogiCORE 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 an MMCM instantiation for the SPI-4.2 core using the Clocking 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 I, SPI-4.2 Source Interface Timing Budget for more information.

Table 7-1: **Sink Core Clocking Option for Virtex-7 and Kintex-7 FPGAs**

Clocking Option	BUFR	BUFG[a]	MMCM
Regional clocking	1	0/1	0
Global clocking	0	4/5	1

a. The Sink Core requires SnkIdelayRefClk to be 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.

An illustration of the Sink user clock inputs and the sink clock configurations are shown in Figure 7-1. The inputs are defined in Table 3-6, page 40.

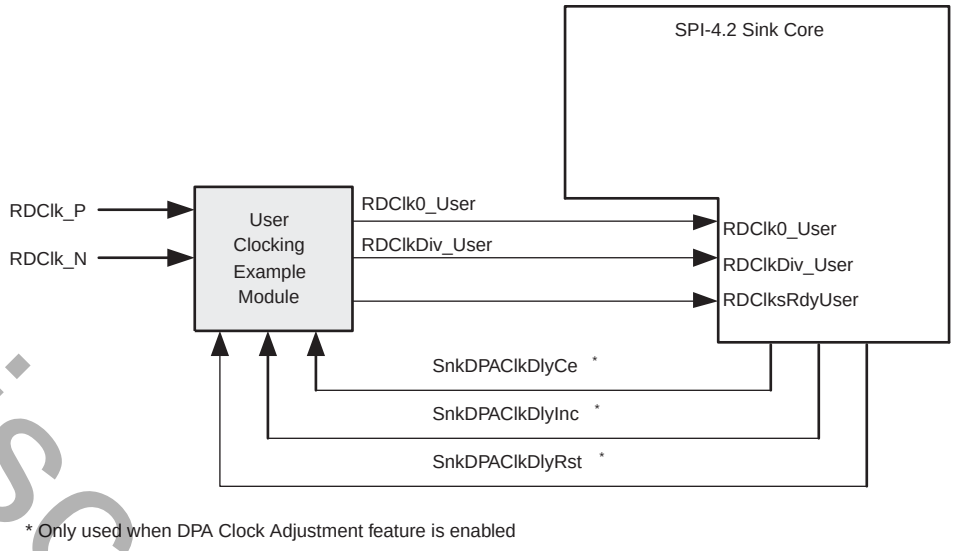

Figure 7-1: **Sink Core User Clocking Example**

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. This configuration is illustrated in Figure 7-2. This implementation is the same for both static and dynamic alignment configurations.

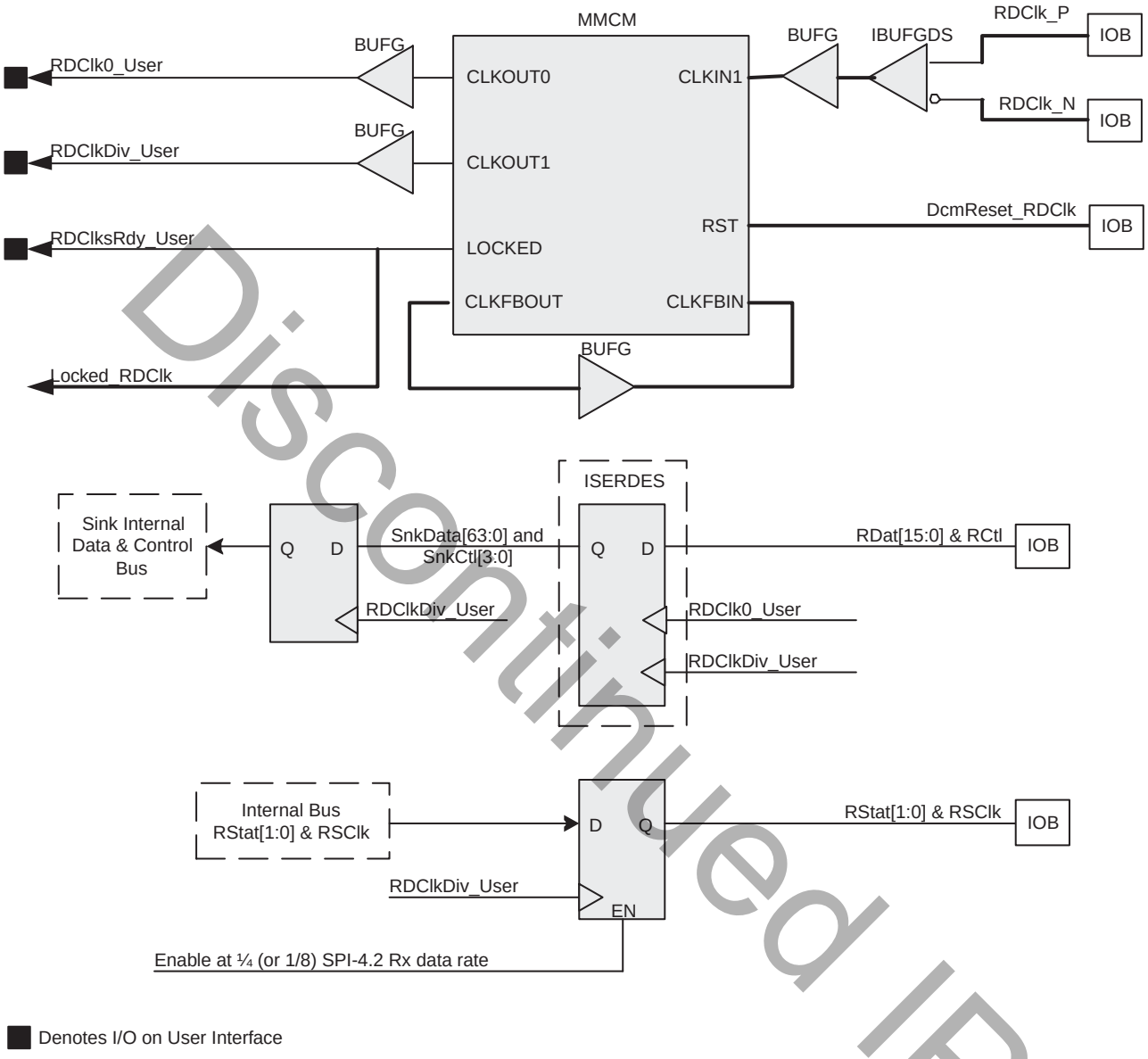

Figure 7-2: **RDClk Global Clocking**

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 7-3. This implementation is the same for both static and dynamic alignment configurations. For 7 series designs, this clocking method adds

restrictions on the placement of the Sink core. See Regional Clocking Considerations, page 159 for more information.

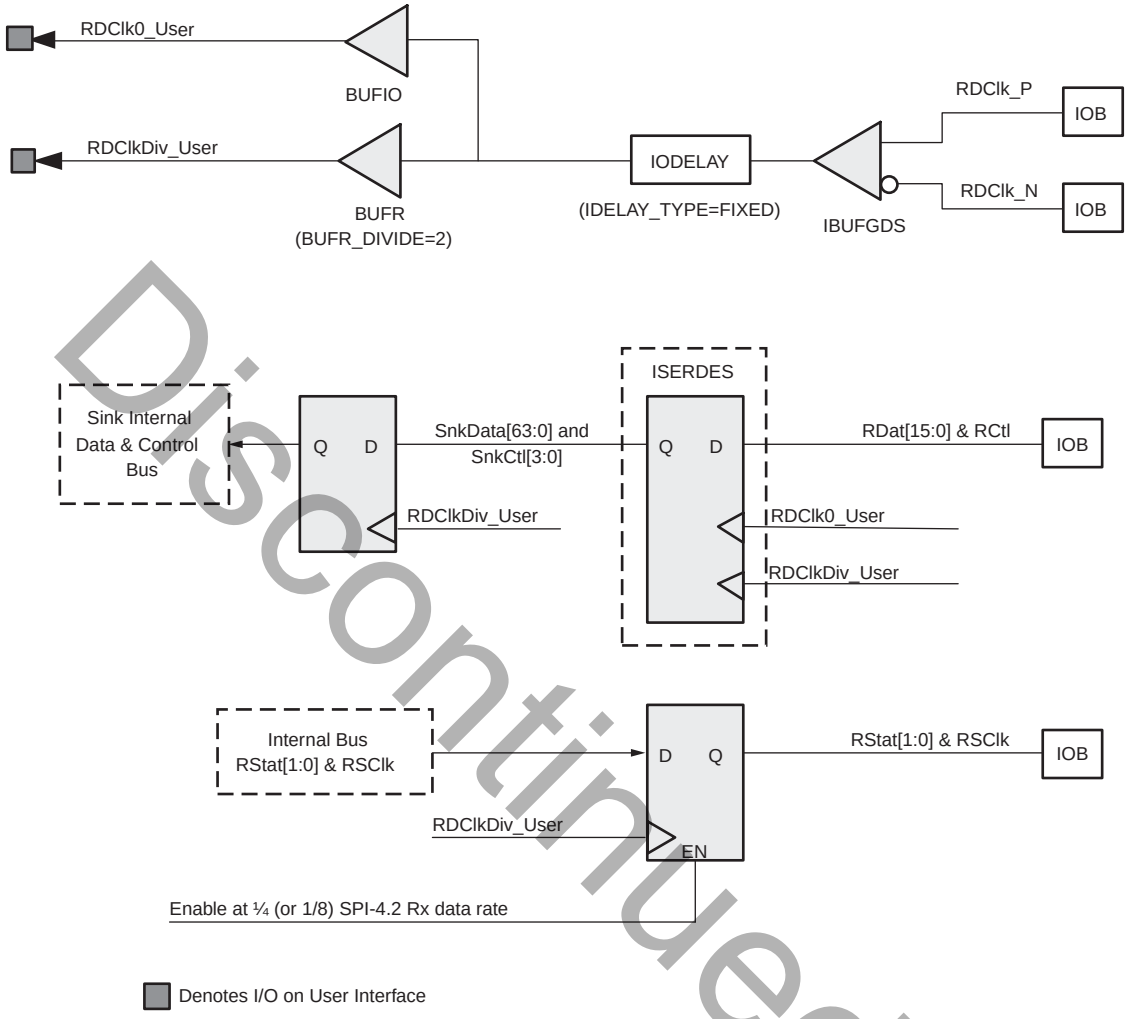

Figure 7-3: **RDClk Regional Clocking**

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 7-4 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.

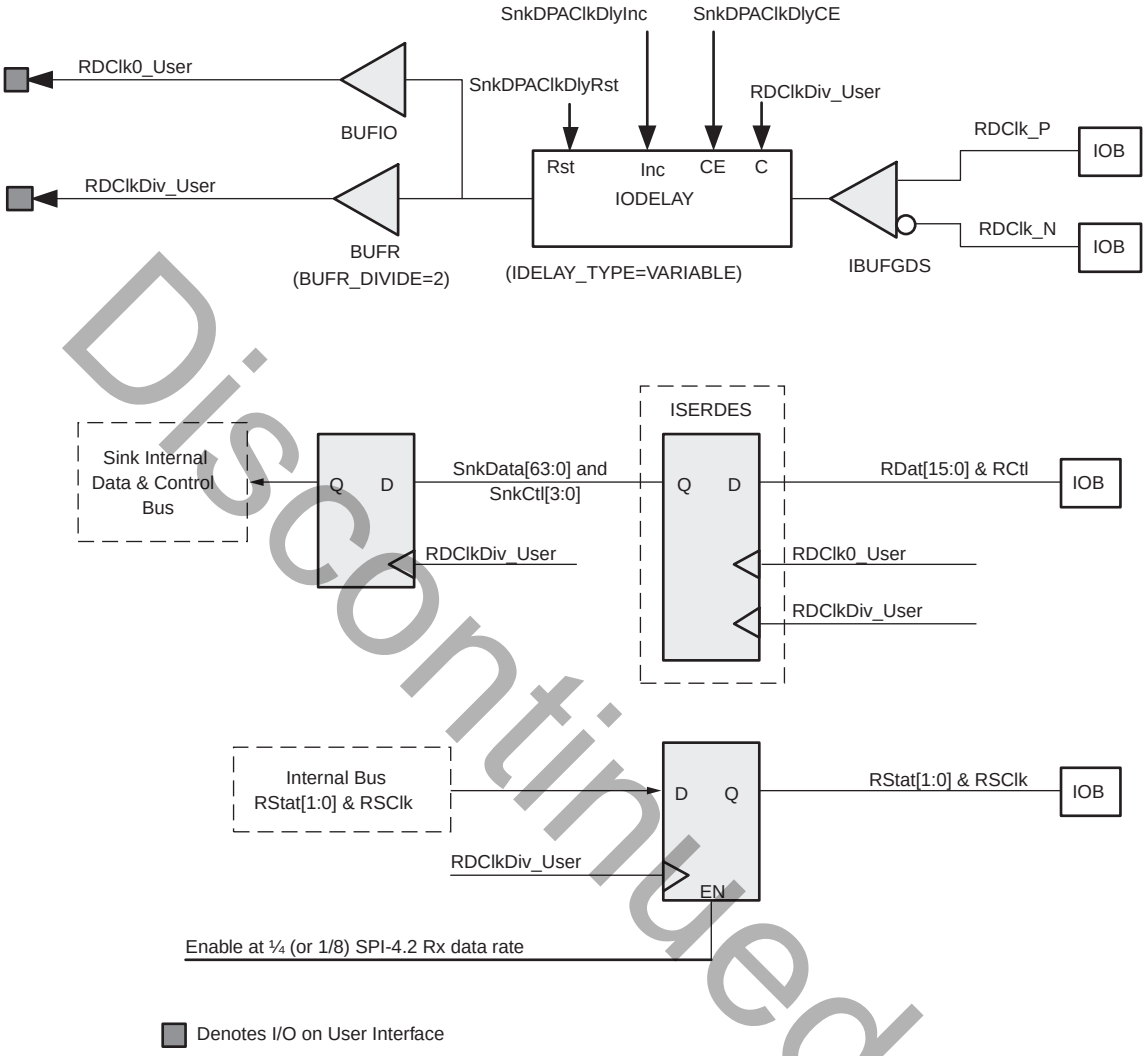

Figure 7-4: **IDELAY on RDClk for DPA Clock Adjustment**

Source Clocking Options

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 7-5.

