SPI4 MACO IP Core

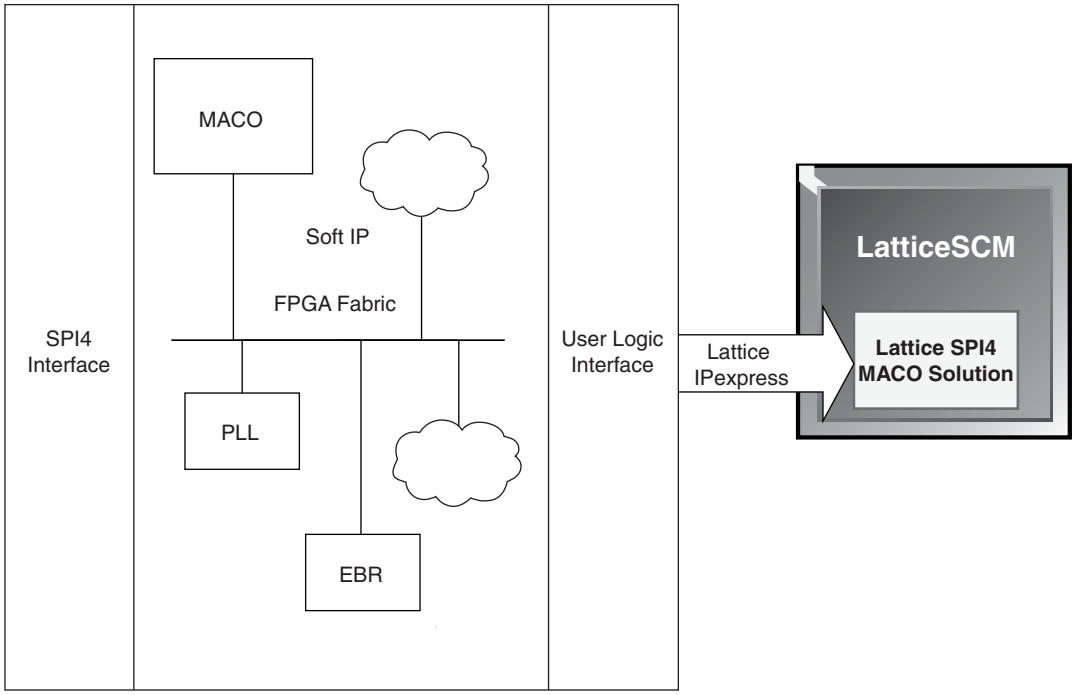
User's Guide

December 2009
ipug44_02.5

Introduction

Lattice's SPI4 MACO Core assists the FPGA designer's efforts by providing pre-tested, reusable functions that can be easily plugged in, freeing designers to focus on their unique system architecture. These blocks eliminate the need to "re-invent the wheel," by providing an industry-standard OIF-compliant System Packet Interface Level 4 Phase 2 Revision 1 (SPI4.2.1) module. This proven core is optimized utilizing the LatticeSCM™ device's MACO™ architecture, resulting in fast, small cores that utilize the latest architecture to its fullest.

Figure 1. Lattice MACO Conceptual Diagram



Complementing the Lattice ispLEVER® software is the support to generate a number of user-customizable cores with the IPexpress™ utility. This utility helps designers input design information into a parameterized design flow. Designers can use the IPexpress software tool to help generate new configurations of this IP core. Specific information on bus size, clocking, and memory device requirements are prompted by the GUI and compiled into the FPGA design database. The utility generates templates and HDL-specific files needed to synthesize the FPGA design.

IPexpress, the Lattice IP configuration utility, is included as a standard feature of the ispLEVER design tools. Details regarding the use of IPexpress can be found in the IPexpress and ispLEVER online Help systems. For more information on the ispLEVER design tools, visit the Lattice web site at www.latticesemi.com/software.

The SPI4 MACO core enables user instantiation of up to two OIF-compliant System Packet Interface Level 4 Phase 2 Revision 1 (SPI4.2.1) cores in LatticeSCM Series FPGAs. This core has been implemented using a combination of hard ASIC gates and FPGA gates where the majority of gates (>90%) will reside in ASIC, freeing up array space for user logic functions. If more than two SPI4 functions are required, the user can easily instantiate additional fully soft SPI4 IP cores having identical functionality to the SPI4 MACO core for ease of use and integration.

The SPI4 MACO core supports up to 256 data channels with aggregate throughputs of between 3 and 12.8Gb/s 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 MACO core and how it can be applied to interconnect physical and link layer devices in 10Gb/s POS, Ethernet, and ATM applications.

The SPI4 MACO core comes with the documents and files listed below:

- Data sheet
- User's guide
- RTL
- Secured RTL simulation model
- Test Bench
- Various RTL Core instantiation templates
- Evaluation simulation environment

Features

- SPI4 core fully compliant with the OIF System Packet Interface Level 4 Phase 2 Revision 1 (SPI4.2.1) interface standard.
- Supported through ispLEVER IPexpress for easy user configuration and parameterization.
- Multiple SPI4 IP core support per device.
- Supported by System Bus and Serial Management Interface (SMI) for in-circuit controllability.
- Up to 256 independent SPI4 channels.
- "Shared" or "Per-Channel" Buffer Manager (see Features below) modes.
- 156 to 1000MHz DDR Dynamic timing mode operation. Supports non-standard "SPI4 Lite" line rates.
- Requires only 865 slices for a full 256-channel "shared" buffer Dynamic mode core (multi-channel burst interleave mode requires extra logic).
- Supports full bandwidth utilization of the SPI4 line in both directions. Requires no Idle cycles in the receive direction nor 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:
  - Individual receiver/transmitter resets
  - De-skew only reset, AIL only reset
  - 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 (Min-Burst mode) 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.
- Manageable 128-bit, 150MHz user-side FIFO interface for ease of integration.

- 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 when in "shared" buffer mode (status is self managed in "per-channel" buffer mode).
- LVTTL or LVDS Status Channel I/O independently configurable in the Receive and Transmit directions.
- Rising or falling edge selectable Status Channel I/O independently configurable in the Receive and Transmit directions.
- Per-Channel Buffer Manager Features:
  - Up to 256 SPI4 data channels
  - Up to 16 physical FIFOs per transmit / receive direction
  - Parameterizable packet error drop
  - Graceful packet over-flow drop
  - Missing sop & eop error checking with optional correction for non-error drop mode
  - Asynchronous FIFO mode of operation
  - Independent user side FIFO clocks per-channel (N write clocks - input, N read clocks - output)
  - Supports both store & forward and cut-through operation (EOP counter per channel).
  - Independent user side FIFO input/output buses per-channel
  - Parameterizable independent buffer depth per transmit and receive direction (8K, 16K, and 32K)
  - Per channel empty, almost empty, full, and almost full status. Programmable almost empty and almost full thresholds per channel.
  - Programmable RAM Sequencer style transmit Scheduler providing per-channel bandwidth management. Independent of SPI4 calendar/status operations.
  - Programmable MaxBurst1 and MaxBurst2 parameters are provided on a per channel basis.
  - Dynamic channel provisioning/re-provisioning.
  - Receive aggregation through programmable channel mapping - any SPI4 channel (0-255) or number of SPI4 channels to any buffer.
  - Supports the complete 8-bit SPI4 channel address at each transmit buffer input allowing for multiple channels to be passed through a single buffer. Also allows the same channel to be passed through multiple buffers in support of multiple queues (QoS) per channel features. Both logical and physical channels can be supported.
  - Min-Burst mode for Network Processors (see below).
  - Self-managed SPI4 status interfaces based on per-buffer fill levels.

Errata

- The Packing feature referred to throughout this User's Guide can not be supported for multi-channel applications. However, the core does a good job of packing the SPI4 line even without this feature enabled as discussed in detail in Appendix A.
- The value of `TxMAXT specifying the number of repetitions of the training sequence that the SPI4 transmitter shall send must be set to an even number quantity. Odd numbered values are not supported.

Getting Started

Requirements to implement a MACO core include:

- MACO design kit
- MACO license file

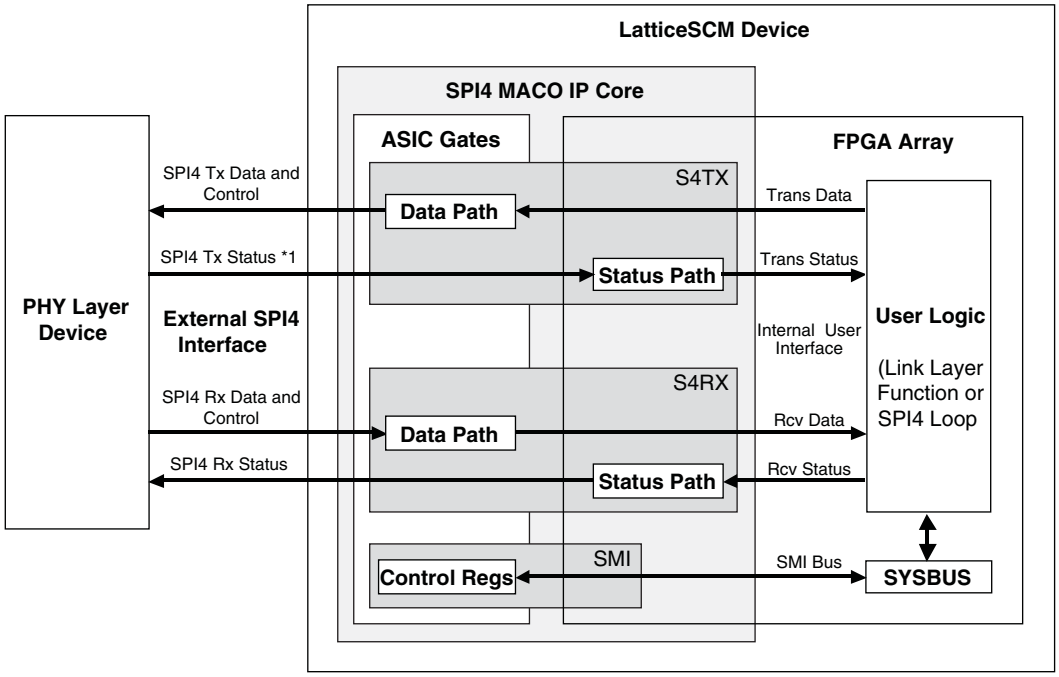
For information on obtaining the above requirements, please contact your local Lattice Semiconductor sales representative.

System Level Block Diagram

Figure 2 shows a system-level diagram of a typical Link layer application where the SPI4 MACO core is implemented in a LatticeSCM series FPGA. This figure depicts the SPI4 IP core configured for "shared" buffer mode (See Appendix B for operational detail in the "per-channel" buffer mode of operation). At the top level, the overall IP core is broken into three different blocks referred to as the SPI4 Transmitter (S4TX), SPI4 Receiver (S4RX), and Serial Management Interface (SMI). The S4RX and S4TX blocks provide data path functionality and are made up of both ASIC and FPGA gates. They provide a direct interface to primary I/O of the device on one side (SPI4) and a device internal FIFO interface to user logic on the other. The SMI block is common to both data path blocks and provides a real-time control interface for configuring core behavior. In this context, the ASIC section can be viewed as a stand-alone ASB block/component capable of providing only interior level SPI4 data functions surrounded by standard programmable logic. It is instantiated at the SPI4 MACO top level where connections to the I/O, FPGA based Status blocks, and EBR memory remains flexible and follow the standard FPGA design flow.

Also included in this IP core is a user-side SPI4 "loop-around module" and 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 overall top-level rtl template design provided (several examples), including the loop around module, 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 LatticeSCM device via SPI4 interconnect and easily verify the speed and functionality of core.

Figure 2. SPI4 MACO Core - System Level Context



Overview

The SPI4 MACO core is used with additional user-side application logic that interfaces with the IP core via separate receive and transmit FIFO interfaces (128-bit data plus control). In Shared buffer mode, the data FIFOs (8K Bytes) are implemented using Embedded Block RAM (EBR) and provide shared channel buffering on an SPI4 line basis; there is no per-channel buffering within the core in this mode. 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 trans-

mitting 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.

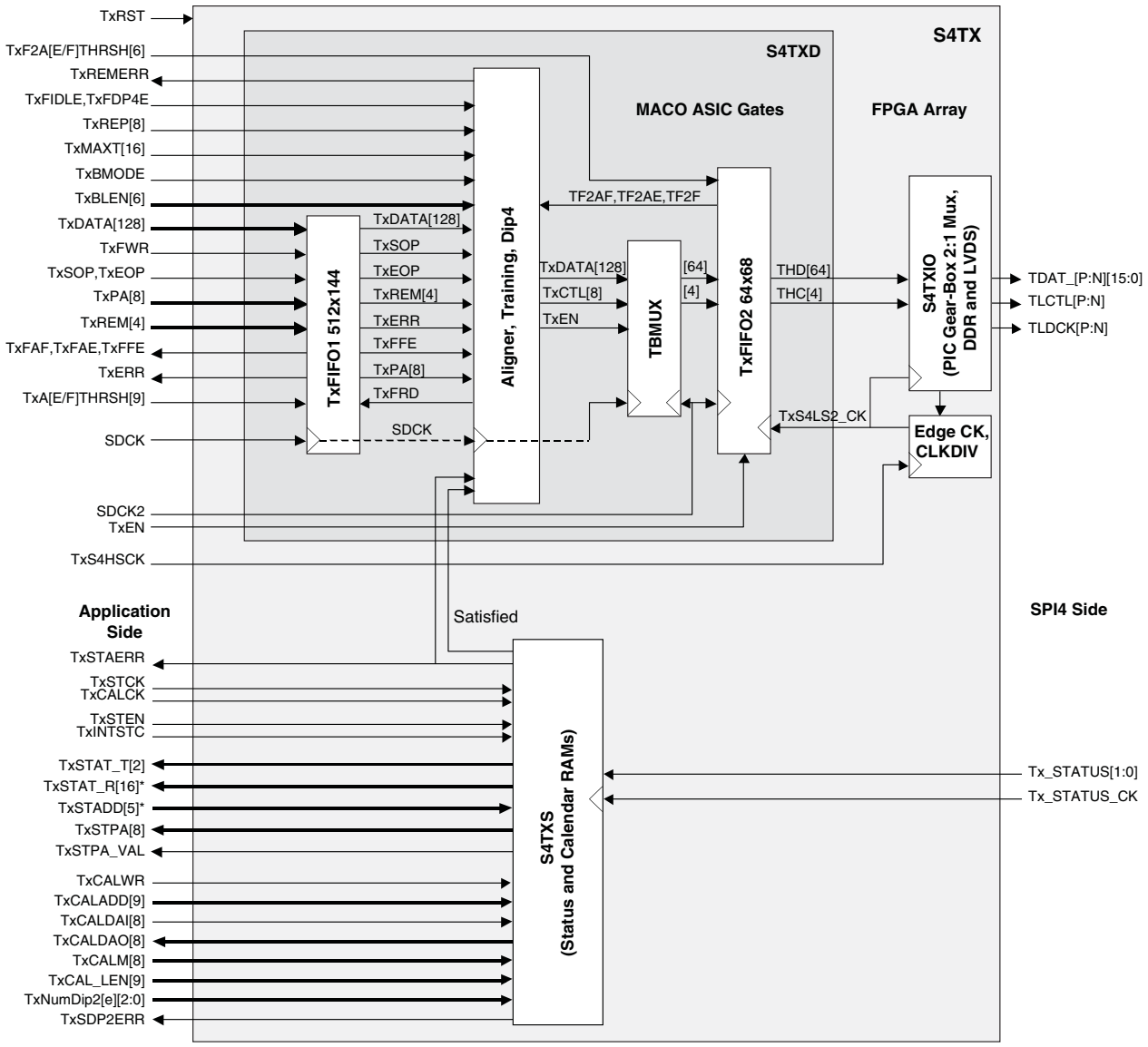
Operational Description

For the following descriptions, designations enclosed in single quotes (e.g. 'SDCK') refer to specific SPI4 MACO core I/O port names, internal memory based registers, or synthesis parameters. Please refer to the associated section (e.g. Signal Descriptions, SMI Memory, and Parameter Descriptions) later in the document for detailed descriptions on the functionality of these items. With regard to timing diagram examples, there are a number of other simulation scenarios all of which cannot be captured here but are available for user simulation and viewing through evaluation simulation (see "Functional Simulation under ModelSim" for a list of simulation scenarios available).

SPI4 Transmitter - S4TX

The transmit path, shown in Figure 3, 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 3 indicates through shading the boundary between ASIC and FPGA components and identifies three distinct sub-sections of the S4TX block as follows:

Figure 3. S4TX - SPI4 Transmit Path



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

SPI4 Transmit Data - S4TXD

The SPI4 Transmit Data (S4TXD) block is contained entirely in ASIC gates except for the memory used in support of the user side transmit FIFO (TxFIFO1) which is implemented in standard FPGA EBR. The SPI4 line side FIFO (TxFIFO2) memory is located in ASIC gates and does not subtract from available FPGA memory.

In this direction, the S4TXD 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 20% greater than the equivalent line side FIFO rate at 128 bits wide in order to carry out a "packing" operation. Packing is required for all cases where the end-of-packet byte does not result in a fully valid 128-bit FIFO entry. In this case, there is SPI4 line bandwidth available that can only be taken advantage of through over-speed here 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 8K bytes organized as 512 locations x 144 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 synthesis programmable as well as real-time programmable via the SMI interface 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.

