Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

Soft SPI4 IP Core User's Guide

September 2010
IPUG59_01.7

Table of Contents

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

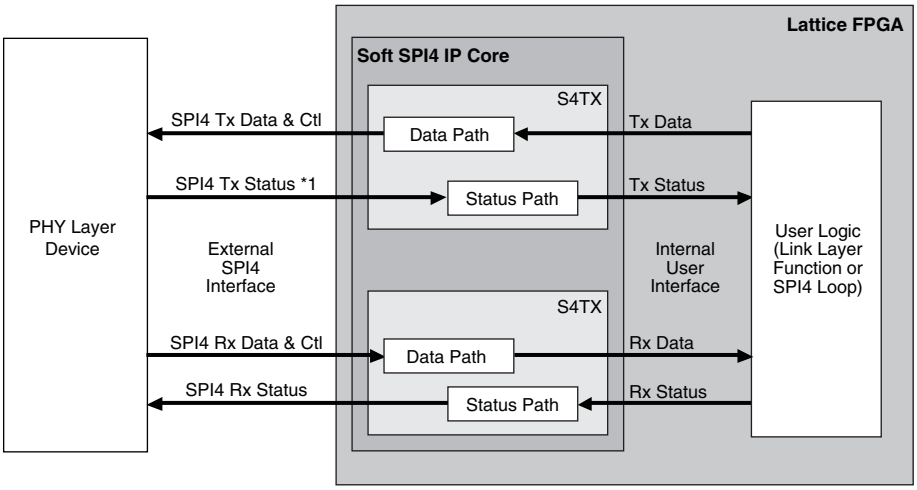
- The Soft SPI4 IP core is fully compliant with the OIF System Packet Interface Level 4 Phase 2 Revision 1 (SPI4.2.1) interface standard
- Supported through Diamond or ispLEVER IPexpress™ tool for easy user configuration and parameterization
- Supports up to 256 independent channels
- 400 to 500MHz DDR 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