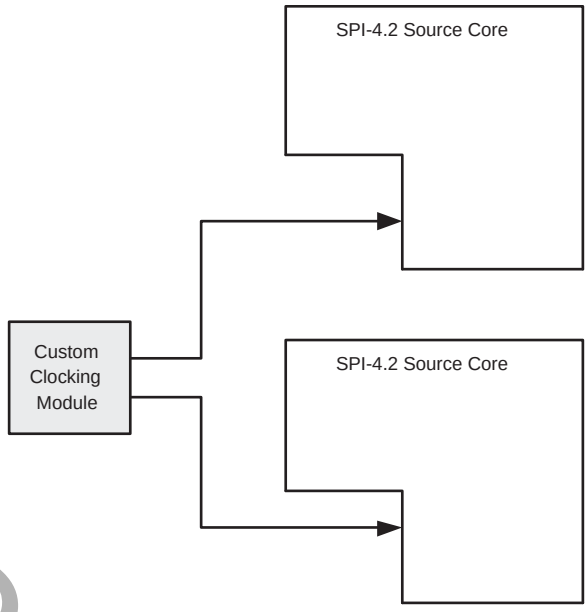

Figure 7-5: **Source Core User Clocking Example**

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. See Appendix I, SPI-4.2 Source Interface Timing Budget for an example of a Static Alignment timing budget. Depending on the chosen clocking option, different clock resources are used. Table 7-2 and Table 7-3 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 an MMCM instantiation for the SPI-4.2 core using the Clocking 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 I, SPI-4.2 Source Interface Timing Budget for more information.

Note: Use regional clocking to generate SysClk for a better timing budget. See Appendix I, SPI-4.2 Source Interface Timing Budget for more details.

Table 7-2: **Source Core SysClk Clocking for Virtex-7 and Kintex-7 FPGAs[a]**

Clocking Option	BUFR	BUFG	MMCM
Regional clocking	1	0	0
Global clocking[b]	0	4	1

a. When Sink core is configured with static alignment, only regional clocking is supported for Source core.

b. Global clocking is only supported for source cores that send data to Sink cores that are configured with Dynamic Phase Alignment.

Table 7-3: **Source Core TSClk Clocking for Virtex-7 and Kintex-7 FPGAs**

Clocking Option	BUFR	BUFG	MMCM
Regional clocking	1	0	0
Global clocking	0	3	1

An illustration of the source user clock inputs and the source clock configurations are shown in Figure 7-6. The inputs are defined in Table 3-11, page 51.

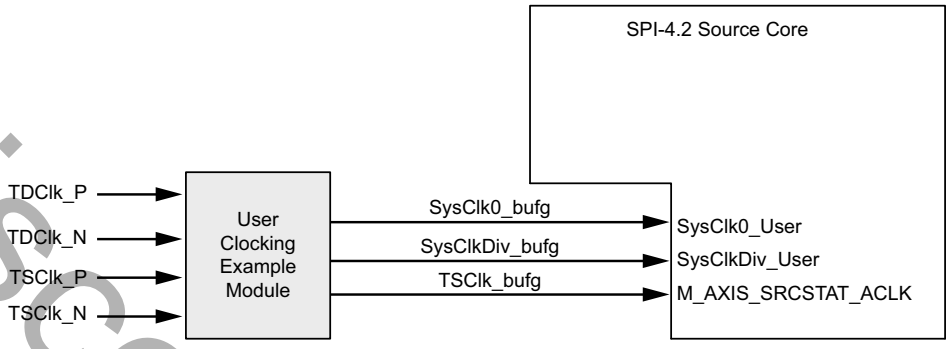

Figure 7-6: **Source Core User Clocking Ports**

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 (LVDS) using the IBUFGDS primitive. This configuration is illustrated in Figure 7-7 and Figure 7-8 showing data and status clocking.

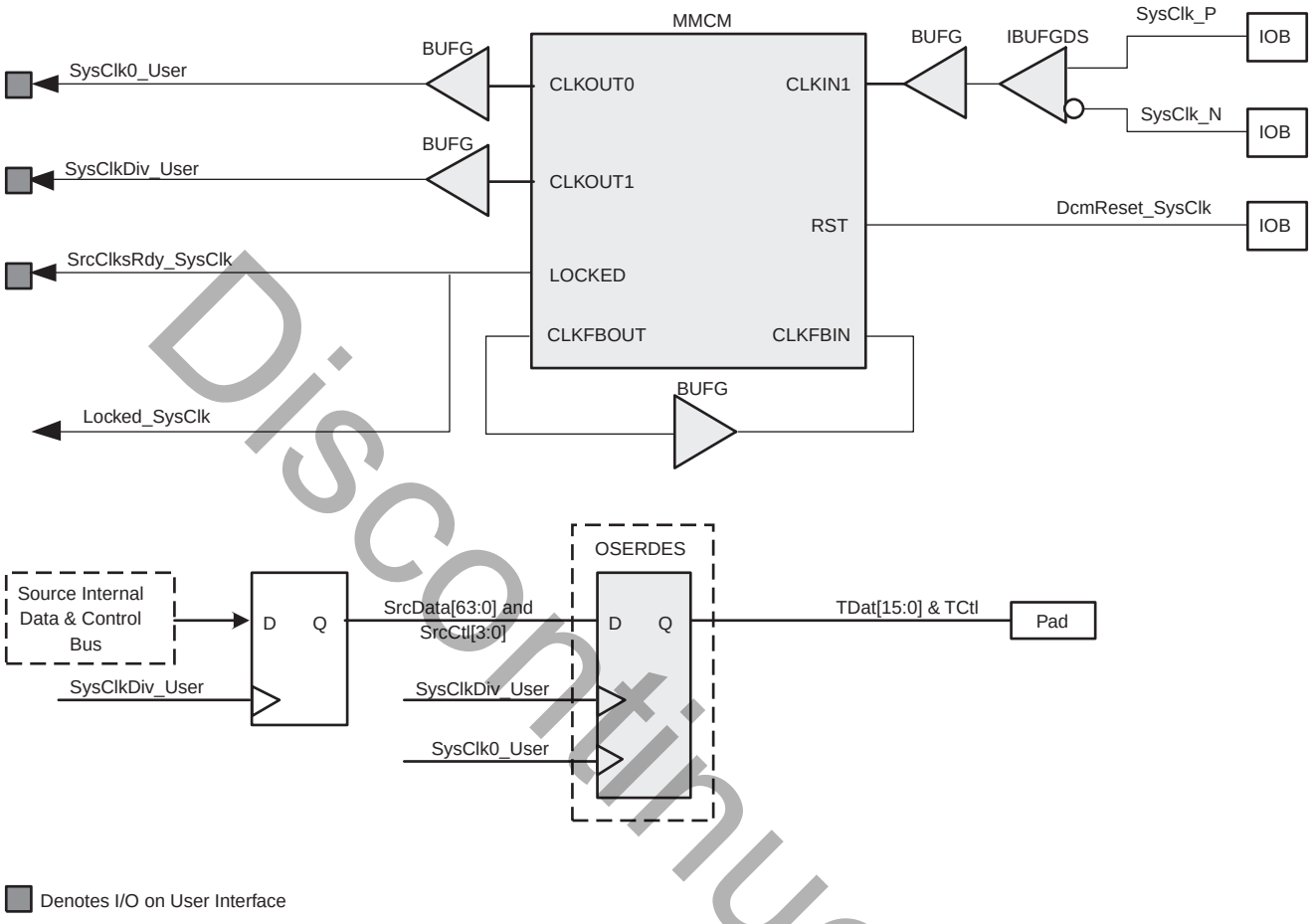

Figure 7-7: **SysClk Global Clocking**

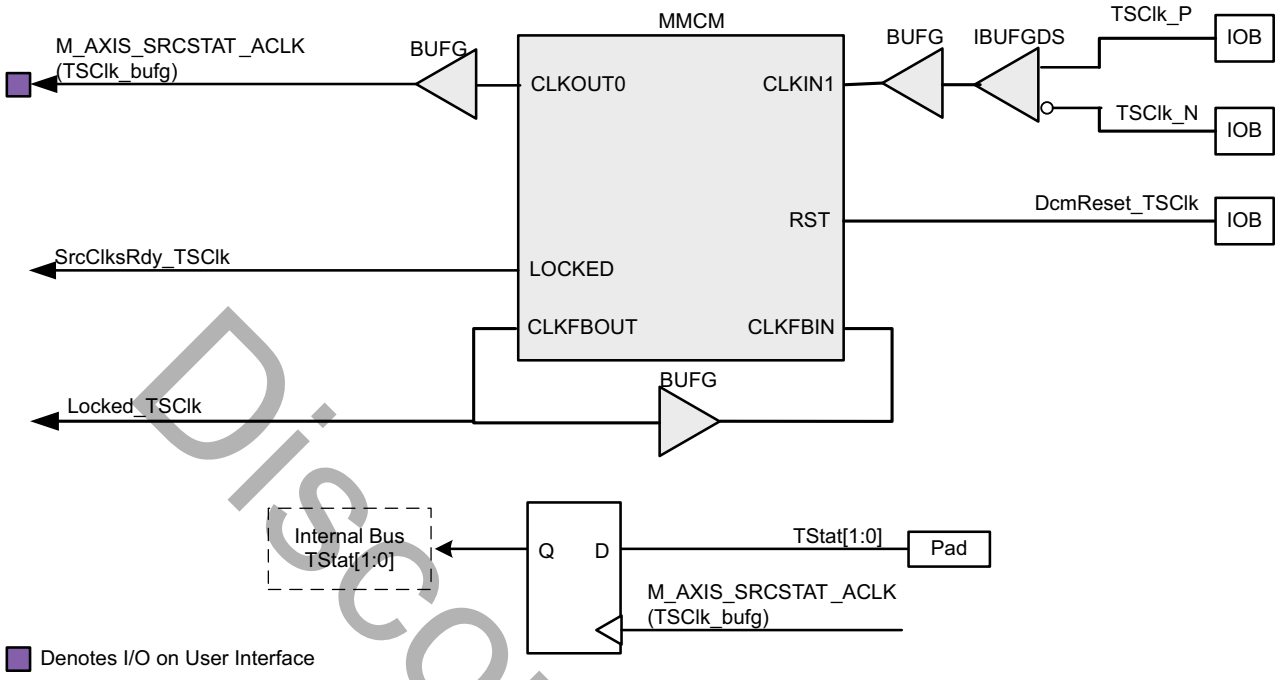

Figure 7-8: **TSClk Global Clocking**

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 7-9 and Figure 7-10 showing data and status clocking. For 7 series designs, this clocking method adds restrictions on the placement of the Sink core. See Regional Clocking Considerations, page 159 for more information