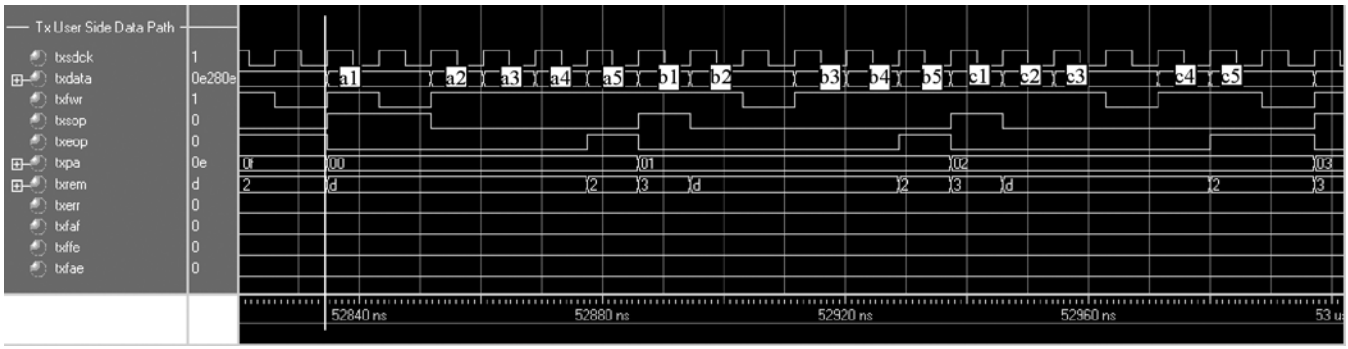
The Aligner formats data read from the user side FIFO into SPI4 control word encapsulated data segments and/or whole packets and writes the data into the line side FIFO. The Aligner monitors the user side Transmit FIFO Empty ('TxF1E') signal, reading data and control information when TxF1E is de-activated, 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 512 bytes organized as 64 locations x 72 bits. A detailed description of SPI4 core clocking and synchronization is given in a subsequent section of this document. The line side FIFO is also protected with four bits of parity generation and checking (see signal description for 'TxF2PERR') in order to ensure data integrity.

Transmit Data Timing Diagram Example: Figure 4 shows the transmission of three 67Byte full packets for channels 0, 1, and 2 over the S4TX transmit user FIFO interface. The interface operates in a synchronous fashion based on the user supplied 'sdck' 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 user side 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 6th 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 wanted 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 affects 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 pipe-lines, and partially the line side FIFO, to fill with Idle control words increasing the latency of the next burst and decreasing the over-all 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 4. S4TX User Data Interface Example



SPI4 Transmit I/O - S4TXIO

The S4TXIO block provides 2-1 gearing/multiplexing, SDR to DDR conversion, and LVDS buffer functions for the transmit data direction. All of these functions are performed at the PIC level and therefore do not require any PLC logic resources. Data (64 bits) and control (4 bits) are received from the S4TXD block through TxFIFO2 at the low 1/2 clock speed rate and 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 high-speed line rate clock is provided by the user and can be generated from an internal PLL or received via primary I/O (see the Clocking and Synchronization section of this document). The high-speed clock is divided by two internally and used in gearing operations.

Minimum Burst Size - Burst Mode: Burst Mode is essentially enabled all of the time 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 SMI memory array 'TxBLEN[5:0]', where a value of one results in standard SPI4 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 TxFIFO1 Almost Empty Burst Threshold ('TXF1AEBTHRSH[]') must be set to a value of at least 2 greater than the burst size. 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. In addition, the TXF1AFTHRSH should be set to a value of at least 32 greater than TXF1AEBTHRSH when using burst mode. Parameters 'TxBLEN[]' and 'TXF1AEBTHRSH[]' are set during the user GUI capture phase and can also be modified in real-time via the SMI interface.

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 'SDCK' 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 set during synthesis but can be programmed in real-time as well using the SMI interface.

Transmit FIFO2 Threshold Optimizations: There are three user programmable thresholds associated with the transmit line side FIFO of which, two 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 the SMI interface. The default values found during GUI capture setup for the compromise selection.

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

The absolute minimum value of 'TXF2AETHRSH' before under-run occurs is dependent upon the number of channels, amount of over-speed, packet size, and the value of the Transmit Almost Full Threshold ('TXF2AFTHRSH'). Applications with low channel count and reasonable over-speed can typically run with values as low as 10. 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 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 (+1) 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 greater the fill level in the FIFO the longer the period of potential packing. 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 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 - S4TXS

The SPI4 Transmit Status (S4TXS) block provides the entire SPI4 status function for the transmit direction. Note that though this a Transmit data function, Status information is actually received and is therefore an input to the core and to the user logic.

It provides a stand-alone status reporting function, for the most part, between the SPI4 line and the user. The only connections between the S4TXS 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 S4TXS is properly framed to the incoming status (if Training and Ctl is not enabled, only idles are sent until framed up) and some composite status information for internal status mode.

The S4TXS 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 should 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 S4TXS 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 S4TXS framer must go through the re-frame procedure in order to return to alignment. See also section Status and Calendar RAM Layout.

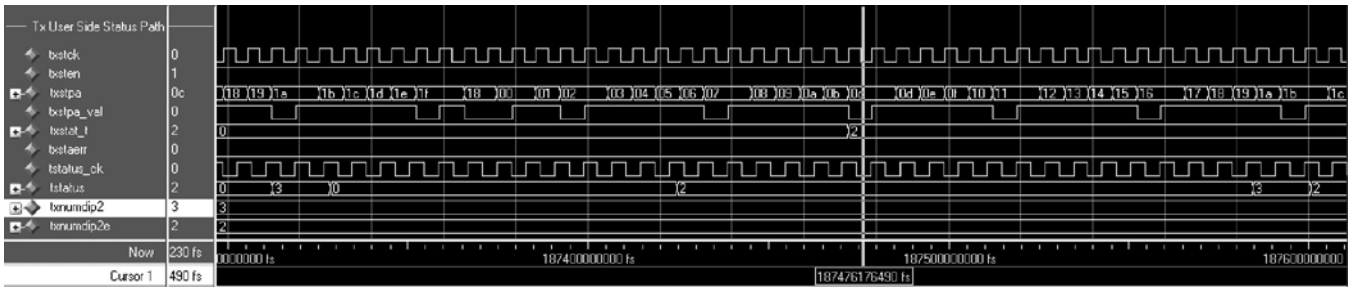
Two different synthesis-selectable configurations for reporting user status are supported: "RAM" mode and "Transparent" mode. In either mode, status and all user side status related signals are provided to the user synchronously via the user supplied 'txstck' clock signal, which can be asynchronous relative to the SPI4 transmit status clock ('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 5 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 ('tstatus[]') 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 'tstatus_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 ('tstatus_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 'tstatus_ck' are operated using the same clock, only two cycles per status frame (DIP2 & Framing) will be invalidated.

Figure 5. S4TXS - 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 6, 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 'tstatus_ck'. It must however, be equal to the frequency of 'tstatus_ck' or greater but not less than 'tstatus_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 per read operation 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 182,860ns the external input status channel ('tstatus[1:0]') reflects a constant Starved (0x0) indication for all channels and hence 'txstat_r' reflected a constant value of 0x0000. After this time, tstatus[1:0] transitions to the Satisfied state (0x2). Counting 'tstatus_ck' clock cycles back from the DIP2 and Framing bits (0x3) on this bus, the transition is shown to occur on channel 20. Looking now at the user side 'txstat_r[]' bus, channel 20 is the first channel to reflect the new Satisfied status. Note that 'txstadd[1:0]' is equal to 2 (8 channels/address) meaning that the channels reported for this address are channels 16-23. Some of the delay that is incurred between the external and internal user side interfaces is due synchronization that must take place between 'tstatus_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

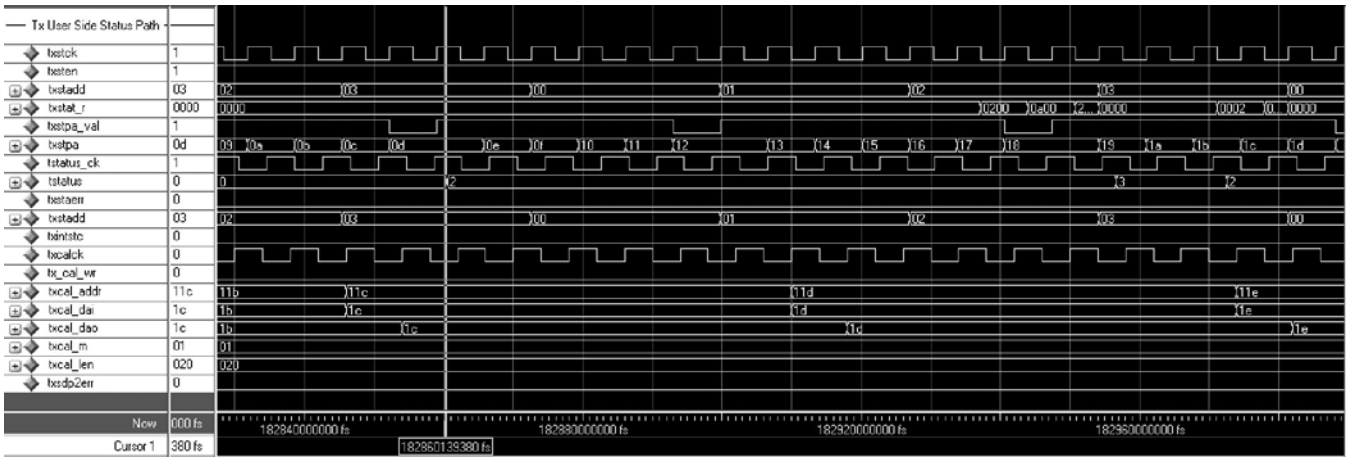
RAM is linear and synchronized to the framing contained within the status channel. For systems where the channels are not symmetrical in terms of bandwidth, the same channel can be programmed into multiple locations in the Calendar RAM resulting in multiple and more frequent status updates per status 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 description of the Calendar RAM interface is based on an EBR based 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 8 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. 0x11c w/0x1c). 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 6. S4TXS - RAM Mode Status and Calendar RAM Interface



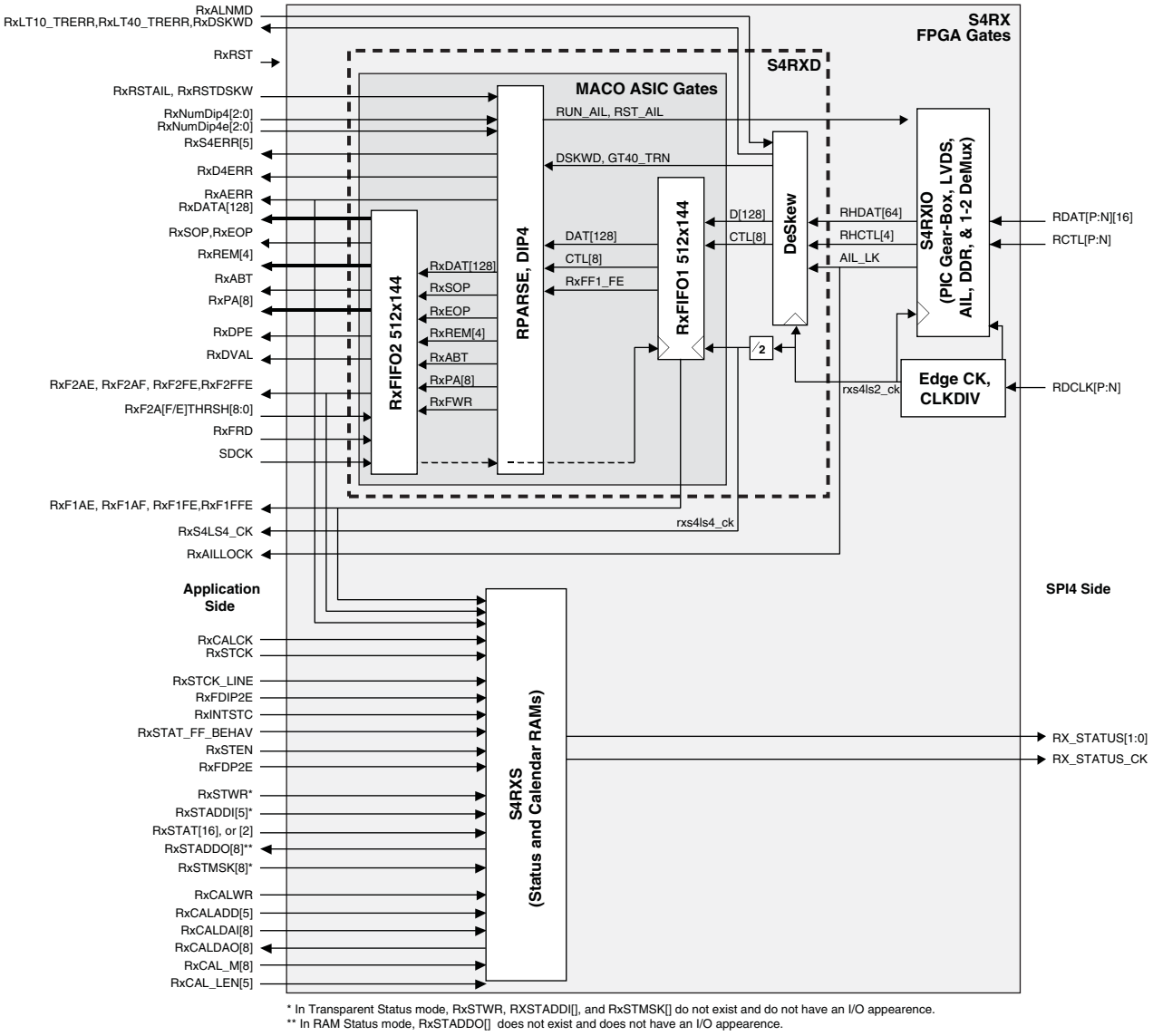
Internal Status Control: This design also provides an option ('TXINTSTC'=1) to eliminate the need for user interrogation of received status. When this option is selected, transmission of data towards the SPI4 line is controlled internally through analysis of the hungry, starved, and satisfied states 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 again observed 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 could 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 status in order to eliminate the unused Status RAMs resulting in the smallest design possible.

SPI4 Receiver - S4RX

The receive path, shown in Figure 7, 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 7 indicates through shading the boundary between ASIC and FPGA components and identifies three distinct sub-sections of the S4RX block as follows:

Figure 7. S4RX - SPI4 Receive Path



* In Transparent Status mode, RxSTWR, RXSTADDI[], and RxSTMSK[] do not exist and do not have an I/O appearence.
** In RAM Status mode, RxSTADDO[] does not exist and does not have an I/O appearence.

SPI4 Receive Data - S4RXD: The SPI4 Receive Data (S4RXD) block is contained mostly in ASIC gates with the exception of the actual memory used in support of both FIFOs (EBR memory only - FIFO controllers are located in ASIC gates) and logic required to support the deskew module which is implemented in standard FPGA.

The SPI4 receiver 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 affects 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 (8 SPI4 words wide) 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 proto-

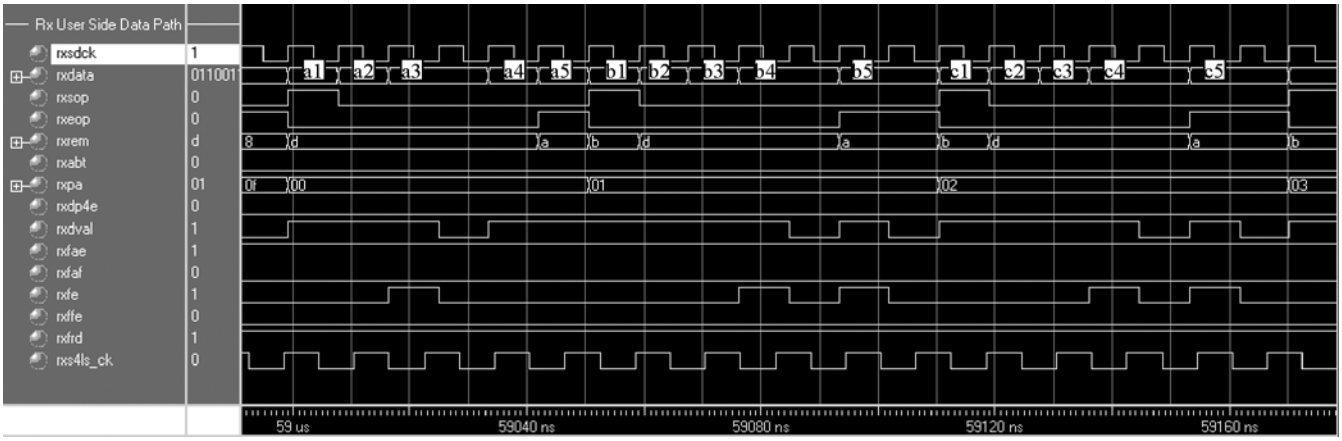
col, format, and DIP4 parity. As control and data is received from the line, error status (good/bad) is reported to the user interface through core I/O and used internal to the core in determining whether to declare the SPI4 line in-alignment or out of alignment based on user programmable error thresholds (see also section Link Start-Up Procedures). 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 should monitor primarily the user 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 de-asserted. 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 section SPI4 Receive Status). Thresholds for almost empty and full flags are set through the GUI capture phase but can also be set in real-time through the SMI interface.

Receive Data Timing Diagram Example: Figure 8 shows the reception of three 75Byte 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 affects of the over-speed are apparent very early in the transfer in that 'rxdval' is used to invalidate one of the 6 clock cycles associated with this transfer (note also the assertion of 'rxfe' - user side FIFO empty). In the 6th 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 2nd and 3rd packet transfers are similar to the first except that different cycles are invalidated by 'rxdval'.

