Soft SPI4 IP Core User's Guide

September 2010
IPUG59_01.7

Table of Contents

Chapter 1. Introduction 4
Quick Facts 4
Features 4
Chapter 2. Functional Description 6
Overview 6
Operational Description 6
SPI4 Transmitter - S4TX 7
SPI4 Transmit Data Protocol - S4TXDP 7
SPI4 Transmit I/O - S4TXIO (TXGB) 9
SPI4 Transmit Status - S4TXSP 11
SPI4 Receiver - S4RX 14
SPI4 Receive Data Protocol - S4RXDP 15
SPI4 Receive Status Protocol - S4RXSP 18
SPI4 Receiver I/O - S4RXIO (RXGB) 21
Calendar and Status RAM Access 22
Start-Up Procedures 23
Receive Direction Start-Up 23
Dynamic Mode Start-up and Recovery (SMSR) FSM 23
Static Mode Start-up and Recovery (SMSR) FSM 23
Transmit Direction Start-Up 24
Signal Descriptions 24
Chapter 3. Parameter Settings 33
Global Tab 36
User Data Interface 36
Generation Options 36
Transmit Tab 37
Transmit Data Path Options 37
Transmit Line Side FIFO Thresholds 37
Transmit User Side FIFO Thresholds 37
Transmit Packing Enable 38
Receive Tab – LatticeECP 38
Receive Data Path Options 38
Receive Tab – Lattice SC/SCM 39
Receive Data Path Options 39
Status Tab 40
Status Channel Options 40
Transmit Status Path Options 40
Receive Status Path Options 40
Calendars Tab 41
Transmit Calendar Options 41
Receive Calendar Options 41
Chapter 4. IP Core Generation 42
Licensing the IP Core 42
Getting Started 42
IPexpress-Created Files and Top Level Directory Structure 44
Instantiating the Core 46
Running Functional Simulation 46
Synthesizing and Implementing the Core in a Top-Level Design 47
Hardware Evaluation 48

© 2010 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal. All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Enabling Hardware Evaluation in Diamond ... 48
Enabling Hardware Evaluation in ispLEVER ... 48
Updating/Regenerating the IP Core ... 48
Regenerating an IP Core in Diamond ... 49
Regenerating an IP Core in ispLEVER ... 49
Chapter 5. Application Support ... 51
Hard-Core Physical Placement ... 51
SPI4 Line-Side I/O ... 51
Clocking and Synchronization ... 51
Clock List ... 51
Clock Usage Diagram ... 52
System-Level Synchronization ... 55
Selecting a System Data Clock Frequency ('SDCK') - Receiver ... 56
Selecting a System Data Clock Frequency ('SDCK') - Transmitter ... 58
Chapter 6. Core Verification ... 59
Chapter 7. Support Resources ... 60
Lattice Technical Support ... 60
Online Forums ... 60
Telephone Support Hotline ... 60
E-mail Support ... 60
Local Support ... 60
Internet ... 60
References ... 60
LatticeECP3 ... 60
LatticeSCM ... 60
Revision History ... 61
Appendix A. Resource Utilization ... 62
LatticeECP3 FPGAs ... 62
Supplied Netlist Configurations ... 62
LatticeSC/M FPGAs ... 62
Supplied Netlist Configurations ... 62

The Soft System Packet Interface 4 (SPI4) Intellectual Property (IP) core enables user instantiation of OIF-compliant System Packet Interface Level 4 Phase 2 Revision 1 (SPI4.2.1) cores in Lattice Field Programmable Gate Arrays (FPGAs).

The Soft SPI4 IP core supports up to 256 data channels with aggregate throughputs of between 3 and 12.8Gbps and can be used to connect network processors with OC192 framers, mappers, and fabrics, as well as Gigabit and 10-Gigabit Ethernet MACs. This user's guide explains the functionality of the SPI4 core and how it can be applied to interconnect physical and link layer devices in 10Gbps POS, Ethernet, and ATM applications.

Quick Facts

Table 1-1 gives quick facts about the Soft SPI4 IP core.

Table 1-1. Soft SPI4 IP Core Quick Facts

		Soft SPI4 IP Core Configuration			
Core Requirements	FPGA Families Supported	**LatticeECP3™**		**LatticeSC/SCM™**	
	Minimal Device Needed	LFE3-35EA-8FN484CES	LFE3-35EA-8FN484CES	LFSC3GA15E-6F900C	LFSC3GA15E-6F900C
Resources Utilization	Target Device	LFE3-17EA-7FN484CES	LFE3-17EA-7FN484CES	LFSC3GA15E-6F900C	LFSC3GA15E-6F900C
	Status Mode	Transparent	RAM	Transparent	RAM
	Data Path Width	100	200	100	200
	LUTs	2500	4100	3200	5300
	sysMEM EBRs	12	18	12	18
	Registers	3000	4800	3000	4900
Design Tool Support	Lattice Implementation	Diamond® 1.0 or ispLEVER® 8.1			
	Synthesis	Synopsys® Synplify™ Pro for Lattice D-2009.12L-1			
	Simulation	Aldec® Active-HDL™ 8.2 Lattice Edition			
		Mentor Graphics ModelSim™ SE 6.3F (			

Features

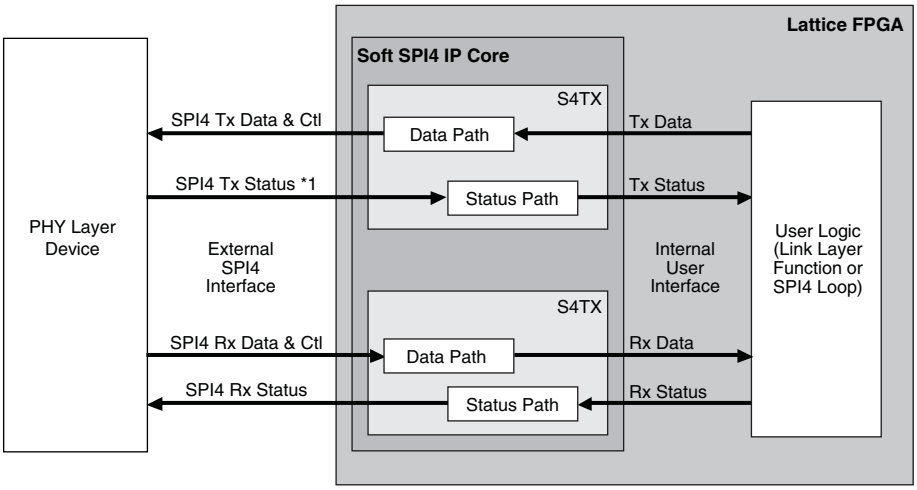
- The Soft SPI4 IP core is fully compliant with the OIF System Packet Interface Level 4 Phase 2 Revision 1 (SPI4.2.1) interface standard
- Supported through Diamond or ispLEVER IPexpress™ tool for easy user configuration and parameterization
- Supports up to 256 independent channels
- 400 to 500MHz DDR Dynamic mode operation in LatticeSC and LatticeSCM devices
- 156 to 350MHz DDR Static timing mode operations for LatticeECP3 devices. Supports non-standard “SPI4 Lite” line rates.
- Supports both 64b and 128b internal architectures for optimization of either speed or size
- Requires only ~2000 slices (64b mode) for a full 256-channel Static mode core
- Supports full bandwidth utilization of the SPI4 line in both directions - requires no idle cycles in the receive direction or insertion of idles in the transmit direction between bursts (as long as there is data available)

- Parity error checking/generation on all receive and transmit control and data words (DIP4) and status (DIP2) interfaces
- Parity error force capabilities on data (independent controls: control word and data) and status interfaces
- Various run-time user controls
  - Force idles (transmitter)
  - Enable/disable packing (transmitter)
  - Training pattern (CAL_M, MAX_T)
- Complete run-time programmability of all internal FIFO thresholds for efficient management of SPI4 line in terms of Lmax and packing
- Provides a direct interface to primary device I/O at the SPI4 interface and an internal FIFO interface to user logic
- Supports minimum transmit burst sizes in increments of 16 bytes from 16 bytes up to 1008 bytes for optimized network processor applications
- Support for packet sizes down to 4 bytes in length
- Fully configurable 512-location calendar RAM for Rx and Tx directions and associated 256-location status RAMs
- Two independently configurable methods of status reporting in the receive and transmit directions - RAM addressable and Transparent
- Rising or falling edge selectable Status Channel I/O independently configurable in the receive and transmit directions

Figure 2-1 shows a system-level diagram of a typical Link layer application where the Soft SPI4 IP core is implemented in a Lattice FPGA. At the top level, the core is broken into two sub-blocks referred to as the SPI4 Transmitter (S4TX) and SPI4 Receiver (S4RX). The S4RX and S4TX blocks provide both status and data path functionality for the direction they serve. They provide a direct interface to the primary I/O of the device on one side (SPI4) and a device-internal FIFO interface to user logic on the other.

Also included is a user-side SPI4 “loop-around module” and a SPI4 test-bench for optional use. The loop-around module loops receive SPI4 data back to the SPI4 transmitter and transmit status back to the SPI4 receiver. An FPGA top-level RTL template design is provided that includes the IP core and loop-around module which can be used without modification for simulation verification and can also be synthesized, placed, and routed “as is” for initial debugging on physical hardware. With this capability, the user can connect their system to a Lattice FPGA via a SPI4 interconnect and easily verify the speed and functionality of the core.

Figure 2-1. Soft SPI4 IP Core, System-Level Context



Overview

The Soft SPI4 IP core is used with additional user-side application logic that interfaces with the IP core via separate receive and transmit FIFO interfaces for SPI4 data information and separate receive and transmit interfaces for SPI4 flow control information. The data FIFOs (4KB 64b mode, 8KB 128b mode) are implemented using Embedded Block RAM (EBR) and provide shared channel buffering on a SPI4 line basis; there is no per-channel buffering within the core for the base design. User-side application logic is responsible for scheduling the maximum allowable SPI4 burst size and the overall amount of data (through credit accounting) that may be written into the S4TX FIFO and transmitted on a per-channel basis. The information needed by the user to manage the credit accounting procedure is received and transmitted on a per-channel basis via the status channel. Both the S4RX and S4TX modules support RAM mode and Transparent mode interfaces that are user selectable for transmitting and receiving status information as well as individual Calendar RAMs that contain configuration data defining the channel order and duration for which status information is transmitted and received for each channel.

Operational Description

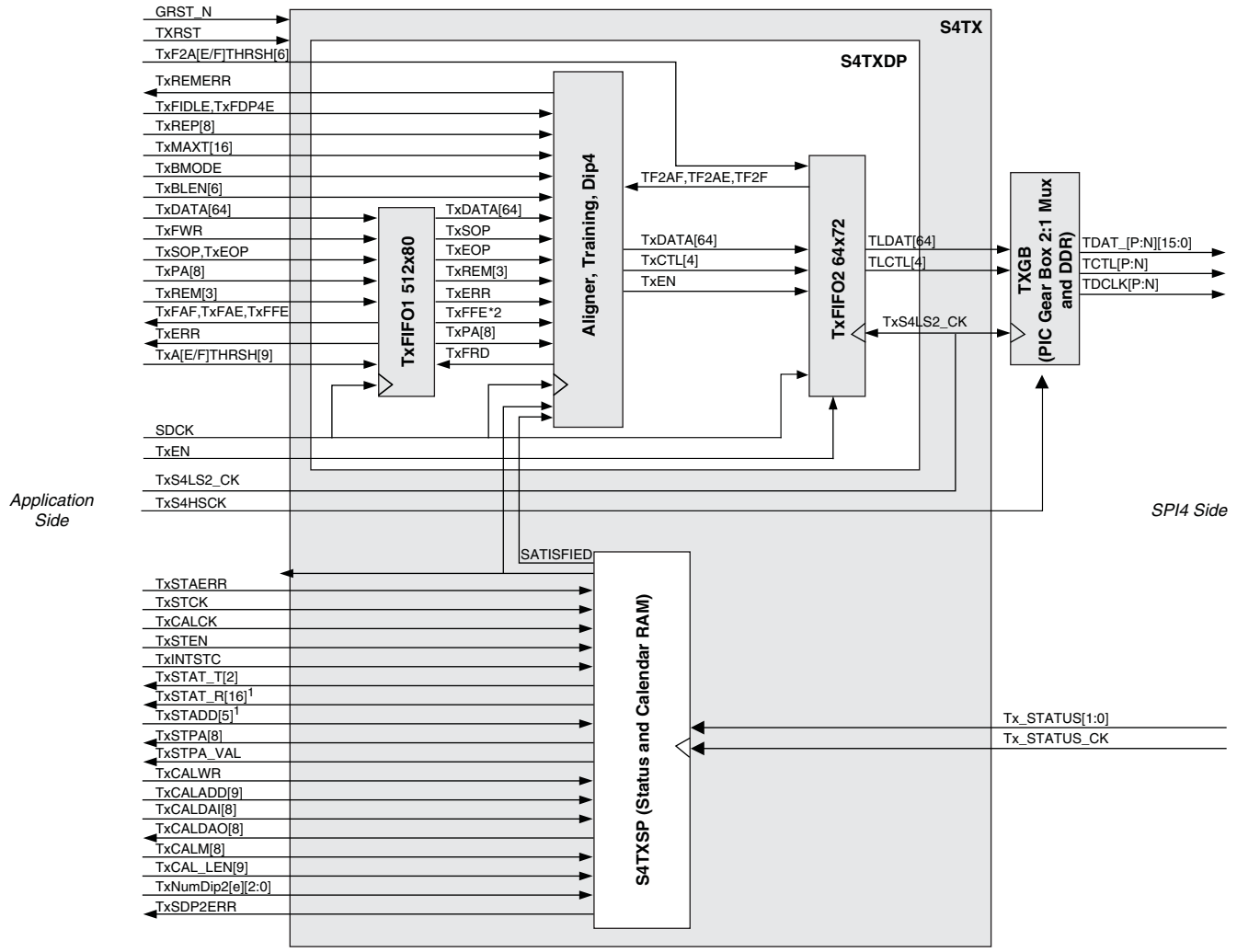
For the following descriptions, designations enclosed in single quotes (e.g. 'SDCK') refer to specific Soft SPI4 IP core I/O port names or synthesis parameters. Refer to “Signal Descriptions” on page 24 and “Parameter Settings” on page 33t for detailed descriptions of the functionality of these items.

With regard to timing diagram examples, there are a number of other simulation scenarios that are not captured here but are available for user simulation and viewing through evaluation simulation (see “Running Functional Simulation” on page 46 for a list of simulation scenarios available).

SPI4 Transmitter - S4TX

The transmit path, shown in Figure 2-2, is the path of data flow from the internal user application function towards the SPI4 line interface and the direction of status flow from the SPI4 line towards the application function. Figure 2-2 indicates through shading some of the hierarchical boundaries and identifies three distinct sub-sections of the S4TX block as described in the following sections.

Figure 2-2. S4TX - SPI4 Transmit Path (64b Mode)



1. In Transparent mode,TxSTADD[5] andTxSTAT_R[16] do not exist and do not have an I/O appearance.

SPI4 Transmit Data Protocol - S4TXDP

In this direction, the S4TXDP block automatically multiplexes data bursts received from the user-side transmit FIFO on a per-channel basis onto the SPI4 line using standard SPI4 port switching “control words”. It uses the sop, eop, abt, and port ID fields received from the user to know when to start, stop, abort, or switch the channel on the SPI4 line. Both read and write sides of the user-side FIFO are operated at a frequency that is typically 10% greater than the equivalent line-side FIFO rate at 64 bits wide in order to carry out a “packing” operation (20% for 128b mode). Packing is required for all cases where the end-of-packet byte does not result in a fully valid 64-bit FIFO entry. In

this case, there is SPI4 line bandwidth available that can only be taken advantage of through over-speed at the user-side FIFO and in the aligner. The amount of instantaneous over-speed required is reduced by averaging the demand for over-speed over time through the smaller line-side FIFO.

User data is written into TxFIFO1 based on the availability of user data to transmit, availability of near-end FIFO space, and availability of FIFO space at the far end of the SPI4 link. The user-side transmit FIFO is either 4K or 8K bytes depending upon mode and is organized as 512 locations x 80 (64b) or 144 (128b) bits. User logic should monitor the Transmit FIFO Almost Full ('TxF1AF') signal as it writes data and control information into the FIFO. When 'TxF1AF' is asserted, writing should be suspended until the Transmit FIFO Almost Empty ('TxF1AE) signal is asserted. Thresholds associated with the almost empty and full flags are real-time controllable via top-level signal array connections to the core and can be set to optimize a minimum or maximum data transfer amount into the FIFO. These thresholds also allow the user to configure the rather large user-side FIFO for “shallow” mode operation that may be needed in some applications to ensure that there is not a large amount of data committed to the line when flow controlled.

The Aligner formats data read from the user-side FIFO into SPI4 control word encapsulated data segments and/or whole packets and writes the data into the line-side FIFO. The Aligner monitors the user-side Transmit FIFO Empty ('TxF1E') signal, reading data and control information when TxF1E is deactivated, and generates the appropriate SPI4 control word containing a DIP-4 parity calculation and control directives (sop, eop, cnt, abt, port ID, etc.) for each packet segment. Data is continually read and transmitted from the user-side FIFO until the 'TxF1E' asserts. If the FIFO empties in the middle of a packet, the segment is terminated with an Idle control word and the SPI4 line goes idle. Transmission resumes when the user-side FIFO is again loaded with data, which can be associated with the same or different channel. Once the user-side begins loading a segment of data into the FIFO, the Aligner block will not be able to over-run the segment as long as the user writes the segment into the FIFO on consecutive clock cycles. This FIFO is operated in a synchronous mode given user loading and Aligner functions both require the over-speed System Data Clock ('SDCK'). This synchronous operation minimizes the response time for flag generation through the FIFO. Before the Aligner block is allowed to transmit data toward the SPI4 line, the associated input direction status channel must be properly framed ('TxSTAERR' inactive). The Aligner will continually send training control and data sequences until this condition has been met.

The timing domains between the user-side System Data Clock ('SDCK') and the SPI4 line-side transmit clock ('TxS4LS_CK') are crossed at the line-side FIFO - TxFIFO2. The line-side FIFO is 4352 or 8704 bytes organized as 64 locations x 68 or 136 bits. A detailed description of SPI4 core clocking and synchronization is given in a subsequent section of this document. The line-side FIFO can be optionally protected with four bits of parity generation and checking (see signal description for 'TxF2PERR' in “Signal Descriptions” on page 24) in order to ensure data integrity.

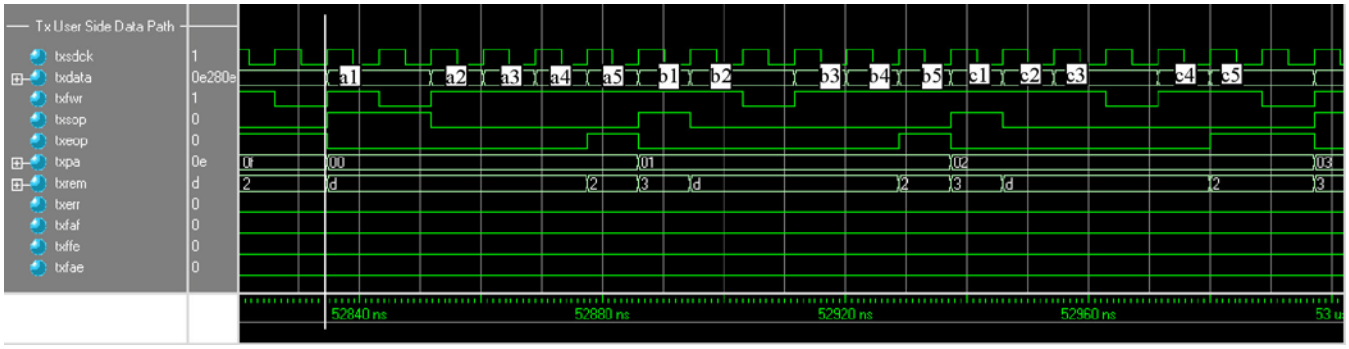
Transmit Data Timing Diagram Example

Figure 2-3 shows the transmission of three 67-byte full packets for channels 0, 1, and 2 over the S4TX transmit user FIFO interface for 128b mode. The interface operates in a synchronous fashion based on the user-supplied 'txsdck' clock input signal. This clock has over-speed relative to the equivalent SPI4 line-side as mentioned above, which is the case for this analysis. The first packet (channel 0) starts at time 52834ns in response to available data to send and an inactive Transmit FIFO Almost Full signal ('txfaf') from the IP core and is marked by the assertion of signals 'txfwr', 'txsop', 'txpa', and txdata[127:0]' from user. In the sixth clock cycle, 'txeop' and 'txrem' are asserted indicating the end of the packet and the amount of remaining bytes (0x2 = 3 bytes) in the last slice (128 bits) of data. It is in this clock cycle that signal 'txabt' (not shown) would be active if the user wants to abort the transfer. An active 'txabt' signal is acted upon by the core only when both 'dval' and 'txeop' signals are also active, otherwise it is ignored.

Although not reflected in this example, the effects of the over-speed will be noticed by the assertion of the 'txfaf' signal, mentioned above, at a regular interval assuming there is constant data to send. When asserted, the user-side must suspend writing to the user FIFO for some period of time. The simplest method is to fill the FIFO until 'txfaf' is asserted and then suspend until 'txfae' (almost empty) is asserted. This arrangement affords the smoothest and most efficient use of the SPI4 line in terms of its maximum bandwidth potential. Allowing the FIFO to run completely dry causes the pipelines, and partially the line-side FIFO, to fill with Idle control words increasing the latency

of the next burst and decreasing the overall bandwidth utilization by reducing opportunities for packing the SPI4 line (no idle control word insertion - back-to-back, single control word separated packets and packet segments).

Figure 2-3. S4TX User Data Interface Example



SPI4 Transmit I/O - S4TXIO (TXGB)

The S4TXIO block provides 2-1 gearing/multiplexing and SDR-to-DDR conversion for the transmit data direction (LVDS buffer insertion is done outside the core). In the LatticeSC device, this block can support 4-1 multiplexing conversion for 128b mode in the SPI4 line output direction. All of these functions are performed at the Programmable Interconnect Cell (PIC) level and therefore do not require any PLC logic resources. In 64b mode, data (64 bits) and control (4 bits) are received from the S4TXDP block through TxFIFO2. Data are received at the lower by 2 clock speed rate and then multiplexed up to a 2x rate, reflecting a 32-bit data bus and 2-bit control bus. A second stage of multiplexing occurs in the PIC where the data and control signals are moved from single-edged format to double-data rate format at the same frequency before being sent off-chip through LVDS output buffers. Data and clock leave the transmitting device in phase. The receiving end is responsible for shifting the clock with respect to the data before using the clock to sample data. All of these signals operate over differential pairs at LVDS levels.

The low and high-speed line clocks are provided by the user and can be generated from an internal PLL or received via the primary I/O (see “Clocking and Synchronization” on page 51 for further information).

Minimum Burst Size - Burst Mode

Burst Mode is essentially always enabled given that the SPI4 protocol is a natural burst interface that requires a minimum burst size of 16 bytes without an EOP as defined in the OIF specification. The burst size is controlled by IP core port array 'TxBLEN[5:0]', where a value of one results in standard SPI4.2 behavior (16 byte minimum burst), a value of 2 results in 32 bytes, and so on up to a value of 63*16bytes=1008 bytes.