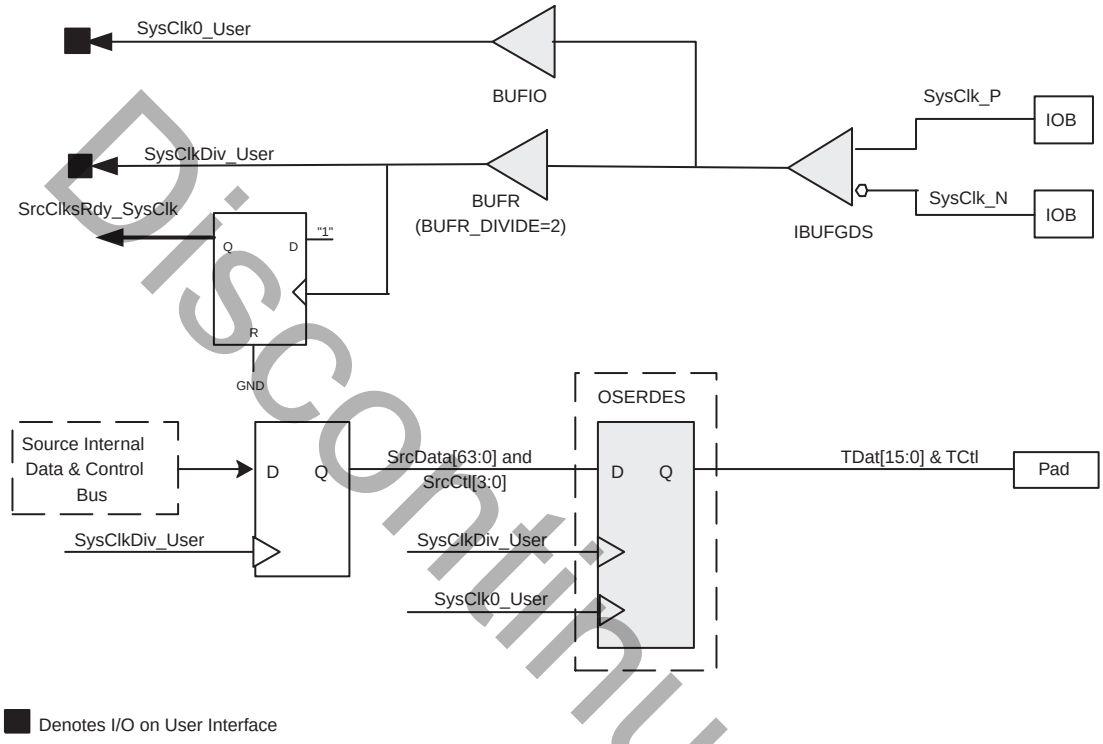

Figure 7-9: **SysClk Regional Clocking**

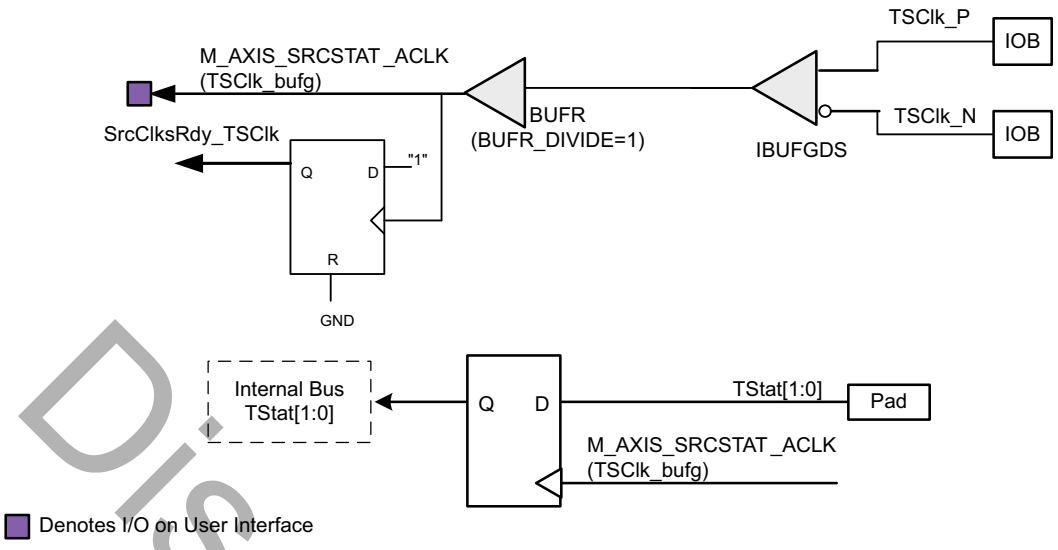

Figure 7-10: **TSClk Regional Clocking**

Clocking Guidelines

There are two parameters to consider when choosing the best clocking option for your system:

- Clock frequency
- Available clock resources

Regional clocking can operate beyond speeds of 1 Gbps, while global clocking with MMCM can operate up to 1 Gbps. Regional clocking requires fewer clock resources than global clocking with MMCM; however, it places an area restriction on the logic that uses the regional clock.

For the maximum and minimum frequencies for different clock resources (for example, MMCM, BUFR), see the appropriate Virtex-7 or Kintex-7 FPGA Data Sheet.

Regional Clocking Considerations

The BUFR in 7 series devices can only be used to drive logic in a single clock region. The SPI-4.2 core example design continues the use of the BUFR with 7 series devices. Usually, the placement of the SPI-4.2 interface clock synthesis logic (as shown in Sink Clocking Options and Source Clocking Options) and the clock distribution selected for these clocks drive the placement requirements for logic and I/O associated with the various user interface clocks. Figure 7-11 and Figure 7-12 show all of the clocks associated with each core, and which blocks of the SPI-4.2 core fall under each clock domain. Use these figures as a guide to determine which portions of user logic and core logic will have placement restrictions based on clocking distribution used for the core. For example, if Regional Clocking is selected for the Sink SPI-4.2 Receive Data clock (`RDClk0_User` and `RDClkDiv_User`) and regional clocking is also desired for the Sink AXI4-Stream FIFO interface clock (`M_AXIS_SNKFF_ACLK`), the user logic associated with `M_AXIS_SNKFF_ACLK` and the Sink Data FIFO block must be placed in the same clock region as the Sink Data Receive block and Sink Status Transmit block (clocked by `RDClk0_User` and `RDClkDiv_User`). In the example design, all user clocks

(M_AXIS_SNKFF_ACLK, AXI_SNK_ACLK, S_AXIS_SRCFF_ACLK, and AXI_SRC_ACLK) are distributed using Global Clocking resources and so they are not required to be placed in the same clock region as the Sink or Source cores.

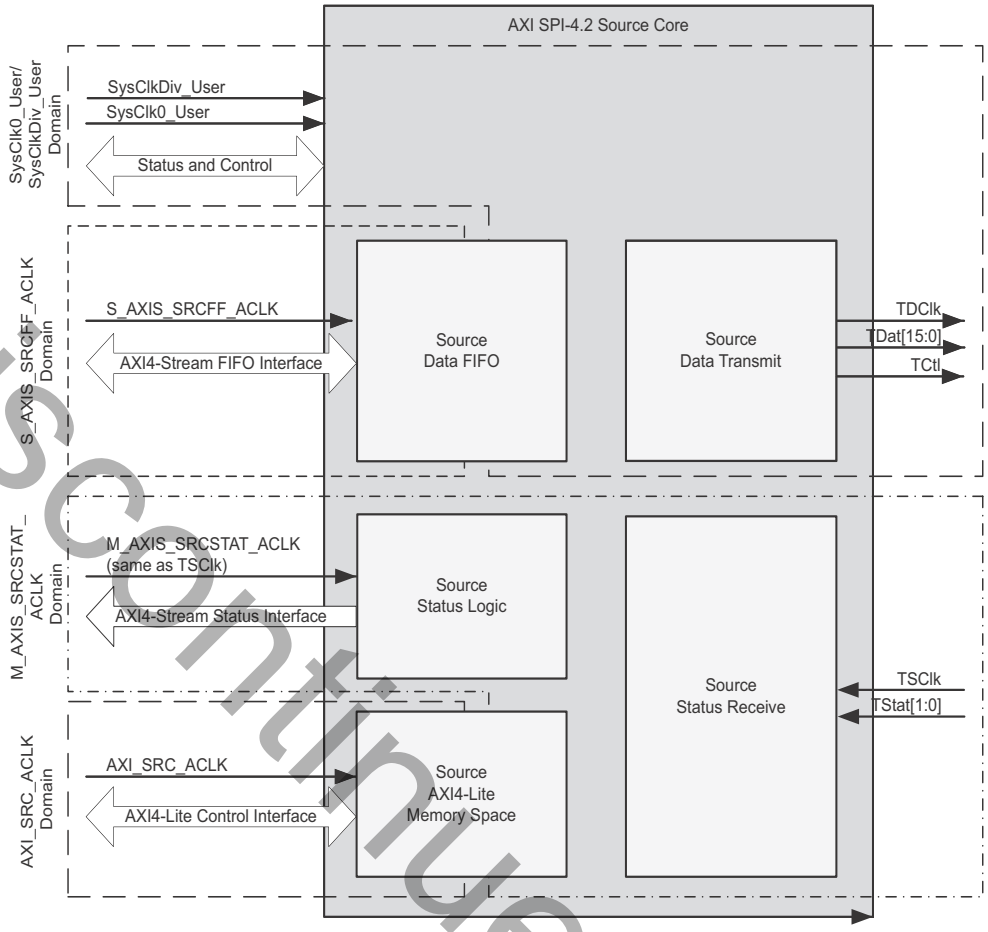

Figure 7-11: **Source Core Clocking**

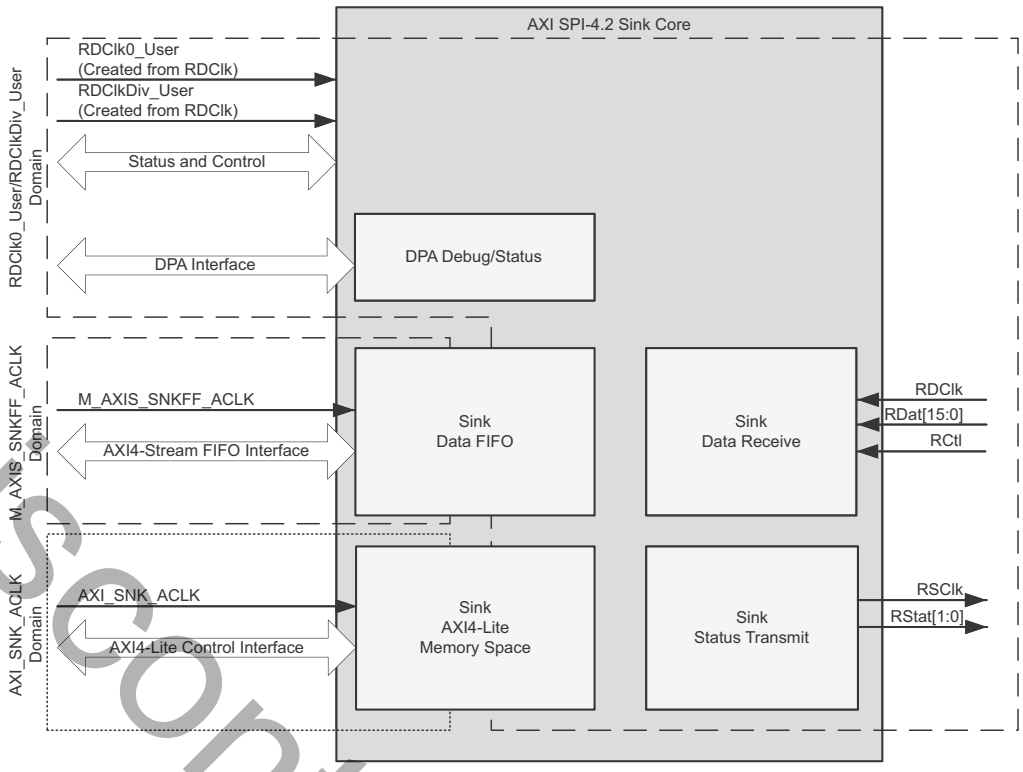

Figure 7-12: **Sink Core Clocking**

Typically, a single SPI-4.2 Sink core or a single SPI-4.2 Source core will use at most one-third of the logic available in a 7 series clock region, leaving the other two-thirds available for user logic. However, there may be some cases where the user wishes to place additional related logic in the two adjacent clock regions when using Regional Clocking. One possibility is if the user logic associated with the Sink or Source core requires more logic than is available in the same clock region in which the core is placed.

The new BUFMRCE primitive provided in the 7 series family provides a solution. Though the SPI-4.2 core example design does not use the BUFMRCE, the multi-region clock buffer can be used to drive separate BUFRs in each clock region. For further information regarding the use of the BUFMRCE with BUFRs in 7 series, see UG472, *7 Series FPGAs Clocking Resources User Guide*, Appendix A. One example of a BUFMRCE clocking scheme

with the SPI-4.2 Source core is shown in Figure 7-13.