Note that a read of an empty FIFO is tolerated but the user needs to disqualify data based on 'rxdval' as mentioned above. In this example, 'rxfrd' is held active constantly and data 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 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 considers 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.

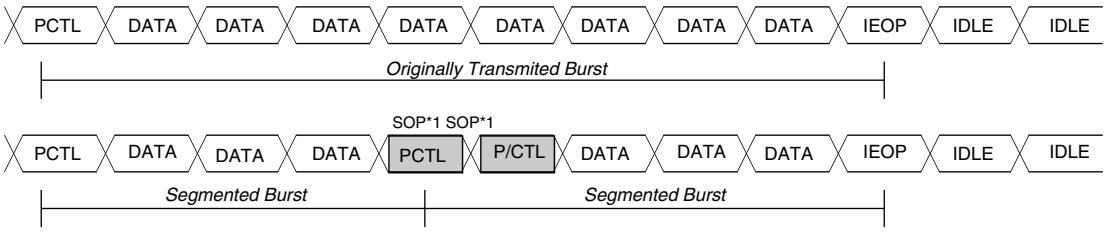
2. The final system where good error reporting can be used to support features like "packet drop" and improved recovery speed.

- **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 2nd and remaining control words, if present, are treated as data starting the continuation burst of the payload control word (see Figure 8) 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 figures 8 and 9 below, 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 affects 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 all together rather than to fill and possibly over-flow the receive FIFO with many zero length entries. There are several cases to consider:

1. 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.

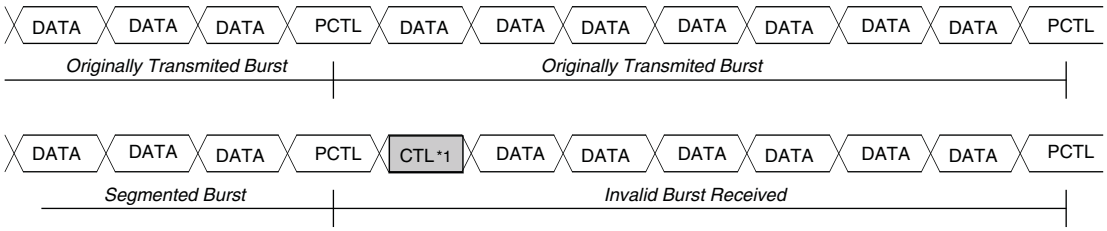
Figure 9. Multiple Illegitimate Control Words In a Data Segment



*1 The Control Signal of the SPI4 Bus has been incorrectly sampled as Control rather than Data

2. 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 10. Multiple Illegitimate Control Words At Burst Boundary



*1 The Control Signal of the SPI4 Bus has been incorrectly sampled as Control rather than Data

For cases similar to the above but no Payload Control words are received (multiple control words where b15=0), will be 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 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 1 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, eop, abt b15=0) is considered an 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 would apparently have to either recover the entire link given that there is no valid address identification with this error or ignore it allowing 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 without 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 - S4RXS

The SPI4 Receive Status (S4RXS) 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 for the most part having 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 ('RxSTAT_FF_BEHAV'= 0 or 1) whether the Almost Full, or Full signal is used to cause the over-ride 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 over-ride 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'.

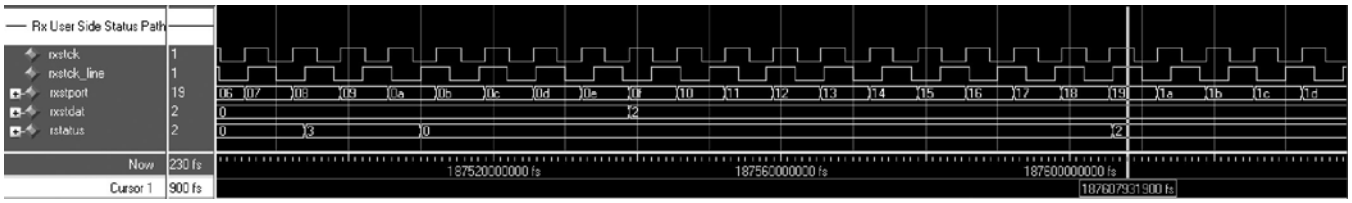
Transparent Mode: In transparent mode, the user supplies status in a 2-bit per channel bus format. This data passes through this module and to the external SPI4 status interface with out being stored in RAM. The Calendar

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 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'). The core does retain the Calendar RAM and is therefore the controller in terms of determining which channel and at what time it's status is to be transmitted. The core provides the user with an 8b address ('rxstport') informing which channels status is needed. A fixed relationship exists between a change in address by the core and when the expected status (3 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 'rxstport' address output 0xb (note: 3 'rxstck' clock cycle delay). The affected output channel can be identified by counting status slots using 'rxstck_line' clock from the framing marker of "3" equal to 0xb or 11 clock cycles away in the status sequence.

Figure 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 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 'rxstadd[4:0]' and a 16b status/(data) bus 'rxstdat[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 183,080ns, 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 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 'rstatus[]' for all channel as shown.

After this time, the Satisfied state for all channels (again 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 value is location 0x2 corresponding to channels 20-23 (8 channels per location). This write takes place in time such that the new status for channel 20 of the 8 channel group makes it out on the external 'rstatus[]' interface before the next full status cycle starts. Using 'RxSTCK_LINE', a count of 20 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.

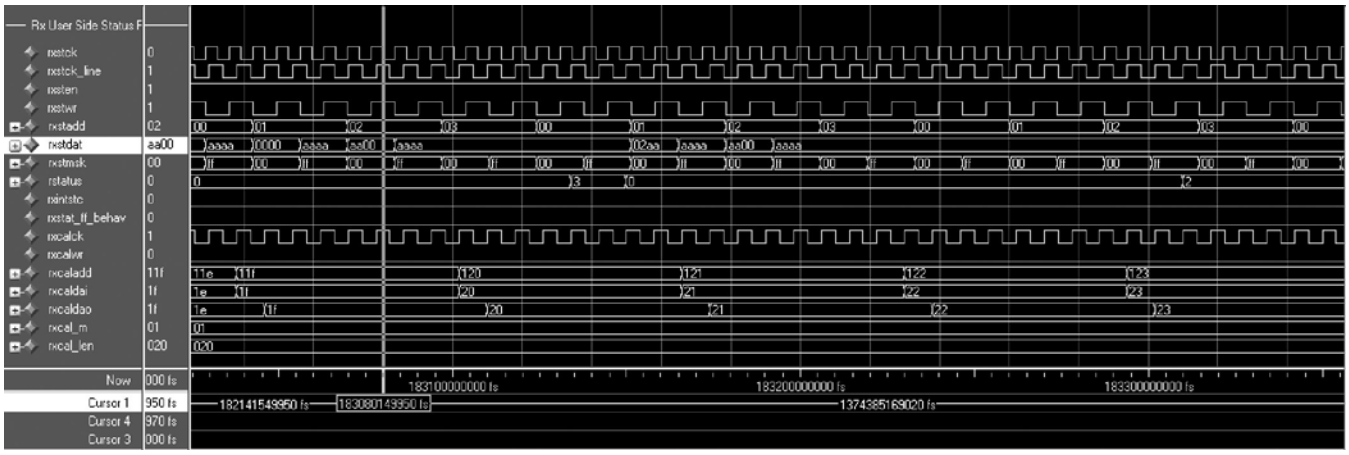
The Calendar RAM interface is based on a True Dual Port RAM in which the user utilizes one port and the internals of the core utilize the other. The Calendar RAM interface is a user side "read/write" interface that operates in a synchronous fashion based on a rising edge active user supplied 'rxcalck' 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 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.

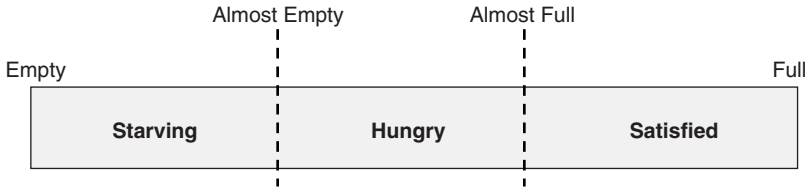
Note that as the address input changes and 'rxcalwr' is de-asserted, so does the corresponding output data ('rxcaldao') based on 'rxcalck' and corresponds to the value written.

Figure 12. S4RX User Calendar RAM Interface



Internal Status Control: This design also 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 and the appropriate three-state status to send on the receive status channel as defined in the OIF specification and shown in Figure 13. The user application could 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 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

The S4RXIO provides 1 to 2 gearing/de-multiplexing, DDR to SDR conversion, clock/data alignment, and LVDS input buffer functions for the receive data direction. All of these functions are performed at the PIC level on an individual signal basis and therefore does 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 1 control signals 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 to 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. A PIC level built in Adaptive Input Logic (AIL) capability is used to perform this function. See also section Link Start-Up Procedures in this document as well as TN1085, LatticeSC MPI/System Bus, for more information on the AIL features.

Serial Management Interface - SMI

The SPI4 MACO core supports a Serial Management Interface (SMI) for the purpose of providing user applications with the ability to change core behavior in real-time while the system is operating. This eliminates the need to re-synthesize, place, and route, in order to change core behavior speeding up debug time. It can also be used to reduce the number of bit-streams required to support projects with multiple SPI4 IP cores since the behavior can be configured independently and dynamically. The SMI port is typically driven from the LatticeSCM built-in System Bus but can also be driven directly from user logic. See TN1085, LatticeSC MPI/System Bus for details on how to interface to the SMI port.

SMI Memory Address Map and Bit Assignments

There are a total 128 bits of SRAM based register storage of which 126 are used. They are used to configure ASIC level functionality for the most part and therefore are associated with data path. Address assignments are listed in Table . Note that some registers span multiple addresses.

Table 1. SMI Memory Address Map and Bit Assignments

Register #	SMI Addr	Register Name	Number Of Bits	Bit Locations
1	0	rxalnmd	1	b[0]
2	0	rxrst	1	b[1]
3	0	rxnumdip4	5	b[6:2]
4	1,0	rxnumdip4e	5	b[3:0], b[7]
5	2,1	rxf1aethrsh	9	b[4:0], b[7:4]
6	3,2	rxf1afthrsh	9	b[5:0], b[7:5]
7	4,3	rxf2aethrsh	9	b[6:0], b[7:6]
8	5,4	rxf2afthrsh	9	b[7:0], b[7]
9	6	txblen	6	b[5:0]
10	6	txrst	1	b[6]
11	8,7,6	txmaxt	16	b[6:0], b[7:0], b[7]
12	9,8	txrep	8	b[6:0], b[7]
13	10,9	txf1aethrsh	9	b[7:0], b[7]
14	12,11	txf1afthrsh	9	b[0], b[7:0]
15	12	txf2aethrsh	6	b[6:1]
16	13,12	txf2afthrsh	6	b[4:0], b[7]

Table 1. SMI Memory Address Map and Bit Assignments (Continued)

Register #	SMI Addr	Register Name	Number Of Bits	Bit Locations
17	13	txf2afothrsh	2	b[6:4]
18	13	rxloop	1	b[7]
19	14	rxrst_dskw	1	b[0]
20	14	rxrst_ail	1	b[1]
21	14	txfdip4e	2	b[3:2]
22	14	txfidle	1	b[4]
23	14	rxtrain_en	1	b[5]
24	14	rxdeskw_en	1	b[6]
25	15,14	txf1aebthrsh	7	b[5:0], b[7]
26	15	unused	2	b[7:6]
Total	**16**		**128**	**b[127:0]**

SMI Memory Descriptions

Table provides a description of the SMI memory based programming. Note that some actions such as receiver reset (RxRST) can be accomplished through IP core I/O as well as through the SMI interface. The middle column indicates signals that can be activated through either method.

Table 2. SMI Memory Descriptions

Memory Name	I/O Support	Description
S4RX/S4TX Common Memory		
RxLOOP	Yes	Receive Loop – Unused at this point.
S4RX Memory		
RXF1AFTHRSH[8:0]	No	RxFIFO1 Almost Full threshold value in 16-byte increments. Affects Rx Flow Control.
RXF1AETHRSH[8:0]	No	RxFIFO1 Almost Empty threshold value in 16-byte increments. Affects Rx Flow Control.
RXF2AFTHRSH[8:0]	No	RxFIFO2 Almost Full threshold value in 16-byte increments.
RXF2AETHRSH[8:0]	No	RxFIFO2 Almost Empty threshold value in 16-byte increments.
RXNUMDIP4[4:0]	No	Number Of DIP4 Words (7-0) – This parameters specifies the number of correct DIP4 words required before the S4RXD receiver declares alignment.
RXNUMDIP4E[4:0]	No	Number Of DIP4 Error Words (7-0) – This parameter specifies the number of incorrect DIP4 words that are required before the S4RXD receiver declares an out-of-alignment error condition.
RxRST	Yes	Receive Reset – This signal when active causes a reset of the entire S4RX receiver.
RxALNMD	No	Receive Alignment Mode – This signal, when active (=1), sets the receiver to operate in Dynamic Timing Alignment mode. The inactive state is not supported (Static mode). This signal should always be a '1'.
RxRSTAIL	Yes	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	Yes	Receive Reset De-Skew – This signal when active causes a reset of the de-skew module forcing it to perform the de-skew operation.
RxTRAIN_EN	No	Receive Training Enable – This signal when active means that the receiver must see Training & Control before declaring the receiver in-alignment. When inactive, the receiver can declare in-alignment state with merely correct DIP4.
RxDESKW_EN	No	Receive Deskew Enable – This signal when active means that the deskew module is allowed to de-skew the input data stream.

Table 2. SMI Memory Descriptions (Continued)

Memory Name	I/O Support	Description
S4TX Memory		
TXF1AETHRSH[8:0]	No	TxFIFO1 Almost Empty threshold value in 16-byte increments.
TXF1AEBTHRSH[6:0]	No	TxFIFO1 Almost Empty Burst threshold value in 16-byte increments. Must be set to at least 2 greater than TxBLEN[5:0] below.
TXF1AFTHRSH[8:0]	No	TxFIFO1 Almost Full threshold value in 16-byte increments. For burst mode behavior (TxBLEN[] > 1), this threshold should be set to a value of at least 32 greater than TXF1AEBTHRSH.
TXF2AETHRSH[5:0]	No	TxFIFO2 Almost Empty threshold value in 16-byte increments. Provides performance and latency tuning. See also section "SPI4 Transmitter -S4TX" for a discussion on setting this threshold. Should not be set to a value of less than 9.
TXF2AFTHRSH[5:0]	No	TxFIFO2 Almost Full threshold value in 16-byte increments. Provides performance and latency tuning. See also section "SPI4 Transmitter -S4TX" for a discussion on setting this threshold. Should not be set to a value of less than 10.
TXF2AFOTHRSH[1:0]	No	TxFIFO2 Almost Full Offset threshold value in 16-byte increments. Provides performance and latency tuning.
TXBLEN[5:0]	No	Tx Max Burst Size – Maximum number of 16-byte blocks that are transmitted for a given channel before segmentation is allowed (i.e. Training & Control Insertion).
TXMAXT[15:0]	No	Tx Min Training Interval – This field indicates the minimum number of cycles * 4 between scheduling of the training sequences on the data path. A value of zero will disable training sequence generation. This parameter needs to be set as an even number, odd ones are not supported.
TXREP[7:0]	No	Tx Training Pattern Repetitions – This field indicates the number of repetitions of the training sequence (10 training control + 10 training data words).
TxFIDLE	Yes	Transmit Force Idles – This signal when active causes the S4TX transmitter to insert idle control words at the soonest available boundary.
TxFDIP4E[1:0]	Yes	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 DIP4 parity
TxRST	Yes	This signal when active (=1) causes a reset of the entire S4TX transmitter.

Calendar and Status RAM Access

There are at most four RAMs within the core that are user accessible in terms of read/write each having their own address and data buses as shown in Table 3. The calendar RAMs are always present in the design and must be initialized through the bus provided or through an initialization file captured via the GUI interface by the user. The Status RAMs are optioned in or out depending upon mode chosen for sending and receiving status. In Transparent Mode, there are no Status RAMs and, therefore, no bus-associated internal I/Os. In RAM Mode, a read/write bus is provided.

Table 3. 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 8b channel ID of the incoming status
Rx STATUS	W only	32	5-bit	16-bit	Each location holds 8, 2b status fields for 8 received status channels.
Tx CALENDAR	R/W	512	9-bit	8-bit	Each location holds a 8b channel ID of the outgoing status
Tx STATUS	R only	32	5-bit	16-bit	Each location holds 8, 2b status fields for 8 transmitted status channels.