When operating with burst sizes greater than one, the TxFIFO2 Almost Empty Burst Threshold ('TXF1AEBTHRSH[]') must be set to a value of at least 2 greater than the burst size in 128b mode and 4 greater in 64b mode. For example, a fixed burst size of 64 bytes would require an Almost Empty Burst Threshold of at least 6. The transmitter waits until the associated almost empty burst flag is de-asserted indicating that there is at least enough data in the FIFO to send the programmed burst size or there is at least one EOP in the FIFO before beginning a SPI4 burst. Once a burst has begun, the transmitter will not allow it to be interrupted/segmented until it is completely transmitted. Parameters 'TxBLEN[]' and 'TXF1AEBTHRSH[]' are set during the user GUI capture phase.

Training Pattern Generation

Training Control and Data Pattern generation is enabled by setting 'TxMAXT[15:0]' to a value other than 0x00. The Training Pattern Generator will maintain a schedule based on the value of 'TxMAXT[]' and request the transmission of the training pattern, 'TxREP[7:0]' times, at the appropriate intervals and boundaries as specified in the SPI4 standard. The system data clock 'TxSDCK' is used to time the interval on a one-for-one count based on the value of 'TxMAXT[]'. The default value is set through an RTL parameter obtained during GUI capture and can also be programmed by an IP core port array.

Transmit FIFO2 Threshold Optimizations

There are four user-programmable thresholds associated with the transmit line-side FIFO of which three can be used to optimize the behavior of the transmitter for optimal SPI4 line packing, minimal L(max), or a compromise of the two. All thresholds are captured in the GUI phase but can be reprogrammed in real time using port-level array connections to the core. The default values found during GUI capture set up the compromise selection.

The transmit FIFO almost empty threshold ('TXF2AETHRSH') and the transmit FIFO almost empty idle threshold ('TXF2AEITHRSH') controls the point at which the Aligner must respond with data or control (i.e. idles) before the FIFO under-runs; an error condition that should never be allowed to occur through proper settings of the thresholds. There are two cases to consider when crossing either of these thresholds high-to-low:

- When there is no data flowing in the system. In this case, the Aligner responds by simply writing Idles into the FIFO when crossed to keep the SPI4 line active. During this condition, the 'TXF2AETHRSH' threshold/flag is ignored.
- When data is flowing in the system. This is the more critical case in terms of recovery in the FIFO because the empty signal arrives during a time when there has been opportunities for packing where multiple write side clock cycles are required to perform a single packed write. When data is flowing, the only way to cross this threshold is if extensive packing has occurred eroding write side bandwidth to below line-side levels. Once the almost empty signal is received, further packing is inhibited, but pipelines leading towards the FIFO have already been loaded with partially valid data slices that will be packed and therefore written into the FIFO at reduced bandwidth increasing the chances for an under-run condition. Because of this, the 'TXF2AETHRSH' will typically need to be set a little higher than the 'TXF2AEITHRSH'. Having a separate threshold just for the almost empty during idles condition allows the transmit line-side FIFO to be run very shallow when no data is flowing, which results in low latency when data flow again resumes.

The absolute minimum value of 'TXF2AETHRSH' before under-run occurs is dependent upon the mode (64/128b), number of channels, amount of over-speed, packet size, and the value of the Transmit Almost Full Threshold ('TXF2AETHRSH'). Applications with low channel count and reasonable over-speed can typically run with values as low as 8. Worst-case conditions may require a value as high as 14. One factor contributing to the magnitude of the threshold is the large round-trip delay between almost empty flag activation and data being written into the FIFO. Note that the almost empty flag is generated from the read clock domain and must be passed back to the write clock domain. If latency is critical, the user should simulate their design using worst-case channel count, clock frequency etc. to establish the lowest possible value. Error signal 'TxF2FERR' can be used in both simulation and in-circuit to determine if the threshold has been set too low. The absolute minimum value of 'TXF2AETHRSH' before under-run is around 5.

The transmit FIFO almost full threshold ('TXF2AFTHRSH') controls the point at which the Aligner must stop writing data into TxFIFO2 before an over-flow condition occurs. The almost full flag asserts on 'TXF2AFTHRSH' + 'TXF2AFOTHRSH' (offset threshold) crossing low-to-high, and de-asserts on crossing of only 'TXF2AFTHRSH' high-to-low. The user can alter the almost full threshold in order to set the desired depth of the line in terms of data storage. The offset threshold is not adjusted by the user but is a simple fixed addition (+2) to the almost full threshold done in the GUI phase. The higher the value of almost full threshold the greater the degree of packing that will occur. This is because the FIFO will have stored up a greater amount of data to sustain the SPI4 line during a period when FIFO write side bandwidth is lower than the read side. The higher the fill-level in the FIFO the longer the period of potential packing will be. Valid tested ranges are between 20 and 54 and depend upon the degree of packing desired.

For this condition to occur (almost full), data must be flowing in the system since the Aligner maintains a fill level near the Almost Empty level when there is no data to send. When there is no data flowing in the system, the aligner uses only the almost empty idle flag to maintain the FIFO as shallow as possible without under-run so that the next piece of data has the lowest possible latency.

SPI4 Transmit Status - S4TXSP

The SPI4 Transmit Status Protocol (S4TXSP) block provides the entire SPI4 status function for the transmit direction. Note that though this is a transmit data function, status information is received and is therefore an input to the core and to the user logic. It provides a stand-alone status reporting function between the SPI4 line and the user. The only connections between the S4TXSP block and the data path is a Transmit Status Alignment Error ('TxSTAERR') signal which forces the data path to send constant training control and data words until the S4TXSP is properly framed to the incoming status and some composite status signals for internal status mode.

The S4TXSP block frames on the incoming status channel, extracts per channel status information, and then forwards the information to user-side logic. This status/flow control information provides the user with an indication of how “full”/“empty” the FIFOs are at the far-end of the SPI4 link. User-side logic will use this information to determine the appropriate amount of data (SPI4 MaxBurst1, MaxBurst2, or no data) that can be written into transmit user-side FIFO on a per-channel basis without causing an overflow at the far end.

User input 'TXSTEN' provides enable/disable control over operation of the transmit Status interface and may be used to ensure that the S4TXSP block does not incorrectly interpret status information from the far end during initialization and/or re-initialization (i.e. adding/subtracting a channel). When 'TXSTEN' is inactive, the transmit Status framer section is forced into the Out-of-Alignment state. This action inhibits user status updates and no data is transmitted on the data interface. The user is able to program the Calendar RAM during this period. When 'TXSTEN' is returned to the active state, the S4TXSP framer must go through the re-frame procedure in order to return to alignment. Additional details are given in the Status and Calendar RAM Layout section of this document.

Two different synthesis-selectable configurations for reporting user status are supported: “RAM” mode and “Transparent” mode. In either mode, status information and all user side status related signals are provided to the user synchronously via the user supplied 'txstck' clock signal, which can be asynchronous relative to the SPI4 transmit status clock ('x_tstatus_ck'). Because logic costs are very minimal and some of the transparent mode functionality may be used in RAM mode, the Transparent Mode Status interface in the transmit direction is always available (I/O and logic is not optioned out) either by itself, or in addition to the RAM mode interface.

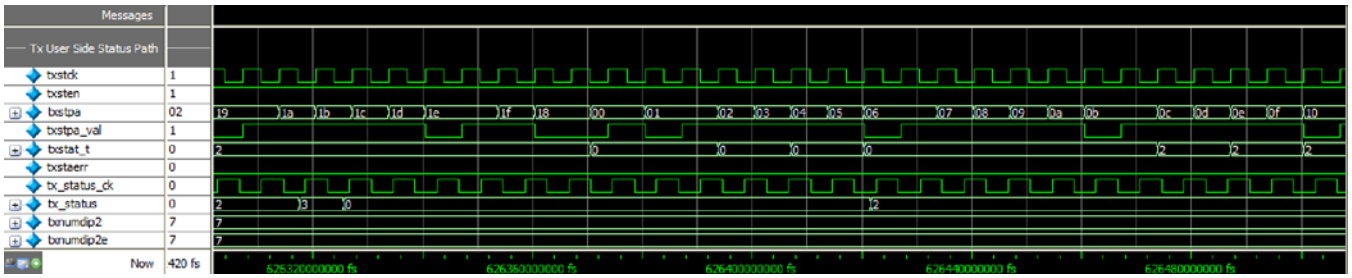
Transparent Mode

Transparent mode is provided for applications in which a user-specific Status access style is preferred. With this mode, the internal RAM storage and associated logic are eliminated and Status is presented to the user in a transparent manner as it is received from the external SPI4 status lines. The core provides the user with an 8-bit channel address, the 2-bit status indication, and a valid signal qualified by proper alignment. The Calendar RAM still exists inside the core for this mode and provides the channel address identification.

Figure 2-4 shows an example of the S4TX Status Interface operating in Transparent Mode with 32 channels single entry per channel. In this mode, status is not stored in RAM inside the core but rather is passed from the SPI4 input status channel directly to the user interface transparently via a 2b user side status bus ('txstat_t'). The core does retain the Calendar RAM and status channel framing and DIP2 parity error checking function and is therefore the controller in terms of determining which channel and at what time it's status is delivered to the user interface. The core provides the user with an 8b address ('txstpa') informing which channels status is currently available via the 'txstat_t[]' bus and a valid signal ('txstpa_val') to qualify the status. These signal are in phase meaning that in a given clock cycle, the status and address correspond to one another.

In the example below, the input status channel from the SPI4 interface ('tx_status[]') changes from a value of 0 to 2 coincident with channel 0xc. The affected input channel can be identified by counting status clock cycle slots from the framing marker of “3” using 'tx_status_ck'. Several cycles later, the status appears on the internal user side bus ('txstat_t') along with a corresponding port address ('txstpa' = 0xc). In this example, the SPI4 line status clock ('tx_status_ck') and the user side status clock signal ('txstck') are asynchronous to one another. 'txstck' is driven with the system data clock ('SDCK'). Because of the user side over-speed, there are a number of 'txstck' clock cycles that are invalidated via signal 'txstpa_val'. When 'txstck' and 'tx_status_ck' are operated using the same clock, only two cycles per status frame (DIP2 and Framing) will be invalidated.

Figure 2-4. S4TXSP - Transparent Status Mode



RAM Mode

In RAM mode, an internal Status RAM is used to store the status received from the far end of the SPI4 link. Status information is written into the memory using the user supplied 'txstck' clock signal. The specific location written each clock cycle is specified by the contents of the Calendar RAM. In this mode, user logic reads the Status RAM using a clock and address that it supplies (clock may be asynchronous to external status line clock).

Referring to Figure 2-5, the S4TX Status RAM interface is a user side “read only” interface that operates in a synchronous fashion based on the user supplied 'txstck' clock input signal. This clock signal is only used for user read access of the RAM through the user interface and does not have any phase relationship requirement with respect to the status channel input clock 'tx_status_ck'. It must however, be equal to the frequency of 'tx_status_ck' or greater but not less than 'tx_status_ck'.

The top half of the diagram shows continual reads of 1/8 of the Status RAM based on the 'txstadd[4:0]' input address bus. Only a single clock cycle is required for read operations but as shown, four cycles of the same address are read. Eight channels worth of status are available per RAM access. The user can sample status for a new address on the following clock edge. Address location 0 corresponds to channels 0 - 7, address location 1 to channels 7 - 15 and so on.

In the example below, prior to time 455,160ns the external input status channel ('tx_status[1:0]') reflects a constant Starved (0x0) indication for all channels and hence 'txstat_r' reflected a constant value of 0x0000. After this time, tx_status[1:0] transitions to the Satisfied state (0x2). Counting 'tx_status_ck' clock cycles back from the DIP2 and Framing bits (0x3) on this bus, the transition is shown to occur on channel 26. Looking now at the user side 'txstat_r[]' bus, channel 26 is the first channel to reflect the new Satisfied status. Note that 'txstadd[]' is equal to 3 (8 channels/address) meaning that the channels reported for this address are channels 24-31. Some of the delay that is incurred between the external and internal user side interfaces is due synchronization that must take place between 'tx_status_ck' and 'txstck' clock domains.

Signals 'txstpa' and 'txstpa_val' are provided in RAM mode for optional use to indicate where the transmit status processor is it any given time.

Calendar RAM

A Calendar RAM of up to 512 locations (user accessible read/write) is used to identify the port/channel address of the incoming status information as it is received and to notify the user that updated status information has been received for the given port and needs to be read. The reading of locations in the Calendar

RAM is linear and synchronized to the framing contained within the status channel. For systems where the channels are not symmetrical in terms of bandwidth, the same channel can be programmed into multiple locations in the Calendar RAM resulting in multiple and more frequent status updates per status frame for a given channel. The corresponding Calendar RAM used to source status information at the other end of the link must be programmed to the same length and channel order.

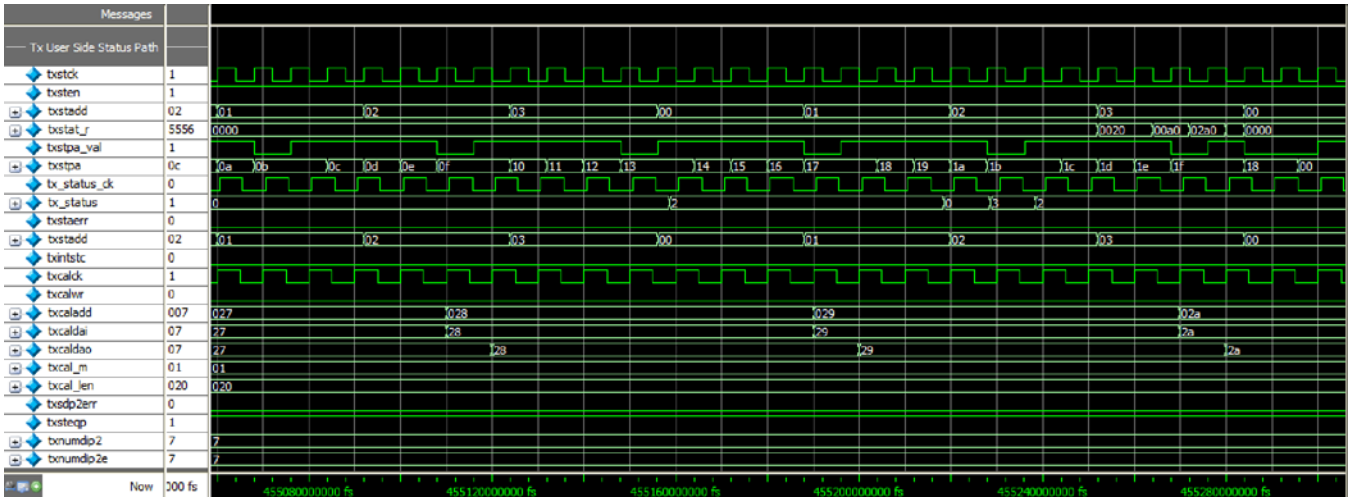
This design of the Calendar RAM interface is based on an EBR True Dual Port RAM providing the underlying memory function in which the user utilizes one port and the internals of the core utilize the other. The S4TX Calendar RAM interface is a user side “read/write” interface that operates in a synchronous fashion based on a rising edge

active user supplied 'txcalck' clock input signal. This clock signal is used for write access as well as for read access of the RAM through the user interface in which the address (and data for write cycles) is sampled before being applied to the RAM core (data is not clocked out of the RAM however). Input signal 'txcalwr' (act high) controls which operation is to be performed on a per cycle basis (reading does not take place when 'txcalwr' is active). Reading or writing is allowed to take place on consecutive clock edges.

The example shown in Figure 2-5 shows small piece of what could be an initialization of the 512 location Calendar RAM if signal 'txcalwr' were to be asserted. For this example, the RAM is used to support only 32 channels ('txcal_len' = 0x20) where each channel has the same amount (1) of Calendar entries and therefore status bandwidth. Addresses beyond the calendar length are simply written similarly as though all 256 channels were being used. Location 0 is programmed to a value of 0 (ch-0), location 1 is programmed to a value of 1 (ch-1), and so on ending with location 0x1f programmed to a value of 0x1f. A single cycle can be used to perform the write cycle and as shown the locations above the calendar length are simply written in a like fashion even though they are not used (i.e. 0x028 with 0x28). Once the calendar RAM is initialized, there is no need to continue writing.

Note that as the address input address changes so does the corresponding output data ('txcaldao') based on 'txcalck' and corresponding to the value that was written. Reading can also be done in a single cycle even though in the example, the address is held for several cycles.

Figure 2-5. S4TXSP - RAM Mode Status and Calendar RAM Interface



Internal Status Control

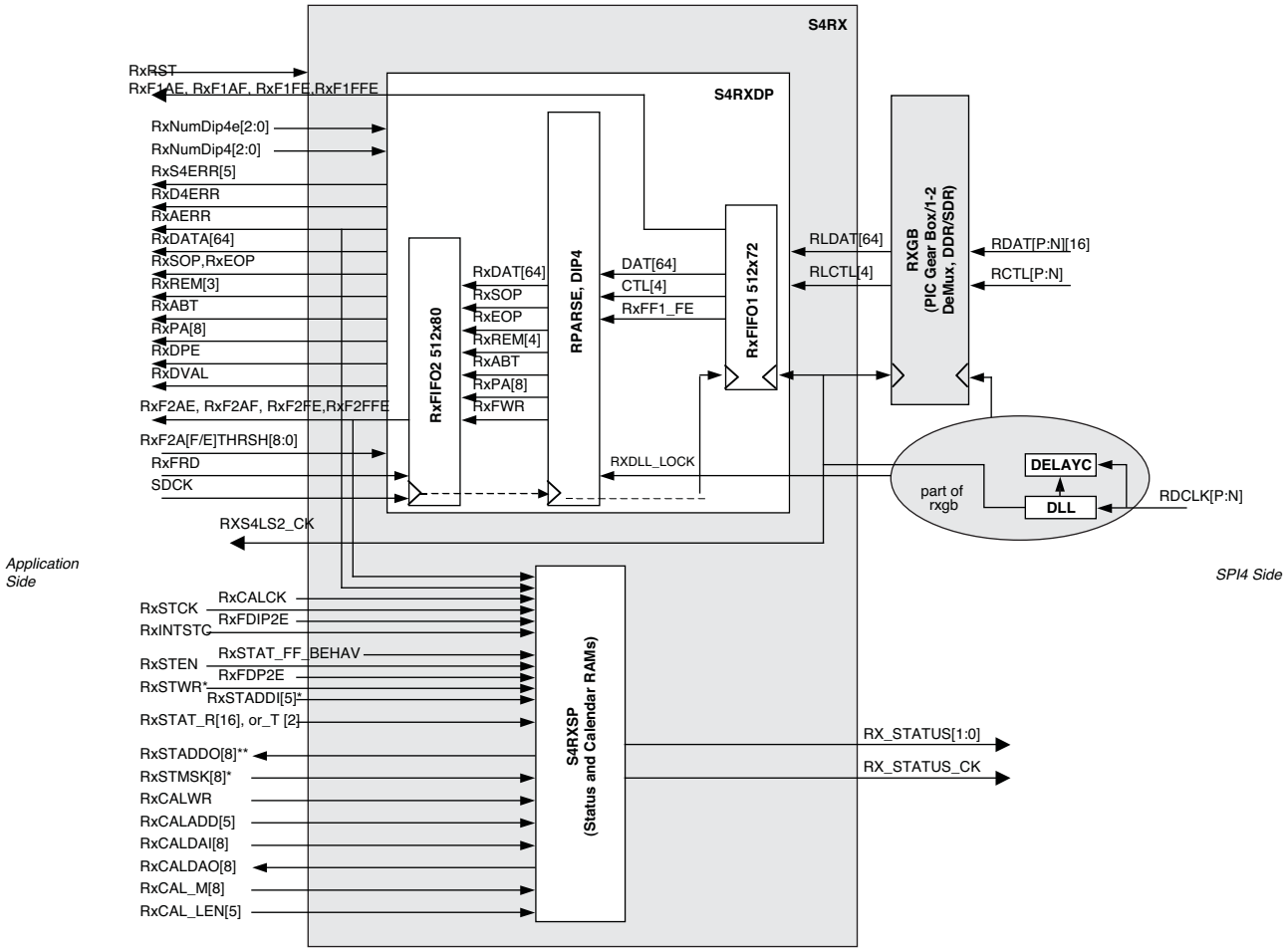
This design provides an option ('TXINTSTC'=1) to eliminate the need for user interrogation of received status. When this option is selected, transmission of data towards the SPI4 line is controlled internally through analysis of the hungry, starved, and satisfied states received on the input status path. If the satisfied state is received for any channel, data transmission is stopped until one full iteration of the calendar sequence has been observed again where all channels are reporting the hungry or starved states. One iteration is defined to be the length of the calendar sequence only, not the full calendar cycle that may include multiple repetitions of the calendar through CAL_M. In this mode, the user can simply load the user-side transmit FIFO taking into consideration only the state of the FIFO Almost Full flag and stopping when it is active. If the core is flow-controlled and data transmission stops, the fill level of this FIFO would naturally build causing this flag to assert assuming user logic continues to load data into the FIFO.

This mode can be used for single channel/single pipe applications or other applications where blocking is not a consideration since any one channel can stop transmission of the other channels. When this option is selected, the user should also select transparent status mode to eliminate the unused Status RAMs, resulting in the smallest design possible.

SPI4 Receiver - S4RX

The receive path, shown in Figure 2-6, is the path of data flow from the SPI4 line towards the internal user application function and the direction of status flow from the application function towards the SPI4 line. Figure 2-6 indicates through shading some of the hierarchical boundaries and identifies three distinct sub-sections of the S4RX block as described in the following sections. Figure 2-6 shows the static mode that is implemented in the LatticeECP3 devices. Similarly, Figure 2-7 shows the dynamic mode that is implemented in LatticeSC/M devices in 128-bit mode.