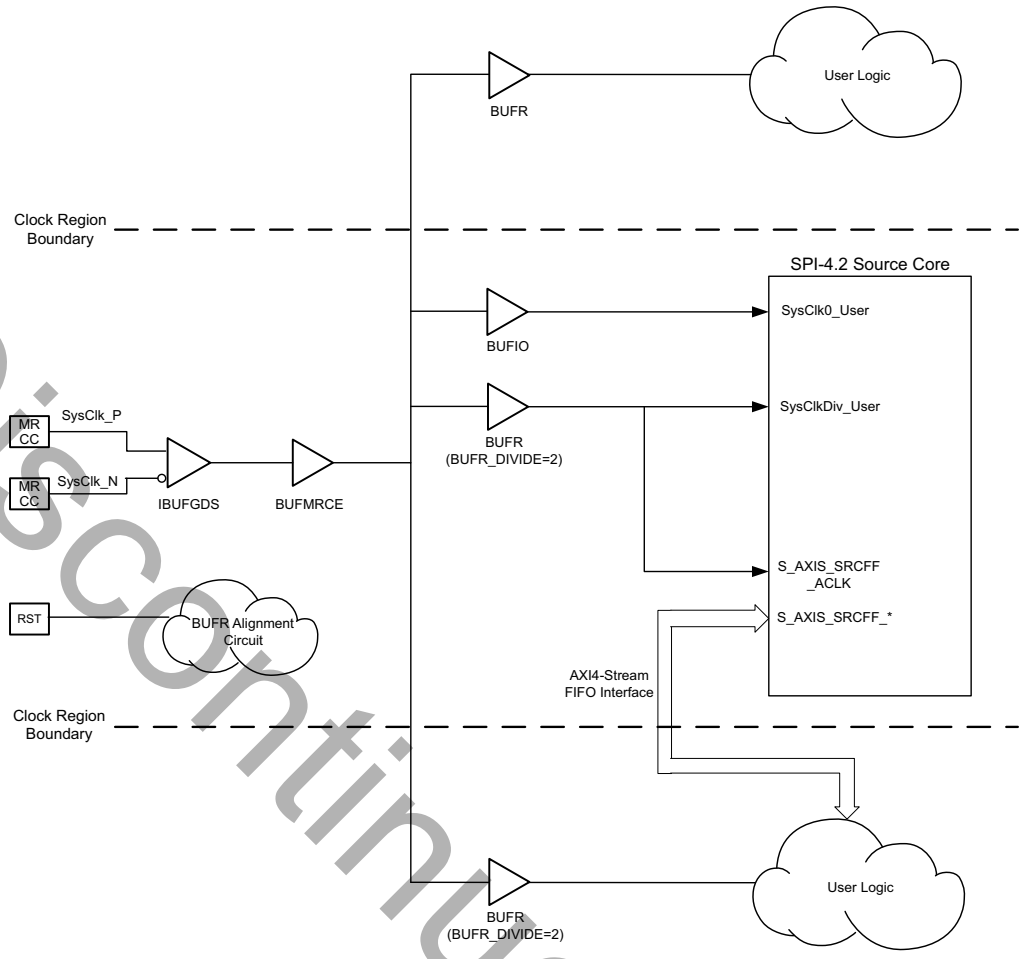

Figure 7-13: **BUFMRCE Clocking Scheme**

7 Series LVTTL Status I/O Considerations

When targeting 7 series devices, the SPI-4.2 Sink and Source cores offer the option to use either LVTTL or LVDS I/O for the status pins (`RSClk/RStat` and `TSClk/TStat`). The following points must be considered when using LVTTL Status I/O with 7 Series devices:

- LVTTL is supported in High Range I/O banks only. LVDS (required for the SPI-4.2 `RDat/RCtl` and `TDat/TCtl` pins and associated clocks) is supported in High Performance banks, or in High Range banks as LVDS_25. Note that `LVDS_25` I/O pins cannot be placed in the same bank as LVTTL pins since their source voltages (V_{CCO}) differ.
- For Kintex-7 devices, High Range banks are always on the left side of the device (for example, Banks 13 and 14), and High Performance banks are always on the right side (for example, Banks 33 and 34).
- For Virtex-7 devices, High Range banks are always located at the lower left of the die (for example, Banks 12 and 13). High Performance banks are located on both sides of the die (for example, Bank 14 or 35). There are currently two Virtex-7 devices that offer High Range I/O banks: XC7V585T-FF1761 and XC7VX330T-FF1761.

- Regional Clocking cannot cross from one half of the device to the other, regardless of use of BUFR and BUFMRCE buffers.
- Regional Clocking using standard BUFRs can drive logic in one clock region (that is, one I/O bank) in 7 series, as described in Regional Clocking Considerations. If using the BUFMRCE, Regional Clocking can clock the two adjacent regions/banks as well (for a total of three clock regions).

Taking these points into account for 7 series devices, LVTTL status I/O cannot be used when the Sink core uses Regional Clocking because the Sink status logic is driven by the same clock (`RDClkDiv_User`) as the rest of the Sink core.

For the Source core, LVTTL status I/O cannot be used when either SysClk distribution or TSClk distribution are Regional Clocking, because some signals used by the Source status logic are created in the SysClk clock domain (`Reset_n` and `SrcEn`).

Note that timing closure of the design is more challenging when using LVTTL status I/O because some paths may need to cross from one half of the device to the other. The Example Design generated with the core provides an example of how timing closure can be achieved in these cases using Area Group constraints. See Chapter 6, Constraining the Core for details.

Instantiating IDELAYCTRL Modules

For Virtex-7 and Kintex-7 FPGA 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. It is necessary to only instantiate one IDELAYCTRL module--the duplication and placement of the IDELAYCTRL is done automatically by the Vivado Design Suite. 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-7 or Kintex-7 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_User clock domain.
--- The RDClkDiv_User 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_User_i,
          R  =>  SnkIdelayCtlRst
            );


  register_rdy : process (SnkIdelayCtlRst, RDClkDiv_User_i)
  begin
     if (SnkIdelayCtlRst = '1') then
       Ready_int         <= '0' after TFF;
       SnkIdelayCtlRdy   <= '0' after TFF;
     elsif (RDClkDiv_User_i'event and RDClkDiv_User_i = '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_User => RDClkDiv_User_i,
    SnkIdelayCtlRdy    => SnkIdelayCtlRdy,