Figure 14. Status RAM Layout

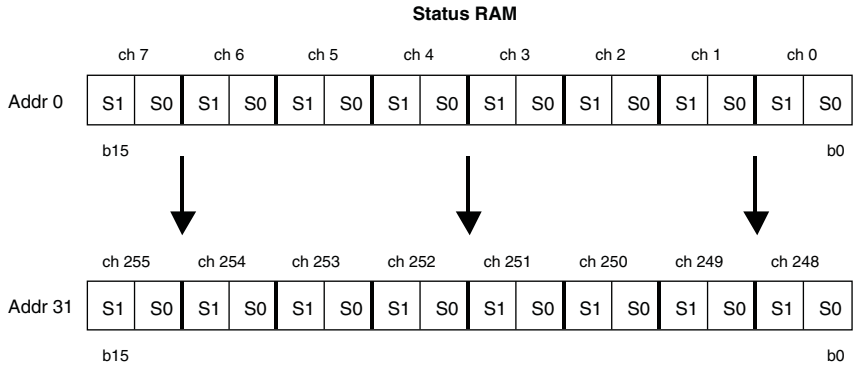
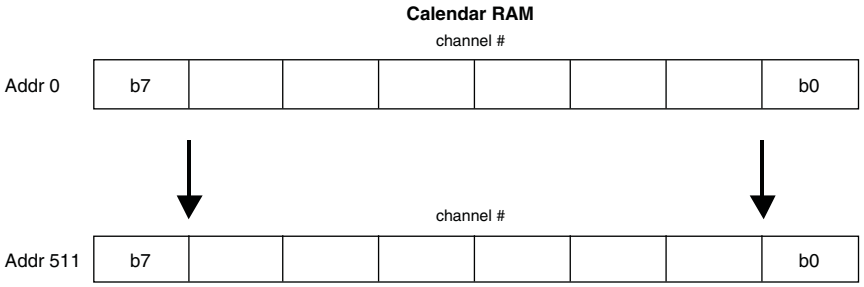


Figure 15. Calendar RAM Layout

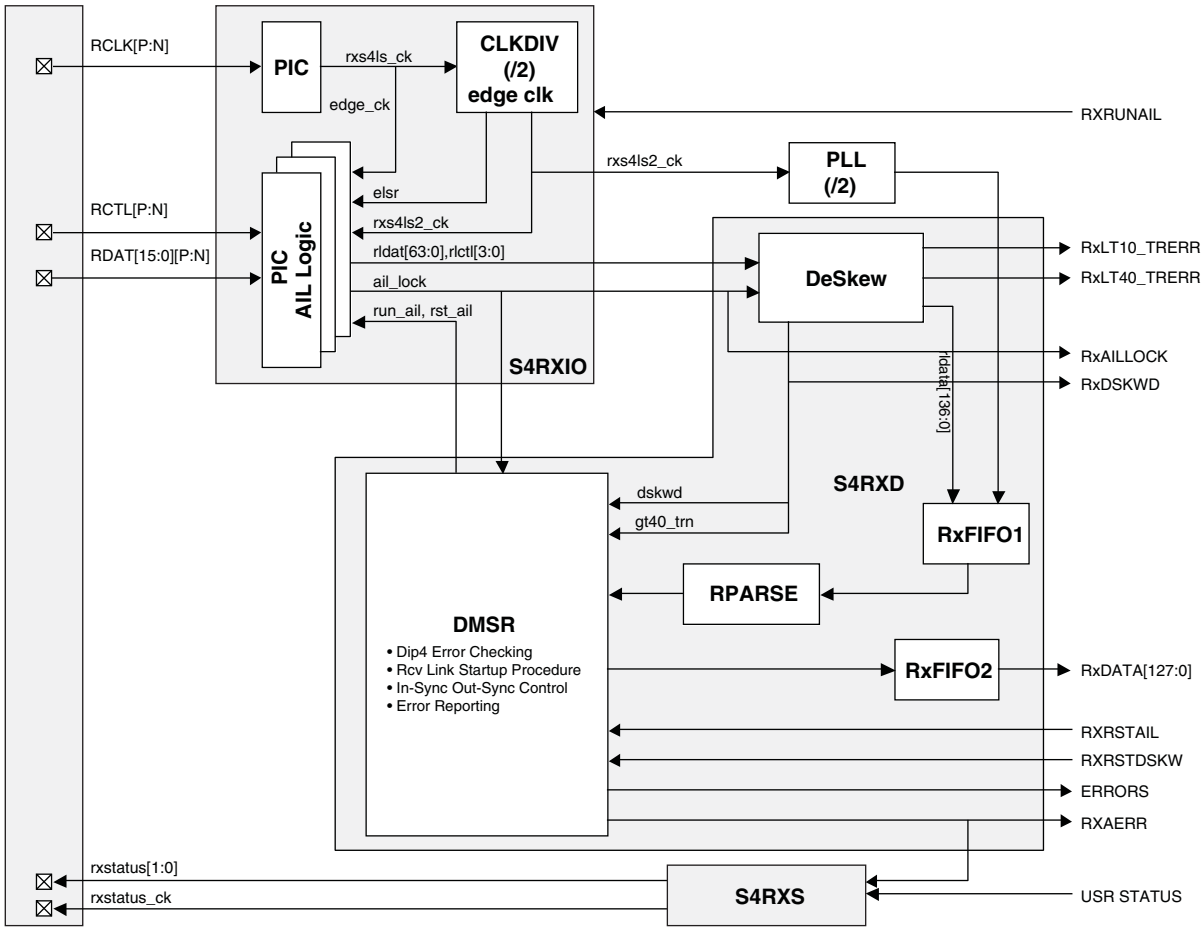


Start-Up Procedures

Receive Direction Start-Up

This section describes the SPI4 Link Start-Up and Recovery procedures for the receive direction. Figure 16 provides a block diagram view of the relevant modules and signal nets that are involved and discussed in this section. Note that in this initial offering, only Dynamic mode operation is supported and therefore Static mode is not discussed.

Figure 16. Start-Up and Recovery Control Block Diagram



Dynamic Mode Start-up and Recovery (DMSR) FSM

Figure 17 shows a diagram of the Dynamic Mode Startup and Recovery (DMSR) Finite State Machine (FSM) illustrating the receivers link start-up and recovery sequence. After a core reset, the DMSR FSM begins in the "reset" state and then transitions unconditionally into the "train" state one clock cycle later. DMSR output signal 'rst_ail' is active only in the "reset" state and forces a full AIL reset. While in the "reset" and "train" states as well as all others except "normal", DMSR output signal 'RXAERR' is forced inactive 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 the AIL and deskew functions to lock-on and de-skew the data and control bus.

Once the AIL logic has successfully locked onto the data edges and is confidently sampling the data and control signals ('RxAIL_LOCK'=1), the deskew block will then attempt to deskew all 17 bits of the input bus (16 data and 1 control). To do this, a minimum of 40 repetitions of the Training Control and Data pattern is required and when detected, signal GT40_TRN will be asserted and state flow proceeds to state "deskew". If at this time the deskew block has successfully deskew'ed the entire bus, signal 'RxDSKWD' will also be asserted. Throughout this time, it is expected that the far-end is sending constant Training Control and Data patterns as specified by the OIF. If it is detected that this is not occurring, an error is generated ('RxLT40_TRNERR') and could be caused by a faulty Status or Data interface.

After the input data and control bus has been de-skewd, state "dip4" is entered and an attempt can now be made 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 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 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 processes begins all over again. Additionally, if the de-skew block detects a non-deskewed condition, 1) state flow proceeds back to state "de-skew", 2) out-of-synchronization state is declared, 3) again the far-end is sent constant Status framing. Lastly, if at any time during normal operation the AIL logic indicates an AIL unlocked condition, 1) state flow will return to the "train" state, 2) out-of-synchronization state is declared, 3) again the far-end is sent constant Status framing, 4) 'RxRUN_AIL' is temporarily forced inactive causing the lock criteria to be met before the de-skew process is allowed to start.

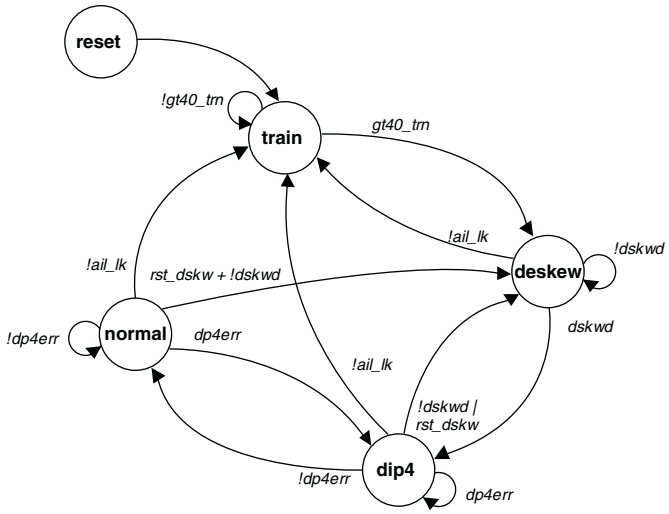
The above actions are taken autonomously by the DMSR-FSM as it continuously seeks to achieve a "normal" state where the receiver is in synchronization with the data (in-sync) without user involvement. The user can however force certain recovery actions from any state such as a full DMSR-FSM reset ('RxRST_AIL') comprised of full AIL initialization, re-Deskew, and data re-sync or, a partial reset which includes only re-deskew and data re-sync.

Debug Mode: For lab debug and possible non-compliant devices, the user can selectively disable ('RxTRAIN_EN'=0) the requirement that Training and Control be observed before allowing state flow into state "normal" and de-asserting "RxAERR". When 'RxTRAIN_EN' =0, signals 'RxAIL_LOCK', 'RxGT40_TRN', and 'RxDSKWED' are all forced to their active state. AIL lock must be forced in addition because the expected data link pattern in lieu of Training and Control is Idles which has no edge content needed for lock. AIL lock must be forced to ensure DMSR-FSM recovery does not occur which would result in a lock-up state (waiting for 'RxAIL_LOCK' with a pattern of Idles). Note that these signals are only forced active in the connection to the DMSR-FSM; the connection to the user level I/O is the raw and can be used by the user in it's recovery decision making.

In this mode, real AIL_LOCK will not actually occur until the user begins to send data. Depending on the initial setup of the AIL logic (tap selection centering), the first few packets/segments may or may not be received with errors. Eventually however, the AIL logic will achieve an optimal lock point and things will settle out.

Lastly, regarding de-skew, the de-skew module can be directed via control bit 'RXDESKEW_EN' to either perform the de-skew operation or not (=0 forces center selection). Detection however, is always enabled. This allows a configuration for example where 'RXTRAIN_EN'=0 for start-up in dynamic mode but desire de-skew if and when scheduled Training and Control occurs.

Figure 17. Dynamic Mode Start-Up and Recovery FSM



FSM Inputs:

- **gt40_trn (rxgt40_trn)** – Greater than 40 training and ctl received.
- **dskwd (rxdskwd)** – Deskew block has sucessfuly de-skewed input bus.
- **dp4err** – Prog number of correct dip4 received.
- **!dp4err** – Prog number of incorrect dip4 received.
- **rxrstail** – Reset request with AIL and de-skew initialization.
- **rxrstdskw** – Reset request with only de-skew initialization.
- **ail_lk (rxaillock)** – AIL circuit has acheived lock.
- **grst_n** – Global reset.
- **rxrst** – S4RX-specific reset.

FSM Outputs:

- rxaerr - Inactive only in state "normal".
- run_ail - Active in all states except "reset". Inactive for one two cycles on transiions to state "train".
- rst_ail - Active only in state "reset".

Notes:

- User request rxrstail active results in transition to the reset state regardless of current state.
- runail and rstail min two cycles duration
- Transitions to state "train" are accompanied with temporary removal of run_ail forcing lock criteria to be met.
- The de-skew block waits for AIL_LOCK before searching for training patterns.

AIL Attributes

In Dynamic Mode, the AIL control logic is used to control data delay settings ensuring proper set-up and hold. Listed below are properties used in design capture for the AIL feature. Please refer to technical note TN1088, LatticeSC PURESPEED I/O Usage Guide for a detailed discussion of the AIL capability and how it works using the settings listed below.

1. MARGIN WINDOW = Four

2. BESTFIT = OFF

3. BITSWAP = ON

4. CLOCK EDGE = Two

5. AIL = ON

De-Skew Attributes

1. 'RxAILLOCK' inactive (not locked) causes the de-skew block to indicate a non de-skewed state via signal 'rxd-skwd'.

2. 'RxAILLOCK' inactive causes the de-skew block to essentially reset and wait for an AIL locked condition before proceeding with a search for training and control patterns. While unlocked, internal signal 'rxgt40_trn' is not activated regardless of input data stream content.

3. Once AIL locked, the de-skew block must observe 40 consecutive repetitions of the training control and data pattern before asserting 'rxgt40_trn' and asserting signal 'RXDSKWD'. A four-cycle minimum active high signal is provided for the receiving clock domain. 'rxgt40_trn' stays active until the instance of training control and data patterns ceases and then returns low in anticipation of returning to the "training" state due to regularly schedule training patterns.

4. Once de-skewed, via constant training control and data, a minimum of 10 repetitions are required to ensure (check) and maintain the de-skewed state. This is the value that should be programmed into the far-end's Alpha.

5. Once de-skewed, via constant training control and data, absence of Training and Control patterns will not change or affect the current skew selection.

6. While in the de-skewed state ('RxDSKWD' active), less than 10 repetitions (1-9) observed results in an 'RxLT10_TRERR' error signal being activated. A four-cycle active high signal is provided for the receiving clock domain.

7. While not de-skewed, less than 40 (1-39) repetitions observed for the 2nd and following times results in the 'RXLT40_TRERR' signal being activated. The first instance could come from in-flight regularly scheduled training control and data of 10 and so it is ignored. A four-cycle active high signal is provided for the receiving clock domain.

8. If 'RxTRAIN_EN' is inactive, the de-skew block forces a common skew selection for all data and the control line.

DMSR Attributes

1. As the over-all link startup and recovery controller, the DMSR provides enough intelligence and capabilities to never allow a recoverable condition result in non-recovery.

2. Provides several levels of user requested recovery that over-ride current autonomous actions. Signal input 'RxRSTAIL' forces full reset from any state including AIL re-lock and de-skew. Signal input 'RxRSTDSKW' forces partial reset from "dip4" and "normal" states.

3. Output signal 'RxAERR', when active (all states except normal), forces constant framing on the output status channel causing our receiver to observe constant training control and data patterns.

4. If in the "deskew" state for a considerably long period of time (2x) than what is required for de-skew, transitions back to state "train" causing 'rxrun_ail' to go inactive and perform AIL re-lock. This loop between "train" and "de-skew" continues forever until the input bus has been deskewed.

5. Before leaving state "deskew" for entry into state "dip4", all parity error counters, good and bad, are cleared.

6. 'GRST_N' or 'RxRST' active from any state forces the "reset" 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 it's 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 transmitted 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.

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

Hard-Core Physical Placement

The LatticeSCM series of devices provides a total of two different SPI4 hard-core blocks upon which up to two SPI4 IP cores are built upon. Each hard-core block is located in the center along the edge of the left and right sides. Placement of the hard-core block drives placement of the I/O and the relatively small amount of FPGA logic as well. Knowing the placement of various functions in the design allows the user to come up with a good floor plan for their design.

SPI4 Line Side I/O

Although the user has the ability, following the standard FPGA design flow, to specify whatever I/O placement is desired, Lattice has worked out a pre-defined set of I/O pins for both the left and right sides in a number of package types that allow the user to migrate from smaller to larger FPGA device sizes (25K LUTs to 40K LUTs, etc.) while maintaining the exact same pin-outs. All arrangements have been placed and routed at speed and several have been tested in the lab.

The pin-out arrangement simply spans the line side differential I/O along the left or right side depending upon site chosen by the user during GUI capture. I/O (LVDS buffers) and PIC (gear-box) level functionality associated with both receive and transmit data paths span either the top and bottom left quadrants or, top and bottom right quadrants. I/O functionality associated with both receive and transmit status paths exist in only the bottom left or bottom right quadrant.

Soft-Logic Placement

The soft-logic part of the SPI4 MACO core is fairly small (<1700 LUTs) given that most of the functionality has been captured in ASIC gates. The soft-logic that does exist, is placed along the left or right sides of the device near the hard-core and I/O blocks that it serves.

Clocking and Synchronization

This section provides a general discussion of synchronization dealing with various approaches that may be taken for delivery and extraction of clock signals to/from the IP core. There is the potential with this core to employ the use of just a few clocks or to employ the use of a lot of different clocks and therefore clock domains depending upon needs. Additionally, there is a great deal of flexibility in terms of 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 4 provides a comprehensive view of the total number of clock nets possible when using the SPI4 MACO core as well as a host of additional information about each net that should serve useful when determining a synchronization strategy. While the number of clocks seems like a lot, there are a few things to consider when arriving at a plan:

- 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 even be further consolidated into one of transmit or receive data path divide by four clock nets ('rxs4ls4_ck') if desired.
- Since the SPI4 IP core will be placed exclusively on either the left or right sides of the device, Primary clock drivers will only be used in these quadrants (top and bottom left or, top and bottom right). Since the LatticeSCM

device provides 12 Primary clock drivers per quadrant, there should still be plenty of drivers left for user functions assuming the Primary clock are floor planned at the quadrant level.

Table 4. Clock Listing

Clock Net Name	Originating IP Core Port Name	Driver Source	Clock Driver Type	Quadrants[1] (Left or Right)	Max. Freq.
rxs4hs_ck	'RCLK'	PIO	edge	bottom + top	500MHz
rxs4ls2_ck	'RCLK'	CLKDIV	primary	bottom + top	250MHz
rxs4ls4_ck	'RCLK'(/2)	FPGA (flip-flop)	primary	bottom	125MHz
txs4hs_ck	'TXS4HS_CK'	PLL/PIO	edge	bottom + top	500MHz
txs4ls2_ck	'TXS4HS_CK'	CLKDIV	primary	bottom + top	250MHz
sdck	'SDCK'[2]	PLL	primary	bottom + top	150MHz
sdck2	'SDCK2'[2]	PLL	primary	bottom	300MHz
rxstck	'RXSTCK'	PIO/PLL	primary/local	bottom	150MHz
rxstck_line	'RxSTCK_LINE'	PIO/PLL	primary/local	bottom	125MHz
rxcalck	'RXCALCK'	PIO/PLL	primary/local	bottom	125MHz
txstck	'TXSTCK'	PIO/PLL	primary/local	bottom	150MHz
txcalck	'TXCALCK'	PIO/PLL	primary/local	bottom	125MHz
smi_clk	'SMI_CLK'	PIO/SB	primary/local	top	50MHz

1. Quadrants: left or right depends on which SPI4 MACO site is chosen.
2. SDCK and SDCK2 must have a specific phase relationship - see below.