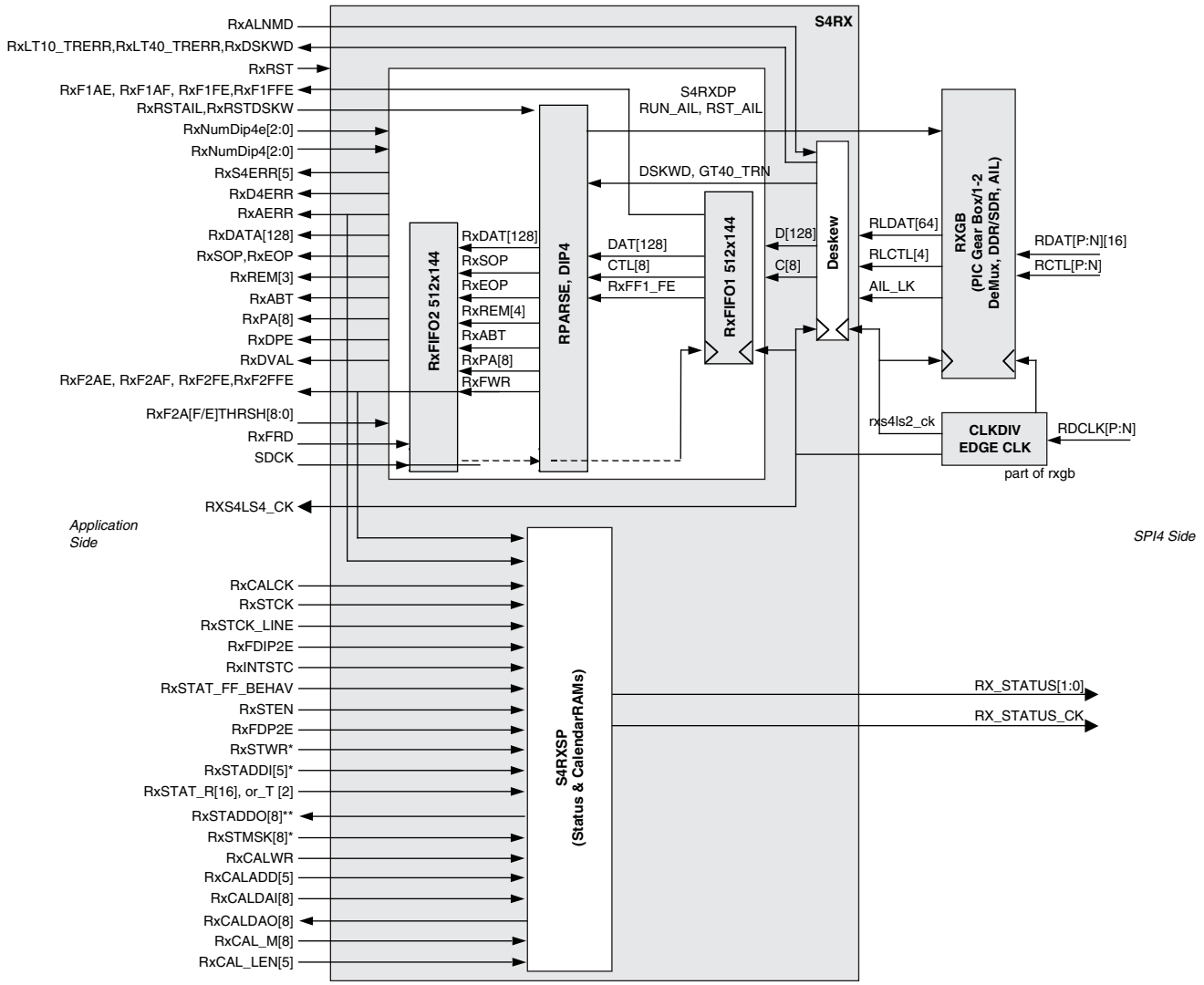
Figure 2-6. S4RX - SPI4 Receive Path



* In Transparent Status mode,RxSTWR, RXSTADDI[], andRxSTMSK[] do not have I/O appearance in the wrapper.

** In RAM Status mode,RxSTADDO[] does not have an I/O appearance in the wrapper.

Figure 2-7. S4RX - SPI4 Dynamic Mode Receive Path



* In Transparent Status mode,RxSTWR, RXSTADDI[], andRxSTMSK[] do not have I/O appearance in the wrapper.
** In RAM Status mode,RxSTADDO[] does not have an I/O appearance in the wrapper.

SPI4 Receive Data Protocol - S4RXDP

The SPI4 Receive Data Protocol (S4RXDP) block performs the reverse operations of the transmitter using SPI4 control words received from the SPI4 line to delineate packet starts, stops, and port switches. Similar to the transmit direction, sop, eop, abt, and port id fields are passed to the user via a user-side FIFO (RxFIFO2). Over speed as well as another line-side FIFO (RxFIFO1) are also used in order to successfully reverse the effects of potential “packing” operations by the transmitter. Unpacking is required for cases where there are multiple channel operations (end-of-packet) per 128-bit data slice received from the SPI4 line. For these cases, multiple writes to the user-side FIFO are required per line-side clock cycle in order to maintain packet separation through a period of time where the line-side FIFO reading is temporarily inhibited. The line-side FIFO is operated in an asynchronous mode bridging the user and line-side clock domains while the user-side FIFO is operated synchronously within the user clock domain.

The Parser receives a de-multiplexed SPI4 bit-stream (4 or 8 SPI4 words wide depending on mode) through “read” operations of the line-side FIFO and continually parses the incoming data stream one word at a time looking for proper SPI4 line protocol, format, and DIP4 parity. As control and data is received from the line, error status (good/bad) is reported to the user interface through core I/O and used internal to the core in determining whether

to declare the SPI4 line in-alignment or out of alignment based on user programmable error thresholds (see also “Start-Up Procedures” on page 23). Once the SPI4 line is in alignment, and a valid data segment is detected, the data and relevant control (sop, eop, abt, port address, etc.) are aligned, left justified, and written into the user-side FIFO as long as it is not full (if full, the data is discarded). Assuming a functional Status channel, the user-side FIFO will never over-flow. There is an output error signal that indicates the full condition should it occur due to a faulty or unequipped status channel.

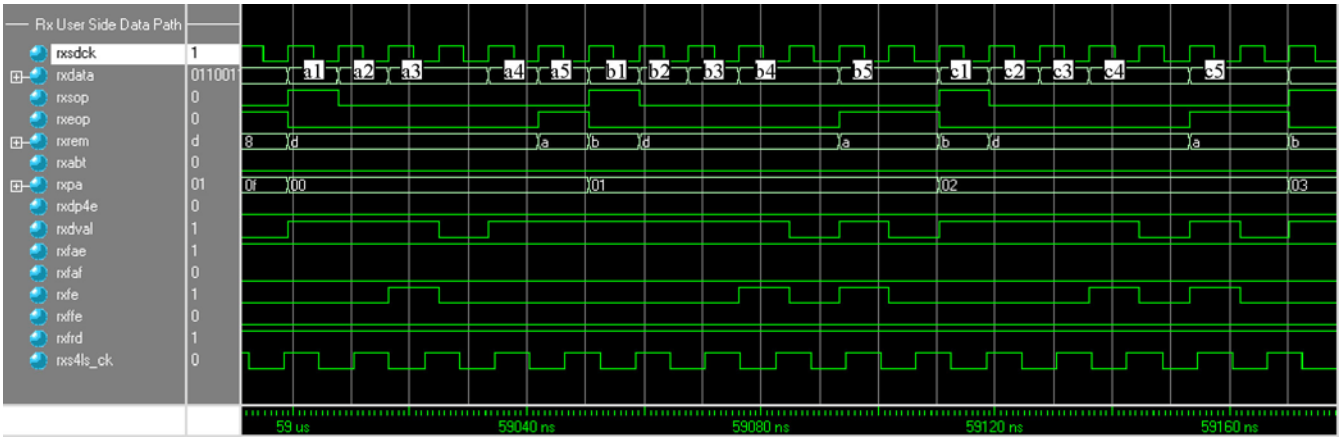
User logic monitors primarily the user-side receive FIFO Empty flag ('RxF2E') as it reads data and control information. When the empty flag is asserted, reading should be suspended until it is again deasserted. Since both the write and read side clocks are the same, once the user-side begins reading, it will not be able to overrun the current burst segment. FIFO Almost Full ('RxF2AF'), FIFO Almost Empty ('RxF2AE'), and FIFO Full ('RxF2FE' - error condition) output signals are also provided to the user, but may or may not be used. The FIFO almost Full and Full signals are used internally affecting the transmitted status under certain abnormal circumstances (see “SPI4 Receive Status Protocol - S4RXSP” on page 18 for further information). Thresholds for almost empty and full flags are set through the GUI capture phase but can also be set in real time via IP core port connections.

Receive Data Timing Diagram Example

Figure 2-8 shows the reception of three 75-byte full packets for channels 0, 1, and 2 (labeled a, b, and c) through the receive user FIFO interface. The interface operates in a synchronous fashion based on the user-supplied 'sdck' clock input signal. This clock, as mentioned earlier, should have some over-speed relative to the equivalent SPI4 line-side, which is the case for this analysis. The first packet (channel 0) starts at time 58900ns in response to an active read input signal ('rxfrd') from the user and is marked by the assertion of signals 'rxsop', 'rxdval', 'rxpa', and 'rxdata[127:0]' by the IP core. The effects of the over-speed are apparent very early in the transfer since 'rxdval' is used to invalidate one of the six clock cycles associated with this transfer (note also the assertion of 'rxfe' - FIFO empty). In the sixth clock cycle, 'rxeop' and 'rxrem' are asserted, indicating the end of the packet and the amount of remaining bytes (0xa = 11 bytes) in the last slice (128 bits) of data. It is in this clock cycle that signal 'rxdp4e' would be active if the packet had been received with a DIP4 error or the assertion of 'rxabt' if the packet was sent with an abort by the far end of the link. These two signals are valid only when both the 'dval' and 'rxeop' signals are also active. The second and third packet transfers are similar to the first except different cycles are invalidated by 'rxdval'.

Note that a read of an empty FIFO is tolerated but the user must disqualify data based on 'rxdval' as mentioned above. In this example, 'rxfrd' is held active constantly and data is taken only when validated. In terms of latency delay through the core from the SPI4 line to the receive user interface, it takes 17 'sdck' clock cycles from the arrival of the first SPI4 data word to de-assertion of 'txfe' (FIFO empty).

Figure 2-8. S4RX User Data Interface



Error Handling

The SPI4 receiver checks for a number of error conditions and raises individual error flags ('RS4ERR[4:0]') for each, as described in this section. When considering the type and level of support for error checking, one consid-

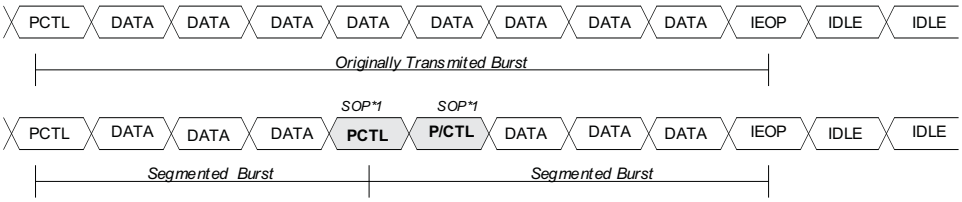
ers two distinctly different needs 1) the system/device debug effort where the user incorrectly loads two SOPs in a row into the output FIFO at the far-end for example and is quickly lead to the problem by adequate error reporting and 2) the final system where good error reporting can be used to support features like “packet drop” and improved recovery speed.

Control Word Preceded by a Payload Control Word (RS4ERR[0])
Any control (rctl=1) word that is preceded by a payload control (rctl=1 and b15=1) word is treated as a SPI4 line protocol violation since only “data” is allowed to immediately follow a payload control word. When this condition is detected, the burst associated with the payload control word is terminated in the normal fashion (i.e. DIP4 parity checking and passed to the output FIFO). The second and remaining control words, if present, are treated as data starting the continuation burst of the payload control word (see Figure 2-9) and also written to the output FIFO. This causes erroneous data in the second segment (the extra control word/s) and it will therefore fail DIP4 error checking when concluded. In addition, an error flag (not FIFO aligned) is activated to notify the user of the condition.

One assumption for system level analysis is that the hardware at the transmitting end has been debugged and operating correctly and will not send multiple control words in a row without the presence of a fault or a noisy SPI4 line. It is therefore believed that in most cases, for the examples shown in Figure 2-9 and Figure 2-10, that it is the control signal (rctl) that is sampled incorrectly and what is actually present on the SPI4 bus is a data word and not a control word. When considering the effects a situation like this, it is important that many packets or packet segments are not written to the output FIFO that would complicate and lengthen recovery. It would be better to either add the consecutive control words to a single segment or to drop them altogether rather than to fill and possibly over-flow the receive FIFO with many zero length entries. There are several cases to consider:

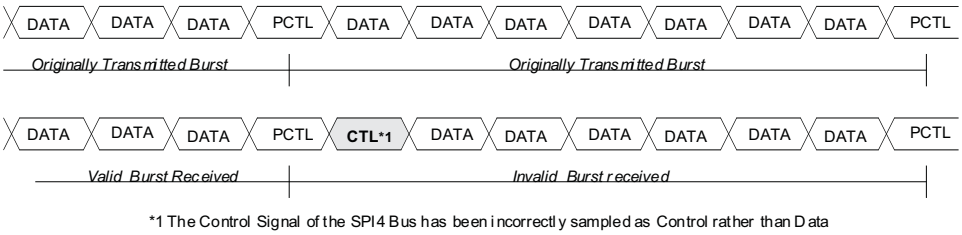
- Payload Control with one or multiple illegitimate Control Words following in the middle of a valid data segment. When this condition occurs, the first segment will fail DIP4 parity, and N number of continuing control words will be tacked on to the next segment and it too will fail parity.

Figure 2-9. Multiple Illegitimate Control Words In a Data Segment



- A valid payload control word is sampled correctly but due to signal integrity/noise problems, rctl is again sampled, this time incorrectly, as control during a time when data is actually on the bus. For this case, the first segment will pass parity and be passed on to the user. The second segment will fail parity.

Figure 2-10. Multiple Illegitimate Control Words At Burst Boundary



Cases similar to the above but for which no Payload Control words are received (multiple control words where b15=0), are handled through other error checking mechanisms as follows:

- Reception of an Idle control word sequence (PURE, w/EOP, or ABT) will terminate any in-progress segment. If this condition occurs in the middle of a valid segment, the first part will fail with DIP4 error and the second part

will appear as a “data preceded by idle” error condition and all data up to the next valid control word will be dropped inside the core (see "Reserved Control Word Detected (RS4ERR[2])" below).

- Reception of a Reserved Word sequence will also terminate any in progress segment. If this condition occurs in the middle of a valid segment, the first part will fail with DIP4 error and the second part will appear as a “data preceded by idle” error condition.

EOP Preceded by Idle (RS4ERR[1])
All EOP bursts on the SPI4 line must contain at least one byte of data, which means that data must precede an EOP control. This error will be activated when an EOP control word (idle or payload control) is observed preceded by an Idle (B15=0). Nothing is written to the output FIFO when this occurs. This error can be used to indicate possible missing SOPs.

Data Preceded by Idle (RS4ERR[3])
All data bursts must be preceded by a valid payload control word (b15=1) specifying, among other things, the channel address of the data to come. Data that is observed preceded by an idle (PURE, w/EOP, ABT b15=0) is considered a SPI4 protocol violation. In this case, the data is dropped and the user notified of the error. This could be an indication of a missing SOP. The user will need to either recover the entire link given that there is no valid address identification with this error or ignore it and allow a higher-level application handle it.

Reserved Control Word Detected (RS4ERR[2])
All reserved control words are considered SPI4 protocol violations. When observed, any in-progress segment will be terminated.

Invalid Burst (RS4ERR[4])
All non-EOP bursts on the SPI4 line are expected to conform to the credit (16 byte) boundary (a multiple of 16 bytes) rule. Any burst detected with out an EOP that is not a multiple of 16 bytes is flagged as an error. This could be an indication of a missing EOP. This error signal is aligned and “or”ed into the DIP4 error lane allowing for the support of a packet-drop capability.

SPI4 Receive Status Protocol - S4RXSP

The SPI4 Receive Status (S4RXSP) function receives status information (starved, hungry, and satisfied) from the user in either RAM or Transparent mode selectable during the GUI capture phase and implemented via synthesis parameter 'RxSTAT_MD' (modes discussed later in this section). Flow control information is sent to the far end through the status channel providing an indication of the full/empty status of the receive user-side FIFO and any user-supplied per-channels FIFOs at the near end. User logic at the far end uses this information to determine the appropriate amount of data (MaxBurst1, MaxBurst2, or no data) that can be sent on a per-channel basis without causing near-end buffer overflow.

The status path operates independently from the data path and has only minimal connection between the two for presenting receive alignment status (in/out of frame) and both receive user and line-side FIFO fill level status. All of these connections affect the status sent to the far end under abnormal circumstances. When the receive data path is out-of-frame, for example, a constant “11” pattern is sent to the far end on the status channel over-riding user status. The far end is expected to respond with Training Control and Data patterns on the data path to aid in resolving the alignment issue. Additionally, receive user-side FIFO fill levels can over-ride user status and force a Satisfied state for all channels while in the aligned state. The user can optionally select ('RxST_SEL_FF_FAF'= 0 or 1) whether the Almost Full, or Full signal is used to cause the override condition. Regardless of the selection, the condition persists until the Almost Empty signal is asserted upon which status reverts back to being sourced from the Status RAM or transparent Status interface. The same behavior exists for the line-side FIFO except that there is no option to use the Full signal. Almost Full is used to trigger the override and almost empty is used to clear it. These features protects against cases where user logic is late in servicing the receive user FIFO for whatever reason and for cases where there is not enough over-speed in the system clock domain to deal with a potential burst of small segment EOPs from the line-side FIFO.

User input 'RXSTEN' provides enable/disable control over operation of the receive Status interface and may be used to ensure that incorrect status information is not transmitted on the SPI4 link during initialization and/or re-initialization (i.e. adding/subtracting a channel). When 'RXSTEN' is inactive, the output Rx Status interface is forced to send constant framing regardless of the state of the input data path. Although the Calendar RAM is always 'write' accessible by the user; this is the best time to initialize the Calendar.

The following sections describe the two status modes, RAM and Transparent, which are provided for transferring status information into the core. For both modes, a user-supplied clock ('rxstck') is used as the user side interface clock to synchronously transfer status into the core. A second user-supplied clock ('rxstck_line'), which may be asynchronous to 'rxstck', is used to drive status on the SPI4 interface. Only one of the following modes is provided at any given time and is determined during synthesis based on parameter 'RXSTAT_MD'.

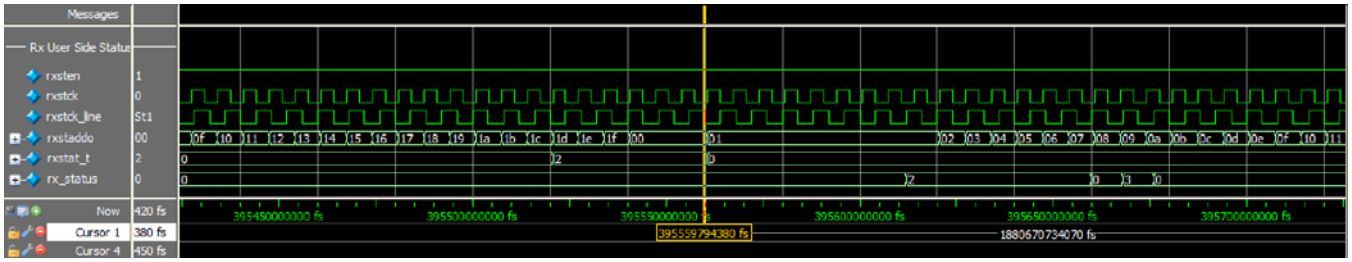
Transparent Mode
In transparent mode, the user supplies status in a 2-bit per channel bus format. This data passes through this module and to the external SPI4 status interface with out being stored in RAM. The Calendar

RAM discussed below provides the user logic with an 8-bit channel address for which status is requested on a continual basis according to the calendar sequence. At the appropriate time, status transmission is suspended and correct framing and dip2 parity are inserted. This mode does not contain RAM-based status storage and therefore utilizes less device resources.

Figure 2-11 shows a timing diagram example of the Transparent Status Mode operating with 32 channels and a single calendar entry per channel. In this mode, status is not stored in RAM but rather is passed from the user interface to the external status SPI4 channel interface transparently via a 2b user side status bus ('rxstat_t'). The core does retain the Calendar RAM and is therefore the controller in terms of determining which channel and at what time its status is to be transmitted. The core provides the user with an 8b address ('rxstaddo') informing which channels status is needed. A fixed relationship exists between a change in address by the core and when the expected status (4 cycle delay) for that channel address must appear on the input bus.

In the example below, 'rxstck' and 'rxstck_line' are asynchronous. 'rxstck' is driven with the user system data clock ('SDCK') while 'rxstck_line' is driven by _ rate version of the transmit SPI4 line clock ('TxS4LS4_CK'). The input status bus from the user changes from a value of 0 to 2 coincident with 'rxstaddo' address output 0x1a (note: a 4 'rxstck' clock cycle delay before sampling status for a given address). The affected output channel can be identified by counting status slots using 'rxstck_line' clock from the framing marker of “3” equal to 0x1a or 6 clock cycles away in the status sequence.

Figure 2-11. S4RX User Transparent Status Mode Interface



RAM Mode
In RAM mode, the user writes status to an internal RAM in 16-bit bus format carrying 2-bit status fields for 8 channels per write cycle. A write strobe, 8-bit write mask field, and associated clock are also used in the write processes. The mask field allows the user to write the status for a single channel or any number of the other seven channels within the 16-bit memory location without affecting the status of the other channels.

The RAM interface is a user side write-only interface that operates in a synchronous fashion based on a user supplied 'rxstck' clock input signal. This clock signal is also used internal to the core for reading the Status RAM. Since there is no synchronization between locations written to the RAM by the user and locations read from the RAM by

internal logic, it is possible to both write and read the same location within the same clock cycle. This condition is covered within the design and ensures that static timing is met should the condition arise.

The timing diagram shown in Figure 2-12 presents a 32-channel application with continual writes ('rxstwr' active) to

1/8 of the Status RAM made up of 4 address locations 0 to 3 via the five bit address bus 'rxstaddi[4:0]' and a 16b status/(data) bus 'rxstat_r[15:0]'. Status for up to 8 channels is written into the RAM on each clock cycle that the write signal is active and associated channel mask bits ('rxstmsk[7:0]') are clear. Address location 0 corresponds to channels 0 to 7, address location 1 to channels 8 to 15, and so on. The least significant mask bit corresponds to the least significant channel and the mask is active high. For this example, there are two writes performed per memory location, one with the mask field set to zero's allowing the write and a second where the mask field is active illustrating the mask capability.

Up until time 455,420ns, a status of Starved (“00”) is written using a value of 0x0000 as well as a status of Satisfied (“10”) using a value of 0xaaaa for each location (two writes per location). The first write of the two is successful but the second is not due to an alternating mask field value of 0x00 and 0xff as shown ('rxstmsk') for each channel resulting in a Starved output status on 'rx_status' for all channel as shown.

After this time, the Satisfied state for all channels (0xaaaa) is written into the RAM during cycles where, this time, the mask field is clear resulting in new status. The first RAM location written with the new status value is location 0x3 corresponding to channels 24-31 (8 channels per location). For this first write, only 6 of the channels have changed and so a value of 0xaaa0 is actually written. This write takes place in time such that the new status for channel 26 of the 8 channel group makes it out on the external 'rx_status' interface before the next full status cycle starts. Using 'rxstck_line', a count of 6 clock cycles from the frame marker (value of 0x3) shows the first channel with the new status.