    ..................................
  );
```

XILINX®

Chapter 8

Example Design

This section provides detailed information about the example design, including a description of the file groups, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

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 8-1 shows the example design modules' architecture and interfaces to the SPI-4.2 core.

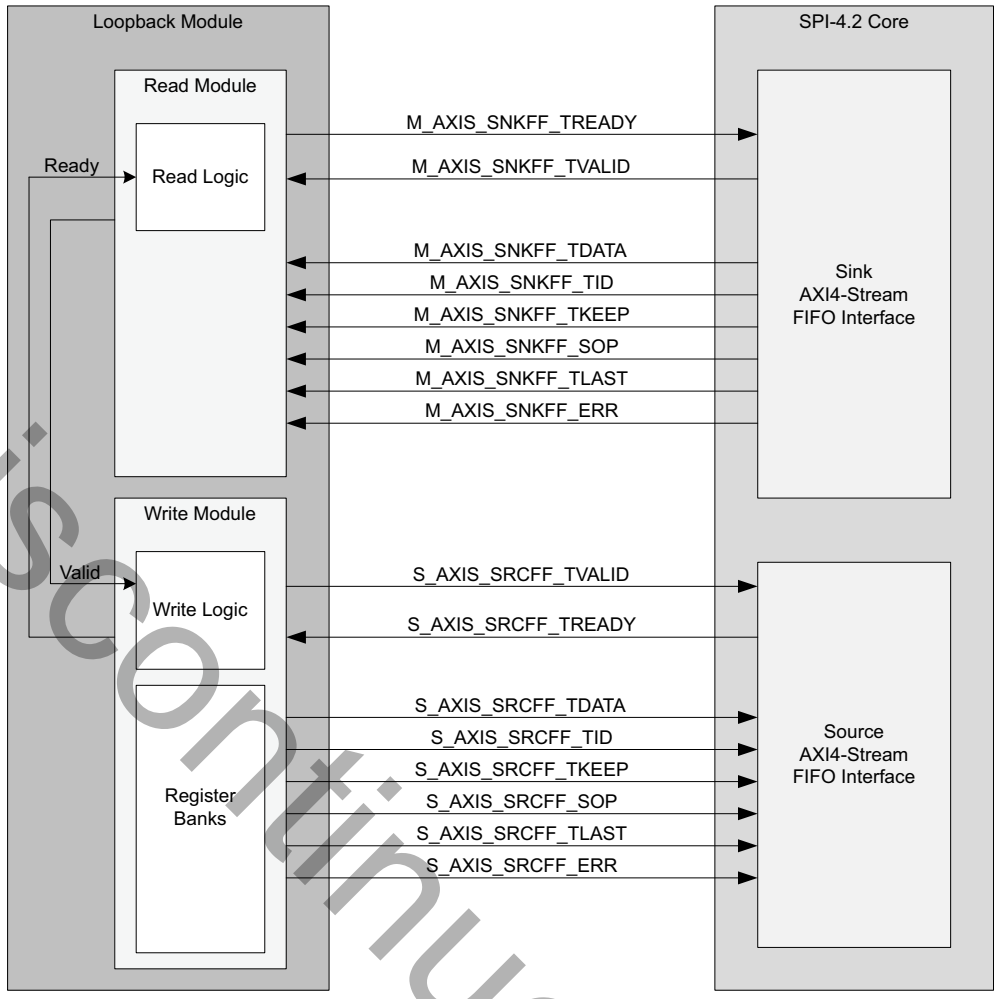

Figure 8-1: **Example Design Configuration**

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 AXI4-Stream FIFO interface and a Write Module that transfers data into the Source AXI4-Stream FIFO. The Write Module checks `S_AXIS_SRCFF_TREADY` to determine whether it can transfer data into the Source FIFO. The Read Module also accesses Source core status information using either the AXI4-Lite Control interface or the AXI4-Stream Status interface (depending on Status mode selected) and then passes this information to the Write Module to be written into the Sink AXI-4 Lite Control interface to update the Sink status.

Basic Loopback Operation

When the Source FIFO is not Almost Full (`S_AXIS_SRCFF_TREADY` is asserted), the Write Module passes a registered version of `S_AXIS_SRCFF_TREADY` to the Read Module. This Ready is used in the Read Module to complete the AXI4-Stream `TREADY/TVALID` handshake on the Sink FIFO interface. The data is then registered twice in the Write Module and written into the AXI4-Stream interface of 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 (TREADY de-asserts), the transfer of data between the FIFOs is stalled immediately. One additional data beat is read from the Sink FIFO and held in the registers until the Source FIFO re-asserts TREADY.

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 8-2.

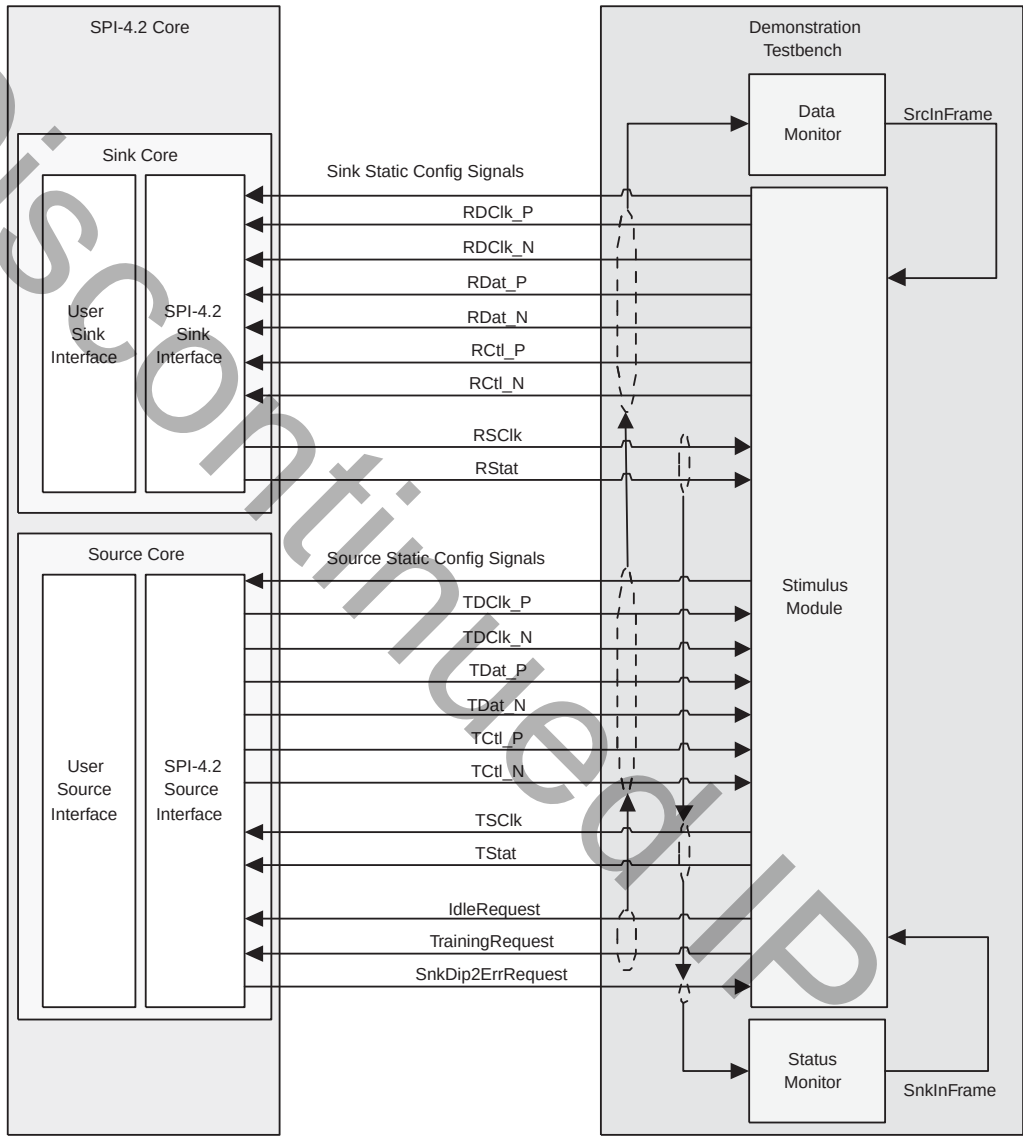

Figure 8-2: **Demonstration Test Bench Connections**

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 8-3, the demonstration test bench consists of the following modules:

- Clock Generator
- Startup
- Stimulus
- Data Monitor
- Status Monitor
- Testcase

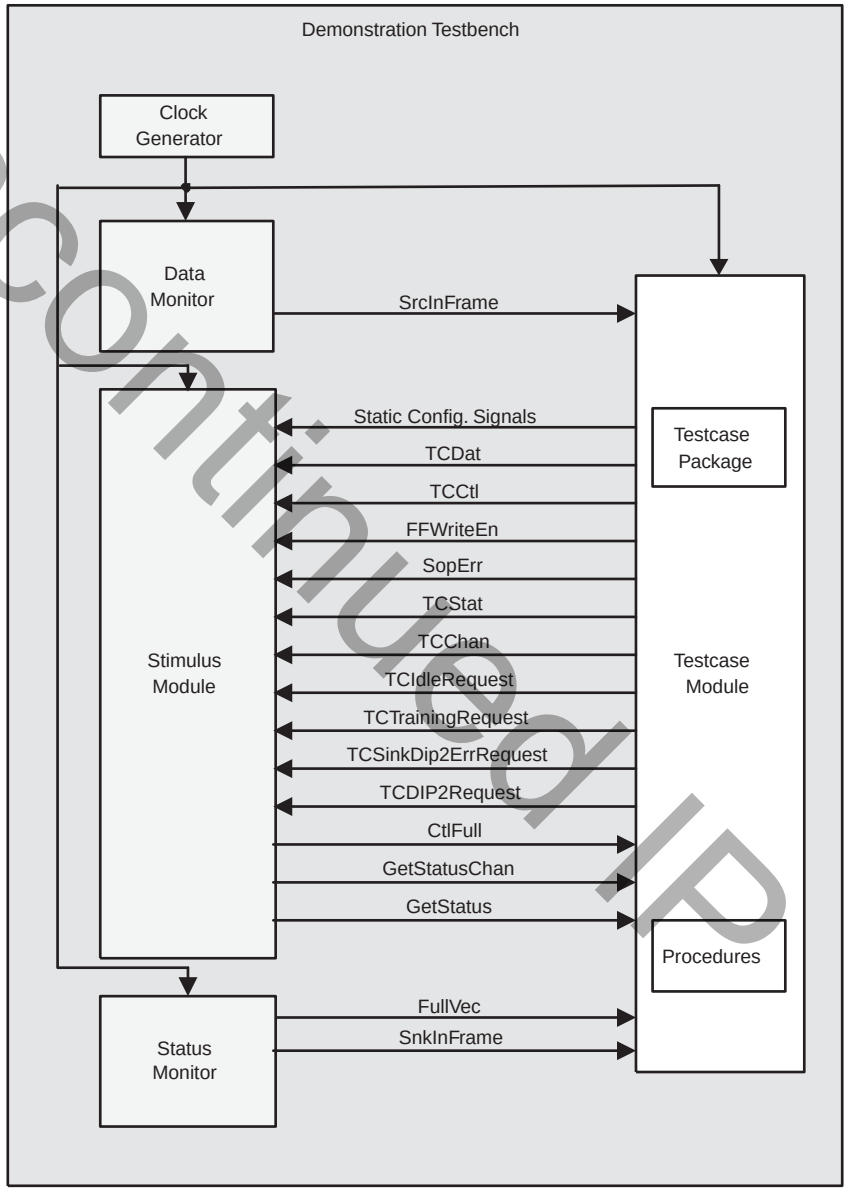

Figure 8-3: **Test Bench Modules**

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 8-1.

Startup Module

The Startup Module contains three functions: MMCM setup, calendar loading, and Dynamic Phase Alignment (DPA) Initialization. These functions are described in detail in the following sections.

MMCM Startup

The MMCM Startup is a state machine that ensures that the MMCMs are reset in the appropriate order. If they are not reset appropriately, the 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, AXI_SNK_ARESETN, AXI_SRC_ARESETN, M_AXIS_SNKFF_ARESETN, and S_AXIS_SRCFF_ARESETN` signals are deasserted only after this occurs. All operations are performed in the `SysClk` domain.

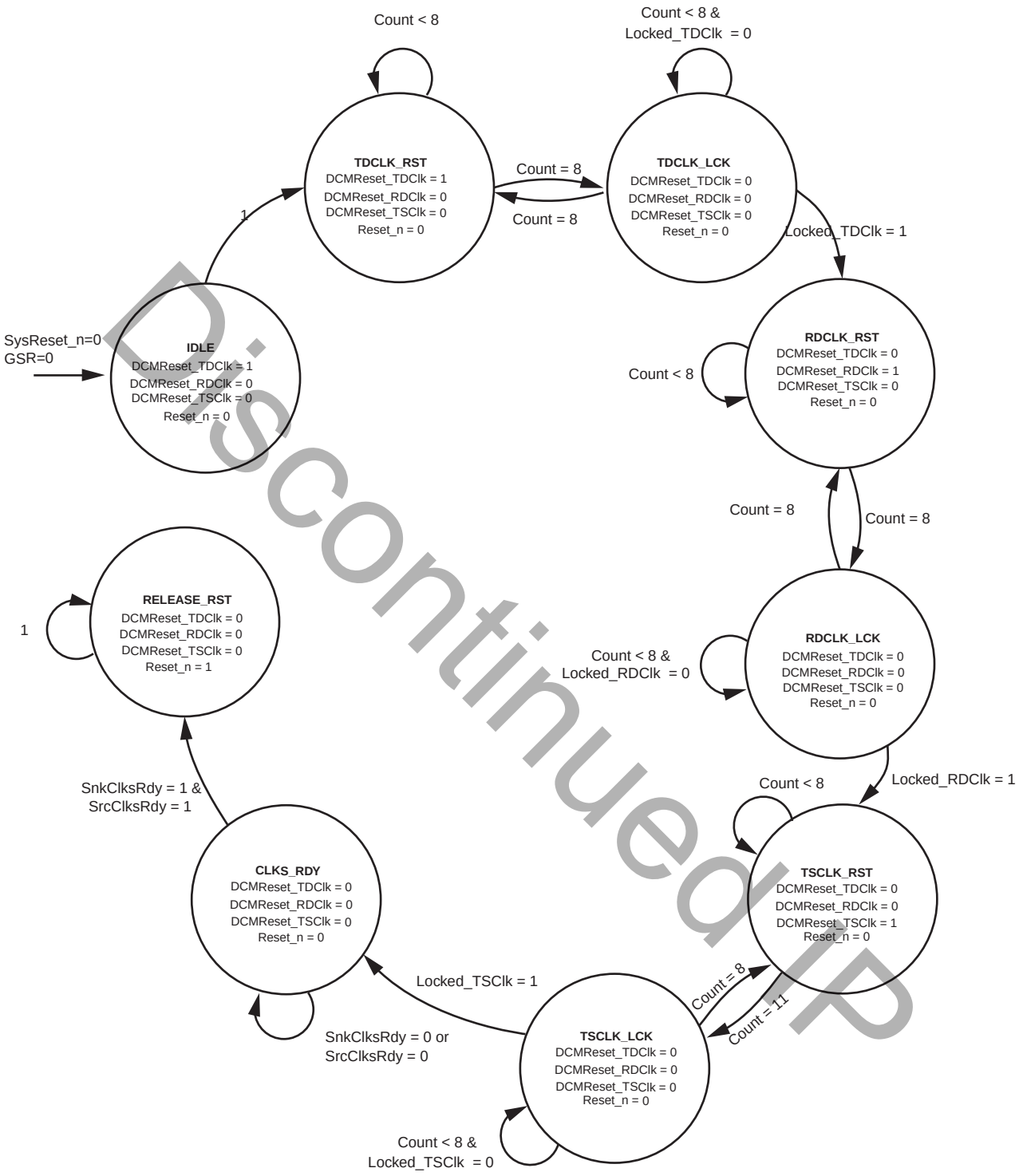

Figure 8-4: **Startup State Diagram**

Figure 8-4 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`, `AXI_SNK_ARESETN, AXI_SRC_ARESETN, M_AXIS_SNKFF_ARESETN, and S_AXIS_SRCFF_ARESETN.`

The MMCM reset duration in this sequence is used for general simulation only. In hardware, the user must follow the MMCM reset duration as specified in the appropriate data sheet:

- http://xilinx.com/support/documentation/7_series.htm

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, using the AXI4-Lite Control interfaces, 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 177].

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 Vivado IP catalog 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 8-2 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 8-1 provides test bench constants that can be modified. These constants are modified by regenerating the core in the Vivado IP catalog.

Table 8-1: **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).
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).

Table 8-1: **Testcase Package User-Defined Constants *(Cont'd)***

Name	Constant Type	Default Value (Range)	Description
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>	
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.

Table 8-1: **Testcase Package User-Defined Constants *(Cont'd)***

Name	Constant Type	Default Value (Range)	Description
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:

- SPI-4.2 and demonstration test bench requests
- Source core status and Sink core data

Figure 8-2 shows the interface between the testcase and stimulus modules.

The static configuration parameters are set when the SPI-4.2 core is generated; these signals can also be modified in circuit. The description of these parameters 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 8-2 contains a list of common useful test case signals and descriptions.

Table 8-2: **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.
Reset_n	Reset signal to the Sink and Source core (active low).
SnkEn	Enable signal to the Sink core.
SnkInFrame	Asserted when the Sink core is in frame (as interpreted by the status monitor).
SnkOof	Out-of-Frame signal from the Sink core.

Table 8-2: **Useful Testcase Signals**

SrcEn	Enable signal to the Source core.
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 8-2, page 167). The fifth is encapsulated with the generated data sent to the stimulus module. Table 8-3 details request signals.

Table 8-3: **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 and SrcEn.` 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.

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.

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 Vivado IP catalog.

Name	Description
Example Design Files (<proj_dir>/proj_dir.srcs/sources_1/ip/component_name/component_name/example_design)	
<component_name>_top.xdc	SPI-4.2 Core Wrapper Design Example Constraints File
<component_name>_top.v[hd]	Core VHDL or Verilog Top Level Wrapper File (required for simulation)
<component_name>_pl4_fifo_loopback.v[hd]	FIFO Loopback VHDL or Verilog Top-level File
<component_name>_pl4_fifo_loopback_read.v[hd]	FIFO Loopback VHDL or Verilog Read Module
<component_name>_pl4_fifo_loopback_write.v[hd]	FIFO Loopback VHDL or Verilog Write Module
<component_name>_pl4_snk_clk.v[hd]	Example Sink core clocking module
<component_name>_pl4_src_clk.v[hd]	Example Source core clocking module
Demonstration Test Bench Simulation Files (<proj_dir>/proj_dir.srcs/sources_1/ip/component_name/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 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]	Demo Test Bench Testcase Module
pl4_testcase_pkg.v[hd]	Demo Test Bench 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

Name	Description
Functional Simulation Scripts (<proj_dir>/proj_dir.srcs/sources_1/ip/component_name/component_name/simulation/functional)	
simulate_mti.do	ModelSim script for compiling the SPI-4.2 core functional simulation netlist and demo test bench 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 test bench files for Incisive Enterprise Simulator (IES); this script also loads and runs the simulation.
simulate_ncsim.bat	DOS shell script for compiling the SPI-4.2 core functional simulation netlist and demo test bench 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.

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.

Appendix F

SPI-4.2 Calendar Programming

This appendix lists examples that describe how to program the Sink and Source Calendar Memory 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 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.

Appendix G

Debugging

This appendix contains details about debugging the core.

Alignment Issues

If alignment issues occur during hardware operation (for example, DIP4 Errors or Sink is out of frame), the DPA (Dynamic Phase Alignment) diagnostic ports can be used for debugging purposes.

See WP249, *SPI-4.2 Dynamic Phase Alignment White Paper*, for details about the SPI-4.2 Dynamic Phase Alignment Sink core and how to debug hardware issues using the DPA diagnostic ports.

Appendix H

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®-7 FPGAs -1, -2, and -3 devices up to 1.25 Gbps (depending on speed grade) on the SPI-4.2 Interface and 312 MHz on the User Interface (`M_AXIS_SNKFF_ACLK` and `S_AXIS_SRCFF_ACLK` clocks).
  - Kintex®-7 FPGAs -1, -2, and -3 devices up to 1Gbps (depending on speed grade) on the SPI-4.2 Interface and 250MHz on the User Interface (`M_AXIS_SNKFF_ACLK` and `S_AXIS_SRCFF_ACLK` 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 TKEEPs.

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
- `S_AXIS_SRCFF_ACLK` and `M_AXIS_SNKFF_ACLK`: 150 MHz to 350 MHz