Special Clock Requirements

Clock signals 'SDCK' and 'SDCK2' must have a specific phase relation since data is transferred inside the core from the lower speed 'SDCK' clock to the higher speed 'SDCK2' clock. Edges of the low speed clock must be co-incident with the rising edges of the high-speed clock as opposed to the negative edge. Also, the edges of the low speed clock must have at least 200ps delay with the edges of the high-speed clock. The high-speed clock must not lag the low-speed clock. The easiest way to generate these clocks is through an internal PLL.

SPI4 Operating Speeds and Device Speed Grade Dependency

LatticeSCM devices can be used at internal core supply voltage of VCC=1.2V (normal mode) or VCC=1V (low power mode). Various characterization tests were run on the SPI4 MACO blocks to determine the relationship between their speeds of operation and operating conditions for the device. Test were performed with the FPGA core voltages ranging from 0.95V (worst case for VCC =1V) to 1.26V (best case for VCC=1.2V) and junction temperatures ranging from -40°C to 105°C. Characterization test results are summarized in Table 5 for different speed grades.

Table 5. Maximum SPI4 Data Rate per Speed Grade

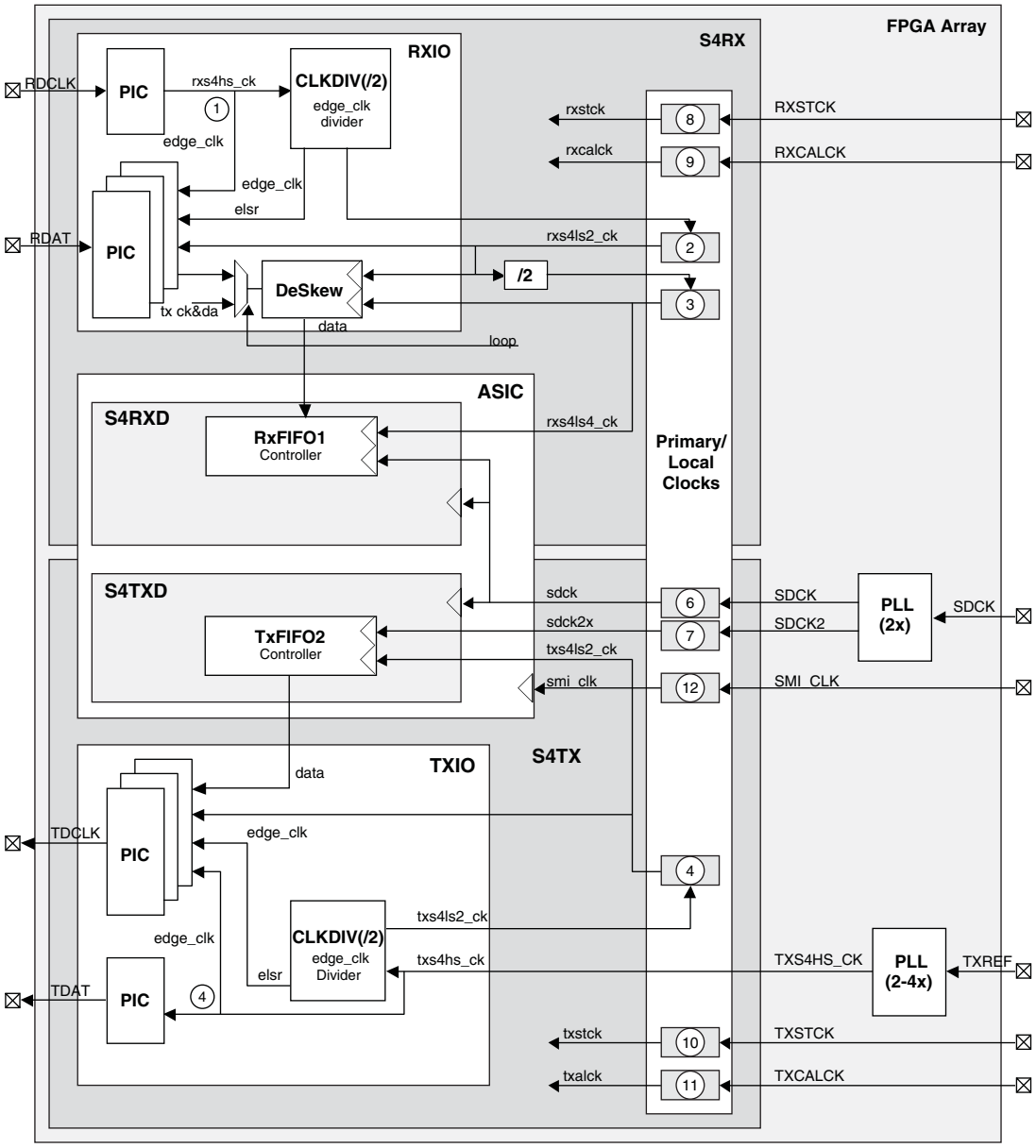
Speed Grade	-5	-6	-7	Units
MAX SPI4 Data Rate Across All Operating Conditions	800	900	1000	Mbps

Depending on the SPI4 speed requirements of their design, users should pick the appropriate speed grades as specified in Table 5. For example, if the SPI4 needs to run at an 800Mbps data rate, any of the speed grades (-5, -6, or -7) would suffice. However, if a design needs 1000Mbps SPI4 operation, you are advised to pick a -7 speed grade device.

Clock Usage Diagram

Figure 18 provides a graphical representation of some of the information presented in Table 4. It also shows one method for generating the 'SDCK' and 'TXS4HS_CK' clock signals through FPGA PLLs.

Figure 18. Clock Usage Diagram



System Level Synchronization

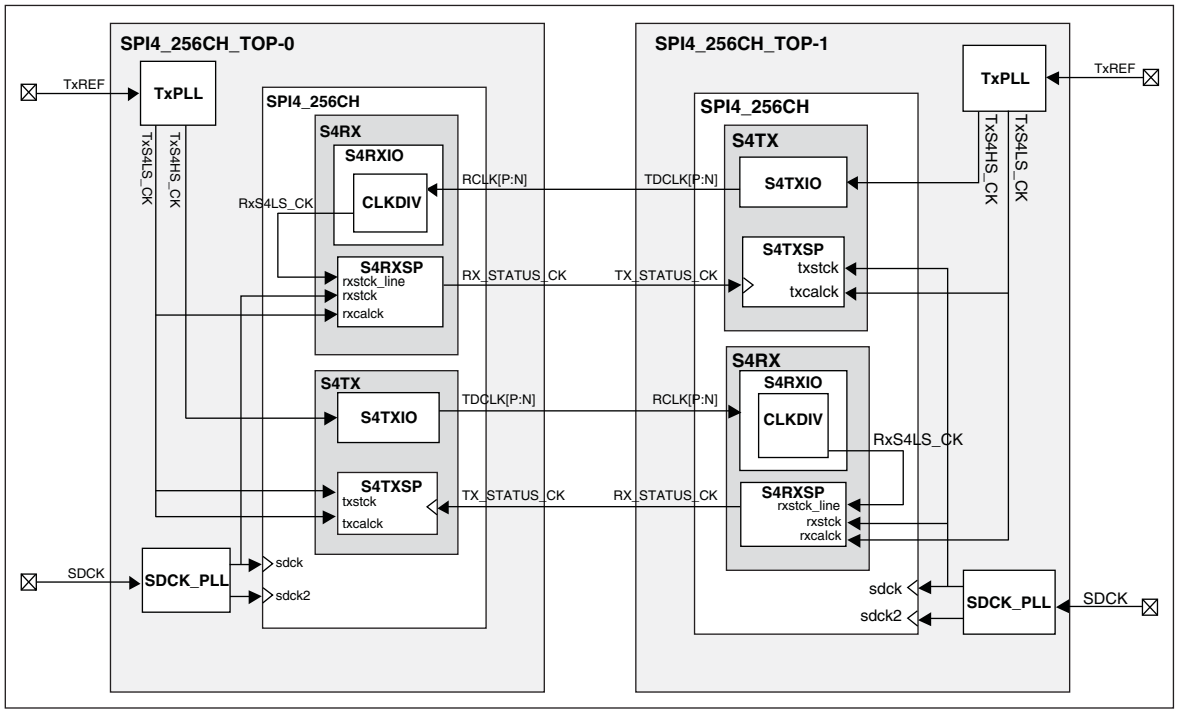
Figure 19 provides an example of a system level synchronization scheme chosen where at each end the timing reference supplied to the Transmit PLL (TxREF) is the starting point from which timing can be analyzed. In this example, status is driven back to the far-end using the divide by four version ('RxS4LS4_CK') of the SPI4 line clock. This arrangement results in status being received from the far-end that is at least frequency locked to the near-end version of the same clock.

A slight modification to this arrangement is to allow the received divide by four version ('RxS4LS4_CK') of the SPI4 line clock to be used as the timing reference to the internal Transmit PLL at one end of the link. A timing loop would be created if done at both ends; somewhere their needs to be a fixed timing reference.

The system level clock domains (SDCK) do not leave 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.

Figure 19. Top and System Level Clocking (Both SPI4 Ends)



Selecting a System Data Clock Frequency ('SDCK') - Receiver: Data received from the SPI4 line is written directly into RxFIFO1 as full 136-bit data slices (128 bits of data [eight 16-bit words] and 8 bits of control [one bit per word]) using a divide by four version of the receive SPI4 line clock. There is no filtering of control words or any other component of SPI4 traffic as it is written into RxFIFO1; all SPI4 traffic, control and data, pass through RxFIFO1. The receiver Parser module reads full 136-bit data slices from RxFIFO1 using the system data clock ('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 8 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 (2-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 (2-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 4 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 (2 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 would need 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% over-speed. Assuming an 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. Probably somewhere around 20% should 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% over-speed, 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.

Lastly, it should be pointed out that although hitting the high-water mark on RxFIFO1 due to a lack of over-speed results in flow-control and no loss of data, it can have a significant negative affect on the maximum bandwidth utilization of the SPI4 line and should therefore be avoided at all costs (this feature was added as a "stop-gap" measure only and not meant for normal operation). This occurs because the far-end, though it stops transmission of

data, switches to Idles Control words which must also pass through RxFIFO1 and parsed by the receiver parser to ensure proper SPI4 protocol. Idle control words are processed more efficiently than Eop Control words but also requires bandwidth and can make the recovery time transitioning from high-water (flow-control) to low-water (non-flow-control) longer than might be expected. Once the "read" side has processed all of the EOP data leading up to the Idle Control words resulting from flow-control, the FIFO may still be fairly full. The period of time required before hitting low-water mark and resuming data flow is based on the amount of over-speed and how full the FIFO is. For this case (Idles) there is one to one correspondence between read and write cycles but it may still take a fairly long time for the FIFO to drain and eventually hit the low-water mark.

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. In order guarantee a fully utilized SPI4 line the maximum over-speed required for this design is between 10-15%.

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

Parameter Descriptions

Table 6 lists the parameters generated from the GUI configuration process necessary to tailor an SPI4 core for specific user requirements. Some of the parameters are associated with the internal data path (static SMI type ones - functionality provided in ASIC gates) and the rest are associated with status path and I/O selections.

Synthesis affecting parameters are divided into two categories, those affecting synthesis of the core and those affecting synthesis of the top and are labeled in this manner. This type of parameter actually changes the "shape" of the IP in terms of the number of I/O or types of components used in the design.

All parameters with type of "Static SMI" are considered Static in that their values in-circuit come from memory inside the device (SMI Memory) initialized via the program bit stream. During user synthesis, the values of these parameters will be picked up from the params.v file and used in "defparam" statements to assign and then pass the values through the EDIF netlist to the back-end software where the specific memory bits are initialized.

Table 6. Parameters

Number	Parameter	Description	Range	Default	Type
S4RX Parameters					
1	RXF1AETHRSH	Receive FIFO1 Almost Empty Threshold (16 byte units)	1-510	40	Static SMI
2	RXF1AFTHRSH	Receive FIFO1 Almost Full Threshold (16 byte units)	2-511	400	Static SMI
3	RXF2AETHRSH	Receive FIFO2 Almost Empty Threshold (16 byte units)	1-510	40	Static SMI
4	RXF2AFTHRSH	Receive FIFO2 Almost Full Threshold (16 byte units)	2-511	80	Static SMI
5	RXNUMDIP4	Receive Number of Correct DIP4 Before In-Sync	1-7	3	Static SMI
6	RXNUMDIP4E	Receive Number of In-Correct DIP4 Before Out-Sync	1-7	7	Static SMI
7	RXCAL_M	Receive Calendar Repetition	1-255	1	Static Input

Table 6. Parameters (Continued)

Number	Parameter	Description	Range	Default	Type
8	RXCAL_LEN	Receive Calendar Length	1-512	4	Static Input
9	RXINTSTC	Receive Internal Status Control[1]	True/False	False	Static Input
10	RX_STAT_FF_BEHAVIOR	Receive Status FIFO Flag (Full/Almost Full) Behavior[2]	Full/Almost Full	Full	Static Input
11	RXSTREDGE	Receive Status Output Active Clock Edge Used	Rising/Falling	Rising if defined	User Synthesis
12	RXSTLVDSBUF	Receive Status Buffer Type	LVTTL/LVDS	LVTTL if not defined	User Synthesis
13	RXSTAT_MD	Receive Status Mode (RAM / Transparent)[3]	RAM/Trans	Trans if defined	Core Synthesis
14	RXALNMD[4]	Receive Alignment Mode (Static/Dynamic)	Static/Dynamic	Dynamic	Static SMI
15	RXTRAIN_EN[4]	Receive Training Enable	Enabled/Disabled	Enabled	Static SMI
16	RXDESKEW_EN[4]	Receive Deskew Enable	Enabled/Disabled	Enabled	Static SMI
S4TX Parameters					
17	TXNUMDIP2	Transmit Number of Correct DIP2 Before In-Sync	1-7	3	Static Input
18	TXNUMDIP2E	Transmit Number of In-Correct DIP2 Before Out-Sync	1-7	3	Static Input
19	TXCAL_M	Transmit Calendar Repetition	1-255	1	Static Input
20	TXCAL_LEN	Transmit Calendar Length	1-512	4	Static Input
21	TXINTSTC	Transmit Internal Status Control	True/False	False	Static Input
22	TXSTREDGE	Transmit Status Input Active Edge Used	Rising/Falling	Falling if not defined	User Synthesis
23	TXBLEN	Transmit Burst Length (16-byte units)	1-63	4	Static SMI
24	TXPACK_EN	Transmit Packing Enable	Enabled/Disabled	Disabled	Static Input
25	TXMAXT	Transmit Maximum Training Interval	0-65536	32768	Static SMI
26	TXREP	Transmit Training Pattern Repetitions	0-255. If SPI4 TX is connected with Lattice SPI4 RX, 1-9 is not allowed.	10	Static SMI
27	TXF1AEBTHRSH	Transmit FIFO1 Almost Empty Burst Threshold (16-byte units)	1-64	6	Static SMI
28	TXF1AETHRSH	Transmit FIFO1 Almost Empty Threshold (16-byte units)	1-510	40	Static SMI
29	TXF1AFTHRSH	Transmit FIFO1 Almost Full Threshold (16-byte units)	2-511	160	Static SMI
30	TXF2AETHRSH	Transmit FIFO2 Almost Empty Threshold (16-byte units)	1-62	14	Static SMI
31	TXF2AFTHRSH	Transmit FIFO2 Almost Full Threshold (16-byte units)	2-63	24	Static SMI
32	TXF2AFOTHRSH	Transmit FIFO2 Almost Empty Offset Threshold (16-byte units)	0-3	1	Static SMI
33	TXSTLVDSBUF	Transmit Status Buffer Type	LVTTL/LVDS	LVTTL if not defined	User Synthesis

Table 6. Parameters (Continued)

Number	Parameter	Description	Range	Default	Type
34	TXSTAT_MD	Transmit Status Mode (RAM / Transparent)	RAM/Trans	Trans if 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 SMI-based Static parameter is used inside the ASIC block and also pushed up to the soft logic level and used there as well.

Signal Descriptions

Two tables are provided below representing the entirety of the SPI4 MACO core I/O. Table 7 provides the internal user interface side, and Table 8 provides the SPI4 line side external I/O.

Table 7. User Side Signal Descriptions

Signal Name	Direction	Description
S4RX/S4TX Common Signals		
SDCK	Input	**System Data Clock** - Should have over-speed relative to the SPI4 receive line clock / 4 (i.e. RCLK/4). Also used in user domain logic to transfer data to and from the IP core. See the Special Clock Requirements section of this document for 'SDCK' and 'SDCK2' phase requirements.
SDCK2	Input	**System Data Clock Multiplied By 2** - Should have over-speed relative to the SPI4 receive line clock (i.e. RCLK/2). Not used in user domain logic due to high speed (300 MHz maximum rate).
S4RX Internal Data Path Related Signals		
GRST_N	Input	**SPI4 MACO LatticeSCM Global Core Reset** (active low).
RxRST	Input	**Receive Reset** - This signal when active causes a reset of the entire S4RX receiver.
RXDATA[127:0]	Output	**Receive System Data** - User side system data outputs from RxFIFO2 (8 - 16 bit data words).
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[3:0]	Output	**Receive Remainder** - Indicates the byte lane position of the last valid data byte (meaningful if corresponding data slice contains an EOP). A value of 0 = 1 byte, left justified MSB = b127-b120). A value of 15 = all bytes valid.
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 the SPI4 Burst which may or may not be an EOP boundary. Error relevant for Packets and packet segments only - not active for single control words. See also RxS4ERR. (Clock=SDCK 1 cycle active)
RXDVAL	Output	**Receive FIFO Data Valid** - This signal when active (=1) qualifies RxFIFO2 data (RxDATA[127:0]) as being valid. There is no fixed relationship to the RxFRD input and this RxDVAL signal - RxDATA[] should be ignored when inactive.