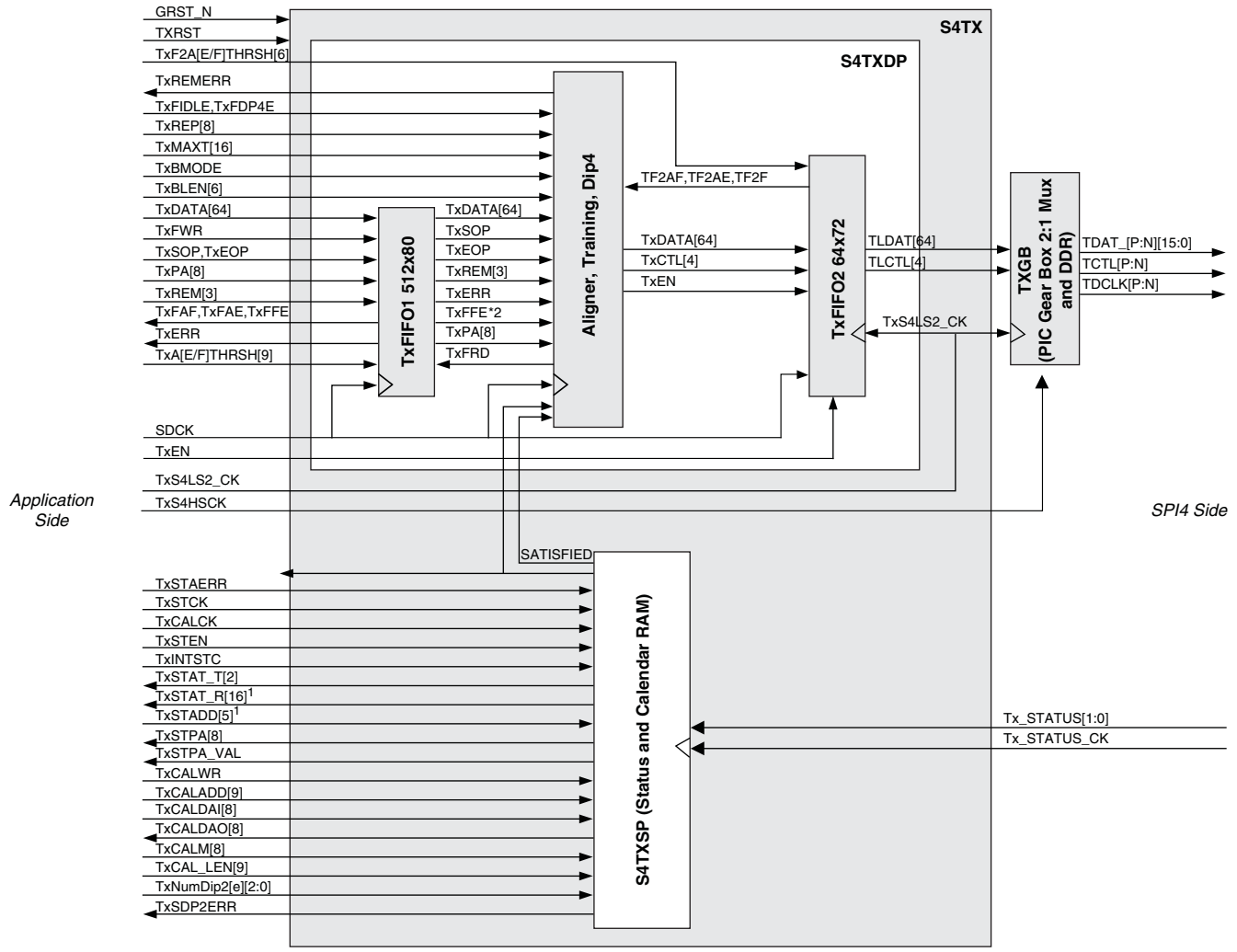
For the following descriptions, designations enclosed in single quotes (e.g. 'SDCK') refer to specific Soft SPI4 IP core I/O port names or synthesis parameters. Refer to “Signal Descriptions” on page 24 and “Parameter Settings” on page 33t for detailed descriptions of the functionality of these items.

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.

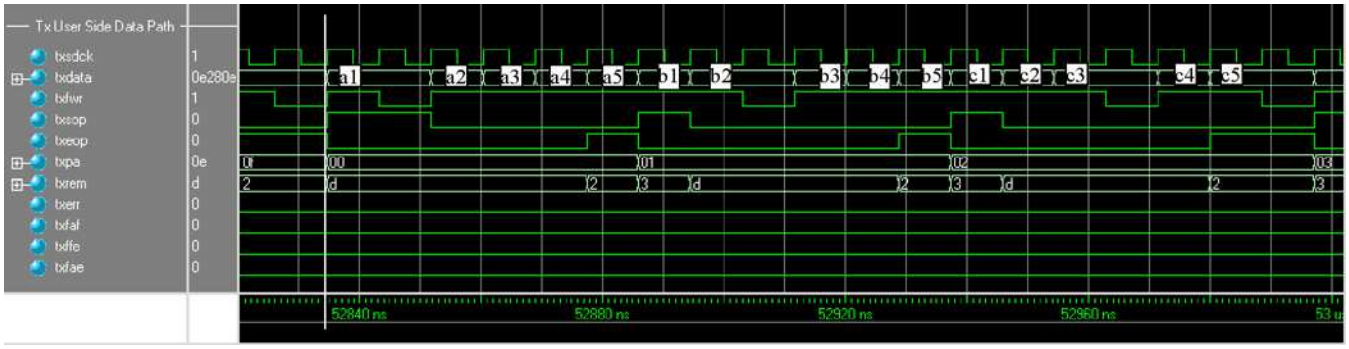
Transmit Data Timing Diagram Example

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.