- `AXI_SNK_ACLK` and `AXI_SRC_ACLK`: 75MHz to 175MHz

XILINX®

Appendix I

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 Clock Interface 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.

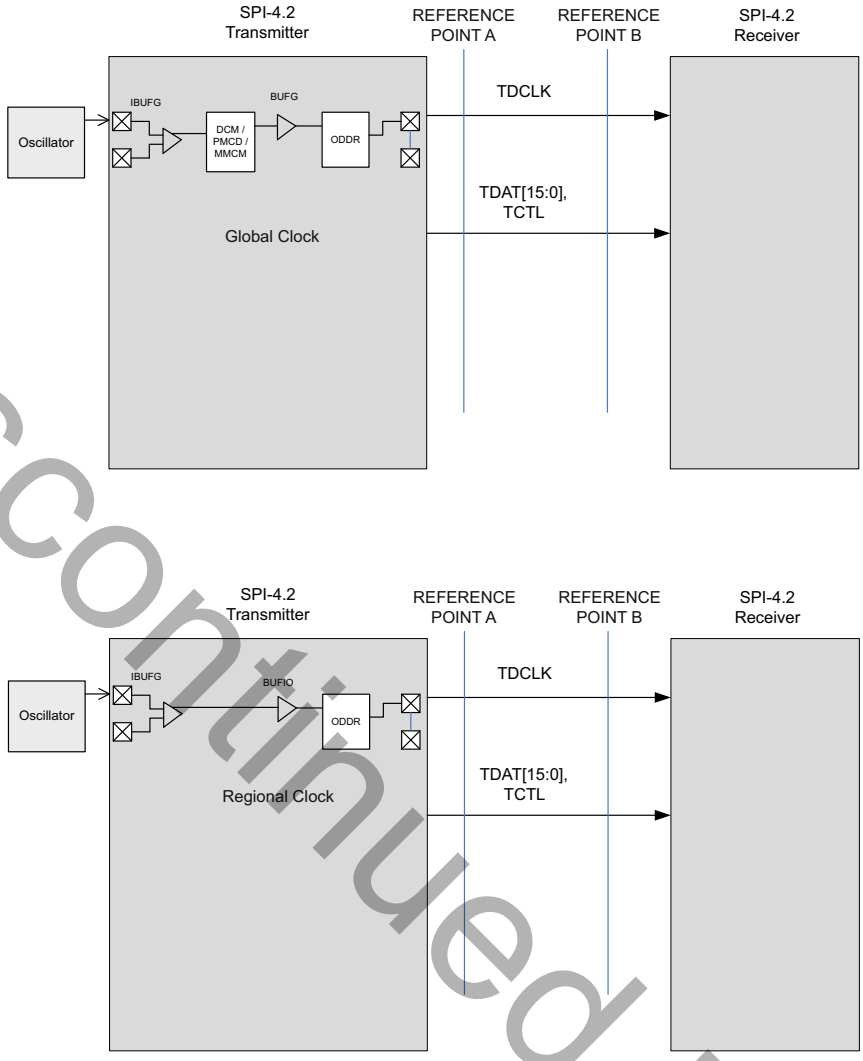

Figure I-1: **OIF Specification Reference Points**

XILINX®

Appendix J

Migrating to SPI-4.2 v11.2 and Later

Note: For details about migrating from the ISE Design Suite to the Vivado Design Suite, see UG911, *Vivado Design Suite Migration Methodology Guide.*

Starting with the SPI-4.2 v11.1 core, the user interfaces to the core have undergone significant changes. All of the user interfaces, except for the general core "Status and Control" interfaces, have been standardized to an AXI4 type of interface. Specifically, the FIFO data interfaces now operate as an AXI4-Stream interface, and the three separate FIFO Status, Calendar, and Static Configuration interfaces have been combined into a single AXI4-Lite interface. With the change to AXI4-Stream for the FIFOs also comes a different behavior for data transfer: the Sink FIFO now "pushes" data to the user when it is available, and both Sink and Source FIFOs require a "ready/valid" handshake to transfer data. The changes also affect the initialization sequence for the cores. The following

figures offer a general comparison between the pre-v11.1 interfaces and the new v11.1 interfaces.

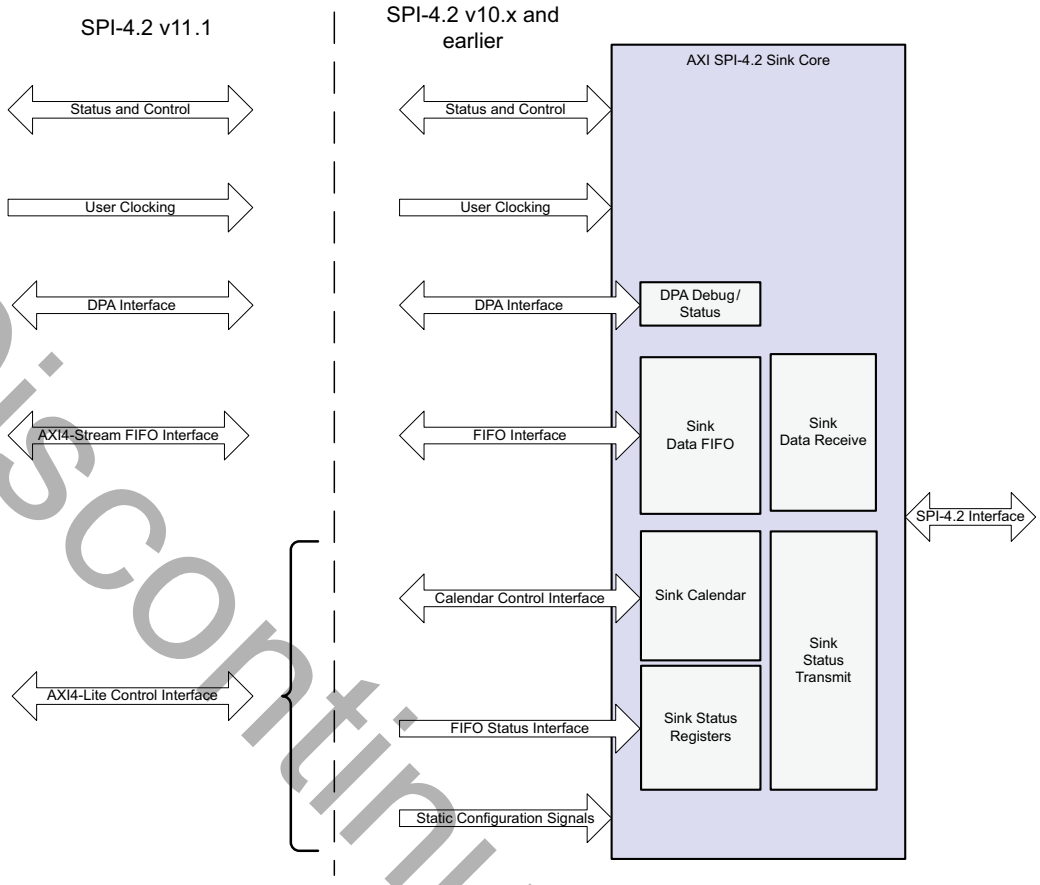

Figure J-1: **Sink Core Interface Migration**

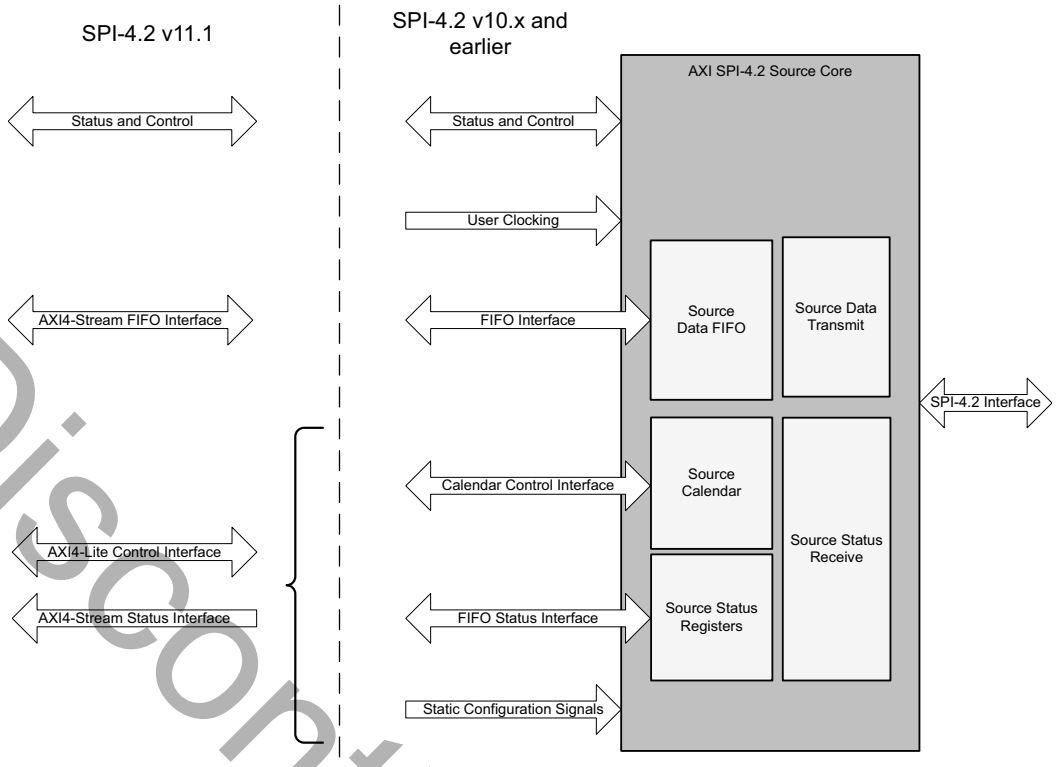

Figure J-2: **Source Core Interface Migration**

The following sections describe in more detail the key changes to each interface, how the previous cores' ports map to the new interfaces, and how the initialization sequence has changed.

Interface Differences

The pre-v11.1 SPI-4.2 Sink core has five separate user interfaces: The Status and Control interface, the Sink FIFO Interface, the Sink Calendar Control Interface, the Sink FIFO Status Interface, and the Sink Static Configuration Interface. The pre-v11.1 SPI-4.2 Source core has five separate user interfaces: The Status and Control interface, the Source FIFO Interface, the Source Calendar Control Interface, the Source FIFO Status Interface, and the Source Static Configuration Interface. The only interfaces that remain the same in v11.1 are the Sink and Source Status and Control interfaces. The following section describes, using pre-v11.1 interface names, how the other interfaces have been modified for the SPI-4.2 v11.1 core.

Sink FIFO Interface

The Sink FIFO now uses an AXI4-Stream master interface to transfer data to the user application. The signals for this interface have similar general behavior, with some key differences:

- The `SnkFFValid` signal is now `M_AXIS_SNKFF_TVALID`. Previously, the FIFO operated in a "pull" behavior, where the user would first assert `SnkFFRdEn_n` and the FIFO would respond with data and a `SnkFFValid` assertion. The new AXI4-Stream interface requires a change to a "push" behavior. This means if data is

available to be read from the FIFO, the core will automatically present (push) the data to the user and assert `M_AXIS_SNKFF_TVALID`, and will wait until this data beat has been accepted via the `TREADY/TVALID` handshake before presenting the next data beat.

- The `SnkFFRdEn_n` signal is now `M_AXIS_SNKFF_TREADY`, and has changed from active-low to active-high. The behavior of this signal is different with AXI as well. Previously, asserting `SnkFFRdEn_n` was used to request a read from the FIFO, and one data beat would appear at the FIFO output on the next clock cycle for each request.

  With the SPI-4.2 v11.1 core, the `M_AXIS_SNKFF_TREADY` signal is part of the TREADY/TVALID handshake and is used to confirm that the user application has received the data beat. Each simultaneous assertion of both `TREADY` and `TVALID` on a clock edge indicates that one data beat has been transferred. Unlike previous versions of the core where `SnkFFValid` would assert in response to `SnkFFRdEn_n`, the assertion of `TVALID` does not require that `TREADY` is first asserted.

- `SnkFFAlmostEmpty_n` and `SnkFFEmpty_n` are removed. With "push" FIFO behavior, the de-assertion of TVALID indicates the FIFO is empty.
- `SnkFFMod[2:0]` (`SnkFFMod[3:0]` for 128-bit interface) is replaced by `M_AXIS_SNKFF_TKEEP[7:0]` (or `[15:0]`). The new signal still indicates which bytes of a data beat are valid, but now provides the information in the form of a per-lane byte enable.
- The ordering of data on `M_AXIS_SNKFF_TDATA` is now little endian. This means the bits `M_AXIS_SNKFF_TDATA [7:0]` were the first byte received on `RDat[15:0]`. Note that since RDat is 16-bits wide, the OIF spec indicates that `RDat[15:8]` is the "first" byte received, and `RDat[7:0]` is the "last." This is further described in Table 5-1, page 72.

Sink Calendar Control Interface

The Sink Calendar Control interface is now combined with the FIFO Status interface and Static Configuration interface into a single Sink AXI4-Lite Control Interface. Reads and writes to the Calendar memory are performed on the Sink AXI4-Lite Control Interface as described in Sink Calendar Memory Initialization in Chapter 5. As with previous cores, writes to the Calendar memory can only be performed while the core is disabled (`SnkEn`=0).

Sink FIFO Status Interface

The Sink FIFO Status interface is now combined with the Calendar interface and Static Configuration interface into a single Sink AXI4-Lite Control Interface. Differing from previous cores, the Status memory is now readable in addition to writeable. Reads and writes to the Status memory are performed on the Sink AXI4-Lite Control Interface as described in Sink Flow Control in Chapter 5. During typical in-frame core operation, the Sink AXI4-Lite Control interface will be only used for Status writes, so the core gives precedence to writes on this interface.

Sink Static Configuration Interface

The Sink Static Configuration interface is now combined with the FIFO Status interface and Calendar Control interface into a single Sink AXI4-Lite Control Interface. Reads and writes to the Static Configuration memory are performed on the Sink AXI4-Lite Control

Interface as described in Sink Static Configuration Parameters in Chapter 5. Previous cores provide access to the Static Configuration parameters via tie-offs whose value could be changed in the core wrapper; if the user wanted to change the value of these parameters in-circuit they would need to create registers to drive these signals. In the AXI SPI-4.2 core, all static configuration values are stored in registers within the core, accessible via the Sink AXI4-Lite Control Interface, and the outputs of these registers are routed to the necessary logic. Unchanged from before, modification of the Static Configuration parameters must be done while the core is in reset (`Reset_n` = 0), as described in Sink Static Configuration Parameters in Chapter 5.

Source FIFO Interface

The Source FIFO now uses an AXI4-Stream slave interface to transfer data from the user application. The signals for this interface have similar general behavior, with some key differences:

- The `SrcFFWrEn_n` signal is now `S_AXIS_SRCFF_TVALID`, and has changed from active-low to active-high. In pre-v11.1 cores, the assertion of `SrcFFWrEn_n` by the user application along with valid data would cause a data beat to be written into the FIFO. The new `S_AXIS_SRCFF_TVALID` signal is used in the same way, but is qualified by the `TVALID/TREADY` handshake now. This means one data beat is written into the Source FIFO for each simultaneous assertion of both `TREADY` and `TVALID` on a clock edge.
- The `SrcFFAlmostFull_n` signal is now `S_AXIS_SRCFF_TREADY`. This change provides the AXI-required `TVALID/TREADY` handshake. Previously, the core would allow data to continue to be written in after `SrcFFAlmostFull_n` assertion-now the core will not accept data if `TREADY` de-asserts. Note that the de-assertion of `TREADY` is still controlled by the Static Configuration parameters SrcAFThresAssert and `SrcAFThresNegate`, just as `SrcFFAlmostFull_n` was controlled.
- The `SrcFFOverflow_n` signal is no longer necessary based on the explanation of `SrcFFAlmostFull_n` changes, and has been removed. In the v11.1 core, the Source FIFO cannot overflow.
- `SrcFFMod[2:0]` (`SrcFFMod[3:0]` for 128-bit interface) is replaced by `S_AXIS_SRCFF_TKEEP[7:0]` (or `[15:0]`). The new signal still indicates which bytes of a data beat are valid, but now provides the information in the form of a per-lane byte enable.