Table 7. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
RXF2AE	Output	**Receive FIFO Almost Empty** - This signal when active (=1), indicates RxFIFO2 is almost empty as determined using synthesis parameter RXF2AETHRSH[8:0] above. This signal is used by the S4RXS block but may be used by the user side interface. See also output signal RXF2AF below.
RXF2AF	Output	**Receive FIFO Almost Full** - This signal when active (=1), indicates RxFIFO2 is almost full as determined using synthesis parameter RXF2AFTHRSH[8:0] above. Also, when this signal is asserted (edge), the "Satisfied" condition can optionally be transmitted on the RXSTATUS[1:0] channel until the corresponding RxFAE signal is asserted (edge).
RXF2E	Output	**Receive FIFO Empty** - This status signal when active (=1), indicates RxFIFO2 is empty and the user side should stop reading.
RXF2FE	Output	**Receive FIFO 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.
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. (Clock=SDCK, at least 1 cycle active)
RXAERR	Output	**Receive Alignment Error** - This signal when active (=1), indicates the S4RXD block has not achieved alignment (consecutive number of correct DIP4 words). (Clock=SDCK, at least 2 cycles active)
RS4ERR[4:0]	Output	**Receive SPI4 Line Protocol Violation Errors** - These signals when active (=1) indicate that the S4RXD block has received a sequence of data and control words that are in violation of the SPI4 protocol. All signals are synchronous to the SDCK clock domain and will be active for at least 1 full cycle. Note: RxS4ERR[4] must be sampled in the SDCK clock domain before being analyzed because it is not a direct output from a FF. The others are direct FF outputs and do not require sampling. RS4ERR[4] - A valid write to RxFIFO2 where less than 16 bytes are marked valid without an EOP active. SPI4 burst aligned and "or"ed into the 'RXDP4E' 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.
RXF1FERR	Output	**Receive FIFO1 Full Error** - This signal when active (=1) indicates that FIFO1 has become full and has over flowed. With an operational status channel, this error should never occur.
RXF1_PARERR	Output	**Receive FIFO1 Parity Error** - This signal when active (=1) indicates that a parity error was detected on data read from RxFIFO1.
RxRSTAIL	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. (Clock=rxs4ls2_ck, 4 cycles active)

Table 7. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
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. (Clock=rxs4ls2_ck, 4 cycles active)
RxDSKWD	Output	**Receive De-Skewed** - This signal when active indicates that the de-skew block has successfully de-skewed the input SPI4 line. (Clock=rxs4ls2_ck, at least 4 cycles active)
RxAIL_LOCK	Output	**Receive AIL Locked** - This signal when active (=1) indicates that all AIL circuits are locked. (Clock=rxs4hs_ck, at least 4 cycles active)
RxRUNAIL	Input	**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.
RxLOOP	Input	**Receive Loop** - Unused in this version of the core.
RxDCNTL[8:0]	Input	**Receive Delay Control** - This array specifies the data delay value relative to clock in Static mode - currently unused since static mode is not offered.
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 = 125MHz).
RXSTCK	Input	**Receive Status Clock** - This clock signal is used as the receive direction user status interface clock. All receive status signals passing into or out of the IP core on the user side are synchronous to this clock. Can be a different clock than 'RxSTCK_LINE' but must be in the range of, on the low side, 'RxSTCK_LINE and on the high side 150MHz. Cannot be lower than 'RxSTCK_LINE'. Used in both Transparent and RAM modes.
RxSTCK_LINE	Input	**Receive Status Line Clock** - This clock signal is used to clock status off chip onto the SPI4 line. See also signal description 'rxstck'.
RXFDIP2E	Input	**Transmit Force DIP2 Error** - This signal when active (=1) causes the S4RXS to insert DIP2 parity errors. Framing remains valid.
RXINTSTC	Static 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, & FF - Satisfied). This mode should be used only for single channel applications. See also static input RxSTAT_FF_BEHAV. When RxINTSTC is inactive, only the AF or FF flag is used and will over-ride user status when active as a protective measure.
RxSTAT_FF_BEHAV	Static Input	**Receive Status FIFO Full Behavior** - This control signal selects whether the S4RXS block reacts to the Almost Full (AF) = 0, or Full (FF) = 1 RxFIFO2 flag signal 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.
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.

Table 7. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
RXSTDAT[15:0], or [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].
RXSTADD[4:0]	Input	**Receive Status Address** - 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.
RXSTPORT[7:0]	Output	**Receive Status Address Out** - This output status address bus coupled with the TxSTAT[] 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.
RXSTMSK[7:0]	Input	**Receive Status Mask** - This field (a=1) provides a mask for the RxSTAT[8] 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 RXSTCK 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.
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.
S4TX Internal Data Path Related Signals		
TxRST	Input	This signal when active (=1) causes a reset of the entire S4TX transmitter.
TXDATA	Input	**System Data** - User side system data bus inputs into TxFIFO1 (eight 16-bit data words).
TXFWR	Input	**Transmit FIFO1 Write** - When active (=1), the corresponding 8-word data slice is written into the TxFIFO1. If inactive, TXDAT 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.

Table 7. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
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 in MSB location (b127-120). A value of 15 = all bytes valid. During normal data transmission, the value should always be 15.
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 synthesis parameter TxAETHRSH[] 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 synthesis parameter TxAFTHRSH[] 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.
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.
TxFIDLE	Input	**Transmit Force Idles** - This signal when active causes the S4TX transmitter to insert idle control words at the soonest available boundary.
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 DIP4 parity
TXREM_ERR	Output	**Transmitter Remainder Error** - This signal when active (a=1) indicates that TxFIFO1 was written with a remainder of other than 4'b1111 during a cycle when 'TXEOP' was not active; a clear error situation. (Clock=SDCK, 1 cycle active)
TXBURST_ERR	Output	**Transmitter Burst Error** - This 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]. (Clock=SDCK, 1 cycle active)
TXF2_PARERR	Output	**Transmitter FIFO2 Parity Error** - This signal when active (a=1) indicates that a parity error has been detected on a read operation of TxFIFO2.
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 125MHz).

Table 7. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
TXSTCK	Input	**Transmit Status Clock** - 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.
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. (Clock=I/O port tstatus_ck, minimum 4 cycle active)
TXSTAT_T[1:0]	Output (TxSTCK)	**Transmit Status [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.
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 S4TXS 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[] & 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]	Static Input	**Transmit Calendar Repetition** - Alpha.
TXCAL_LEN[8:0]	Static Input	**Transmit Calendar Length**.

Table 7. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
TXSDP2ERR	Output	**Transmit Status Parity Error** - This signal when active (=1), indicates that the S4TSP block has detected a parity error on status received. (Clock=I/O port tstatus_ck, 1 cycle active)
TXNUMDIP2[2:0]	Static 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]	Static 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	Output	**Transmit SPI4 Low Speed Divide By 2 Clock** - This is the internal divide by 2 version of S4TXHS_CK. General purpose output.
SMI Interface Signals		
SMI_RSTN	Input	**SMI Reset** - This active low reset affects the internal SMI controller.
SMI_CLK	Input	**SMI Clock** - SMI Clock
SMI_RD	Input	**SMI Read** - Active high read strobe
SMI_WR	Input	**SMI Write** - Active High write strobe
SMI_WDATA	Input	**SMI Write Data** - Serial write data
SMI_ADDR[9:0]	Input	**SMI Address** - SMI address
SMI_RDATA	Output	**SMI Read Data** - SMI Read data

The SPI4 line side I/O signal list in Table 8 reflects that of primary I/O of the device as well as that of 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 (rstatus[]), the clock is driven from the top level netlist through a Primary clock tree rather than through the core. Note that as part of the over-all IP offering, example top_level netlists as well as supporting I/O buffer modules are provided.

Table 8. SPI4 MACO Line Side I/O Signal List

Primary I/O Signal Name	Core I/O Signal Name	Direction	Description
Receive Direction			
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.
RCTL_N	n/a	Input	**Receive SPI4 Control (N)** - Control (=0) / Data (=1) indicator.
RX_STATUS[1:0]	rstatus[1:0]	Output	**Receive Status** - Output status channel associated with input/receive data path.
RX_STATUS_CK	n/a	Output	**Receive Status Clock** - Output status channel clock. Driven at the top_level via Primary Clock network.
Transmit Direction			
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.

Table 8. SPI4 MACO Line Side I/O Signal List (Continued)

Primary I/O Signal Name	Core I/O Signal Name	Direction	Description
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]	tstatus[1:0]	Input	**Transmit Status** - Input status channel associated with output/transmit data path.
TX_STATUS_CK	tstatus_ck	Input	**Transmit Status Clock** - Input status channel clock.

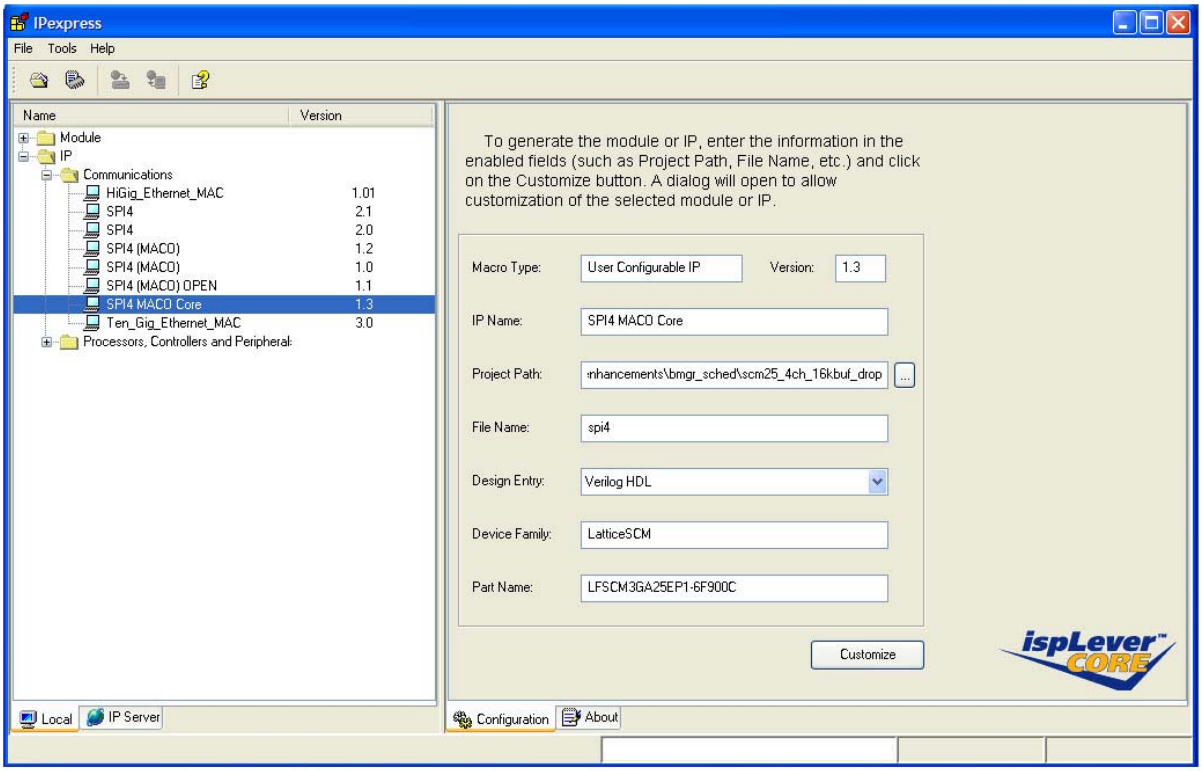
SPI4 MACO Design Kit and Core Generation

The SPI4 MACO core is available for download via IPexpress, the IP configuration utility included with ispLEVER. The IPexpress GUI window for the SPI4 MACO core is shown in Figure 20. 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.
- Design Entry Type – Verilog HDL/VHDL.
- Device Family – Device family to which IP is to be targeted (LatticeSCM).
- Part Name – Specific targeted part within the selected device family.

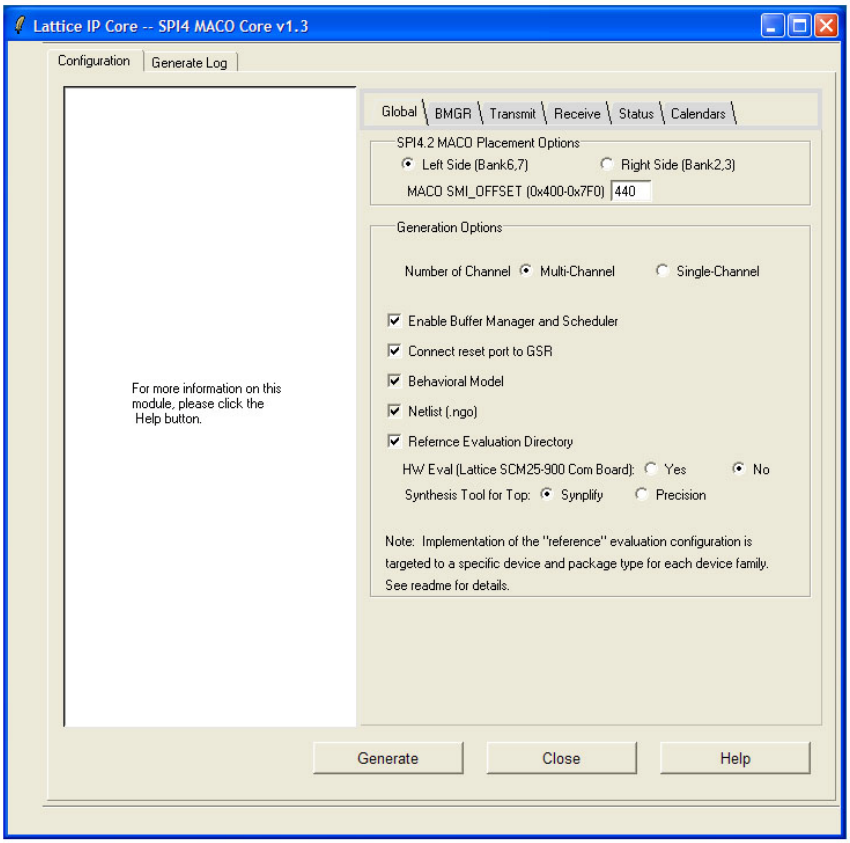
Note that if IPexpress is called from within an existing project, Project Path, Design Entry, Device Family and Part Name default to the specified project parameters. Please refer to the IPexpress on-line help for further information.

Figure 20. IPexpress GUI Window



To create a custom configuration, click on the **Customize** button to display the IP core Configuration GUI, shown in Figures 21 to 26. For the parameter values, please refer to the Parameter Descriptions section of this document for further information.

Figure 21. GUI Dialog Box Global



The BMGR (Buffer Manager) tab is new beginning with the introduction of version 1.3 of the IP core described in this document. Also new in this version is the creation of both a "Core-Only" and "Core with Loop-Back Module" (provided as a reference design) implementation directories during the IP core generation process. And lastly, a "Hardware Evaluation (Lattice SCM25-900)" selection has been added that tailors the I/O for implementation to the LatticeSC Comunications Board for physical demonstration purposes.