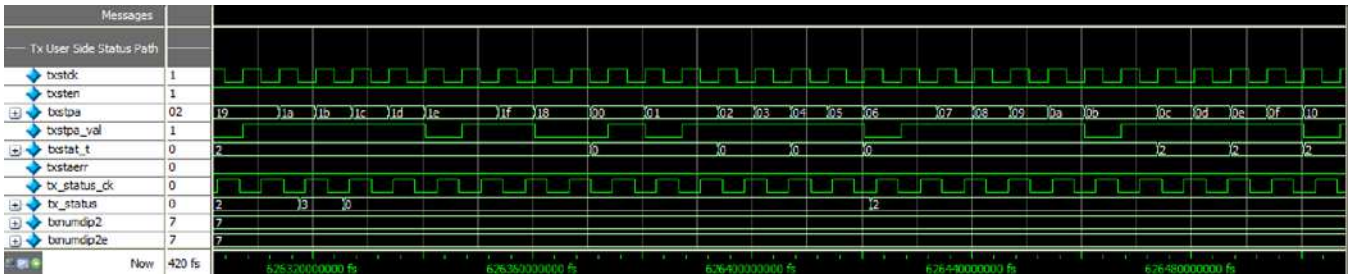
Transparent Mode

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.

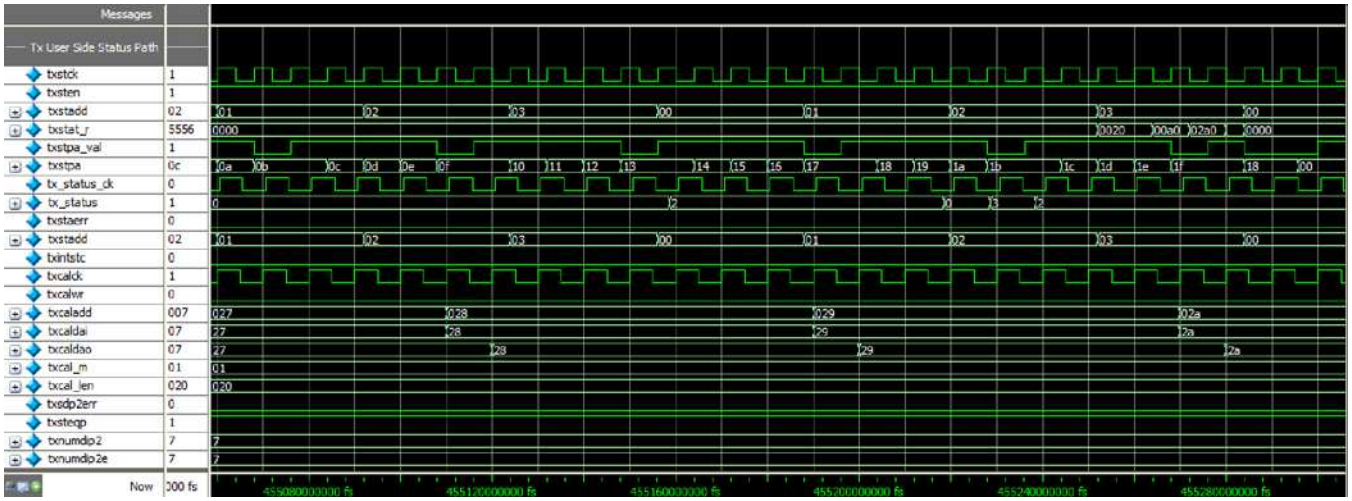
This design of the Calendar RAM interface is based on an EBR True Dual Port RAM providing the underlying memory function in which the user utilizes one port and the internals of the core utilize the other. The S4TX Calendar RAM interface is a user side “read/write” interface that operates in a synchronous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