- The ordering of data on `S_AXIS_SRCFF_TDATA` is now little endian. This means the bits `S_AXIS_SRCFF_TDATA [7:0]` are the first byte sent out on `TDat[15:0]`. Note that since `TDat` is 16-bits wide, the OIF spec indicates that `TDat[15:8]` is the "first" byte sent, and `TDat[7:0]` is the "last." This is further described in Table 5-7, page 108.

Source Calendar Control Interface

The Source Calendar Control interface is now combined with the FIFO Status interface (Addressable Mode) and Static Configuration interface into a single AXI4-Lite Control Interface. Reads and writes to the Calendar memory are performed on the AXI4-Lite Control Interface as described in Source Calendar Memory Initialization in Chapter 5. As with previous cores, writes to the Calendar memory can only be performed while the core is disabled (`SrcEn`=0).

Source FIFO Status Interface

When using Addressable Status Mode, the Source FIFO Status interface is now combined with the Calendar interface and Static Configuration interface into a single Source AXI4-Lite Control Interface. The Status memory is read-only, as with previous cores. Reads from the Status memory are performed on the Source AXI4-Lite Control Interface as described in Source Flow Control: Addressable Status Mode in Chapter 5. During typical in-frame core operation, the Source AXI4-Lite Control interface will be only used for Status reads, so the core gives precedence to reads on this interface. Previously, reads from the FIFO Status interface could retrieve up to 16 channels' status every clock cycle. Because of handshaking requirements for the AXI4-Lite interface, reads now retrieve up to four channels' status every other clock cycle.

When using Transparent Status Mode, the Source FIFO Status information is provided solely on the Source AXI4-Stream Status Interface and is not stored within the AXI4-Lite Control interface memory space. The signals `SrcStatCh[7:0]` and `SrcStatChValid` are renamed `M_AXIS_SRCSTAT_TID[7:0]` and `M_AXIS_SRCSTAT_TVALID`, respectively. The `SrcStat[1:0]` signal is now provided on `M_AXIS_SRCSTAT_TUSER[1:0]`. Use of this interface is described in Source AXI4-Stream Status Interface in Chapter 5. The latency of this interface is the same as with the previous cores.

Source Static Configuration Interface

The Source Static Configuration interface is now combined with the FIFO Status interface (Addressable Mode) and Calendar Control interface into a single Source AXI4-Lite Control Interface. Reads and writes to the Static Configuration memory are performed on the Source AXI4-Lite Control Interface as described in Source Static Configuration Parameters in Chapter 5. Previous cores provide access to the Static Configuration parameters via tie-offs whose value could be changed in the core wrapper--if the user wanted to change the value of these parameters in-circuit they would need to create registers to drive these signals. In the AXI SPI-4.2 core, all static configuration values are stored in registers within the core and are accessible via the Source AXI4-Lite Control Interface. In addition, the outputs of these registers are routed to the necessary logic. Unchanged from before, modification of the Static Configuration parameters must be done while the core is in reset (`Reset_n` = 0), as described in Source Static Configuration Parameters in Chapter 5.

Core Initialization Sequence Differences and Resets

The v11.1 core has some changes to the required initialization sequence, as described in detail in Initializing the SPI-4.2 Core in Chapter 5. The primary differences to this sequence are due to the addition of the AXI4-Lite interfaces on the Sink and Source cores. Specifically, the AXI4-Lite interfaces each have their own reset, where before the Calendar and Status interfaces and memories were reset by the core-wide reset (`Reset_n`). The initialization sequence now also specifies when writes to the Static Configuration parameters can occur (that is, only when `Reset_n` is asserted), since in the reset state the core is not in operation and not using the static configuration parameters to define its output. A reset of the AXI4-Lite interface will revert these values to the values set in the Vivado IP catalog.

The following summarizes the Sink and Source core resets, and which parts of the core are affected:

- `Reset_n`: Resets the entire data path and SPI-4.2 interface, including data FIFO contents and AXI4-Stream FIFO interface, ISERDES/OSERDES, status paths (TStat

and RStat) and DPA logic. Does NOT reset the AXI4-Lite interface and associated memories.

- `M_AXIS_SNKFF_ARESETN`: Resets the Sink FIFO and Sink AXI4-Stream FIFO interface while allowing the Sink core to remain in-frame (SnkOof=0). Requires minimum reset pulse width of three cycles of `M_AXIS_SNKFF_ACLK`.
- `S_AXIS_SRCFF_ARESETN`: Resets the Source FIFO and Source AXI4-Stream FIFO interface while allowing the Source core to remain in-frame (SrcOof=0). Requires minimum reset pulse width of three cycles of `S_AXIS_SRCFF_ACLK`.
- `AXI_SNK_ARESETN`: Resets the Sink AXI4-Lite interface logic. Resets associated memories to default values (that is, Vivado IP catalog selected settings). These memories are Status memory, Calendar memory, and Static Configuration memory. Requires minimum reset pulse width of two cycles of `AXI_SNK_ACLK`.
- `AXI_SRC_ARESETN`: Resets the Source AXI4-Lite interface logic. Resets associated memories to default values (that is, Vivado IP catalog selected settings). These memories are Status memory, Calendar memory, and Static Configuration memory. Requires minimum reset pulse width of two cycles of `AXI_SRC_ACLK`.

Sink Core Port Changes

Table J-1 details changes to the Sink Core port naming, additional or deprecated ports, and polarity changes from v10.x and earlier cores to v11.1. Note that none of the SPI-4.2 Interface signals have changed, as their behavior is described by the OIF-SPI4-02.1 System Packet Interface Phase 2 standard.

Table J-1: **Sink Core Port Map from v10.x to v11.1**

Version 10.x and earlier	Version 11.1	Notes
Sink Control and Status Interface	**Sink Control and Status Interface**	
Reset_n	Reset_n	No Change
SnkEn	SnkEn	Changed to RDClkDiv_User domain
SnkTrainValid	SnkTrainValid	Changed to RDClkDiv_User domain
SnkIdelayCtlRdy	SnkIdelayCtlRdy	No Change
SnkDIP2ErrRequest	SnkDIP2ErrRequest	Changed to RDClkDiv_User domain
SnkOof	SnkOof	Changed to RDClkDiv_User domain
SnkBusErr	SnkBusErr	Changed to RDClkDiv_User domain
SnkBusErrStat[7:0]	SnkBusErrStat[7:0]	Changed to RDClkDiv_User domain
PhaseAlignRequest	SnkDPAPhaseAlignRequest	Rename, Changed to RDClkDiv_User domain
PhaseAlignComplete	SnkDPAPhaseAlignComplete	Rename, Changed to RDClkDiv_User domain
SnkDPAFailed	SnkDPAFailed	Changed to RDClkDiv_User domain
SnkDPARamAddr[5:0]	SnkDPARamAddr[5:0]	No Change
SnkDPARamData[16:0]	SnkDPARamData[16:0]	No Change
SnkDPARamValid	SnkDPARamValid	No Change
SnkCDPAHalt	SnkCDPAHalt	No Change

Table J-1: **Sink Core Port Map from v10.x to v11.1**

Version 10.x and earlier	Version 11.1	Notes
SnkDPADiagWin	SnkDPADiagWin	No Change
SnkDPAAddrRst	SnkDPAAddrRst	No Change
SnkDPAAddrEn	SnkDPAAddrEn	No Change
SnkDPAClkDlyRst	SnkDPAClkDlyRst	No Change
SnkDPAClkDlyCe	SnkDPAClkDlyCe	No Change
SnkDPAClkDlyInc	SnkDPAClkDlyInc	No Change
Sink FIFO interface	**Sink AXI4-Stream FIFO Interface**	
SnkFifoReset_n	M_AXIS_SNKFF_ARESETN	Rename
SnkFFClk	M_AXIS_SNKFF_ACLK	Rename
SnkFFRdEn_n	M_AXIS_SNKFF_TREADY	Rename, change sense (now active high), and requires handshake with M_AXIS_SNKFF_TVALID. Change to "Push" FIFO, meaning if the FIFO contains valid data, it will not wait until M_AXIS_SNKFF_TREADY is asserted before asserting M_AXIS_SNKFF_TVALID
SnkFFValid	M_AXIS_SNKFF_TVALID	Rename, change to "Push" behavior where data is automatically presented to user if available in the FIFO
SnkFFAddr[7:0]	M_AXIS_SNKFF_TID[7:0]	Rename
SnkFFData[63:0],[127:0]	M_AXIS_SNKFF_TDATA[63:0], [127:0]	Rename
SnkFFMod[2:0],[3:0]	M_AXIS_SNKFF_TKEEP[7:0], [15:0]	Rename, change to byte qualifier indicating which are valid bytes
SnkFFSOP	M_AXIS_SNKFF_SOP	Rename
SnkFFEOP	M_AXIS_SNKFF_TLAST	Rename
SnkFFErr	M_AXIS_SNKFF_ERR	Rename
SnkFFDIP4Err	M_AXIS_SNKFF_DIP4ERR	Rename
SnkFFPayloadDIP4	M_AXIS_SNKFF_PAYLOADDIP4	Rename
SnkFFBurstErr	M_AXIS_SNKFF_BURSTERR	Rename
SnkFFPayloadErr	M_AXIS_SNKFF_PAYLOADERR	Rename
SnkFFAlmostEmpty_n		Removed
SnkFFEmpty_n		Removed. De-asserted M_AXIS_SNKFF_TVALID indicates empty
SnkAlmostFull_n	SNKFF_ALMOSTFULL_N	Rename
SnkOverflow_n	SNKFF_OVERFLOW_N	Rename