Calendar RAM

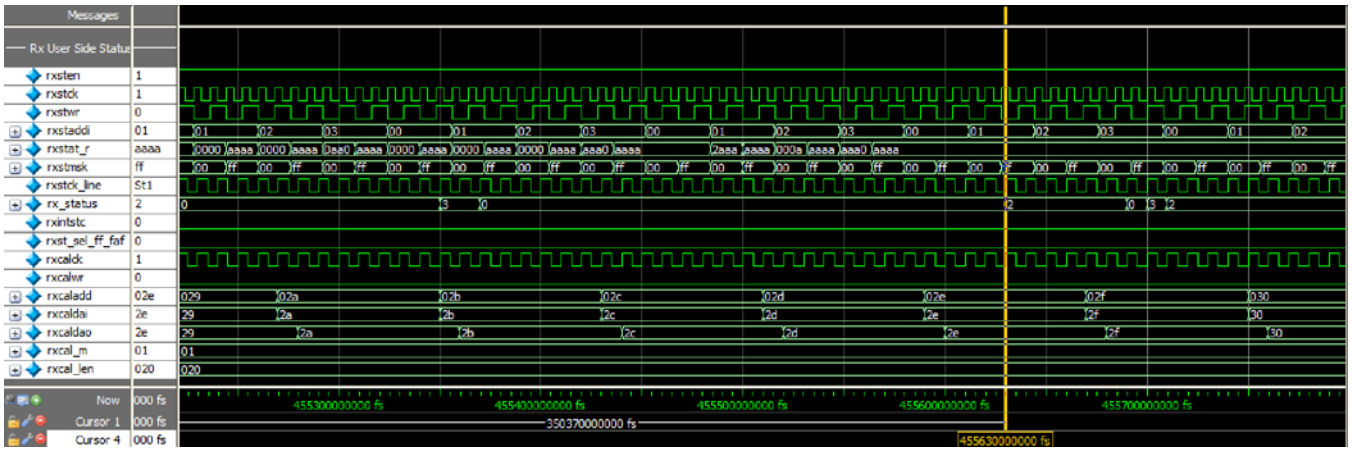
Regardless of the status mode chosen by the user to deliver status to the core, status is transmitted according to a local Calendar RAM of up to 512 locations (user accessible read/write) along with Framing and DIP2 parity information. The contents of this RAM is used to specify which channel's status is to be transmitted on a per-clock cycle basis. The internal reading of locations in the Calendar RAM is linear and the amount of memory read corresponds to a user supplied length field ('rxcal_len[]'). The phase of the address counter is arbitrary and is not synchronized to any other part of the system. For systems where the channels are not symmetrical in terms of bandwidth, the same channel can be programmed into multiple locations in the Calendar RAM resulting in multiple and more frequent status updates per status frame. The corresponding Calendar RAM used to receive status information at the far-end of the SPI4 link must be programmed to the same length and channel order.

The Calendar RAM interface is based on a True Dual Port RAM in which the user utilizes one port and the internals of the core utilize the other. The Calendar RAM interface is a user side “read/write” interface that operates in a synchronous fashion based on a rising edge active user supplied 'rxcalck' clock input signal. This clock signal is used for write access as well as for read access of the RAM through the user interface in which the address, and data for write cycles, are sampled before being applied to the RAM core (data is not clocked out of the RAM). Input signal 'rxcalwr' (act high) controls which operation is to be performed on a per cycle basis and reading does not take place when 'rxcalwr' is active. Reading or writing is allowed to take place on consecutive clock edges

The example shown in Figure 2-12 shows a small piece of what could be an initialization of the 512 location Calendar RAM if 'rxcalwr' were to be asserted. For this example, the RAM is used to support only 32 channels ('rxcal_len' = 0x20) where each channel has the same amount (1) of Calendar entries and therefore status bandwidth. Location 0 is programmed to a value of 0 (ch-0), location 1 is programmed to a value of 1 (ch-1), and so on ending with location 0x1f programmed to a value of 0x1f. Addresses beyond the calendar length are also written in the same fashion although not used. A single cycle is used to perform the write cycle. Once the calendar RAM is initialized, there is no need to continue writing as is shown in the figure.

Note that as the address input changes and 'rxcalwr' is de-asserted, so does the corresponding output data ('rxcaldao') based on 'rxcalck' and corresponds to the value written.

Figure 2-12. S4RX User Calendar RAM Interface



Internal Status Control

This design provides an option ('RxINTSTC'=1) to eliminate the need for user-generated status for single channel/pipe applications and applications where blocking is not a concern. When this behavior is selected, the user-side FIFO flags are used to automatically determine the appropriate three-state status to send on the receive status channel as defined in the OIF specification and shown in Figure 2-13. The user application can ignore the status channel and simply unload the user-side data FIFO in this mode since the far end will automatically be flow controlled based on the FIFO fill levels when the user application gets behind.

Figure 2-13. Internal Status Mode Encoding

- AF active = Satisfied
- AF inactive and AE inactive = Hungry
- AF inactive and AE active = Starved

Both almost empty and almost full flags are user programmable. When this option is selected, the user should select Transparent Mode to eliminate the unused Status RAMs from being implemented in the circuit.

SPI4 Receiver I/O - S4RXIO (RXGB)

The S4RXIO provides 1-2 gearing/de-multiplexing, DDR to SDR conversion, and clock/data alignment functions for the receive data direction (LVDS buffer insertion is done outside the core). De-multiplexing and DDR to SDR conversion functions are performed at the PIC level on an individual signal basis and therefore do not require any PLC logic resources. The input SPI4 bus is comprised of a 16-bit data bus 'RDAT_[P:N][15:0]', a control signal 'RCTL_[P:N]' and a source synchronous clock signal 'RCLK_[P:N]', all which of operate over differential pairs using LVDS levels. These 16 data signals plus one control signal are converted to “single-ended” mode by the LVDS buffers and sampled within the PICs logic using both edges of the received data clock 'RCLK [P:N]' implementing a 1-2 de-multiplexing function and thus doubling the number of signals from 17 to 34. The clock rate/frequency at this point remains the same as that of the SPI4 line but data is now in a single-edged clock format. Another 1:2 stage of de-multiplexing is performed at this point to reduce the 34-bit wide bus clock to a 1/2-rate clock frequency. This de-multiplexing function is also performed in the individual per-I/O signal PICs and results in a 64-bit wide data bus and 4-bit control bus leaving the PIC area of the FPGA.

Data and clock are transmitted by the sending device in-phase and therefore the receiver is responsible for adjusting the clock/data phase relationship such that on a per-signal level, reliable sampling can be achieved. In static mode, the function is performed using a common DLL and per-channel delay lines in order to achieve an exact 90 phase shift of clock relative to data at the sampling flip-flop inside the PICs for each signal of the SPI4 bus. In dynamic mode, this function is performed using AIL in LatticeSC/M devices. See see IPUG44, *LatticeSCM SPI4.2 MACO Core User's Guide* for further information.

Calendar and Status RAM Access

There are at most four RAMs within the core that are user accessible in terms of read/write. Each has their own address and data buses as shown in Table 2-1. The calendar RAMs are always present in the design and must be initialized through the bus provided. The status RAMs are optioned in or out depending upon the mode chosen for sending and receiving status. In Transparent Mode, there are no Status RAMs and therefore no bus-associated internal I/O. In RAM Mode, a read/write bus is provided.

Table 2-1. Signal Definitions

RAM Name	Access Type	Number Locations	Addr Bus	Data Bus	Description
Rx CALENDAR	R/W	512	9 bit	8 bit	Each location holds a 8-bit channel ID of the incoming status
Rx STATUS	W only	32	5 bit	16 bit	Each location holds 8, 2-bit status fields for 8 received status channels.
Tx CALENDAR	R/W	512	9 bit	8 bit	Each location holds a 8-bit channel ID of the outgoing status
Tx STATUS	R only	32	5 bit	16 bit	Each location holds 8, 2-bit status fields for 8 transmitted status channels.

Figure 2-14. Status RAM Layout

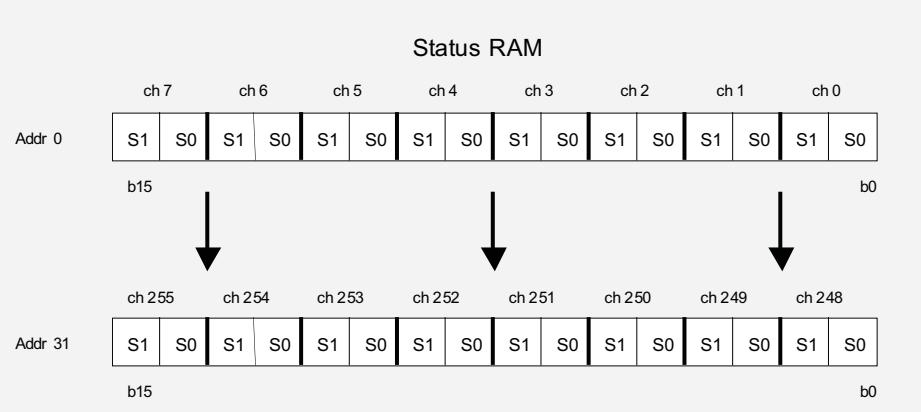
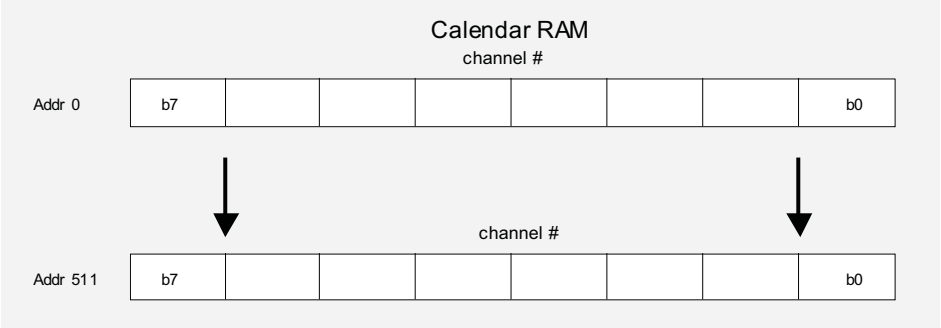


Figure 2-15. Calendar RAM Layout



Start-Up Procedures

Receive Direction Start-Up

This section describes the SPI4 Link Start-Up and Recovery procedures for the receive direction. Only static mode operation is supported in this offering.

Dynamic Mode Start-up and Recovery (SMSR) FSM

Dynamic mode is used in the LatticeSC device only. For more information about SPI4 dynamic mode, see IPUG44, *LatticeSCM SPI4.2 MACO Core User's Guide*.

Static Mode Start-up and Recovery (SMSR) FSM

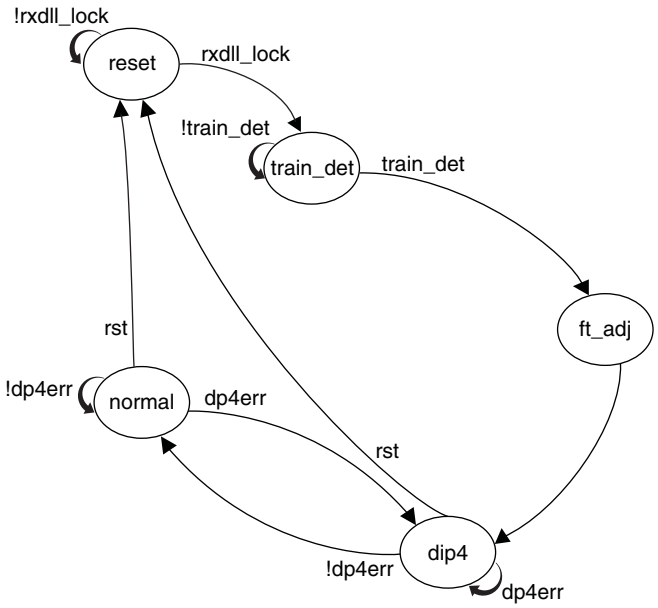
Figure 2-16 shows the Static Mode Startup and Recovery (SMSR) Finite State Machine (FSM) illustrating the receivers link start-up and recovery sequence. After a core reset, the SMSR FSM begins in the "reset" state and waits there until 'RXDLL_LOCK' is asserted. While in the "reset" state, as well as all others, except "normal", SMSR output signal 'RXAERR' is forced active causing constant framing patterns ("11") to be sent to the far end via the receive status channel and an inhibit on movement of data into the receive user interface FIFO. Sending constant Framing patterns to the far end causes it to respond with sending constant Training Control and Data words in return on the data interface which is needed for start-up functions.

Once the Receive DLL has successfully locked onto the clock edges and is confidently measuring the exact period of the receive clock ('RxDLL_LOCK'=1), state-flow will progress into state "train_det"'. If at this time the Training Detect circuit has successfully observed the Training and Control pattern, state-flow will progresses to state "ft_adj" (fine-tune adjust). Throughout this time, it is expected that the far-end is sending constant Training Control and Data patterns as specified by the OIF.

After the receiver has locked onto the receive clock and observed training patterns, state "DIP4" is entered and the receiver attempts to achieve synchronization with the data. Both the "good" and "bad" DIP4 parity error counters are cleared and a continuous analysis of the input data stream in terms of DIP4 error checking begins. When a programmable number of consecutive correct DIP4 control words are received, the In-Sync state is declared ('RXAERR'=0) and state "normal" is entered. When this occurs, valid status is now allowed to be sent to the far-end and data received from the SPI4 line is allowed to move into the user-side receive FIFO. At this point normal operation has begun. If at any time during normal operation a programmable number of consecutive DIP4 errors occurs, 1) state flow returns to the "DIP4" state, 2) the out-of-synchronization state is declared, and 3) again the far end is sent constant Status framing and the process begins all over again.

The above actions are taken autonomously by the DMSR-FSM as it continuously seeks to achieve a "normal" state where the receiver is in synchronization with the data (in-sync) without user involvement. The user can, however, force recovery actions from any state such as a full SMSR-FSM reset ('GRST_N').

Figure 2-16. Static Mode Start-Up and Recovery FSM



FSM Inputs:
rxdll_lock – Receive DLL Lock
train_det – Training detected
dp4err – Prog number of correct dip4 received
!dp4err – Prog number of incorrect dip4 received
grst_n – Global reset

FSM Outputs:
rxaerr – Inactive only in “normal” state

Transmit Direction Start-Up

During and immediately after reset is released, the SPI4 Transmitter (S4TX) continuously sends the training pattern on the output data interface and transmit status information is ignored. The duration of this condition is controlled by the status block and is based upon the reception of correct Framing and DIP2 parity from the far end of the link. Once the framing pattern is found and a programmable number of consecutive (based on 'TxNUMDip2') correct DIP2 matches are detected, the link is considered to be “in alignment”. Until alignment is achieved, the user should not read the status RAM since its content is unpredictable at this time. Upon entering the aligned state, data from the user-side transmit FIFO is allowed to be sent out on the SPI4 line and valid transmit status for at least one full status frame will have been written into the Tx Status RAM. In the Transparent Status mode, signal 'TXSTPA_VAL' is used to validate or invalidate 'TxSTPA[]' and 'TxSTAT_T' until alignment is achieved.

While aligned, a programmable number of consecutive (based on 'TxNUMDIP2E') DIP2 parity mismatches will cause a transition back to the Out-Of-Alignment state.

Signal Descriptions

The Soft SPI4 IP core I/Os are specified in Table 2-2 and Table 2-3. Table 2-2 provides the I/O for the internal user interface side and Table 2-3 provides the SPI4 line-side external I/O.

Table 2-2. User-Side Signal Descriptions

Signal Name	Direction	Description
S4RX/S4TX Common Signals		
GRST_N	Input	Soft SPI4 IP Core Global core reset (active low).
RXRST	Input	Async input for SPI4 RX, it will be internally used to sync reset the FIFO controllers, this signal is active high.
TXRST	Input	Async input for SPI4 TX, it will be internally used to sync reset the FIFO controllers, this signal is active high.

Table 2-2. User-Side Signal Descriptions (Continued)

Signal Name	Direction	Description
S4RX Internal Data Path Related Signals		
RXSDCK	Input	Receive System Data Clock – Should have over-speed relative to the SPI4 receive line clock /2 in 64b mode and /4 in 128b mode. Also used in user domain logic to transfer data from the IP core.
RXDATA[127,64:0]	Output	Receive System Data – User-side system data outputs from RxFIFO2 (4 - 16 bit data words = 64b mode, or 8 - 16 bit data words = 128b mode).
RXSOP	Output	Receive Start Of Packet – The corresponding data slice contains the start of a packet (a=1).
RXEOP	Output	Receive End Of Packet – The corresponding data slice contains the end of a packet (a=1).
RXREM[3,2:0]	Output	Receive Remainder – Indicates the byte lane position of the last valid data byte. A value of 0 = 1 byte, left justified MSB = b63 - b56 in 64b mode, b127-b120 in128b mode). A value of 7 (64b) or 15 (128b) = all bytes valid. During normal data transmission, the value should be either 7 (64b) or 15 (128b).
RXABT	Output	Receive Abort – Packet error status indicating the corresponding packet was received with an abort (a=1).
RXPA[7:0]	Output	Transmit Port Address (0 - 255).
RXDP4E	Output	Receive Dip4 Parity Error – Dip4 Parity Error indication (a=1) FIFO aligned to EOP. Error relevant for Packets and packet segments only - not active for single control words. See also RxD4ERR
RXDVAL	Output	Receive FIFO Data Valid – This signal when active (=1) qualifies RxFIFO2 data (RxDATA[]) as being valid. There is no fixed relationship to the RxFRD input and this RxDVAL signal - RxDATA[] should be ignored when inactive.
RXF1AE	Output	Receive FIFO1 Almost Empty – This signal when active (=1), indicates RxFIFO1 is almost empty as determined using parameter RXF1AETHRSH[8:0]below. This signal is used inside the core (S4RXSP) during abnormal conditions and factors into status sent to the far-end. This is a system level debug signal and does not require connection. See also output signal RXF1AF below.
RXF1AF	Output	Receive FIFO1 Almost Full – This signal when active (=1), indicates RxFIFO1 is almost full as determined using parameter RXF1AFTHRSH[8:0] below. When this signal is asserted (edge), the “Satisfied” condition will be transmitted on the RXSTATUS[1:0] channel until the corresponding RxF1AE signal is asserted (edge). This is a system level debug signal and does not require connection.
RXF1E	Output	Receive FIFO1 Empty – This status signal when active (=1), indicates RxFIFO1 is empty. This is a system level debug signal and does not require connection.
RXF1FE	Output	Receive FIFO1 Full Error – This error signal when active (=1), indicates RxFIFO1 is full and should be considered to have over-flowed. This error condition should never happen assuming an operational status path and may be monitored by user logic.
RXF2AE	Output	Receive FIFO2 Almost Empty - This signal when active (=1), indicates RxFIFO2 is almost empty as determined using synthesis parameter RXF2AETHRSH[8:0] below. This signal is used inside the core (S4RXSP) but may be used in the user-side interface. See also output signal RXF2AF below.
RXF2AF	Output	Receive FIFO2 Almost Full - This signal when active (=1), indicates RxFIFO2 is almost full as determined using synthesis parameter RXF2AFTHRSH[8:0] below. Also, when this signal is asserted (edge), the “Satisfied” condition can optionally be transmitted on the RXSTATUS[1:0] channel until the corresponding RxF2AE signal is asserted (edge).
RXF2E	Output	Receive FIFO2 Empty - This status signal when active (=1), indicates RxFIFO2 is empty and the user-side should stop reading.

Table 2-2. User-Side Signal Descriptions (Continued)

Signal Name	Direction	Description
RXF2FE	Output	Receive FIFO2 Full Error - This error signal when active (=1), indicates RxFIFO2 is full and should be considered to have over-flowed.
RXFRD	Input	Receive FIFO Read = This signal when active (=1), causes a read of RxFIFO2.
RXF1AFTHRSH[8:0]	Input	RxFIFO1 Almost Full Threshold - Threshold value in 16 byte increments. Affects Rx Flow Control.
RXF1AETHRSH[8:0]	Input	RxFIFO1 Almost Empty Threshold - Threshold value in 16 byte increments. Affects Rx Flow Control.
RXF2AFTHRSH[8:0]	Input	RxFIFO2 Almost Full Threshold - Threshold value in 16 byte increments.
RXF2AETHRSH[8:0]	Input	RxFIFO2 Almost Empty Threshold - Threshold value in 16 byte increments.
RXNUMDIP4[2:0]	Input	Receive Number Of DIP4 Words (2:0) - This parameter specifies the number of correct DIP4 words required before the S4RXDP receiver declares alignment.
RXNUMDIP4E[2:0]	Input	Number Of DIP4 Error Words (2:0) - This parameter specifies the number of incorrect DIP4 words that are required before the S4RXDP receiver declares an out-of-alignment error condition.
RXF1PARERR	Output	Receive FIFO1 Parity Error - This signal when active (=1) indicates that a parity error was detected on data read from RxFIFO1. This is a debug only signal - no connection is required.
RXRPD4ERR	Output	Receive Parser Data Parity Error - This signal when active (=1), indicates that a DIP-4 parity error has been detected on single control words such as Idles and SOP control words.
RXD4ERR	Output	Receive Data Parity Error - This signal when active (=1), indicates that a DIP-4 parity error has been detected. Includes packet/segment dip4 errors as well as single control word dip4 errors.
RXAERR	Output	Receive Alignment Error - This signal when active (=1), indicates the S4RXDP block has not achieved alignment (consecutive number of correct DIP4 words).
RS4ERR[4:0]	Output	Receive SPI4 Line Protocol Violation Errors - These signals when active (=1) indicate that the S4RXDP block has received a sequence of data and control words that are in violation of the SPI4 protocol. All signals are synchronous to the RxSDCK clock domain and will be active for at least 1 full cycle. RS4ERR[4] - A valid write to RxFIFO2 where less than 8 (64b) or 16 (128b) bytes are marked valid without an EOP active. This is "or"ed with the aligned DIP4 error lane through RxFIFO2 and is activated to mark the segment as bad. RS4ERR[3] - Data preceded by an idle. Error flag is raised. RS4ERR[2] - Reserved Control Word. Error flag is raised. RS4ERR[1] - Any EOP preceded by an idle. RS4ERR[0] - Any control word preceded by a payload control word.

Table 2-2. User-Side Signal Descriptions (Continued)

Signal Name	Direction	Description
RX User-Side Signals Used Only in LatticeSC Devices		
RXRST_AIL	Input	Receive Reset AIL - This signal when active causes a reset and re-acquisition of the SPI4 input signals on a per signal basis by the AIL circuits.
RXRSTDSKW	Input	Receive Reset De-Skew - This signal when active causes a reset of the de-skew module forcing it to perform the de-skew operation.
RXLT10_TRERR	Output	Receive Less Than 10 Training Patterns Error - This signal when active (=1) indicates that less than 10 training patterns were received during an “de-skewed” state. 10 repetitions are required to maintain de-skew once trained.
RXLT40_TRERR	Output	Receive Less Than 40 Training Patterns Error - This signal when active indicates that less than 40 training patterns were received during non de-skewed un-aligned state. 40 repetitions are required initially de-skew the line.
RXDSKWD	Output	Receive De-Skewed - This signal when active indicates that the de-skew block has successfully de-skewed the input SPI4 line
RXLOOP	Input	Receive Loop - Unused in this version of the core.
RXALNMD	Input	Receive alignment mode - This signal when active (=1) indicates that the RX AIL circuits are turned on for dynamic alignment operation.
RXTRAIN_EN	Input	Receive Training Enable - This signal when active means that the receiver must see Training and Control before declaring the receiver in-alignment. When inactive, the receiver can declare in-alignment state with merely correct DIP4.
RXDESKW_EN	Input	Receive Deskew Enable - This signal when active means that the deskew module is allowed to de-skew the input data stream.
S4RX Internal Status Path Related Signals		
RXCALCK	Input	Receive Calendar Clock - This clock signal is used to synchronously read/write the CALENDAR RAM. The frequency of this clock is not critical since once initialized, access is not expected (freq Max = 1/4 of data line clock).
RXSTCK	Input	Receive Status Clock - Receive Status Clock: This clock signal is used for writing in status into the core from the user logic side. It can be the same clock as the RXSTCK_LINE or the user application clock domain clock, such as SDCK (in 128-bit mode), or 1/2 of the SDCK (in 64-bit mode). Used in both Transparent and RAM modes.
RXSTCK_LINE	Input	Receive Status Line Clock - This clock signal is used for clocking status off-chip via the rx_status[1:0] channel (freq max = 1/4th of the data line clock). Used in both Transparent and RAM modes.
RXFDIP2E	Input	Transmit Force DIP2 Error [1:0] - This signal (bit 1) when active (=1) causes the S4RXSP to insert DIP2 parity errors. Framing remains valid.
RXINTSTC	Input	Receive Internal Status Control - When this signal is active (=1), status is generated internally (user status ignored) according to values of the RxFIFO2 flags (AE - Hungry, AF - Satisfied, and FF - Satisfied). This mode should be used only for single channel applications. See also static input RxST_SEL_FF_FAF. When RxINTSTC is inactive, only the AF or FF flag is used and will over-ride user status when active as a protective measure.
RxST_SEL_FF_FAF	Static Input	Receive Status Select FIFO Full FIFO Almost Full Behavior - This control signal selects whether the S4RXSP block reacts to the Almost Full (AF) = 0, or Full (FF) = 1 RxFIFO2 flag signals for generation of the Satisfied state on the Status interface.
RXSTEN	Input	Receive Status Enable - When active status is sent according to the order of the calendar RAM and Status RAM content. When inactive, constant framing pattern (‘11’) is sent.