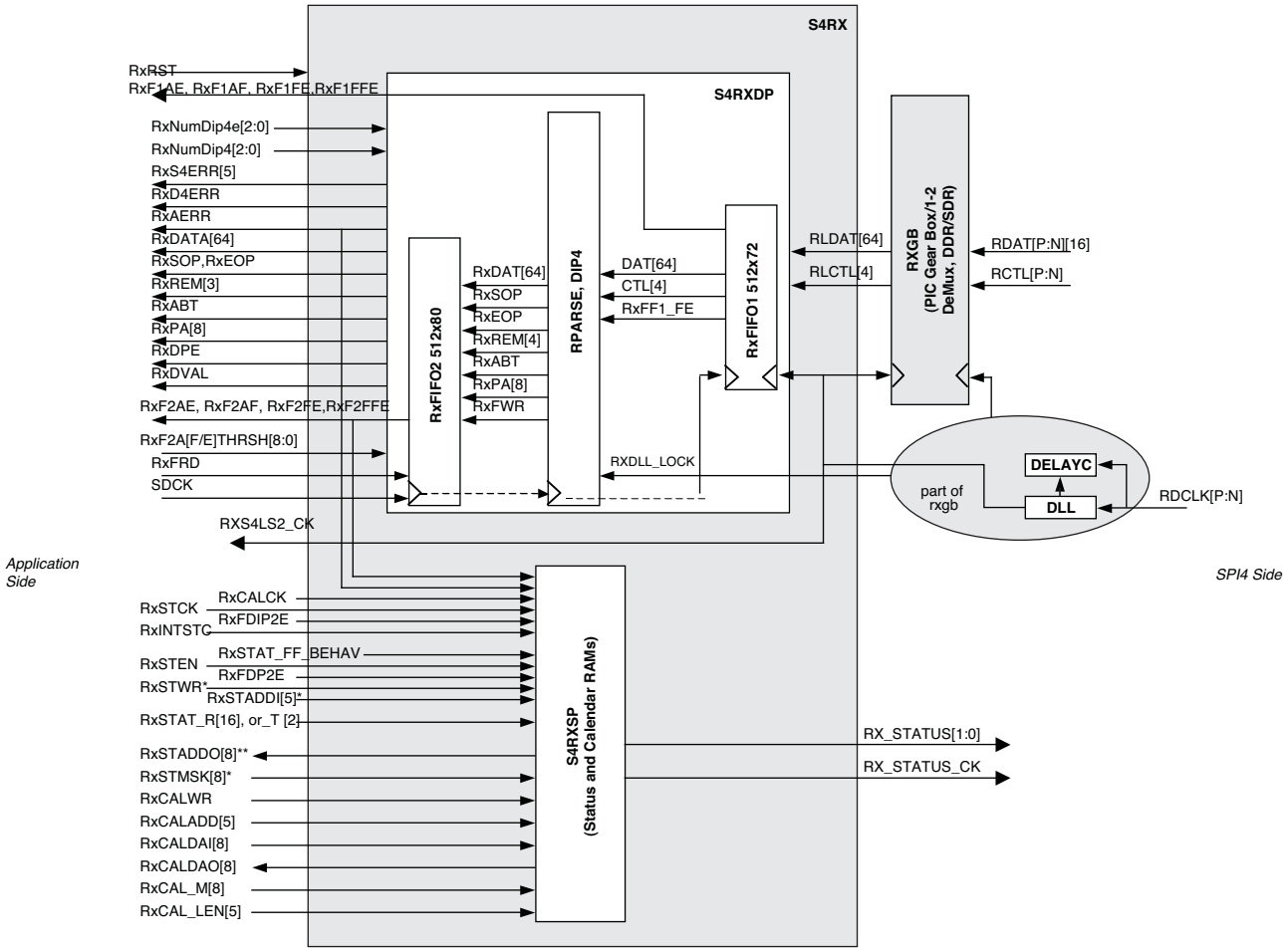
This design provides an option ('TXINTSTC'=1) to eliminate the need for user interrogation of received status. When this option is selected, transmission of data towards the SPI4 line is controlled internally through analysis of the hungry, starved, and satisfied states received on the input status path. 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.