Table J-1: **Sink Core Port Map from v10.x to v11.1**

Version 10.x and earlier	Version 11.1	Notes
Sink Calendar Control Write interface	**Sink AXI4-Lite Control Interface**	
SnkCalClk	AXI_SNK_ACLK	Rename, combine with Status and Config functions
SnkCalData[7:0]	AXI_SNK_WDATA[31:0]	Rename, combine with Status and Config functions, and add AXI protocol channel handshake . Now writes up to 4 calendar spaces at once, instead of one at a time
SnkCalWrEn_n	AXI_SNK_WVALID	
	AXI_SNK_WREADY	
	AXI_SNK_WSTRB[3:0]	
SnkCalAddr[8:0]	AXI_SNK_AWADDR[9:0]	Rename, combine with Status and Config functions, and add AXI protocol channel handshake . Calendar uses AWADDR[9:0] ranges from 0x000 to 0x1FC
	AXI_SNK_AWVALID	
	AXI_SNK_AWREADY	
	AXI_SNK_BVALID	New AXI4-Lite Write Response Channel
	AXI_SNK_BREADY	
	SNK_BRESP_ERR[7:0]	New AXI4-Lite interface error signaling
Sink Calendar Control Read interface	**Sink AXI4-Lite Control Interface**	
SnkCalClk	AXI_SNK_ACLK	Rename, combine with Status and Config functions
SnkCalDataOut[7:0]	AXI_SNK_RDATA[31:0]	Rename, combine with Status and Config functions, and add AXI handshake. Now reads up to 4 calendar spaces at once, instead of one at a time
	AXI_SNK_RVALID	
	AXI_SNK_RREADY	
SnkCalAddr[8:0]	AXI_SNK_ARADDR[9:0]	Rename, combine with Status and Config functions, and add AXI protocol channel handshake. Calendar uses ARADDR[9:0] ranges from 0x000 to 0x1FC
	AXI_SNK_ARVALID	
	AXI_SNK_ARREADY	
	SNK_BRESP_ERR[7:0]	New AXI4-Lite interface error signaling
Sink FIFO Status interface	**Sink AXI4-Lite Control Interface**	
SnkStatClk	AXI_SNK_ACLK	Rename, combine with Calendar and Config functions
SnkStat[31:0]	AXI_SNK_WDATA[31:0]	Rename, combine with Calendar and Config functions, and add AXI protocol channel handshake . Status is now written up to 4 channels at a time instead of 16, so WSTRB is reduced to 4 bits versus 16 bits for SnkStatMask.
SnkStatWr_n	AXI_SNK_WVALID	
	AXI_SNK_WREADY	
SnkStatMask[15:0]	AXI_SNK_WSTRB[3:0]	

Table J-1: **Sink Core Port Map from v10.x to v11.1**

Version 10.x and earlier	Version 11.1	Notes
SnkStatAddr[3:0]	AXI_SNK_AWADDR[9:0]	Rename, combine with Calendar and Config functions, and add AXI protocol channel handshake. Status uses AWADDR[9:0] ranges from 0x200 to 0x2FC
	AXI_SNK_AWVALID	
SnkStatWr_n	AXI_SNK_AWREADY	
	AXI_SNK_RDATA[31:0]	Sink Status information is now readable. Pre-v11.1 cores did not have this capability. Status uses ARADDR[9:0] ranges from 0x200 to 0x2FC. Status is read 4 channels at a time.
	AXI_SNK_RVALID	
	AXI_SNK_RREADY	
	AXI_SNK_ARADDR[9:0]	
	AXI_SNK_ARVALID	
	AXI_SNK_ARREADY	
	AXI_SNK_BVALID	New AXI4-Lite Write Response Channel
	AXI_SNK_BREADY	
	SNK_BRESP_ERR[7:0]	New AXI4-Lite interface error signaling
Sink Static Configuration Interface	**Sink AXI4-Lite Control Interface**	
NumDip4Errors[3:0]	AXI_SNK*, SNK_BRESP_ERR[7:0]	Created new read/write memory space for these parameters. Combined with Calendar and Status functions, and uses AXI4-Lite Control interface to perform reads and writes. Configuration memory uses AWADDR[9:0] and ARADDR[9:0] ranges from 0x300 to 0x30C
NumTrainSequences[3:0]		
SnkCalendarM[7:0]		
SnkCalendarLen[8:0]		
SnkAFThresAssert[8:0]		
SnkAFThresNegate[8:0]		
RSClkDiv		
RSClkPhase		
FifoAFMode[1:0]		
Sink Clocking interface	**Sink Clocking interface**	
RDClk0_User	RDClk0_User	No Change
RDClkDiv_User	RDClkDiv_User	No Change
SnkClksRdy_User	SnkClksRdy_User	No Change

Source Core Port Changes

Table J-2 details changes to the Source Core port naming, additional or deprecated ports, and polarity changes from v10.x and earlier cores to v11.1. Note that none of the SPI-4.2 Interface signals have changed, as their behavior is described by the OIF-SPI4-02.1 System Packet Interface Phase 2 standard.

Table J-2: **Source Core Port Map from v10.x to v11.1**

Version 10.x and earlier	Version 11.1	Notes
Source Control and Status	**Source Control and Status**	
Reset_n	Reset_n	No Change
SrcEn	SrcEn	Changed to SysClkDiv_User domain
SrcOof	SrcOof	Changed to SysClkDiv_User domain
SrcDIP2Err	SrcDIP2Err	Changed to M_AXIS_SRCSTAT_ACLK domain
SrcStatFrameErr	SrcStatFrameErr	Changed to M_AXIS_SRCSTAT_ACLK domain
SrcPatternErr	SrcPatternErr	No Change
IdleRequest	IdleRequest	Changed to SysClkDiv_User domain
TrainingRequest	TrainingRequest	Changed to SysClkDiv_User domain
SrcTriStateEn	SrcTriStateEn	Changed to SysClkDiv_User domain
SrcOofOverride	SrcOofOverride	Changed to SysClkDiv_User domain
Source FIFO Interface	**Source AXI4-Stream FIFO Interface**	
SrcFifoReset_n	S_AXIS_SRCFF_ARESETN	Rename
SrcFFClk	S_AXIS_SRCFF_ACLK	Rename
SrcFFWrEn_n	S_AXIS_SRCFF_TVALID	Rename, change sense (now active high), and requires handshake with S_AXIS_SRCFF_TREADY
SrcFFAddr[7:0]	S_AXIS_SRCFF_TID[7:0]	Rename
SrcFFData[63:0],[127:0]	S_AXIS_SRCFF_TDATA[63:0], [127:0]	Rename
SrcFFMod[2:0],[3:0]	S_AXIS_SRCFF_TKEEP[7:0], [15:0]	Rename, change to byte qualifier indicating which are valid bytes
SrcFFSOP	S_AXIS_SRCFF_SOP	Rename
SrcFFEOP	S_AXIS_SRCFF_TLAST	Rename
SrcFFErr	S_AXIS_SRCFF_ERR	Rename
SrcFFAlmostFull_n	S_AXIS_SRCFF_TREADY	Rename. If FIFO is AlmostFull, S_AXIS_SRCFF_TREADY de-asserts
SrcFFOverflow_n		Removed. FIFO cannot overflow since S_AXIS_SRCFF_TREADY will de-assert on Almost Full.

Table J-2: **Source Core Port Map from v10.x to v11.1**

Version 10.x and earlier	Version 11.1	Notes
Source Calendar Control Write Interface	**Source AXI4-Lite Control Interface**	
SrcCalClk	AXI_SRC_ACLK	Rename, combine with Status and Config functions
SrcCalData[7:0]	AXI_SRC_WDATA[31:0]	Rename, combine with Status and Config functions, and add AXI protocol channel handshake. Now writes up to 4 calendar spaces at once, instead of one at a time
SrcCalWrEn_n	AXI_SRC_WVALID	
	AXI_SRC_WREADY	
	AXI_SRC_WSTRB[3:0]	
SrcCalAddr[8:0]	AXI_SRC_AWADDR[9:0]	Rename, combine with Status and Config functions, and add AXI handshake. Calendar uses AWADDR[9:0] ranges from 0x000 to 0x1FC
SrcCalWrEn_n	AXI_SRC_AWVALID	
	AXI_SRC_AWREADY	
	AXI_SRC_BVALID	New AXI4-Lite Write Response Channel
	AXI_SRC_BREADY	
	SRC_BRESP_ERR[7:0]	New AXI4-Lite interface error signaling
Source Calendar Control Read Interface	**Source AXI4-Lite Control Interface**	
SrcCalClk	AXI_SRC_ACLK	Rename, combine with Status and Config functions
SrcCalDataOut[7:0]	AXI_SRC_RDATA[31:0]	Rename, combine with Status and Config functions, and add AXI protocol channel handshake. Now reads up to 4 calendar spaces at once, instead of one at a time
	AXI_SRC_RVALID	
	AXI_SRC_RREADY	
SrcCalAddr[8:0]	AXI_SRC_ARADDR[9:0]	Rename, combine with Status and Config functions, and add AXI protocol channel handshake. Calendar uses ARADDR[9:0] ranges from 0x000 to 0x1FC
	AXI_SRC_ARVALID	
	AXI_SRC_ARREADY	
	SRC_BRESP_ERR[7:0]	New AXI4-Lite interface error signaling
Source FIFO Status Interface (Transparent)	**Source AXI4-Stream Status Interface**	
TSClk_User	M_AXIS_SRCSTAT_ACLK	Derive from TSClk_P/N from SPI-4.2 interface
SrcStat[1:0]	M_AXIS_SRCSTAT_TUSER[1:0]	Rename only
SrcStatCh[7:0]	M_AXIS_SRCSTAT_TID[7:0]	Rename only
SrcStatChValid	M_AXIS_SRCSTAT_TVALID	Rename only

Table J-2: **Source Core Port Map from v10.x to v11.1**

Version 10.x and earlier	Version 11.1	Notes
Source FIFO Status Interface (Addressable)	**Source AXI4-Lite Control Interface**	
SrcStatClk	AXI_SRC_ACLK	Rename, combine with Calendar and Config functions
SrcStat[31:0]	AXI_SRC_RDATA[31:0]	Rename, combine with Calendar and Config functions, and add AXI protocol channel handshake. Status is now read 4 channels at a time instead of 16. Status space is not writeable via the AXI4-Lite interface.
	AXI_SRC_RREADY	
	AXI_SRC_RVALID	
SrcStatAddr[3:0]	AXI_SRC_ARADDR[9:0]	Rename, combine with Calendar and Config functions, and add AXI protocol channel handshake. Status uses ARADDR[9:0] ranges from 0x200 to 0x2FC
	AXI_SRC_ARREADY	
	AXI_SRC_ARVALID	
	SRC_BRESP_ERR[7:0]	New AXI4-Lite interface error signaling
Source Configuration Interface	**Source AXI4-Lite Control Interface**	
SrcBurstMode	AXI_SRC*, SRC_BRESP_ERR[7:0]	Created new read/write memory space for these parameters. Combined with Calendar and Status functions, and uses AXI4-Lite Control interface to perform reads and writes. Configuration memory uses AWADDR[9:0] and ARADDR[9:0] ranges from 0x300 to 0x310
SrcBurstLen[9:0]		
SrcAFThresAssert[8:0]		
SrcAFThresNegate[8:0]		
SrcCalendar_M[7:0]		
SrcCalendar_Len[8:0]		
DataMaxT[15:0]		
AlphaData[7:0]		
NumDip2Errors[3:0]		
NumDip2Matches[3:0]		
Source Clocking Interface	**Source Clocking Interface**	
SysClk0_User	SysClk0_User	No Change
SysClkDiv_User	SysClkDiv_User	No Change
TSClk_User	M_AXIS_SRCSTAT_ACLK	Combine with M_AXIS_SRCSTAT_ACLK.