Table 2-2. User-Side Signal Descriptions (Continued)

Signal Name	Direction	Description
RXSTWR	Input (RxSTCK)	Receive Status Write - This signal when active (=1) causes the data on input array RxSTAT[] to be written into the Status RAM location indexed by input array RxSTADD[]. The update is synchronous to the RxSTCK input clock. This input is used only in RAM status mode.
RXSTAT_R[15:0] or RXSTAT_T[1:0]	Input	Receive Status - When in RAM Status mode, this input status bus provides the application side interface with a mechanism for writing the “status” (starved, hungry, satisfied) of up to 8 channels at a time into the Status RAM. The most significant channel appears in the upper portion of the RxSTAT[] bus. RxSTADD[] location zero corresponds to channels 7 - 0. When in Transparent Status mode, this 2 bit input array provides a mechanism for writing status for a single channel per clock cycle (RxSTCK synchronous). The application side presents status for the channel requested by the core when in this mode via output array RxSTADDO[7:0].
RXSTADDI[4:0]	Input	Receive Status Address Input - This status address bus coupled with the TxSTAT[] bus provides the application side interface with a mechanism for writing the “status” (starved, hungry, satisfied) of up to 4 channels at a time into the Status RAM. Used only in the RAM Status mode.
RXSTADDO[7:0]	Output	Receive Status Address Out - This output status address bus coupled with the RxSTAT[] input bus provides the mechanism for implementing “Transparent” Status mode. In this mode, the core supplies, on a per clock cycle basis, the channel address (based on internal Calendar) for which Status is needed in order to supply the external SPI4 Rx_STATUS[] bus in real time - status is not stored in a local RAM. This bus is only used in Transparent Status mode.
RX_STATUS	Output	Receive Output Status - This is the final output receive status bound for the SPI4 line. This bus along with RXSTCK make up the SPI4 status bus.
RXSTMSK[7:0]	Input	Receive Status Mask - This field (a=1) provides a mask for the RxSTAT[15:0] bus so that 1 or any combination of the 8 channels associated with a single memory write location can be modified. Used only in the RAM Status mode.
RXCALWR	Input	Receive Calendar Write - This signal when active (=1) causes the data on input array TxCALDAI[] to be written into the Calendar RAM location indexed by input array TxCALADD[]. The update is synchronous to the RXCALCK input clock.
RXCALADD[8:0]	Input	Receive Calendar Address - Address index into the 512x8 Calendar RAM addressed as 512, 8-bit locations.
RXCALDAI[7:0]	Input	Receive Calendar Data Input - Calendar RAM data input bus.
RXCALDAO[7:0]	Output	Receive Calendar Data Output- Calendar RAM data output bus.
RXCAL_M[7:0]	Static Input	Receive Calendar Repetition - This field specifies the number of times that the calendar sequence is repeated before the DIP2 code word and framing are inserted (Alpha).
RXCAL_LEN[8:0]	Static Input	Receive Calendar Length - This field specifies the length of the calendar sequence. Note this value does not have to be equal to the number of channels populated.
RXS4LS2_CK	Output	Receive SPI4 Low-Speed Divide By 2 Clock - This is the divide by 2 version of the SPI4 receive line clock. General purpose use.
RXS4LS4_CK	Output	Receive SPI4 Low-Speed Divide By 4 Clock - This is the divide by 4 version of the SPI4 receive line clock. General purpose use.
S4TX Internal Data Path Related Signals		
TXSDCK	Input	Transmit System Data Clock - Should have over-speed relative to the SPI4 receive line clock /2 in 64b mode and /4 in 128b mode. Also used in user domain logic to transfer data to the IP core.
TXDATA	Input	Transmit System Data - User-side system data inputs (4 - 16 bit data words = 64b mode, or 8 - 16 bit data words = 128b mode.

Table 2-2. User-Side Signal Descriptions (Continued)

Signal Name	Direction	Description
TXFWR	Input	Transmit FIFO1 Write - When active (=1), the corresponding 4 or 8-word data slice is written into the TxFIFO1. If inactive, TXDATA is ignored.
TXSOP	Input	Transmit Start Of Packet - When active (=1), the corresponding data slice contains the start of a packet.
TXEOP	Input	Transmit End Of Packet - When active (=1), the corresponding data slice contains the end of a packet.
TXREM[3:0]	Input	Transmit Remainder - Indicates the byte lane position of the last valid data byte. A value of 0 = 1 byte, left justified MSB = b63 - b56 in 64b mode, b127-b120 in128b mode). A value of 7 (64b) or 15 (128b) = all bytes valid. During normal data transmission, the value should be either 7 (64b) or 15 (128b).
TXPA[7:0]	Input	Transmit Port Address (0 - 255).
TXERR	Input	Transmit Error - Control input (=1) that causes an EOP with Abort to be generated for the current packet.
TXF1FE	Output	Transmit FIFO1 Full Error - This error signal when active (=1), indicates TxFIFO1 is full.
TXF1AE	Output	Transmit FIFO1 Almost Empty - This signal when active (=1), indicates the TxFIFO1 is almost empty as determined using parameter TxF1AETHRSH[] below. This output can be used to determine when it is ok to begin writing TxFIFO1 after it has hit the almost full state.
TXF1AF	Output	Transmit FIFO1 Almost Full - This signal when active (=1), indicates TxFIFO1 is almost full as determined using parameter TxF1AFTHRSH[] below.
TXF2E	Output	Transmit FIFO2 Empty Error - This error signal when active (=1), indicates TxFIFO2 is empty; a condition which should never occur.
TXF2FE	Output	Transmit FIFO2 Full Error - This error signal when active (=1), indicates TxFIFO2 is full; a condition which should never occur given proper settings of TxFIFO2 thresholds.
TXF1AETHRSH[8:0]	Input	TxFIFO1 Almost Empty Threshold - Threshold value in 16 byte increments.
TXF1AEBTHRSH[6:0]	Input	TxFIFO1 Almost Empty Burst Threshold - Threshold value in 16 byte increments. Must be set to at least 2 greater than TxBLEN[5:0] below in 128b mode and 4 grater in 64b mode.
TXF1AFTHRSH[8:0]	Input	TxFIFO1 Almost Full Threshold - Threshold value in 16 byte increments.
TXF2AETHRSH[5:0]	Input	TxFIFO2 Almost Empty Threshold - Threshold value in 16 byte increments. Provides performance and latency tuning.
TXF2AEITHRSH[5:0]	Input	TxFIFO2 Almost Empty Idle Threshold - Threshold value in 16 byte increments. Provides performance and latency tuning.
TXF2AFTHRSH[5:0]	Input	TxFIFO2 Almost Full Threshold - Threshold value in 16 byte increments. Provides performance and latency tuning.
TXF2AFOTHRSH[1:0]	Input	TxFIFO2 Almost Full Offset Threshold - Threshold value in 16 byte increments. Provides performance and latency tuning.
TXBLEN[5:0]	Input	Tx Max Bust Size - Maximum number of 16 byte blocks that are transmitted for a given channel before segmentation is allowed (i.e. Training and Control Insertion).
TXMAXT[15:0]	Input	Tx Maximum Training Interval - This field indicates the maximum number of cycles * 4 between scheduling of the training sequences on the data path. A value of zero will disable training sequence generation.
TXREP[7:0]	Input	Tx Training Pattern Repetitions - This field indicates the number of repetitions of the training sequence (10 training control + 10 training data words). If SPI4 core TX is connected with Lattice SPI4 dynamic mode RX, it can be 0 or at least 10.
TxFIDLE	Input	Transmit Force Idles- This signal when active causes the S4TX transmitter to insert idle control words at the soonest available boundary.

Table 2-2. User-Side Signal Descriptions (Continued)

Signal Name	Direction	Description
TxENPACK	Input	Transmit Enable Packing - This signal when active causes the S4TX transmitter to pack the SPI4 line during cases where due to non-multiple of 16 byte EOPs there is bandwidth available. This control allows the user to turn packing off for early devices that may not be able to handle a packed line.
TxFDIP4E[1:0]	Input	Transmit Force DIP4 Error [1:0] - This signal (bit 1) when active (=1) causes the S4TX transmitter to insert DIP4 parity errors. Bit 0 determines whether all control words are sent with bad parity or just control words ending data segments are generated with bad DIP3 parity
TX_REM_ERR	Output	Transmitter Remainder Error - This error signal when active (a=1) indicates that TxFIFO1 was written with a remainder of other than 3'b111 (64b mode) or 4'b1111 (128b mode) during a cycle when 'TXEOP' was not active; a clear error situation.
TXBLEN_ERR	Output	Transmitter Burst Error - This error signal when active (a=1) indicates that transmitter has segmented a burst different than the burst size set through TxBLEN[5:0]. Ensures correct programming of TxF1AEBTHRSH[], which must be set to a value 2 greater than TxBLEN[5:0] in 64b mode and 4 grater in 128b mode.
TXF2PERR	Output	Transmit FIFO2 Parity Error - This signal when active (a=1) indicates that a parity error has been detected on a read operation of TxFIFO2. This is a debug only signal - no connection is required.
S4TX Internal Status Path Related Signals		
TXCALCK	Input	Transmit Calendar Clock - This clock signal is used to synchronously read/write the CALENDAR RAM. The frequency of this clock is not critical since once initialized, access is not expected (Freq Max 100Mhz).
TXSTCK	Input	This clock signal is used in the transmit user side status interface. All signals exchanged over this interface are synchronous to this clock. Can be a different clock than TX_STATUS_CK but must be in the range of, on the low side, TX_STATUS_CK and on the high side 150MHz. Can not be lower than TX_STATUS_CK.
TXSTEN	Input	Transmit Status Enable - This signal when active (=1), status channel processing is enabled. When inactive, signal TxSTAERR below is forced active.
TXSTEQP	Input	Transmit Status Equipped - This signal when active (=1) causes the data transmitter to factor in the state of the Status path (in/out of frame) before sending data. Forcing this signal inactive allows the transmitter send data without regard to the status path.
TXINTSTC	Input	Transmit Internal Status Control - This signal when active (=1), forces internal control over reading of TxFIFO1 based on internal analysis of the received status (can be used only in single channel applications).
TXSTAERR	Output (TxSTCK)	Transmit Status Alignment Error - This signal when active (=1) indicates the S4TSP block has not framed properly on the incoming SPI4 status channel.
TXSTAT_T[1:0]	Output (TxSTCK)	Transmit Status Transparent[1:0] - This status bus coupled with the TxSTPA[] address bus and TxSTPA_VAL output signals provides the application side interface with a transparent view of status as it is received on a per-channel basis from the far-end. This output is valid in both Transparent and RAM modes.
TXSTAT_R[15:0]	Output (TxSTCK)	Transmit Status [15:0] - When in the "RAM" status mode, this status bus coupled with the TxSTADD[] address bus provides the application side interface with a mechanism for reading the "status" (starved, hungry, satisfied) of up to 8 channels at a time. The most significant channel appears in the upper portion of the TxSTAT_R[] bus. TxSTADD[] location zero corresponds to channels 7 - 0. This I/O will not be present in Transparent mode.
TXSTADD[4:0]	Input (TxSTCK)	Transmit Status Address - This status address bus coupled with the TxSTAT_R[] data bus provides the application side interface with a mechanism for reading the "status" (starved, hungry, satisfied) of up to 8 channels at a time from the Status RAM when in RAM mode. This I/O will not be present in Transparent mode.

Table 2-2. User-Side Signal Descriptions (Continued)

Signal Name	Direction	Description
TXSTPA[7:0]	Output (TxSTCK)	Transmit Status Port Address - This signal array provides the user with an indication of which ports status is currently being received via the input TXSTATUS[] bus and processed by the S4TXSP logic. Available in both RAM and Transparent status modes.
TXSTPA_VAL	Output (TxSTCK)	Transmit Status Port Address Valid - This signal when active (=1) indicates that the value on TxSTPA[] and TxSTAT_T is valid. It will be inactive during the framing and dip2 time periods. In RAM mode, this signal with TxSTPA[] can be used to indicate which channels have recently received new status and can be read. The user must also take into consideration the alignment error signal when reading status (TxSTAERR).
TXCALWR	Input	Transmit Calendar Write - This signal when active (=1) causes the data on input array TxCALDAI[] to be written into the Calendar RAM location indexed by input array TxCALADD[]. The update is synchronous to the TxCALCK input clock.
TXCALADD[8:0]	Input	Transmit Calendar Address - Address index into the 512x8 location Calendar RAM. Addresses 512 8 bit locations of this RAM.
TXCALDAI[7:0]	Input	Transmit Calendar Data Input - Calendar RAM data input bus.
TXCALDAO[7:0]	Output	Transmit Calendar Data Output- Calendar RAM data output bus.
TXCAL_M[7:0]	Input	Transmit Calendar Repetition - Alpha.
TXCAL_LEN[8:0]	Input	Transmit Calendar Length.
TXSDP2ERR	Output	Transmit Status Parity Error - This signal when active (=1), indicates that the S4TSP block has detected a parity error on status received.
TXNUMDIP2[2:0]	Input	Transmit Number Of DIP 2 (0-7) - Specifies the number of correct DIP2 code words that are required before declaring alignment.
TXNUMDIP2E[2:0]	Input	Transmit Number Of DIP 2 Error (0-7) - Specifies the number of Incorrect DIP2 code words that are required before declaring out of alignment.
TXEN	Input	Transmit Enable - This signal when active (=1) shuts off the transmit data path at TxFIFO2 by inhibiting reading.
TXS4HS_CK	Input	Transmit SPI4 High Speed Clock - Line rate clock supplied from user logic.
TXS4LS2_CK	Input	Transmit SPI4 Low Speed Divide By 2 Clock - This is the divide by 2 version of S4TXHS_CK.
TXS4LS4_CK	Input	Transmit SPI4 Low Speed Divide By 4 Clock - This is the divide by 4 version of S4TXHS_CK.

The SPI4 line-side I/O signals in Table 2-3 reflect the of primary I/O of the device as well as the IP core itself. Since the IP core does not contain I/O buffers, they are not exactly the same. For example, on the output status interface (rx_status[]), the clock is driven from the top level netlist through a Primary clock tree rather than through the core. As part of the overall IP offering, example top-level netlists and supporting I/O buffer modules are provided as part of the evaluation package.

Table 2-3. SPI4 line-side I/O Signals

Primary I/O Signal Name	Core I/O Signal Name	Direction	Description
Receive Direction (Top Level)			
RCLK_P	rdclk	Input	Receive SPI4 Line Clock (P) – Used to sample the RDAT_[P:N][15:0] data bus.
RCLK_N	n/a	Input	Receive SPI4 Line Clock (N) – Used to sample the RDAT_[P:N][15:0] data bus.
RDAT_P[15:0]	rdat[15:0]	Input	Receive SPI4 Data (P) – SPI4 16 bit input DDR data bus.
RDAT_N[15:0]	n/a	Input	Receive SPI4 Data (N) – SPI4 16 bit input DDR data bus.
RCTL_P	rctl	Input	Receive SPI4 Control (P) – Control (=1) / Data (=0) indicator.

Table 2-3. SPI4 line-side I/O Signals (Continued)

Primary I/O Signal Name	Core I/O Signal Name	Direction	Description
RCTL_N	n/a	Input	Receive SPI4 Control (N) – Control (=0) / Data (=1) indicator
RX_STATUS[1:0]	rx_status[1:0]	Output	Receive Status – Output status channel associated with input/receive data path.
Receive Direction (Core Level)			
RX_STATUS_CK	n/a	Output	Receive Status Clock – Output status channel clock. Driven at the top-level via Primary Clock network.
RLDAT[63/127:0]	rldat[63/127:0]	Input	Receive SPI4 Data From RXGB – SPI4 64 or 128 bit input data bus.
RLCTL[3/7:0]	rlctl[3/7:0]	Input	Receive SPI4 Control From RXGB – SPI4 4 or 8 bit input data bus.
RX_STATUS[1:0]	rx_status[1:0]	Output	Receive Status – Output status channel associated with input/receive data path. Synchronized to RX_STATUS_CK
Receive Line-Side Signals Used Only in LatticeECP3 Devices			
RXDLL_LOCK	rxdll_lock	Input	Receive DLL Lock Error – This signal when active (=1) indicates that the DLL has lost lock with the incoming SPI4 line clock.
Receive Line-Side Signals Used Only in LatticeSC Devices			
RXAIL_LOCK (from rxgb)	rxail_lock	Input	Receive AIL Locked – This signal when active (=1) indicates that all AIL circuits are locked.
RUN_AIL (to rxgb)	run_ail	Output	Receive Run AIL – This signal when active (=1) allows the AIL circuitry to continue to refine its selection for clock data phase. When inactive, sampling continues but no further adjustments are made to clock/data phase.
RST_AIL (to rxgb)	rst_ail	Output	Receive Reset AIL – This signal when active causes a reset and re-acquisition of the SPI4 input signals on a per signal basis by the AIL circuits.
Transmit Direction (Top Level)			
TDCLK_P	tdclk	Output	Transmit SPI4 Line Clock (P) – Forward clock associated with the TDAT_[P:N][15:0] data bus.
TDCLK_N	n/a	Output	Transmit SPI4 Line Clock (N) – Forward clock associated with the TDAT_[P:N][15:0] data bus.
TDAT_P[15:0]	tdat[15:0]	Output	Transmit SPI4 Data (P) – SPI4 16 bit output DDR data bus.
TDAT_N[15:0]	n/a	Output	Transmit SPI4 Data (N) – SPI4 16 bit output DDR data bus.
TCTL_P	tctl	Output	Transmit SPI4 Control (P) – Control (=1) / Data (=0) indicator.
TCTL_N	n/a	Output	Transmit SPI4 Control (N) – Control (=0) / Data (=1) indicator
TX_STATUS[1:0]	tx_status[1:0]	Input	Transmit Status – Input status channel associated with output/transmit data path.
TX_STATUS_CK	tx_status_ck	Input	Transmit Status Clock – Input status channel clock.
Receive Direction (Core Level)			
TLDAT[63/127:0]	tldat[63/127:0]	Output	Transmit SPI4 Data To TXGB – SPI4 64 or 128 bit output data bus.
TLCTL[3/7:0]	tlctl[3/7:0]	Output	Transmit SPI4 Control To TXGB – SPI4 4 or 8 bit output data bus.
TX_STATUS[1:0]	tx_status[1:0]	Input	Transmit Status – Input status channel associated with input/receive data path.
TX_STATUS_CK	tx_status_ck	Input	Transmit Status Clock – Input status channel clock.

Parameter Settings

The IPexpress tool is used to create IP and architectural modules in the Diamond and ispLEVER software. Refer to "IP Core Generation" on page 42 for a description on how to generate the IP.

Table 3-1 provides the list of user configurable parameters for the FIR Filter IP core The parameter settings are specified using the Soft SPI4 IP core Configuration GUI in IPexpress. The numerous Soft SPI4 IP core parameter options are partitioned across multiple GUI tabs as shown in this chapter.

Table 3-1. Parameter Specifications for the FIR Filter IP Core

Parameter	Range/Options	Default
User Data Interface	64-Bit, 128-Bit	64-Bit
Generation Options		
Behavioral Model	Enabled, Disabled	Enabled
Netlist [.ngo]	Enabled, Disabled	Enabled
Evaluation Directory	Enabled, Disabled	Enabled
Synthesis Tool for Top	Synplify, Precision	Synplify
Transmit Data Path Options		
Maximum Training Interval [MAX_T]	0-65535]	32768
Training Patter Repetitions [ALPHA]	0-255	10
Minimum Burst Length	1-63	4
Transmit Line Side FIFO Thresholds		
Almost Empty	1-62	8
Almost Full	2-63	16
Transmit User Side FIFO Thresholds		
Almost Empty	1-510	80
Almost Full	2-511	384
Almost Full Offset Threshold	0, 1, 2, 3	2
Receive Data Path Options		
Receive Input Alignment Mode (LatticeSC/SCM Only)	Static, Dynamic	Dynamic
Receive Deskew Enable (LatticeSC/SCM Only)	Off, On	On
Receive Training Enable (LatticeSC/SCM Only)	Off, On	On
Receive Line Side FIFO Thresholds		
Almost Empty	1-510	40
Almost Full	2-511	400
Receive User Side FIFO Thresholds		
Almost Empty	1-510	40
Almost Full	2-511	384
Number of Correct DIP4 for In-Sync	1-7	7
Number of Incorrect DIP4 for Out-Sync	1-7	2
Status Channel Options		
Status Path Mode	RAM, Transparent	Transparent
Status Control	Internal (Automatic), External	External
Transmit Status Path Options		
Number of Correct DIP2 for In-Sync	1-7	7
Number of Incorrect DIP2 for Out-Sync	1-7	7

Table 3-1. Parameter Specifications for the FIR Filter IP Core (Continued)

Parameter	Range/Options	Default
Transmit Status Input Active Clock Edge	Rising, Falling	Falling
Receive Status Path Options		
Receive Status Input Active Clock Edge	Rising, Falling	Rising
Receive Status FIFO Fill Level Flag Select	Almost Full, Full	Almost Full
Transmit Calendar Options		
Transmit Calendar Length	1-512	4
Transmit Calendar Repetition	1-255	4
Receive Calendar Options		
Receive Calendar Length	1-512	4
Receive Calendar Repetition	1-255	4

Table 3-2 provides a list of the parameters generated from the GUI configuration process necessary to tailor a SPI4 core for specific user requirements. Some of the parameters are used in the IP core generation process to shape the IP in terms of the number of I/O or number and types of components used in the design. These parameters are synthesis-affecting and are labeled as “Core Synthesis” in the table.

Most of the parameters listed in Table 3-2 specify static `define values used to define the values of the various port arrays on the core when it is instantiated. The specific values selected in the GUI are used for both simulation and implementation (map, place and route) evaluation of the top-level FPGA evaluation design included in the IP core package (see `\src\rtl\top\ecp3\spi4_referencetop.v` or `spi4_core_only_top.v`). Users may also statically specify these parameter settings in their applications (see `\src\params\params.v`). For all of these types of parameters, users also have the option of making signal connections to these ports in their designs instead to provide real-time control. Parameters of this type are labeled as “Port Connection” since this is the point at which the parameter would be used.

See “IP Core Generation” on page 42 for more information regarding parameters and the “params.v” file.