Figure 22. GUI Dialog Box Buffer Manager

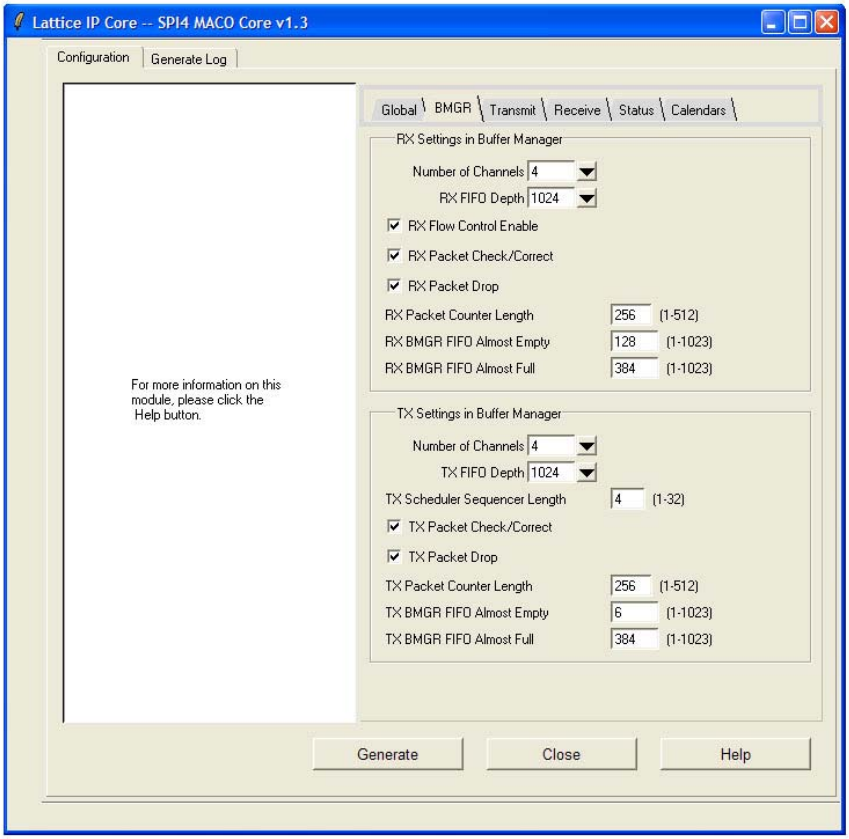


Figure 23. GUI Dialog Box Transmit

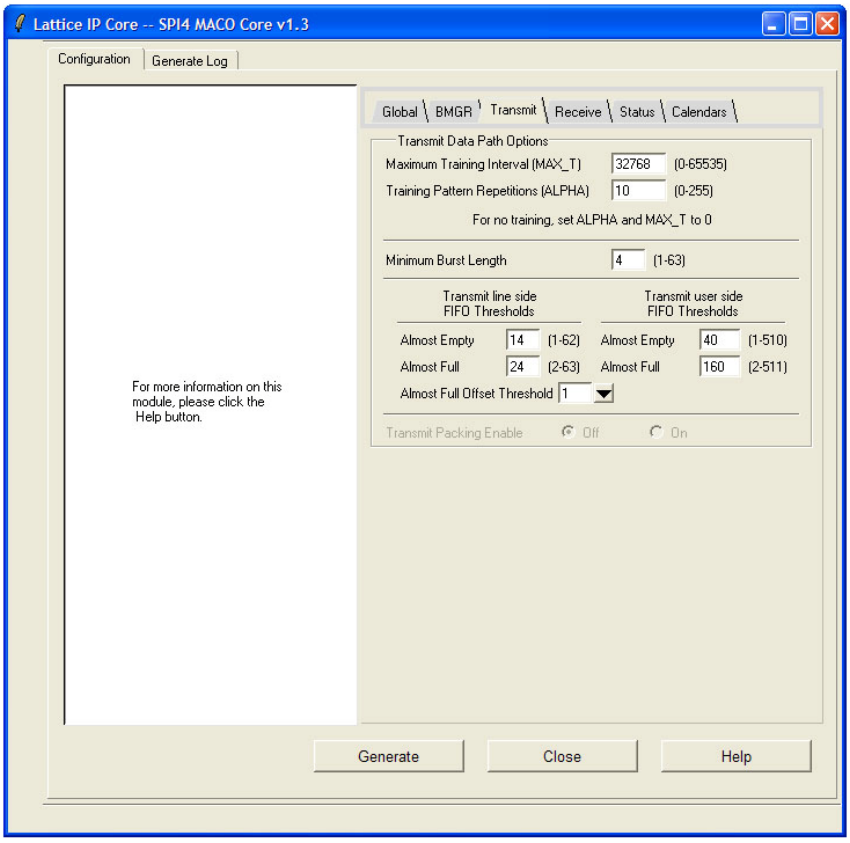


Figure 24. GUI Dialog Box Receive

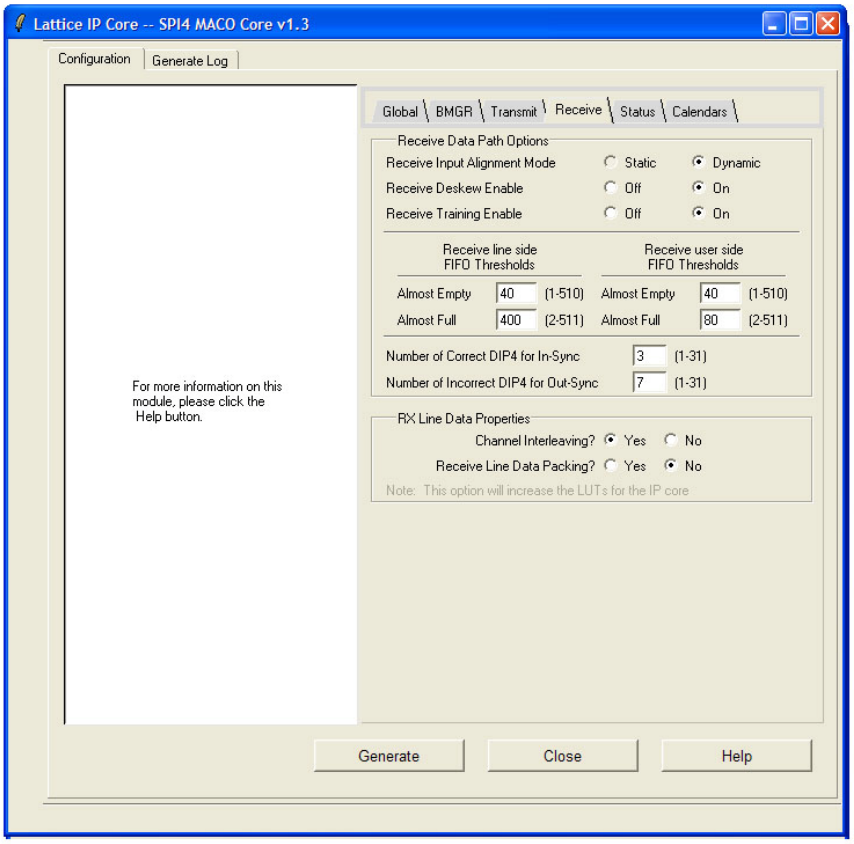


Figure 25. GUI Dialog Box Status

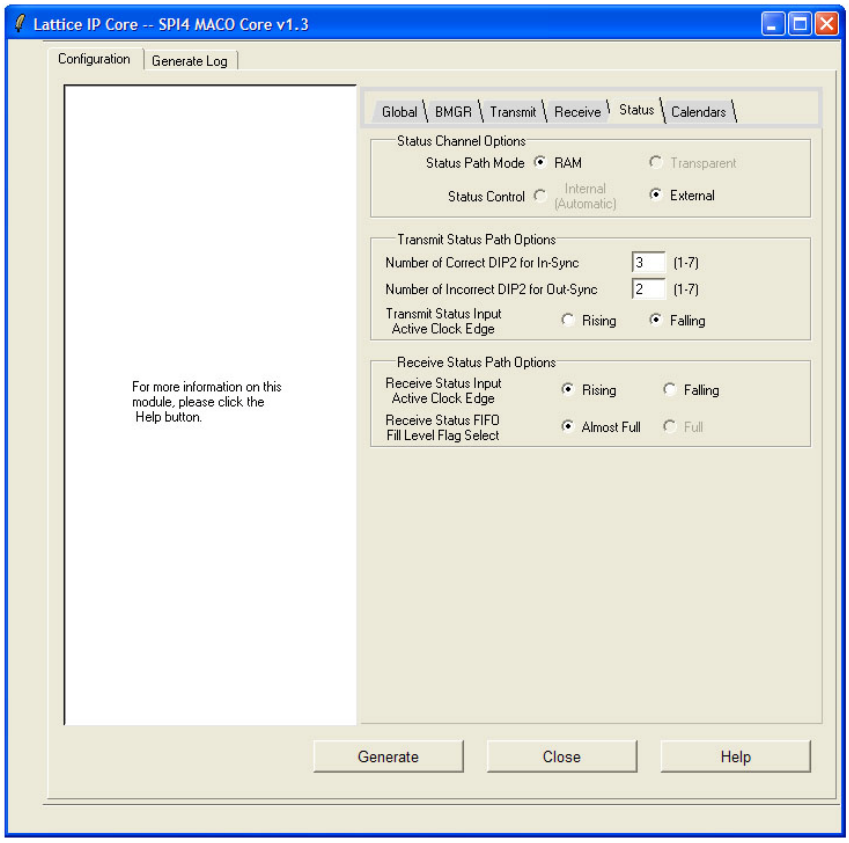
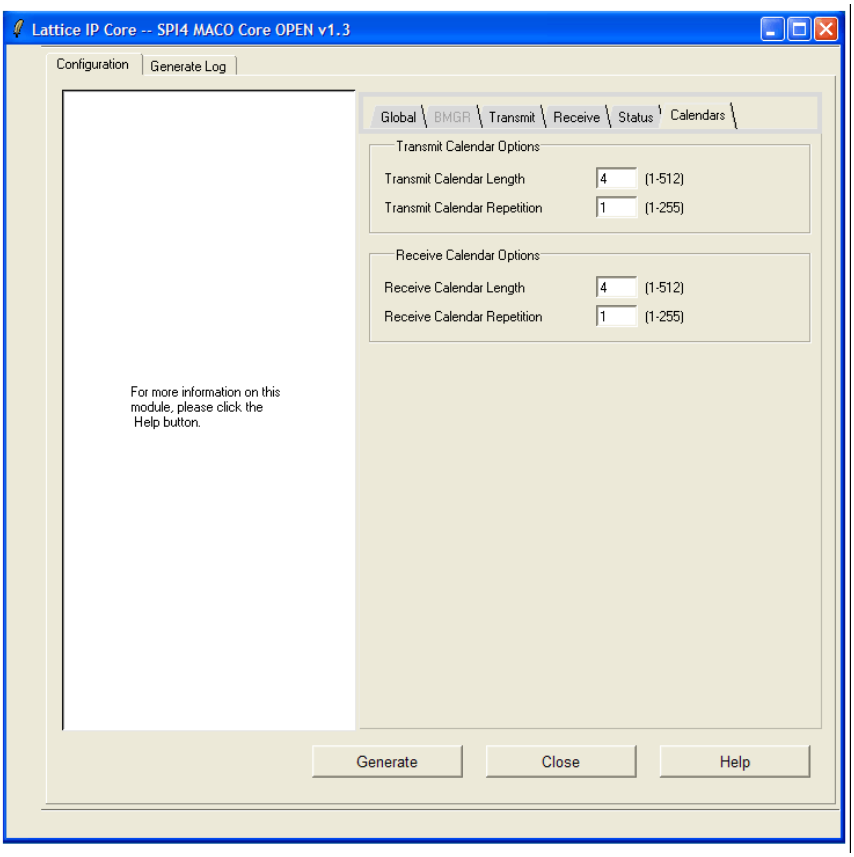


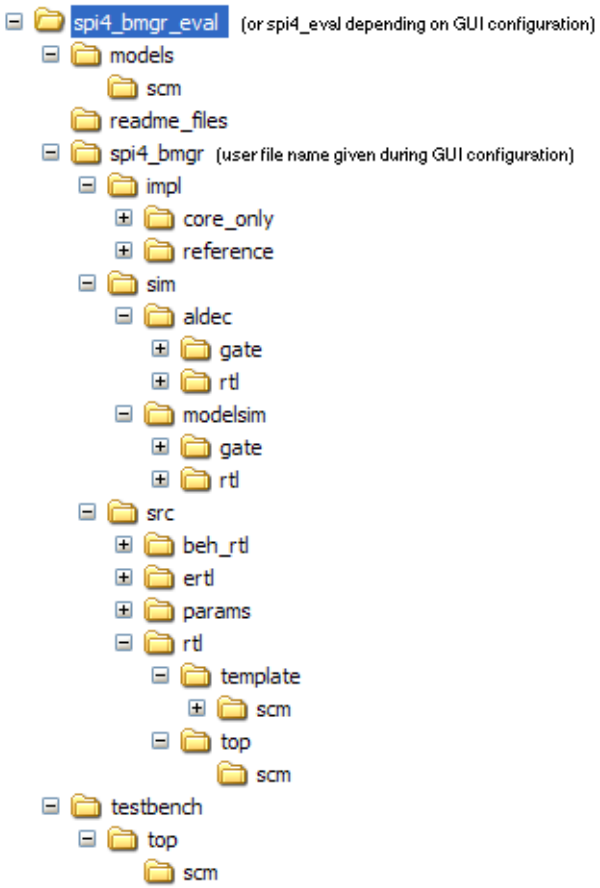
Figure 26. GUI Dialog Box Calendars



If the user prefers to use the evaluation simulation and project included in the IP design kit, the calendar options for TX and RX should match.

When the user clicks the **Generate** button, the configuration-specific IP core and supporting files are generated in the user's project directory. The directory structure of the generated files is shown in Figure 27.

Figure 27. IP Core Generated Directory Structure



The following files are generated in the user's project directory:

- <username>.lpc – IP parameter file, value collected from the GUI inputs.
- <username>.ngo – Synthesized and mapped IP core.
- <username>.v – Wrapper module used for core synthesis and simulation.
- <username>_bb.v – Black box module wrapper for synthesis.
- <username>_inst.v – Example of Verilog instantiation template to be included in the user's design.
- <username>_comp.vhd - Example of VHDL instantiation component to be included in the user's design.
- spi4_maco_core_beh.v – Behavioral simulation model for IP core configuration, needs for rtl simulation only, to be used with <username>.v. It might be in other names depending upon the GUI information, such as: spi4_maco_beh_local.v if the scm15 256pin package is selected. For how to run rtl simulation in evaluation package, please refer to the readme.htm under <spi4_eval> directory in the local generated IP core.
- s4dp_hc_beh.v – Behavioral simulation model for SPI4 MACO hard IP, needs for gate simulation only, to be used with <username>_eval.vo or <username>_eval.vho gate level netlist. For how to run gate simulation in evaluation package, please refer to the readme.htm under <spi4_eval> directory in the local generated IP core.

These are all of the files needed to implement and verify the IP core in a top-level design.

The following additional files providing IP core generation status information and command line generation capability are generated in the user's project directory:

- <username>_generate.tcl – Created when GUI "Generate" button is pushed, invokes generation.
- <username>_generate.log – ispLEVER synthesis and map log file.
- <username>_gen.log – IPexpress IP generation log file.

The \<spi4_eval> and subtending directories provide files supporting core evaluation. The \<spi4_eval> directory shown in Figure 27 contains files/folders with content that is constant for all configurations of the IP. The \<username> subfolder (\spi4_maco in this example) contains files/folders with content specific to the username configuration.

The \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. \<spi4_left>, \<spi4_right>, etc.

Locating the IP

The SPI4 IP core uses a mixture of hard and soft IP blocks to create the full design. This mixture of hard and soft IP requires the user to locate, or place, the core in a defined location on the device array. The hard blocks' fixed locations will drive the location of the IP. Table 9 lists the site names for the hard blocks on the different device arrays.

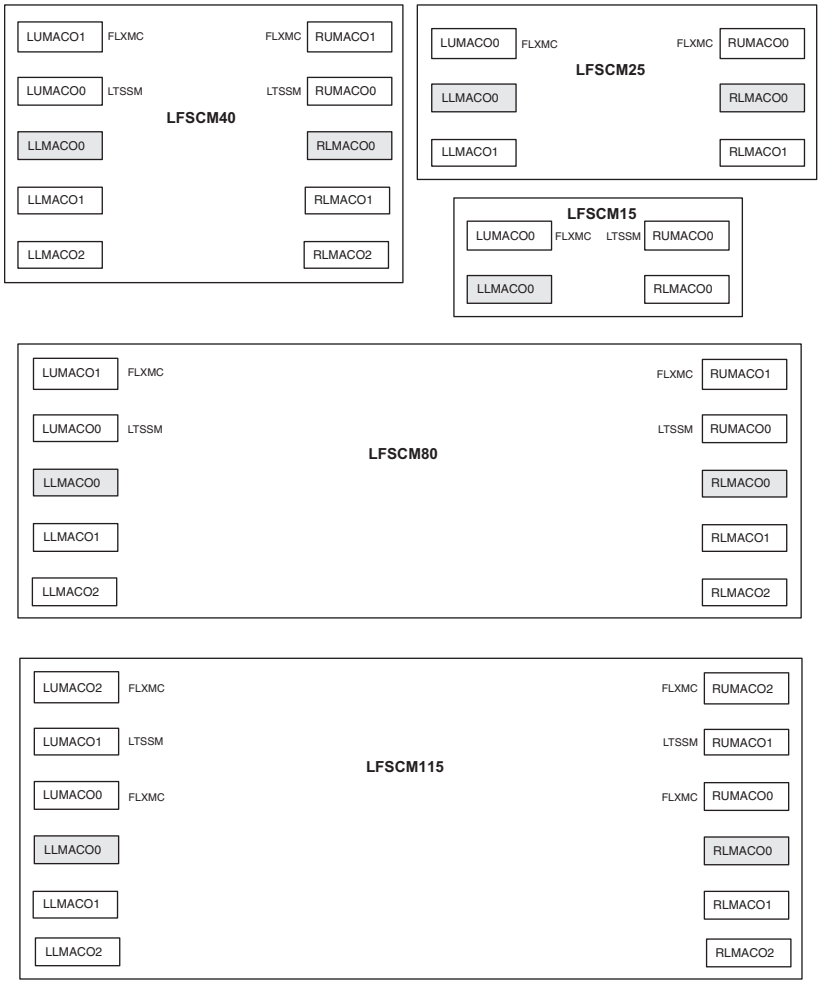
Table 9. SPI4 Hard-Core Location Site Names For Lattice Devices

Device	Hard-Core Location Site Name
LFSCM15	LLMACO0
LFSCM25	LLMACO0, RLMACO0
LFSCM40	LLMACO0, RLMACO0
LFSCM80	LLMACO0, RLMACO0
LFSCM115	LLMACO0, RLMACO0

Locating the LatticeSCM Hard Elements

Figure 28 provides a diagram of the LatticeSCM15, 25, 40, 80, and 115 devices with site names for the SPI4 hard-core elements. The sites shaded in gray represent valid SPI4 hard-core locations.