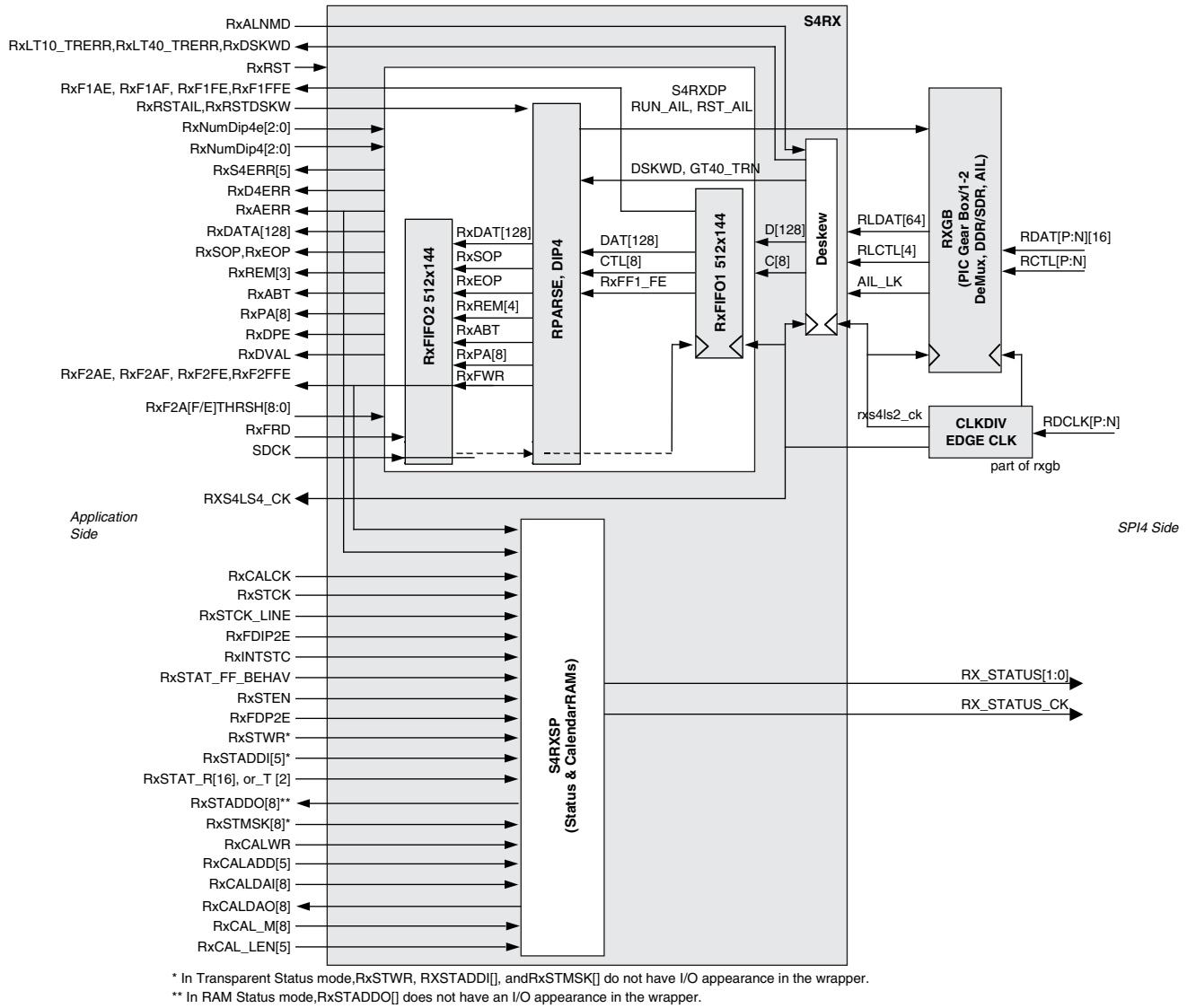
Figure 2-6. S4RX - SPI4 Receive Path



* 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.