Table 3-2. Parameters

Parameter	Description	Range	Default	Type
S4RX Parameters				
RXF1AETHRSH	Receive FIFO1 Almost Empty Threshold (16-byte units)	1-510	40	Port Connection
RXF1AFTHRSH	Receive FIFO1 Almost Full Threshold (16-byte units)	2-511	400	Port Connection
RXF2AETHRSH	Receive FIFO2 Almost Empty Threshold (16-byte units)	1-510	40	Port Connection
RXF2AFTHRSH	Receive FIFO2 Almost Full Threshold (16-byte units)	2-511	384	Port Connection
RXNUMDIP4	Receive Number of Correct DIP4 Before In-Sync	1-7	7	Port Connection
RXNUMDIP4E	Receive Number of Incorrect DIP4 Before Out-Sync	1-7	2	Port Connection
RXCAL_M	Receive Calendar Repetition	1-255	4	Port Connection
RXCAL_LEN	Receive Calendar Length	1-512	4	Port Connection
RXINTSTC	Receive Internal Status Control[1]	Internal/External	External	Port Connection
RXST_SEL_FF_FAF	Receive Status FIFO Flag (Full/Almost Full) Behavior[2]	Full/Almost Full	Almost_Full	Port Connection
RXSTREDGE	Receive Status Output Active Clock Edge Used	Rising/Falling	Rising if defined	FPGA Top Synthesis

Table 3-2. Parameters (Continued)

Parameter	Description	Range	Default	Type
RXSTAT_MD	Receive Status Mode (RAM / Transparent)[3]	RAM/Trans	Transparent if defined	Core Synthesis
Parameters for RX Dynamic Mode (LatticeSC/M devices only)				
RXALNMD	Receive Alignment Mode	Dynamic/Static	Dynamic	Port Connection
RXTRAIN_EN	Receive Training Enable	Enabled/ Disabled	Enabled	Port Connection
RXDESKEW_EN	Receive Deskew Enable	Enabled/ Disabled	Enabled	Port Connection
S4TX Parameters				
TXNUMDIP2	Transmit Number of Correct DIP2 Before In-Sync	1-7	7	Port Connection
TXNUMDIP2E	Transmit Number of In-Correct DIP2 Before Out-Sync	1-7	7	Port Connection
TXCAL_M	Transmit Calendar Repetition	1-255	4	Port Connection
TXCAL_LEN	Transmit Calendar Length	1-512	4	Port Connection
TXINTSTC	Transmit Internal Status Control[1]	Internal/External	External	Port Connection
TXSTREDGE	Transmit Status Input Active Edge Used	Rising/Falling	Falling if not defined	FPGA Top Synthesis
TXBLEN	Transmit Burst Length (16 byte units)	1-63	4	Port Connection
TXPACK_EN	Transmit Packing Enable	Enabled/ Disabled	Enabled	Port Connection
TXMAXT	Transmit Maximum Training Interval[4]	0-65536	32768	Port Connection
TXREP	Transmit Training Pattern Repetitions[5]	0-255	10	Port Connection
TXF1AEBTHRSH	Transmit FIFO1 Almost Empty Burst Threshold (16-byte units)	1-63	6	Port Connection
TXF1AETHRSH	Transmit FIFO1 Almost Empty Threshold (16-byte units)	1-510	80	Port Connection
TXF1AFTHRSH	Transmit FIFO1 Almost Full Threshold (16-byte units)	2-511	384	Port Connection
TXF2AETHRSH	Transmit FIFO2 Almost Empty Threshold (16-byte units)	1-63	8	Port Connection
TXF2AEITHRSH	Transmit FIFO2 Almost Empty Idle Threshold (16-byte units)	1-63	7	Port Connection
TXF2AFTHRSH	Transmit FIFO2 Almost Full Threshold (16-byte units)	2-63	16	Port Connection
TXF2AFOTHRSH	Transmit FIFO2 Almost Empty Offset Threshold (16 byte units)	0-3	2	Port Connection
TXSTAT_MD	Transmit Status Mode (RAM / Transparent)	RAM/ Transparent	Transparent if defined	Core Synthesis

Table 3-2. Parameters (Continued)

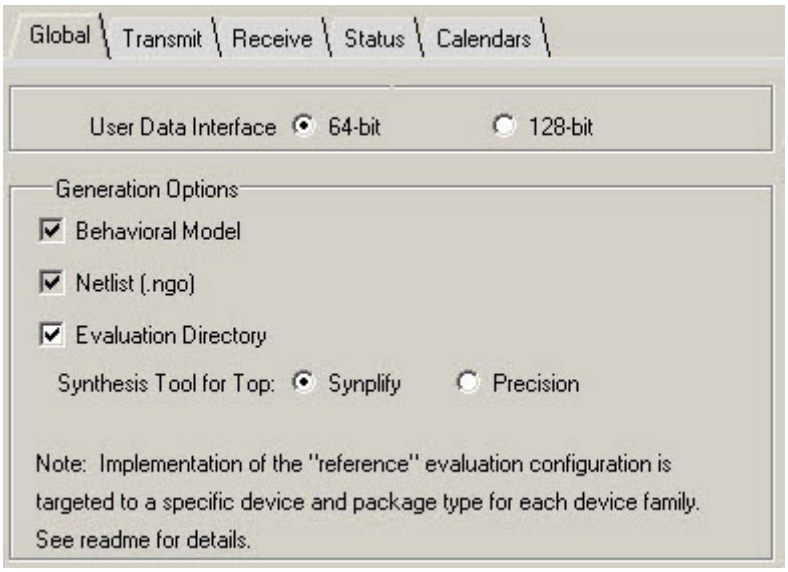
Parameter	Description	Range	Default	Type
Global Parameters				
SPI4_128	IP Core Internal Architecture 64/128b mode	64b,128b	64b if not defined	Core Synthesis

1. This parameter ('RxINTSTC') when set to “True” forces the contents of the receive output status channel to be determined completely from within the core. Three status conditions are sourced (Starved/Hungry/Satisfied) based on the fill level of RxFIFO2. In this mode, the user does not supply channel status.
2. This parameter selects between RxFIFO2 Almost Full and Full flags as to which one will force a Satisfied” condition on the output status channel (forced Almost Full when 'RxINTSTC' = True).
3. This parameter selects between two methods of supplying status to the core for the output status interface. In transparent mode, the user supplies status for each channel in real-time via a two-bit bus based on a pre-scribed order controlled by the core via the Calendar RAM.
4. This parameter (TXMAXT) needs to be even number, odd value is not supported.
5. This parameter (TXREP) can be 0-255 for static mode (in LatticeECP3), for dynamic mode (in LatticeSC/M). If the SPI4 core TX is connected with the Lattice SPI4 RX, 1-9 is not acceptable.

Global Tab

Figure 3-1 shows the contents of the Global tab.

Figure 3-1. Global Tab



User Data Interface

This option selects whether a 64-bit or 128-bit user-side interface is generated. The option also corresponds to the data pipe-line width internal to the core.

Generation Options

These options are general in nature having to do with optional content created during generation.

Behavioral Model

This option selects whether a behavioral simulation model is generated. For this version of the core, a behavioral simulation model is always generated.

Netlist [.ngo]

This option selects whether an .ngo netlist is created during IP generation. For this version of the core, an .ngo file is always generated.

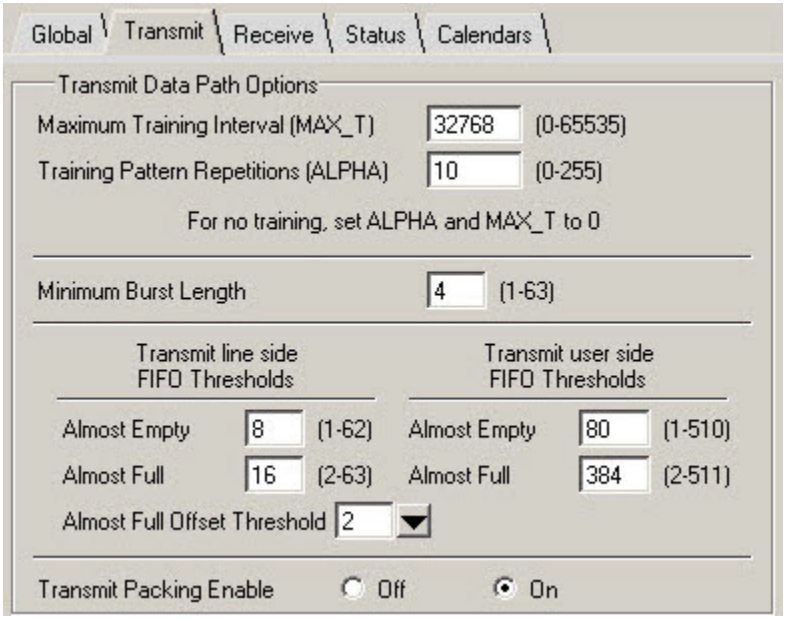
Evaluation Directory
This option selects whether an evaluation simulation and implementation capability is created during IP generation.

Synthesis Tool for Top
This option selects which synthesis tool will be used or the evaluation implementation capability created during IP generation.

Transmit Tab

Figure 3-2 shows the contents of the Transmit tab.

Figure 3-2. Transmit Tab



Transmit Data Path Options

This option specifies the maximum interval between scheduling of Training Sequences on the data path.Maximum Training Interval (MAX_T).

Training Pattern Repetitions (ALPHA)
This option specifies the number of repetitions of the training data sequence that must be scheduled every MAX_T interval.

Minimum Burst Length
This option selects the minimum Burst Length. See “Minimum Burst Size - Burst Mode” on page 9 for details.

Transmit Line Side FIFO Thresholds

These options select the transmit line side FIFO (TxFIFO2) Almost Empty and Almost Full thresholds. See “Transmit FIFO2 Threshold Optimizations” on page 10 for details.

Transmit User Side FIFO Thresholds

These options select the transmit user side FIFO (TxFIFO1) Almost Empty and Almost Full thresholds. See section “SPI4 Transmit Data Protocol - S4TXDP” on page 7 for details.

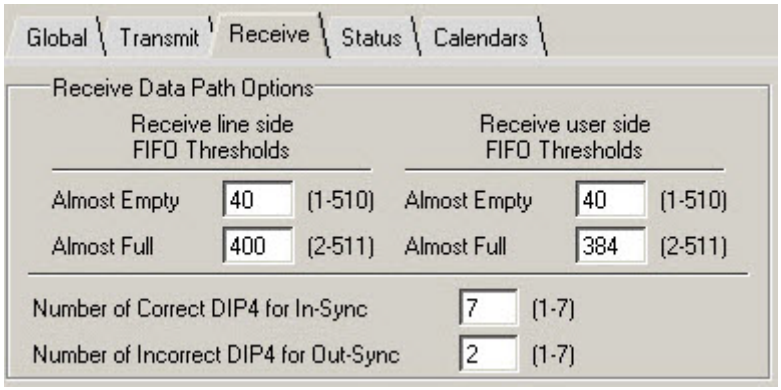
Transmit Packing Enable

This option selects whether or not the S4TX transmitter is allowed to pack the SPI4 line during cases where, due to non-multiple of 16 byte EOPs, there is bandwidth available. This selection allows the user to turn packing off for early devices that may not be able to handle a packed line.

Receive Tab – LatticeECP

Figure 3-3 shows the contents of the Receive tab for LatticeECP.

Figure 3-3. Receive Tab - LatticeECP



Receive Data Path Options

Receive Line Side FIFO Thresholds

These options set the Almost Empty and Almost Full thresholds for the receive line side FIFO (RxFIFO1). The flags that results from these thresholds are used inside the core (S4RXSP) during abnormal conditions and factors into status sent to the far-end.

Receive User Side FIFO Thresholds

These options selects the Almost Empty and Almost Full thresholds for the user side FIFO (RxFIFO2). The flags that result from these thresholds are used to drive receive status during normal operation when internal Status mode is selected.

Number of Correct DIP4 In-Sync

This option specifies the number of consecutive dip4 errors that must be seen before declaring the data link out of alignment/synchronization.

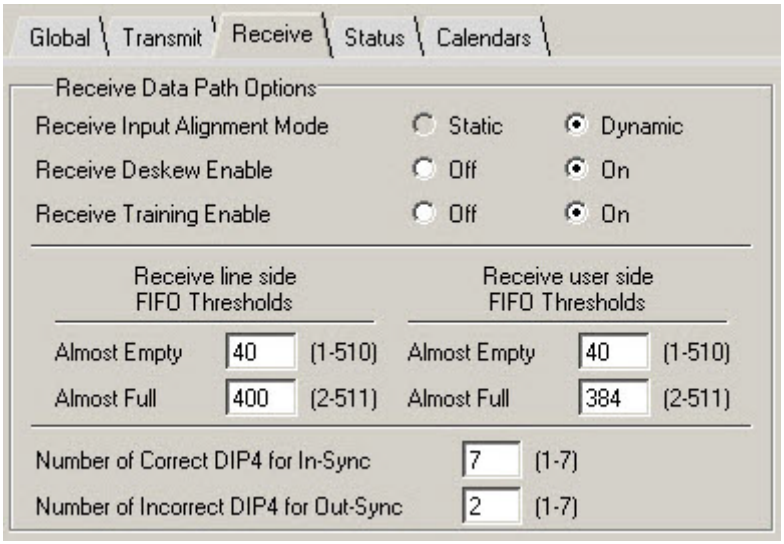
Number of Incorrect DIP4 In-Sync

This option specifies the number of consecutive correct dip4 code words that must be seen before declaring the data link in alignment/synchronization.

Receive Tab – Lattice SC/SCM

Figure 3-4 shows the contents of the Receive tab for LatticeSC/SCM.

Figure 3-4. Receive Tab - LatticeSC/SCM



Receive Data Path Options

Receive Input Alignment Mode
This option selects whether Static Or Dynamic mode operation is used. In this version of the SC/SCM core, only Dynamic mode is support.

Receive Deskew Enable
This option selects whether the Deskew function is enabled or not. This is a debug feature and should be set to enabled for normal operation.

Receive Training Enable
This option selects whether or not the requirement that Training and Control words be observed before declaring in-sync is to be honored. This is a debug option for possible non-compliant devices and should be set to enabled for normal operation.

Receive Line Side FIFO Thresholds
These options set the low and high water thresholds for the receive line side FIFO (RxFIFO1). The flags that results from these thresholds are used inside the core (S4RXSP) during abnormal conditions and factors into status sent to the far-end.

Receive User Side FIFO Thresholds
These options select the thresholds for the user side FIFO (RxFIFO2). The flags that result from these thresholds are used to drive receive status during normal operation when internal Status mode is selected.

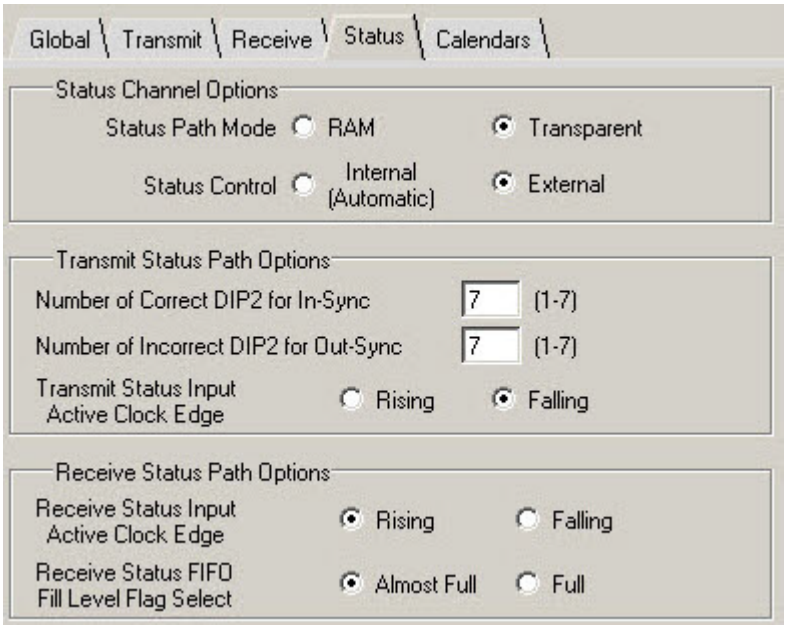
Number of Correct DIP4 In-Sync
This option specifies the number of consecutive correct dip4 code words that must be seen before declaring the data link in alignment/synchronization.

Number of Incorrect DIP4 In-Sync
This option specifies the number of consecutive dip4 errors that must be seen before declaring the data link out of alignment/synchronization.

Status Tab

Figure 3-5 shows the contents of the Status tab.

Figure 3-5. Status Tab



Status Channel Options

Status Path Mode

This option selects the Status mode used to report status to user logic. See "SPI4 Receive Status Protocol - S4RXSP" on page 18 for details.

Status Control

This option selects wether external user logic functions mange the content of the SPI4 status channel or the IP core manages the content of status channel. See section "SPI4 Receive Status Protocol - S4RXSP" on page 18 for details.

Transmit Status Path Options

Number of Correct DIP2 for In-Synch

This option specifies the number of consecutive correct dip2 code words that must be seen before declaring the status channel in alignment/synchronization.

Number of Incorrect DIP2 for Out-Synch

This option specifies the number of consecutive dip2 errors that must be seen before declaring the status channel out of alignment/synchronization.

Transmit Status Input Active Clock Edge

This options selects which clock edge (rising/falling) is used to sample input status of chip on the status interface.

Receive Status Path Options

Receive Status Input Active Clock Edge

This options selects which clock edge (rising/falling) is used to clock status of chip on the output receive status interface.Receive Status FIFO Fill Level Flag Select

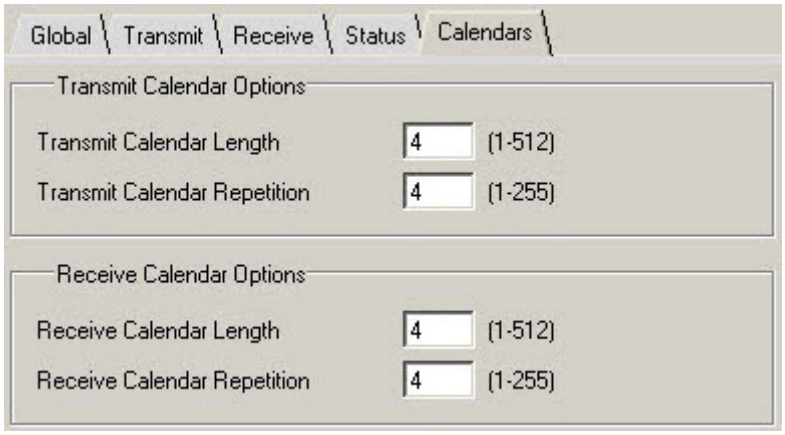
Receive Status FIFO Fill Level Flag Select

This option selects whether the RxFIFO2 Almost Full or Full flag forces/over-rides user status when activated.

Calendars Tab

Figure 3-6 shows the contents of the Calendars tab.

Figure 3-6. Calendars Tab



Transmit Calendar Options

Transmit Calendar Length

This option selects the transmit calendar length (CAL_LEN). See "SPI4 Transmit Status - S4TXSP" on page 11 for details.

Transmit Calendar Repetition

This option selects the transmit calendar repetition value (CAL_M). See "SPI4 Transmit Status - S4TXSP" on page 11 for details.

Receive Calendar Options

Receive Calendar Length

This option selects the receive calendar length (CAL_LEN). See "SPI4 Receive Status Protocol - S4RXSP" on page 18 for details.

Receive Calendar Repetition

This option selects the receive calendar repetition value (CAL_M). See section "SPI4 Receive Status Protocol - S4RXSP" on page 18 for details.

IP Core Generation

This chapter provides information on licensing the Soft SPI4 IP core, generating the core using the Diamond or ispLEVER software IPexpress tool, running functional simulation, and including the core in a top-level design. The Soft SPI4 IP core can be used in LatticeECP3 and LatticeSC/M device families.

Licensing the IP Core

An IP license is required to enable full, unrestricted use of the Soft SPI4 IP core in a complete, top-level design. An IP license that specifies the IP core and device family (ECP3 or SC/M) is required to enable full use of the Soft SPI4 IP core. Instructions on how to obtain licenses for Lattice IP cores are given at:

http://www.latticesemi.com/products/intellectualproperty/aboutip/isplevercoreonlinepurchas.cfm

Users may download and generate the IP core and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license. The Soft SPI4 IP core also supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited time (approximately four hours) without requiring an IP license (see "Hardware Evaluation" on page 48 for further details). However, a license is required to enable timing simulation, to open the design in the Diamond or ispLEVER EPIC tool, and to generate bitstreams that do not include the hardware evaluation timeout limitation.

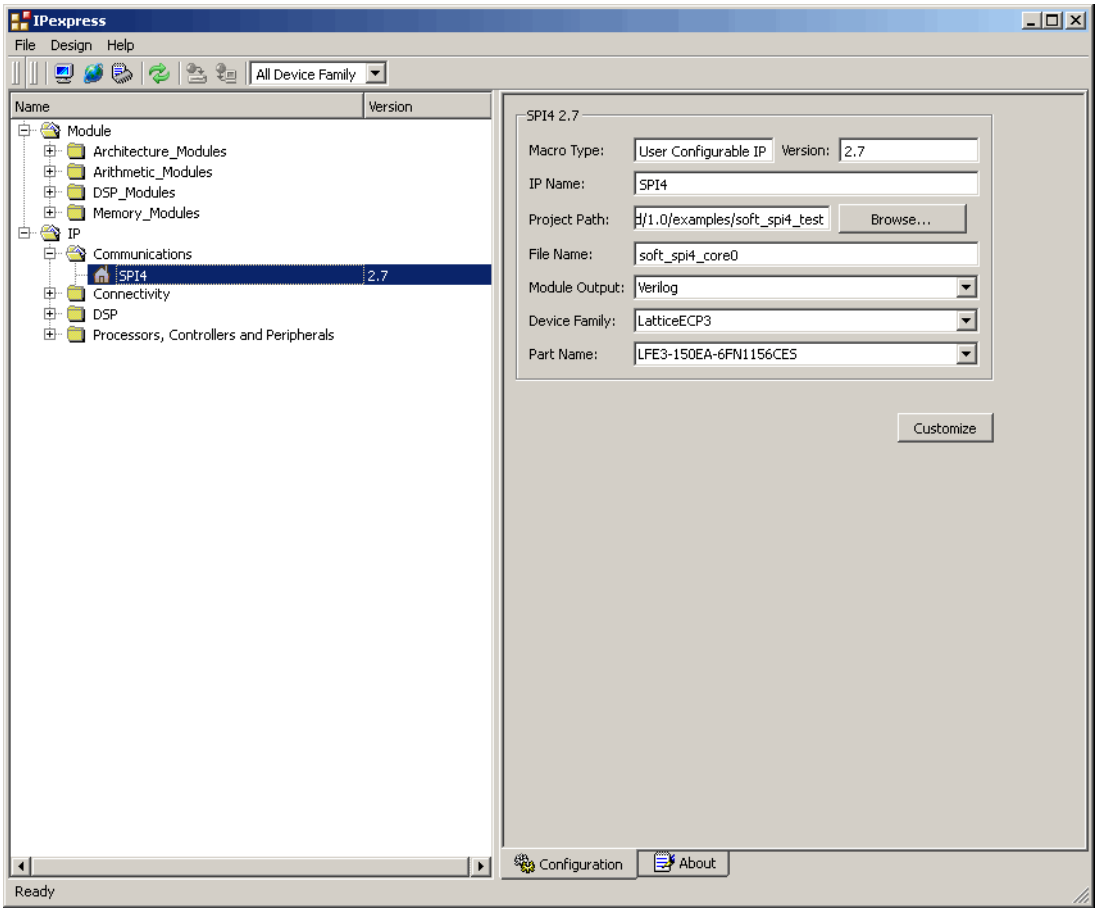
Getting Started

The Soft SPI4 IP core is available for download from the Lattice IP Server using the IPexpress tool in Diamond or ispLEVER. The IP files are automatically installed using ispUPDATE technology in any customer-specified directory. After the IP core has been installed, it will be available in the IPexpress GUI dialog box shown in Figure 4-1.