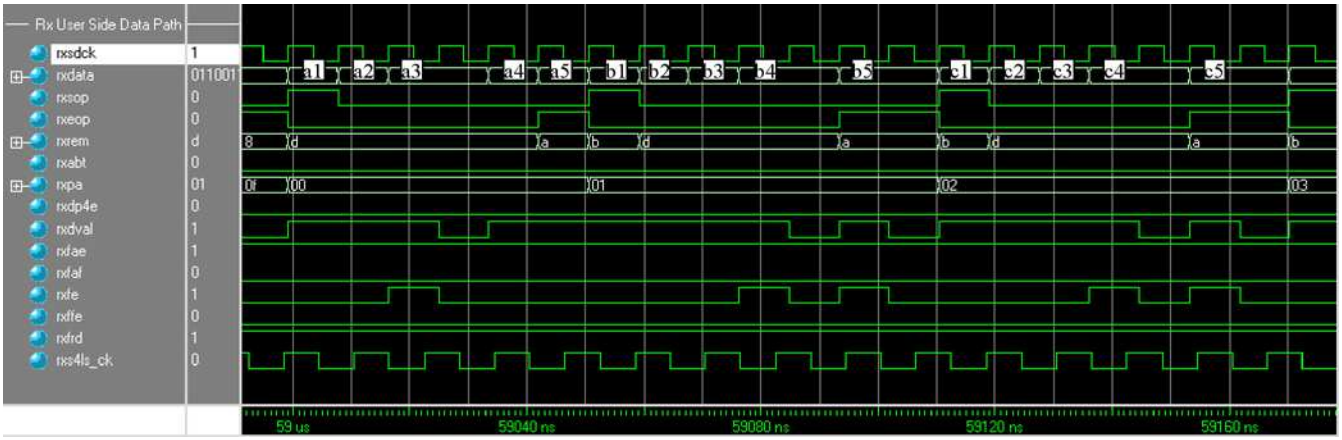
User logic monitors primarily the user-side receive FIFO Empty flag ('RxF2E') as it reads data and control information. When the empty flag is asserted, reading should be suspended until it is again deasserted. Since both the write and read side clocks are the same, once the user-side begins reading, 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-

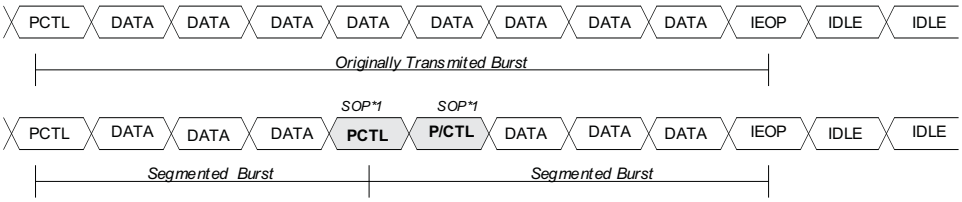
ers two distinctly different needs 1) the system/device debug effort where the user incorrectly loads two SOPs in a row into the output FIFO at the far-end for example and is quickly lead to the problem by adequate error reporting and 2) the final system where good error reporting can be used to support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

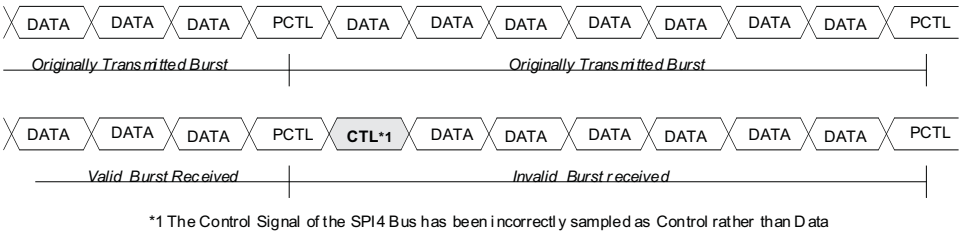
- Payload Control with one or multiple illegitimate Control Words following in the middle of a valid data segment. When this condition occurs, the first segment will fail DIP4 parity, and N number of continuing control words will be tacked on to the next segment and it too will fail parity.