The IPexpress tool GUI dialog box for the Soft SPI4 IP core is shown in Figure 4-1. To generate a specific IP core configuration the user specifies:

- **Project Path** – Path to the directory where the generated IP files will be loaded.
- **File Name** – "username" designation given to the generated IP core and corresponding folders and files.
- **(Diamond) Module Output** – Verilog or VHDL.
- **(ispLEVER) Design Entry Type** – Verilog HDL or VHDL
- **Device Family** – Device family to which IP is to be targeted (e.g. Lattice ECP2M, LatticeECP3, etc.). Only families that support the particular IP core are listed.
- **Part Name** – Specific targeted part within the selected device family.

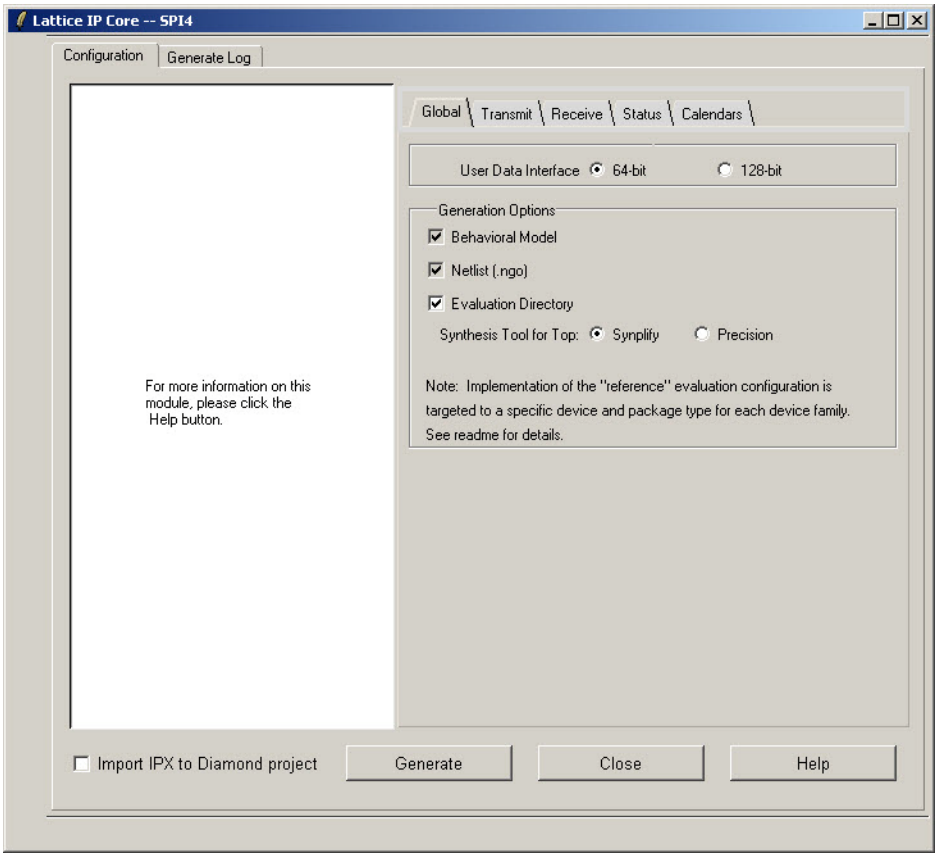
Figure 4-1. IPexpress Dialog Box (Diamond Version)



Note that if the IPexpress tool is called from within an existing project, Project Path, Module Output (Design Entry in ispLEVER), Device Family and Part Name default to the specified project parameters. Refer to the IPexpress tool online help for further information.

To create a custom configuration, the user clicks the **Customize** button in the IPexpress tool dialog box to display the DA-FIR Filter IP Configuration GUI, as shown in Figure 4-2. From this dialog box, the user can select the IP parameter options specific to their application. Refer to "Parameter Settings" on page 33 for more information on the DA-FIR Filter IP parameter settings.

Figure 4-2. Configuration GUI (Diamond Version)



IPexpress-Created Files and Top Level Directory Structure

When the user clicks the **Generate** button in the IP Configuration dialog box, the IP core and supporting files are generated in the specified “Project Path” directory. The directory structure of the generated files is shown in Figure 4-3.

Figure 4-3. LatticeECP3 Core Directory Structure

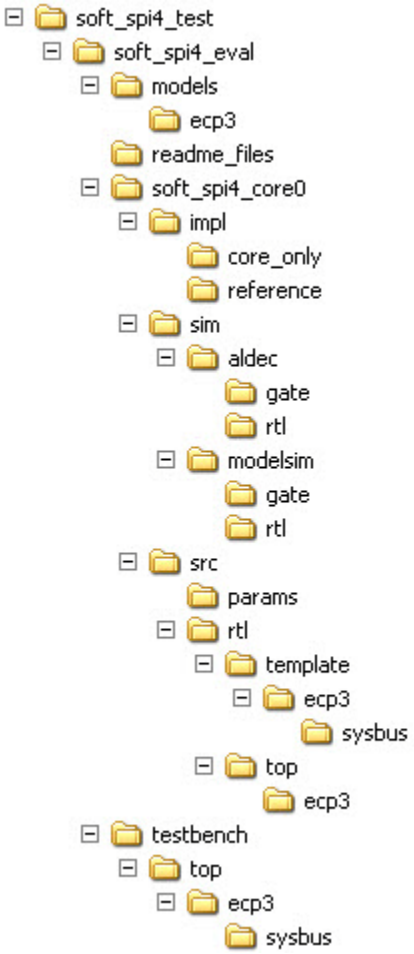


Figure 4-3 provides a list of key files and directories created by the IPexpress tool and how they are used. The IPexpress tool creates several files that are used throughout the design cycle. The names of most of the created files are customized to the user's module name specified in the IPexpress tool.

Table 4-1. File List

File	Description
<*username*>.lpc	This file contains the IPexpress tool options used to recreate or modify the core in the IPexpress tool.
<*username*>.ipx	The IPX file holds references to all of the elements of an IP or Module after it is generated from the IPexpress tool (Diamond version only). The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP/Module generation GUI when an IP/Module is being re-generated.
<*username*>.ngo	This file provides the synthesized IP core.
<*username*>_bb.v/.vhd	This file provides the synthesis black box for the user's synthesis.
<*username*>_inst.v/.vhd	This file provides an instance template for the PCI IP core.
<*username*>_beh.v/.vhd	This file provides the front-end simulation library for the PCI IP core.

Table 4-2 provides a list of key additional files providing IP core generation status information and command line generation capability are generated in the user's project directory.

Table 4-2. Additional Files

File	Description
<*username*>_generate.tcl	This file is created when the GUI "Generate" button is pushed. This file may be run from command line.
<*username*>_generate.log	This is the synthesis and map log file.
<*username*>_gen.log	This is the IPexpress IP generation log file

As mentioned previously, the \<soft_spi4_eval> directory is only generated if the Generation Option "Evaluation Directory" has been selected in the GUI. The \<soft_spi4_eval> and subtending directories provide files supporting SPI4 core evaluation. The \<soft_spi4_eval> directory shown in Figure 4-3 contains files/folders with content that is constant for all configurations of the SPI4. The \<username> subfolder (\soft_spi4_core0 in this example) contains files/folders with content specific to the username configuration.

The \soft_spi4_eval directory is created by IPexpress the first time the core is generated and updated each time the core is regenerated. A \<username> directory is created by IPexpress each time the core is generated and regenerated each time the core with the same file name is regenerated. A separate \<username> directory is generated for cores with different names, e.g. \<soft_spi4_core0>, \<soft_spi4_core1>, etc.

Instantiating the Core

The generated Soft SPI4 IP core package includes black-box (<*username*>_bb.v) and an instance (<*username*>_inst.v) template that can be used to instantiate the core in a top-level design. An example RTL top-level reference source file is also provided in \<*project_dir*>\soft_spi4_eval\<*username*>\src\rtl\top\ecp3\spi4_referencetop.v. Users may use this top-level reference as the starting template for their top-level complete design. In the example, the SPI4 core is instantiated in the FPGA top-level file along with some test logic (user-side spi4 loop, PLLs, I/O buffers, debug, etc.).

The top-level RTL example design is also supported by some of the parameters defined in the params.v file found in \<*project_dir*>\soft_spi4_eval\<*username*>\src\params\params.v. A description of the parameters for this IP core and top-level design is provided in the Parameter Descriptions section of this document.

The top-level file spi4_reference_top.v is the same top-level file that is used in the simulation model described in the next section.

To instantiate this IP core using the Linux platform, users must manually add one environment variable named "SYNPLIFY" to indicate the installation path of SYNPLIFY TOOLS in the local environment file.

Running Functional Simulation

Simulations utilize a SPI4 test-bench top-level file (\testbench\top\spi4_tb_top.v) that connects a SPI4 simulation driver (provides a SPI4 bus source /sink) to the same FPGA top-level file mentioned above and contains the Soft SPI4 IP core and user-side loop-around module. The loop-around module loops packets and packet fragments received from the SPI4 driver back to it through the user-side interface of the SPI4 core inside the FPGA. This mode of simulation testing is referred to as a far-end loop. This evaluation test-bench top-level provides clock sources and the context for SPI4 simulation driver instantiation as well as instantiation of the FPGA top and connection of the two.

The capability provided reflects a configuration-specific simulation of the Soft SPI4 IP core that is consistent with the default settings of the GUI process. Varying GUI parameters are permitted but not all possible combinations are guaranteed to yield a successful simulation, especially those with large channel count numbers due to test-bench limitations. The functional simulation includes a configuration-specific behavioral model of the Soft SPI4 IP Core (spi4_soft_core_beh.v) that is instantiated in an FPGA top level.

Users may run the simulation by doing the following:

1. Open ModelSim

2. Under the File tab, select **Change Directory** and choose folder
   <project_dir>\soft_spi4_eval\<*username*>\sim\modelsim\rtl

3. Execute simulation do script do <username>_eval.do**.**

The following SPI4 data formats will be run. If errors occur, they will be written to the ModelSim dialogue window. The simulation waveform results will be displayed in the ModelSim Wave window.

- // DF00 - Incremental length single/full bursts
- // DF01 - Incremental length interleaved multiple bursts packets
- // DF02 - Random length interleaved multiple bursts packets
- // DF03 - Back-to-back EOPs (up to 4 EOPs per slice)
- // DF04 - Even-even, even-odd, odd-even, odd-odd bursts etc.
- // DF05 - Abort testing
- // DF06 - DIP4 indication
- // DF07 - Training pattern filtering (currently not supported)

Synthesizing and Implementing the Core in a Top-Level Design

The Soft SPI4 IP core itself is synthesized and provided in NGO format when the core is generated. Users may synthesize the a black box of the core in their own top-level design by instantiating the black box (<*username*>_bb.v) in their top-level as described previously and then synthesizing the entire design with either Synplify or Precision RTL Synthesis.

As mentioned above, an evaluation capability of the core in a synthesis and map, place and route context is also created for the user and is based upon the FPGA top-level file described above. The user can, through Diamond or ispLEVER, browse to the generated directory \<*project_dir*>\soft_spi4_eval\<*core_name*>\impl and load the <*project_dir*>.syn file. SPI4 specific map and par options will automatically be loaded having been created by the generation process. The user can then run map and place and route only or run through the entire Diamond or ispLEVER flow including synthesis map, place, and route. All settings are automatically set up by the generation phase.

Place and route is supported for both Core_Only configurations as well as Full_Top configurations that include the SPI4 Loop Module (S4LP). Selection is made in the GUI capture phase. When the Core_Only option is selected and there are no connections to the internal user-side interface, the map -u command line option is used to ensure that unused logic is not minimized away and an accurate count of IP core logic can be made. I/O insertion is disabled during synthesis and so these internal nets are simply left dangling and warnings will be seen during map for them as expected.

When the Full_Top configuration is used, a small amount of extra logic (~200 slices) is added to provide the user-side SPI4 loop-back capability but the design can now be evaluated as a complete FPGA design. This design provides a far-end SPI4 loop-back capability that can be simulated (see below) and then taken into a lab “as is” without modification for hardware evaluation.

For the Diamond or ispLEVER Linux platform, the top-level synthesis must be run separately with a standalone synthesis tool, such as Synplify Pro, since ispLEVER for Linux only accepts an EDIF entry. Synthesis tcl files will be generated for this purpose including spi4_top.tcl in the directory \<*project_dir*>\soft_spi4_eval\<*username*>\impl\syn_eval.

The user can type the following command to synthesize the top_level files:

```
synplify_pro
batch spi4_top.tcl
```

To use this project file in Diamond:

1. Choose **File > Open > Project**.
2. Browse to \<*project_dir*>\soft_spi4_eval\<*username*>\impl\synplify (or precision) in the Open Project dialog box.
3. Select and open <*username*>.ldf. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.

4. Select the **Process** tab in the left-hand GUI window.

5. Implement the complete design via the standard Diamond GUI flow.

To use this project file in ispLEVER:

1. Choose **File > Open Project.**
2. Browse to \<*project_dir*>\soft_spi4_eval\<*username*>\impl\synplify (or precision) in the Open Project dialog box.
3. Select and open <*username*>.syn. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.
4. Select the device top-level entry in the left-hand GUI window.
5. Implement the complete design via the standard ispLEVER GUI flow.

Hardware Evaluation

The Soft SPI4 IP core supports supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited period of time (approximately four hours) without requiring the purchase of an IP license. It may also be used to evaluate the core in hardware in user-defined designs.

Enabling Hardware Evaluation in Diamond

Choose **Project > Active Strategy > Translate Design Settings**. The hardware evaluation capability may be enabled/disabled in the Strategy dialog box. It is enabled by default.

Enabling Hardware Evaluation in ispLEVER

In the Processes for Current Source pane, right-click the **Build Database** process and choose **Properties** from the dropdown menu. The hardware evaluation capability may be enabled/disabled in the Properties dialog box. It is enabled by default.

Updating/Regenerating the IP Core

By regenerating an IP core with the IPexpress tool, you can modify any of its settings including device type, design entry method, and any of the options specific to the IP core. Regenerating can be done to modify an existing IP core or to create a new but similar one.

Regenerating an IP Core in Diamond

To regenerate an IP core in Diamond:

1. In IPexpress, click the **Regenerate** button.
2. In the Regenerate view of IPexpress, choose the IPX source file of the module or IP you wish to regenerate.
3. IPexpress shows the current settings for the module or IP in the Source box. Make your new settings in the T**arget** box.
4. If you want to generate a new set of files in a new location, set the new location in the **IPX Target File** box. The base of the file name will be the base of all the new file names. The IPX Target File must end with an .ipx extension.
5. Click **Regenerate.** The module's dialog box opens showing the current option settings.
6. In the dialog box, choose the desired options. To get information about the options, click **Help**. Also, check the About tab in IPexpress for links to technical notes and user guides. IP may come with additional information. As the options change, the schematic diagram of the module changes to show the I/O and the device resources the module will need.
7. To import the module into your project, if it's not already there, select **Import IPX to Diamond Project** (not available in stand-alone mode).
8. Click **Generate**.
9. Check the Generate Log tab to check for warnings and error messages.
10. Click **Close**.

The IPexpress package file (.ipx) supported by Diamond holds references to all of the elements of the generated IP core required to support simulation, synthesis and implementation. The IP core may be included in a user's design by importing the .ipx file to the associated Diamond project. To change the option settings of a module or IP that is already in a design project, double-click the module's .ipx file in the File List view. This opens IPexpress and the module's dialog box showing the current option settings. Then go to step 6 above.

Regenerating an IP Core in ispLEVER

To regenerate an IP core in ispLEVER:

1. In the IPexpress tool, choose **Tools > Regenerate IP/Module**.
2. In the Select a Parameter File dialog box, choose the Lattice Parameter Configuration (.lpc) file of the IP core you wish to regenerate, and click **Open**.
3. The Select Target Core Version, Design Entry, and Device dialog box shows the current settings for the IP core in the Source Value box. Make your new settings in the Target Value box.
4. If you want to generate a new set of files in a new location, set the location in the LPC Target File box. The base of the .lpc file name will be the base of all the new file names. The LPC Target File must end with an .lpc extension.
5. Click **Next**. The IP core's dialog box opens showing the current option settings.
6. In the dialog box, choose desired options. To get information about the options, click **Help**. Also, check the About tab in the IPexpress tool for links to technical notes and user guides. The IP core might come with additional information. As the options change, the schematic diagram of the IP core changes to show the I/O and the device resources the IP core will need.

7. Click **Generate**.
8. Click the **Generate Log** tab to check for warnings and error messages.

Application Support

This chapter provides application support information for the Soft SPI4 IP core.

Hard-Core Physical Placement

Knowing the placement of various functions in the design allows for effective floorplanning. The relative effects of physical placement are included in the following discussions on IO placement and clocking and synchronization.

SPI4 Line-Side I/O

Following the standard FPGA design flow allows the user to specify whatever I/O placement is desired. However, the IP core evaluation package includes a predefined set of I/O assignments for the external SPI4 bus that have been assigned and lab tested. Assuming these I/O assignments are used, the resulting floorplan results in the receiver (S4RX) being placed roughly in the bottom right quadrant and the transmitter (S4TX) in the lower left quadrant, or left side of the devices. Similar to the SPI4 line-side I/O, the EBRs, especially line-side ones, can be adjusted to be more tightly connected with the I/Os.

According to the OIF-SPI4_02.1, the type of current for SPI4 line-side status channel I/Os is set to LVCMOS33. Depending on the vendor devices the SPI4 interface is connected with, the I/O type may need to be adjusted to LVCMOS25, or another type, if needed.

Clocking and Synchronization

This section describes the various approaches that may be taken for delivery and extraction of clock signals to/from the IP core. There are many potential clock domains that may be specified for this core depending on specific application needs. Additionally, there is a great deal of flexibility in specifying the source (I/O, PLL, flip-flops, etc.) and speed of various clock domains used. Core behavior and performance are affected by the type of and manner in which synchronization is applied to the core.

Clock List

Table 5-1 and Table 5-2 provide a comprehensive view of the total number of clock nets possible when using the SPI4 core as well as additional information about each net that may be useful when determining a synchronization strategy. Issues to consider when determining a synchronization plan include:

- There are a number of ways to consolidate clock nets and therefore clock driver resources. For example, clocks associated with the status interface such as rxstck, rxcalck, txstck, and txcalck all have top-level appearance at the user level and can all be connected to the same primary clock driver if desired or grouped in some other manner. They can be further consolidated into one of the transmit txstck_line, or receive data path divide-by-four clock nets ('rxs4ls4_ck') if desired.
- Clocks can be assigned based on quadrant only needs, freeing up clock routing resources for user functions, as shown in the table below. Lattice FPGAs provide eight primary clock drivers and four secondary clock drivers per quadrant. Therefore, there should be sufficient drivers available for user functions assuming clocks are floorplanned at the quadrant level.

Table 5-1. Clock List for LatticeECP3 Devices

#	Clock Net Name	Originating FPGA Port Name	Driver Source	Clock Driver Type	Quadrants (Left or Right)	Max. Frequency (MHz)
SPI4 Line Data Related Clocks						
1	rxs4hs_ck	RCLK[P:N]	PIO	Edge/FRC	Left or Right	350
2	rxs4ls2_ck	RCLK[P:N]	DLL	Primary	Left or Right	175
3	rxs4ls4_ck[1]	RCLK[P:N]	FPGA (ff)	Secondary	Left or Right	87.5
4	txs4hs_ck	TXREF	PLL/PIO	Edge/FRC	Left or Right	350
5	txs4ls2_ck	TXREF	CLKDIV	Secondary	Left or Right	175
6	txs4ls4_ck[1]	TXREF	PGA (ff)	Primary	Left or Right	87.5
SPI4 Line Status Related Clocks						
7	rxstck	RXSTCK	PIO/PLL	Primary	Left or Right	100
8	rxstck_line	RXSTCK	PIO/PLL	Primary	Left or Right	100
9	rxcalck	RXCALCK	PIO/PLL	Primary	Left or Right	100
10	txstck	TXSTCK	PIO/PLL	Primary	Left or Right	100
11	txcalck	TXCALCK	PIO/PLL	Primary	Left or Right	100
System Clocks						
12	rxsdck	SDCK[2]	PLL/PIO	Primary	Left or Right	200
13	txsdck	SDCK[2]	PLL/PIO	Primary	Left or Right	200

1. Used only in 128b mode
2. SDCK = 200MHz in 64b mode, ~140MHz in 128b mode.

Clock Usage Diagram

Figure 5-1 provides a graphical representation of the information presented in Table 5-1. It also shows one method for generating the 'SDCK' and 'TXS4HS_CK' clock signals through FPGA PLLs. Figure 5-2 provides similar information for Table 5-2.