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'.

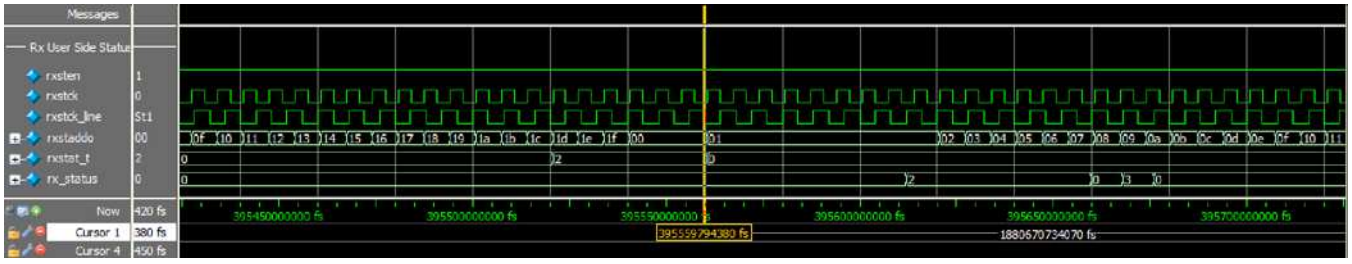
Transparent Mode
In transparent mode, the user supplies status in a 2-bit per channel bus format. This data passes through this module and to the external SPI4 status interface with out being stored in RAM. The Calendar

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

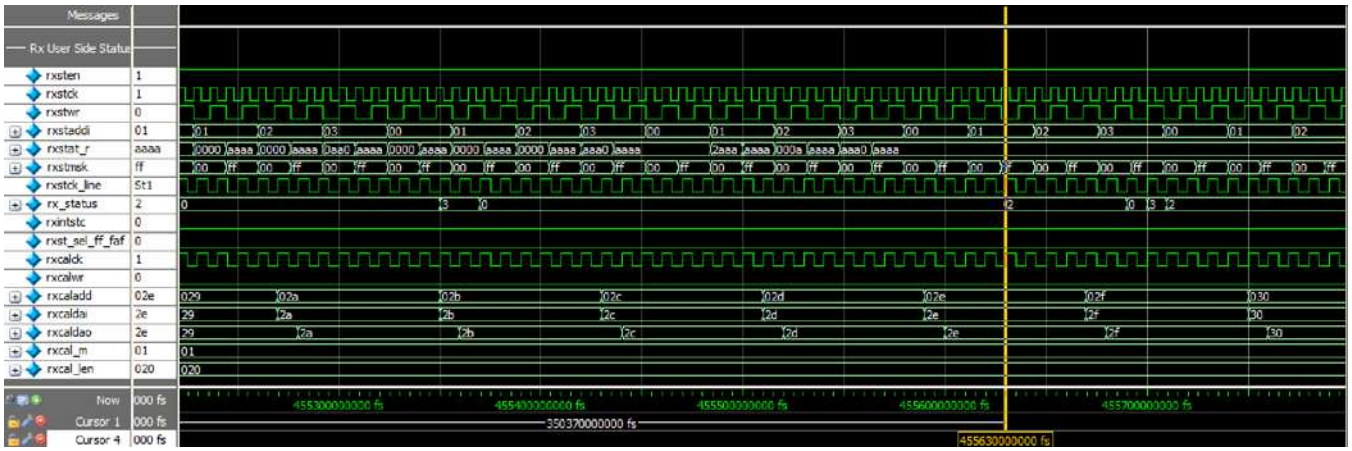
Regardless of the status mode chosen by the user to deliver status to the core, status is transmitted according to a local Calendar RAM of up to 512 locations (user accessible read/write) along with Framing and DIP2 parity information. The contents of this RAM is used to specify which channel's status 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