Figure 5-1. Clock Usage Diagram

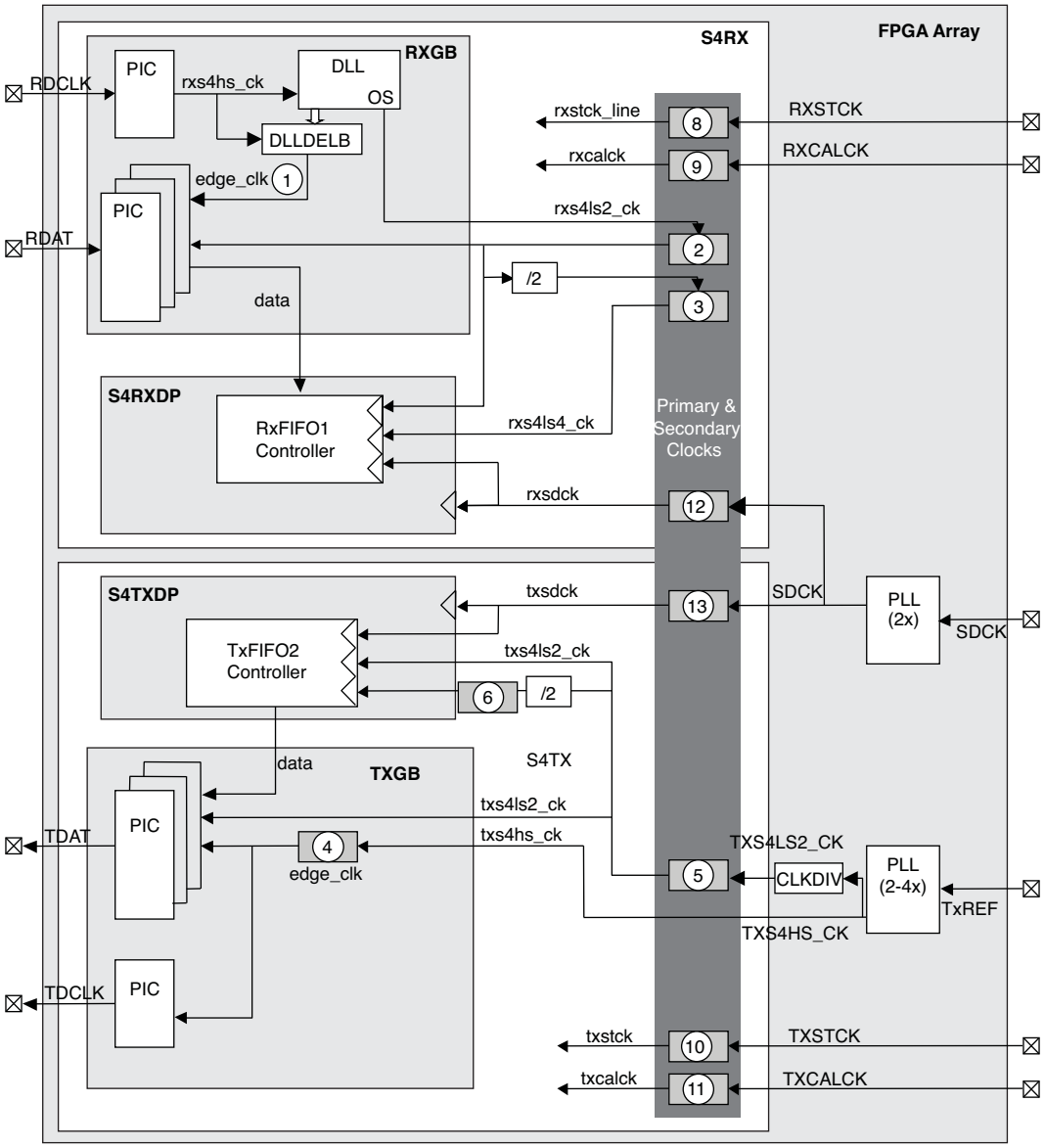
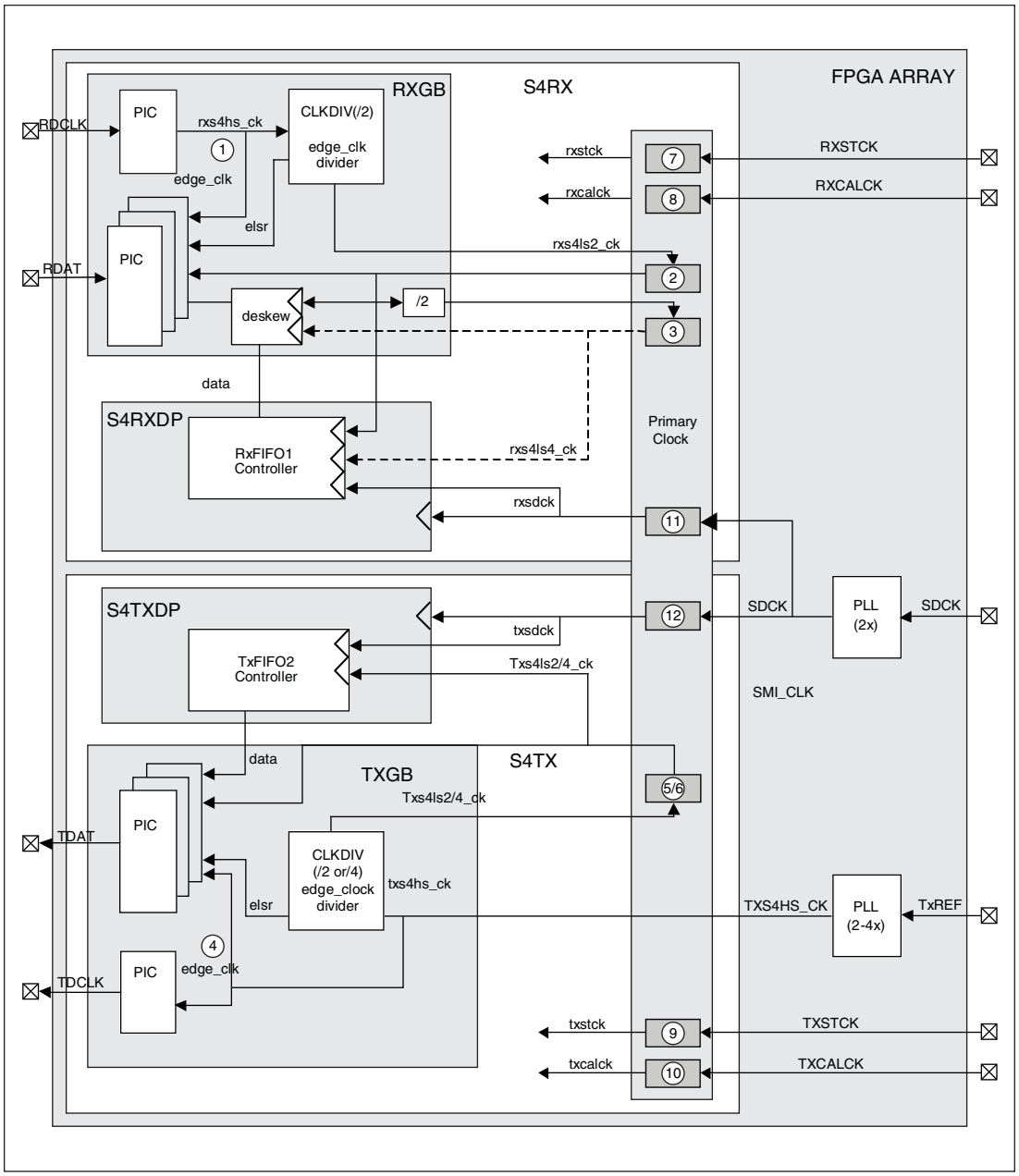


Table 5-2. Clock List for LatticeSC Devices

#	Clock Net Name	Originating FPGA Port Name	Driver Source	Clock Driver Type	Quadrants (Left or Right)	Max. Freq.
SPI4 Line Data Related Clocks						
1	rxs4hs_ck	'RCLK[P:N]'	PIO	Edge/FRC	Bottom + Top	500MHz
2	rxs4ls2_ck	'RCLK[P:N]'	CLKDIV	Primary	Bottom	250MHz
3	rxs4ls4_ck[1]	'RCLK[P:N]'	FPGA (ff)	Primary	Bottom	125MHz
4	txs4hs_ck	'TXREF'	PLL/PIO	Edge/FRC	Bottom + Top	500MHz
5	txs4ls2_ck[2]	'TXREF'	CLKDIV	Primary	Bottom + Top	250MHz
6	txs4ls4_ck[1]	'TXREF'	CLKDIV	Primary	Bottom	125MHz
SPI4 Line Status Related Clocks						
7	rxstck	'RXSTCK'	PIO/PLL	Primary	Bottom + Top	125MHz
8	rxstck_line	'RXSTCK'	PIO/PLL	Primary	Bottom + Top	125MHz
9	rxcalck	'RXCALCK'	PIO/PLL	Primary	Bottom	125MHz
10	txstck	'TXSTCK'	PIO/PLL	Primary	Bottom	125MHz
11	txcalck	'TXCALCK'	PIO/PLL	Primary	Bottom	125MHz
System Clocks						
12	rxsdck	'SDCK'[3]	PLL/PIO	Primary	Bottom + Top	250Mhz
13	txsdck	'SDCK'[3]	PLL/PIO	Primary	Bottom + Top	250MHz

1. Used only in 128b mode.
2. Used only in 64b mode.
3. SDCK = 250MHz in 64b mode, ~150MHz in 128b mode for dynamic mode.

Figure 5-2. LatticeSC (Dynamic RX Mode) Clock Usage Diagram



System-Level Synchronization

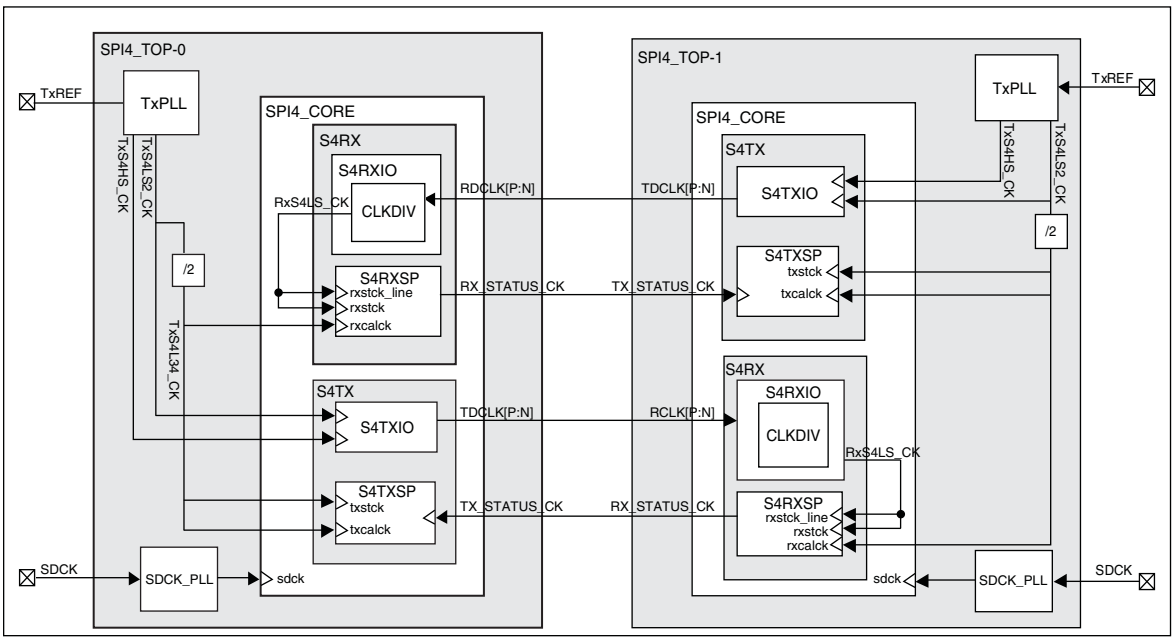
Figure 5-3 shows an example of a system-level synchronization scheme where a timing reference is supplied to an optional Transmit PLL (TxREF) at each end of the SPI4 link. This is the starting point from which system level timing can be analyzed. In this example, status is driven back to the far end using the divide-by-four version ('RxS4LS4_CK') of the SPI4 line clock. This arrangement results in status being received from the far end that is at frequency locked to the near end version of the same clock.

A slight modification to this arrangement is possible where the received divide-by-two or -four version ('RxS4LS4_CK') of the SPI4 line clock can be used as the timing reference to an internal Transmit PLL. A timing loop would be created if this is done at both ends. Somewhere in the user system there needs to be a fixed timing reference.

System-level clocking (SDCK) shown in this example also uses a PLL and a timing reference but the PLL is not required. System clocking is only used inside the device.

In the example diagram below, clocks are doubled up in terms of functionality in order to simplify and reduce the need for clock routing resources (primary/secondary) within the FPGA.

Figure 5-3. Top and System Level Clocking Example for LatticeSC/M Devices (Both SPI4 Ends)



Selecting a System Data Clock Frequency ('SDCK') - Receiver

In 128b mode, the receive Parser module reads full 136-bit data slices (128 bits of data [eight 16-bit words] and 8 bits of control [one bit per word]) from RxFIFO1 using 'SDCK'. Reading occurs continuously unless the FIFO empties (which occurs regularly due to over-speed) or when the Parser encounters an EOP within the data slice. When an EOP is detected, the Parser will stall (stop reading) RxFIFO1 for one or more clock cycles. Slices may contain multiple EOPs for different channels (up to four) or one or more EOPs and an SOP or partial data segment. A single RxFIFO2 entry cannot contain data from more than one channel, so the Parser stops reading RxFIFO1 as it separates (unpacks) the data for different channels and writes the corresponding number of words into RxFIFO2. Performing this function results in a waste of bandwidth since at the end of a packet, some number of the words written to RxFIFO2 are not fully populated. This wasted bandwidth must be compensated for and hence the need for over-speed.

In this core, 2n clock cycles are required to process a data word containing n EOPs. For example, a slice with one EOP requires two read-side clock cycles, a slice with two EOPs requires four read-side clocks, and so on, with a slice containing four EOPs requiring eight read-side clock cycles to process.

The amount of over-speed required is a function of the smallest allowable packet size, the number of active channels, the size of the FIFO to be stalled (i.e. RxFIFO1) and the stall behavior. The following discussion provides a framework upon which the appropriate frequency for 'SDCK' can be determined.

Consider this absolute extreme worst-case scenario where the SPI4 Burst Size is 64 bytes, 128 channels are equipped, and continuous 65 byte packets are received for each channel address without idle insertion between packets. Assume also that packet starts across all channels are synchronized and remain synchronized such that all channels end (EOP) at exactly the same time indefinitely. For this case, an SOP control word (two bytes) followed a 64-byte burst of data will be received at the SPI4 interface for each of the 128 channels in sequence without an EOP for any channel. A total of 66x128=8448 bytes, the SOPs and data, will be written into RxFIFO1 in 16-

byte slices, requiring 528 write clock cycles. Then an EOP control word (two bytes) followed by the remaining data byte (padded to 16 bits) will be received for each of the 128 channels in sequence. The EOPs plus final data segments will be packed 4 per 16-byte slice and will be written into RxFIFO1 in 32 write-side clock cycles. This cycle of 528 write cycles containing SOP+ data followed by 32 write cycles each containing four EOPs per slice will continue indefinitely. Thus, to prevent the contents of RxFIFO1 from building and eventually overflowing, the Parser must completely unload RxFIFO1 and process the 128 65-byte packets, one for each channel, within a period of time corresponding to 560 RxFIFO1 write-side clock cycles.

The Parser can process slices containing SOP+data segments without any additional cycles. Thus 528 read-side clock cycles will be required to process the SOP+data segments for all 128 channels. As indicated previously, the Parser requires 2n clock cycles to process a data word containing n EOPs. Thus the 128 EOPs, which were written into RxFIFO1 in 32 clock cycles, will require 256 read-side clock cycles (two per EOP) to process. During this time, 32 of the 256 clock cycles will result in actual reads of RxFIFO1. The FIFO will be stalled for the remaining 224 cycles, during which time the FIFO fill level will build.

Thus it will require a total of 560 write-side clock cycles to load 128 65-byte packets plus control into RxFIFO1 and 784 read-side clock cycles to unpack and empty RxFIFO1. To ensure that contents of the FIFO do not build over time and eventually result in flow control, the over-speed on the read side needs to be sufficient enough to empty the FIFO each 128-packet cycle. Thus, there must be 784 read cycles for every 560 write cycles, or 40% overspeed. Assuming a SPI4 line rate of 311MHz, the RxFIFO1 write-side clock would be ~78MHz. The read-side clock 'SDCK' would need to be ~110MHz to prevent RxFIFO1 from causing a flow control under this worst case scenario.

As mentioned in the previous discussion, reading from RxFIFO1 will be stalled for the equivalent of 224 read-side clock cycles, or 2.04usec, during EOP processing. The FIFO is continuing to be written while reading is stalled. With a read-side clock of 78MHz, 160 memory locations will be written. RxFIFO1 is 512 words deep, providing significant margin for this worst-case scenario.

Note that the scenario analyzed will typically never happen, let alone be sustained over time in real applications having even moderately variable packet sizes. For most applications it seems reasonable to assume that significantly less than 40% over-speed for 'SDCK' will be required. It is expected that approximately 20% overspeed should typically be more than sufficient. Note that scaling back 'SDCK' does not mean that the worst-case scenario cannot be handled, but only that it cannot be handled indefinitely without eventual flow-control. With ~20% overspeed, many cycles of simultaneous EOPs on 128 channels with minimum size packet, such as the one described above, could be properly handled before the FIFO would eventually hit the high-water mark resulting in flow-control. Note also that this worst-case scenario occurs only with the smallest packet sizes for most systems (~64 bytes). Larger packets increase the FIFO recovery time, reducing the amount of over-speed required.

Although not mentioned above, the SPI4 burst size chosen by the transmitter also affects the required clock frequency. Consider again the worst-case 65-byte synchronous packet scenario just discussed. If the SPI4 burst size is increased to 80 bytes, no partial packet segments are transmitted and multiple EOPs per slice cannot occur. With this scenario it still takes 560 write-side clock cycles to write 128 packets+control into RxFIFO1, but since there is at most only one EOP per slice, only 128 additional read-side clock cycles are required to process the 128 EOPs and the amount of over-speed required is reduced to ~23% worst case.

Note that the S4RX design can handle packets smaller than 64 bytes (e.g. 8 bytes). Just as larger packets increase the FIFO recovery time, reducing the amount of over-speed required, smaller packets reduce FIFO recovery time and coupled with non-optimal burst size settings have the potential to significantly increase the over-speed requirement before flow-control.

The preceding analysis is specific to the 128b mode of IP core operation. However, the results can be easily scaled to address the 64b mode as well. In short, when selecting a System Data Clock Frequency, the amount of overspeed necessary for the 64b mode is one-half the amount for the 128b mode. This is because the amount of wasted bandwidth per cycle is exactly one-half given that the bus cut in half.

Selecting a System Data Clock Frequency ('SDCK') - Transmitter

The Aligner in the transmit path requires over-speed for reasons that are the inverse of the receive path in order to achieve 100% utilization of the SPI4 line bandwidth. However, there are some differences, one of which is the amount of over-speed required.

In the transmit direction, only one TxFIFO1 read per EOP is required, compared to two cycles per EOP in the receive direction. Consider the same worst case scenario discussed in the previous section: 64-byte SPI4 Burst Size, 128 channels, continuous 65 byte packets sent on each channel address without idle insertion, packet transmission start on all channels synchronized such that all channels end (EOP) at exactly the same time indefinitely. When all 128 channels terminate with only a single byte valid, it takes 4 TxFIFO1 read cycles to write the TxFIFO2 output FIFO once in order to pack the line. For this design, approximately 10-15% overspeed is required to guarantee a fully utilized SPI4 line.

Note also that while inadequate overspeed on the receive side may result in flow control, inadequate overspeed on the transmit side results in the transmission of idles between segments and inefficient line utilization.

Core Verification

A summary of compliance tests for the Hard SPI4 IP core in LatticeSC/SCM is given in Lattice Technical Note TN1121, *LatticeSCM SPI4.2 Interoperability with PMC-Sierra PM3388*. The same source was used to build the Soft SPI4 IP core.

Support Resources

This chapter contains information about Lattice Technical Support, additional references, and document revision history.

Lattice Technical Support

There are a number of ways to receive technical support.

Online Forums

The first place to look is Lattice Forums (http://www.latticesemi.com/support/forums.cfm). Lattice Forums contain a wealth of knowledge and are actively monitored by Lattice Applications Engineers.

Telephone Support Hotline

Receive direct technical support for all Lattice products by calling Lattice Applications from 5:30 a.m. to 6 p.m. Pacific Time.

- For USA & Canada: 1-800-LATTICE (528-8423)
- For other locations: +1 503 268 8001

In Asia, call Lattice Applications from 8:30 a.m. to 5:30 p.m. Beijing Time (CST), +0800 UTC. Chinese and English language only.

- For Asia: +86 21 52989090

E-mail Support

- techsupport@latticesemi.com
- techsupport-asia@latticesemi.com

Local Support

Contact your nearest Lattice Sales Office.

Internet

www.latticesemi.com

References

The following documents provide more information on implementing this core:

- IPUG44, *LatticeSCM SPI4.2 MACO Core User's Guide*
- TN1121, *LatticeSCM SPI4.2 Interoperability with PMC-Sierra PM3388*

LatticeECP3

- HB1009, *LatticeECP3 Family Handbook*

LatticeSCM

- DS1004, *LatticeSC/M Family Data Sheet*
- DS1005, *LatticeSC/M Family flexiPCS Data Sheet*

Revision History

Date	Document Version	IP Core Version	Change Summary
August 2006	01.0	0.1	Initial release.
October 2006	01.1	1.1	Added appendix for LatticeECP2M devices.
August 2007	01.2	2.0	Updated appendices for LatticeECP2 and LatticeECP2M devices. Added appendix for LatticeSC/M devices.
March 2009	01.3	2.2	Updates to include asynchronous user side status interfaces.
December 2009	01.4	2.5	Updated footnotes in Appendix tables.
March 2010	01.5	2.6	Removed references to LatticeECP2M family and added support for LatticeECP3 FPGA family.
July 2010	01.6	2.6	Divided document into chapters. Added table of contents.
			Added Quick Facts tables in Chapter 1, “Introduction.”
			Added new content in Chapter 3, “Parameter Settings.”
			Added new content in Chapter 4, “IP Core Generation.”
			Added new content in Chapter 6, “Core Verification.”
September 2010	01.7	2.7	Added support for Diamond software throughout.

Appendix A:

Resource Utilization

This appendix gives resource utilization information for Lattice FPGAs using the Soft SPI4 IP core.

IPexpress is the Lattice IP configuration utility, and is included as a standard feature of the Diamond and ispLEVER design tools. Details regarding the usage of IPexpress can be found in the IPexpress and Diamond or ispLEVER help system. For more information on the Diamond or ispLEVER design tools, visit the Lattice web site at: www.latticesemi.com/software.

LatticeECP3 FPGAs

Table A-1. Performance and Resource Utilization[1]

Configuration							
Bus Mode	**Status Mode**	**SLICEs**	**LUTs**	**Registers**	**I/Os**	**EBRs**	**Line Rate (MHz)**
64	Transparent	2324	2600	3206	80	12	312
128	RAM	3967	4327	5185	80	18	350

1. Performance and utilization data are generated using an LFE3-70EA-8FN672CES device with Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance might vary when using a different software version or targeting a different device density or speed grade within the LatticeECP3 family.

Supplied Netlist Configurations

The Ordering Part Number (OPN) for the SPI4.2 IP core targeting LatticeECP3 devices is SPI-42-E3-U3.

LatticeSC/M FPGAs

Table A-2. Performance and Resource Utilization[1]

Configuration							
Bus Mode	**Status Mode**	**SLICEs**	**LUTs**	**Registers**	**I/Os**	**EBRs**	**Line Rate (MHz)**
64	Transparent	2405	5126	3001	80	12	400
128	RAM	4015	5126	4840	80	18	400

1. Performance and utilization data are generated using an LFSC3GA25E-6FF1020C device with Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance might vary when using a different software version or targeting a different device density or speed grade within the LatticeSC/M family.

Supplied Netlist Configurations

The Ordering Part Number (OPN) for the SPI4.2 IP core targeting LatticeSC/M devices is SPI-42-SC-U3.

Общество с ограниченной ответственностью «МосЧип» ИНН 7719860671 / КПП 771901001
Адрес: 105318, г.Москва, ул.Щербаковская д.3, офис 1107

Данный компонент на территории Российской Федерации

Вы можете приобрести в компании MosChip.

Для оперативного оформления запроса Вам необходимо перейти по данной ссылке:

http://moschip.ru/get-element

Вы можете разместить у нас заказ для любого Вашего проекта, будь то серийное производство или разработка единичного прибора.

В нашем ассортименте представлены ведущие мировые производители активных и пассивных электронных компонентов.

Нашей специализацией является поставка электронной компонентной базы двойного назначения, продукции таких производителей как XILINX, Intel (ex.ALTERA), Vicor, Microchip, Texas Instruments, Analog Devices, Mini-Circuits, Amphenol, Glenair.

Сотрудничество с глобальными дистрибьюторами электронных компонентов, предоставляет возможность заказывать и получать с международных складов практически любой перечень компонентов в оптимальные для Вас сроки.

На всех этапах разработки и производства наши партнеры могут получить квалифицированную поддержку опытных инженеров.

Система менеджмента качества компании отвечает требованиям в соответствии с ГОСТ Р ИСО 9001, ГОСТ РВ 0015-002 и ЭС РД 009

Офис по работе с юридическими лицами:

105318, г.Москва, ул.Щербаковская д.3, офис 1107, 1118, ДЦ «Щербаковский»

Телефон: +7 495 668-12-70 (многоканальный)

Факс: +7 495 668-12-70 (доб.304)

E-mail: info@moschip.ru

Skype отдела продаж:
moschip.ru
moschip.ru_4
moschip.ru_6
moschip.ru_9