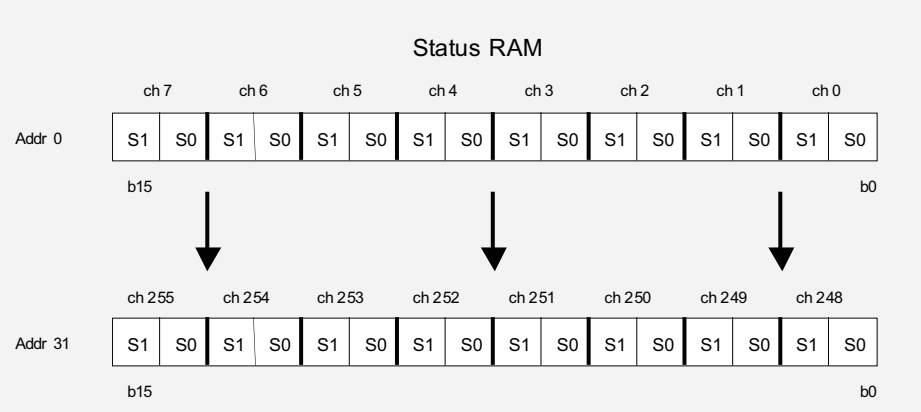
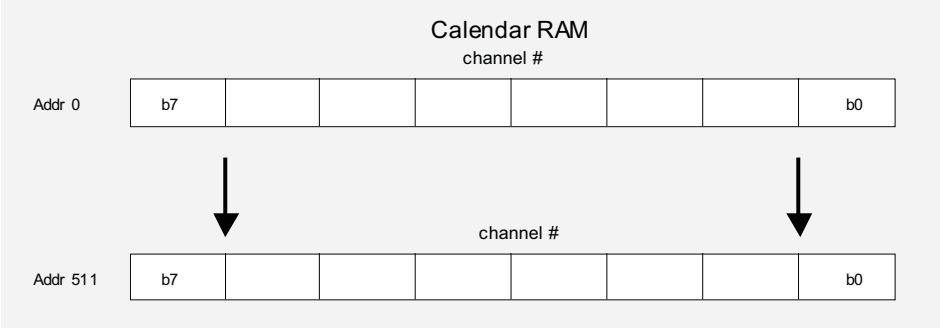


Figure 2-15. Calendar RAM Layout



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

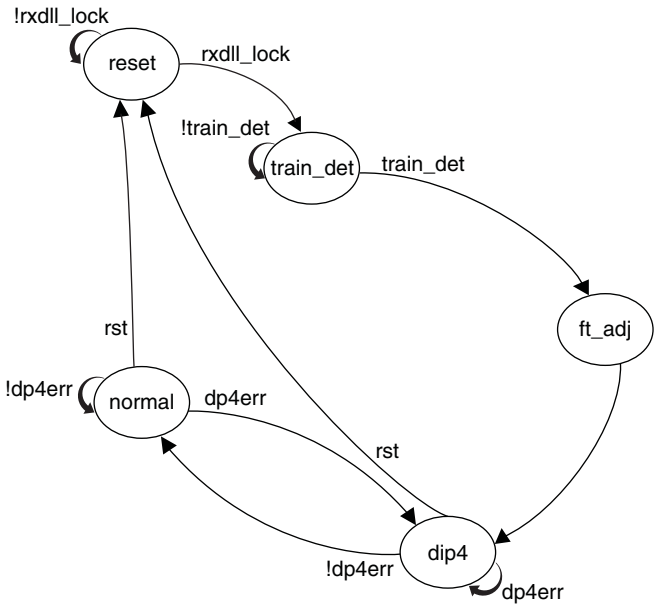
Figure 2-16 shows the Static Mode Startup and Recovery (SMSR) Finite State Machine (FSM) illustrating the receivers link start-up and recovery sequence. After a core reset, the SMSR FSM begins in the "reset" state and waits there until 'RXDLL_LOCK' is asserted. While in the "reset" state, as well as 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.