Figure 28. LatticeSCM Device Arrays and SPI4 MACO Sites



The user should select the SPI4 hard-core site location based on the package pinout and the location of the SPI4 interface on the board layout. In order to locate the SPI4 hard-core, a LOCATE preference in the .lpf file of isp-LEVER is used. The correct preference is generated automatically based on the GUI selection of either Left Side (Bank 6, 7) or Right Side (Bank 2, 3) and the selected device (LFSCM15, 25, etc.) for the project. The LOCATE preference can be found in the .lpf file inside the implementation directory created during IP core generation.

References and Related Information

- LatticeSC/M Family Data Sheet

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
August 2006	01.0	Initial release.
January 2007	01.1	Added "Errata" section.
		Added Appendix A.
		Updated "Selecting a System Data Clock Frequency ('SDCK') Receiver" section.
July 2007	01.2	References to LatticeSC changed to LatticeSCM throughout.
		Added appendix for LatticeSC/M support.
October 2007	01.3	Updated Performance and Resource Utilization table in LatticeSCM appendix.
		Added SPI4 MACO Design Kit and Core Generation text section.
January 2008	01.4	Parameters table - corrected range information for TXREP parameter.
March 2008	01.5	Updated S4TX - SPI4 Transmit Path figure.
April 2008	01.6	Corrected definitions for Receive Protocol errors (rxs4err[4:0]).
June 2008	01.7	Updated status path descriptions to match new asynchronous operation made available in version 1.2 of the IP core.
June 2008	01.8	Document title changed from "LatticeSCM SPI4.2 MACO Core User's Guide" to "SPI4 MACO Core User's Guide".
		Updated appendix.
July 2008	01.9	Updated Serial Memory Interface to Serial Management Interface
		Removed references to the core working in Static mode.
September 2008	02.0	Added "Appendix B. Buffer Manager" describing new Buffer Manager features made available in version 1.3 of the IP core. Add user information section on "Locating the IP".
		Updated Features list.
October 2008	02.1	Added Buffer Manager Mode Resource Utilization table to the Appendix for LatticeSCM FPGAs.
October 2008	02.2	User-side Signal Descriptions table - for TXCALCK and RXCALCK, changed frequency max. from 78 MHz to 125 MHz.
November 2008	02.3	Transmit status path clock (txstck vs txstatus_ck) corrections for ver1.2.
January 2009	02.4	Updated Minimum Burst Size - Burst Mode text section.
		Updated SMI Memory Descriptions table.
December 2009	02.5	Updated footnotes in Appendix tables.

Appendix A. Effects Of Disabling the Packing Feature

The LatticeSCM SPI4 MACO core transmitter does not allow packing in multi-channel applications (single channel applications do work). This section is intended to convey the effects of disabling this feature on line rate performance.

Even with packing turned off, the LatticeSCM SPI4 MACO core does a good job of packing the line as shown below. For users who must have a multi-channel packing feature, a soft-core approach is the recommended remedy.

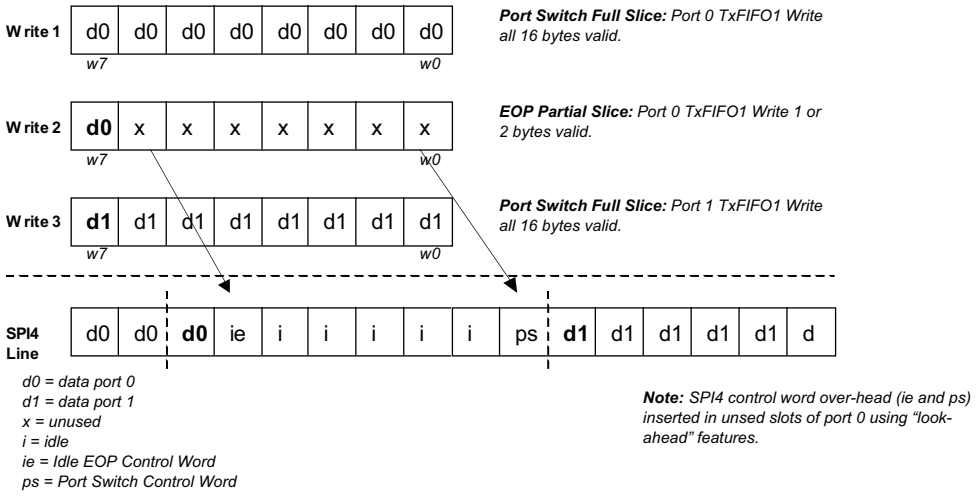
Packing Disabled

The user data interface to the transmitter is 128 bits (16 bytes) wide and for cases where the end-of-packet write does not result in a fully valid 128-bit TxFIFO1 entry, bandwidth here and on the SPI4 line will be wasted with the packing feature disabled. The amount of wasted bandwidth is roughly equal to the number of unused "word" slots in the final EOP write cycle of a given packet into TxFIFO1. There are two cases (best and worst) to consider for a more precise analysis:

EOP 1 Byte Valid

In this case, the amount of wasted bandwidth is equal to only the number of unused slots because the unused slot positions are used to send the appropriate SPI4 control word/s between bursts; no additional SPI4 over-head is required.

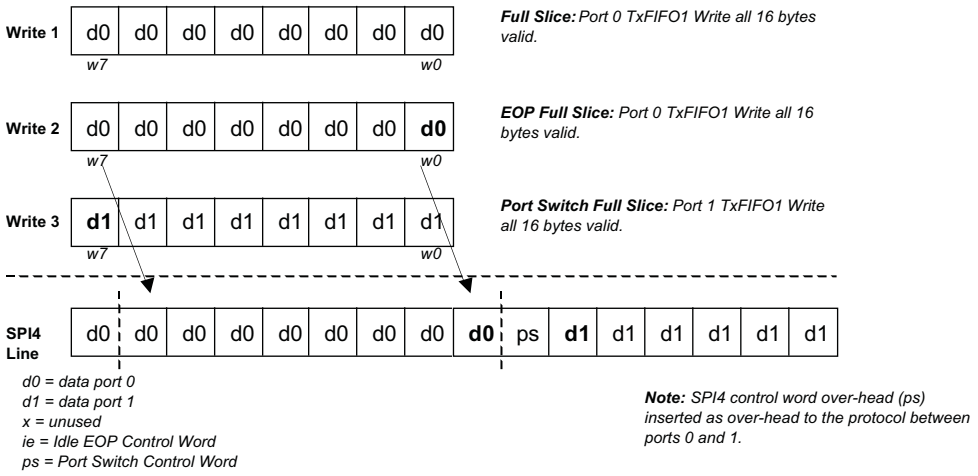
Figure 29. TxFIFO1 Write - 1 Byte Valid



EOP All Bytes Valid

In this case, there is no wasted bandwidth that can be attributed to the lack of packing because the write to TxFIFO1 is already full. The small amount of bandwidth inefficiency is equal to only the amount of over-head needed to support the SPI4 protocol. Additionally, the core does a good job of ensuring that the least number of control words scenario possible is the one that is used. In this case, the EOP and port switch (with or without SOP) will be made with a single SPI4 control word as opposed to two or three which is also possible but less efficient.

Figure 30. TxFIFO1 Write - 16 Bytes Valid



SPI4 Bandwidth Utilization Percentage

The tables below reflect the bandwidth utilization of the SPI4 line as a percentage for the best and worst case scenarios when packing is disabled for several different packet sizes.

Table 10. Bandwidth - Worst Case Packet Sizes

Packet Size	Clock Cycles	% Of SPI4 Line Bandwidth Used
17	2	53%
33	3	69%
49	4	76%
65	5	81%
81	6	84%

Table 11. Bandwidth - Best Case Packet Sizes

Packet Size	Clock Cycles	% Of SPI4 Line Bandwidth Used
16	1	94%
32	2	97%
48	3	98%
64	4	98.5%
80	5	99%

Appendix B. Buffer Manager

The Buffer Manager SPI4 IP core is a follow-on addition to the previously existing SPI4 IP core adding multiple physical buffers and a Bandwidth Manager to the current "shared" buffer design. It uses the existing hard-core based SPI4 design as a base upon which soft-gates are added providing the additional Buffer Manager features. It is offered in a seamless fashion as it relates to the existing IP core where both "shared" and now "multiple" buffer modes are available as selections from within the new SPI4 GUI. These new features are described below within the context of the existing "shared" buffer design previously described throughout the document.

Introduction

The Buffer Manager is made up of between 1 to 16 physical FIFOs, a transmit Bandwidth Manager, and a receive channel mapper. It targets, but is not limited to, applications where a relatively small number of buffers are required and serve independent physical interfaces. An example being a Bridge that aggregates 10 or 12 - 1Gbps interfaces into a single 10Gbps SPI4 interface. Individual FIFOs work well because the number of FIFOs is small and each provides an independent interface including independent clocks for the external interfaces. Applications requiring more than ~16 FIFOs is not practical from both a routing and memory resource point of view.

For the following sections, please refer to the "Block Diagrams" section of this document where applicable.

FIFO Buffers

The FIFO buffers used in the Buffer Manager (BM) each contain a packet checking function (packet_checker) and a standard asynchronous mode FIFO controller equipped with extra features such as error drop, graceful over-flow drop, and an EOP counter in support of "store and forward" switching architectures. Depending upon how the IP core is configured via the GUI, some logic functions are removed in order to save logic. The following describes behavior in the three possible operational modes:

1. **No Drop and No Check/Correct** – In this mode, the packet checker module is not equipped to save logic and none of the other features are supported (check, correct, error drop, over-flow drop, etc.). All packet-checking errors (missing SOP - MSOP, etc.) are forced inactive. In this mode, packets flow through the buffer in a "cut-thru" manner meaning that they flow through the buffer immediately without waiting for the end-of-packet.

2. **No Drop With Packet Check/Correct** – In this mode, the packet checker module is equipped and provides the packet checking/correcting operations. In addition, although packet error drop is not supported, graceful over-flow drop is supported. Two corrective actions can occur in check/correct mode: a) insert a missing EOP and b) drop all data before an EOP that was missing an SOP. All packet-checking errors (missing SOP - MSOP, etc.) are active in this mode. In this mode, packets flow through the buffer in a "cut-thru" manner meaning that they flow through the buffer immediately without waiting for the end-of-packet. This is the low latency mode.

3. **Packet Drop** – In this mode, the packet checker module is equipped providing the features mentioned above and in addition, a packet error drop feature is added. In this mode, packets flow through the buffer in a "store & forward" manner meaning that they do not flow out of the buffer until the complete packet is stored in the buffer. Store and Forward operation is required to support the packet error drop feature. Packet error checking takes place as described above. Packet correction works the same for a missing SOP (drop all data before an EOP that was missing an SOP) but a missing EOP is handled by dropping the entire fragmented packet and is therefore never transmitted.

    - **Missing SOP** – EOP followed by EOP - drop all data up to the next SOP (could be many EOPs, etc.), inhibits FIFO write while data falls on the floor.
    - **Missing EOP** – SOP followed by SOP - drop SOP encapsulated data, begin writing after second SOP into FIFO. If another SOP detected, again drop SOP encapsulated data.
    - **FIFO Full** – Drop current packet up to EOP. Start on next valid SOP. If it results in over-flow, drop it also and repeat forever. *** Graceful drop on over-flow is not dependent on drop mode. Only that the packet checker is equipped. ***
    - **ABT** – Drop current packet. In the transmit direction this signal driven by TX_ABT. In the receive direction it is driven directly by the SPI4 protocol through decoding of the Abort control word.

– **ERR** – Drop current packet. The packet will be dropped at the point where error signal is sampled active while a packet is in progress. All following data up to an EOP will also be dropped. Normal operation begins with the next valid SOP. In the transmit direction this signal is driven by TX_ERR. In the receive direction it is driven directly by the SPI4 DIP4 error and protocol error signals.

*** Note that for applications where aggregation is required and multiple SPI4 channels are passed through a single buffer, that packets must be written into the buffer in whole (SOP -> EOP) or the packet checker must not be enabled. If enabled and packets are written into the buffer in segments, consecutive SOP and or EOPs are possible and will trigger unwanted corrective action.*

*** In the transmit direction, the EOP counter is always in the circuit and never optioned out because it is needed in support of the Min_Burst feature (e.g., at least 64 bytes available or an EOP exists = begin transfer).*

*** Receive & Transmit Packet Counter Length: This GUI item paramterizes the length of the end-of-packet counter which can be cusotmized based on the size of the FIFO selected and expected length of the smallest packet. For example, a 32K buffer / 64Byte min packet = 512.*

Bandwidth Manager - Transmitter

Overview

One of the core components of the Bandwidth Manager is a "Scheduler". The main function of the Scheduler is to manage transmit bandwidth commensurate with the subscribed bandwidth for each SPI4 channel. The Bandwidth Management capability is especially important in Packet-Over-SONET applications where Virtual ConcATenation (VCAT) can be used to size any number of channels from as small as STS-1 (51mbps) to as large as STS-192 (10Gbps) in 10Gbps systems. Many of these systems involve Bridging applications where a given packet may arrive in full at a 10Gbps rate over a XAUI interface for example but needs to be "played out" at a much slower rate, over a much longer period of time, on an SPI4 channel (eg. 150mbps) towards a Framer device. Typically, dynamic channel re-provisioning must also be supported where STS-1s can be added or subtracted to a given channel while the system is live without causing errors or interrupting service on other channels. Supporting these types of applications requires bandwidth management and per-channel buffers.

There are a number of ways to accomplish the scheduling task and in particular the bandwidth management function. One way transmit scheduling can be accomplished is alluded to in the SPI4.2 specification through use of the Calendar sequence that allows more or less status updates based on a specific channels bandwidth allocation. Note that synchronization between status channel processing and data channel scheduling need not occur; it is simply the order and frequency at which given channels are scheduled that the data path has need of and in common with the status path. For example, the status update for a given channel may only require a single status clock cycle while on the data path, a scheduled channel for transmission may take at minimum 4 cycles without an EOP and as many cycles as is necessary to satisfy MAX_BURST1 or 2.

The scheduler provided in this design works in similar fashion based on a direct channel sequence specification programmed into a local RAM. The scheduler is, however, provided with it's own sequence RAM separate from the one used to drive the SPI4 Calendar sequence allowing for greater flexibility. Suppose an example 2.5G VCAT system where minimum granularity is STS-3's (155Mhz) and a maximum of 10 SPI4 channels must be supported. The status Calendar RAM could be programmed to a length of 10 with single entry status reporting while the Scheduler sequence length could be programmed to a length of 16 for example where each entry in the RAM represents a 1/16 portion of the total 2.5G bandwidth. Channels with more or less bandwidth (number of STS-3's) would be represented in the sequence more or less often. This implementation provides the ability to support simple Round Robin (RR) or pseudo Weighted Round Robin (WRR) arbitration functionality that is often called for in these applications. More entries in the RAM in affect increase the weight of a given channel in the context of a true WRR arbitration scheme.

One consideration to make with the Scheduler Sequencer architecture is the time that can be wasted in polling channels that may have nothing to send or their status reflects inability to send (satisfied). The view on this consideration is that it adds a small amount of latency but does not have a negative impact on bandwidth allocations of other channels having data to send. This view is that as long as the maximum wasted polling time for a given chan-

nel is less than or equal to the minimum bandwidth allocation possible for a given channel, it is of no concern. In other words, each channel in the Scheduler sequence is allocated a specific amount of bandwidth and if there is nothing to send, then it is ok to use some of the bandwidth allocated to it without affecting the bandwidth allocations of the other channels. This design requires only a single clock cycle to poll channels, which equates to around 16Mbs, or 1/800 of the bandwidth available on a typical 12.8Gbs SPI4 line.

Functional Description

The over-all scheduler will be composed of two modules: a Channel Poll Sequencer (CPS) and a 32 location Channel Descriptor RAM (CDR). The content of the CDR is addressed in linear order up to a programmable length of from 1 - 32. As each RAM location is addressed, a 20b descriptor becomes available containing a 4b channel identifier (ID), and the MAX_BURST 1 and 2 parameters for the channel. The channel ID number will be used to poll for that channels status through the Status RAM and to select the correct transmit FIFO for determining the availability of data to send (based on FIFO fill levels). Transmission for the given channel begins if: 1) there is enough data available to meet the minimum burst criteria and 2) current status for the channel is in the hungry (MB2) or starved (MB1) state. Transmission for a given channel continues until the selected Max_Burst value is achieved or the source FIFO drains below the Min_Burst value without the presence of an EOP. Otherwise, the Schedulers counter increments addressing/pollong the next potential channel for transmission. The minimum burst criteria refers to the existing Minimum Burst capability supported by the existing IP core now requiring co-operative loading by the Scheduler in order to function properly. This requires a minimum burst FIFO threshold setting obtained from 'TxBLEN[]' and an indication of whether or not at lest one full packet exists in the FIFO. User logic is not required to follow any special procedure while witting data into the Buffer Manager in order to support the Minimum Burst mode feature. This is all managed internally.

Channel number entries written into the CDR by the user are limited to the physical range of the maximum number of FIFOs supported, which are 16. Each physical FIFO however, supports a complete 8b range of SPI4 channel input addresses upon which to identify a given packet - the logical channel number. Providing this capability removes the physical channel ID limitation and also allows for aggregation of multiple channels into a single physical FIFO providing a method to separate them at the far end.

Minimum Burst Mode – A burst counter is used to maintain proper burst boundaries based on TX_BLEN[] in order to ensure that no burst shorter than this is transmitted without the presence of an EOP. The burst counter is used in both Store and Forward and Cut-Thru modes and is reset on each EOP to begin a new alignment. The design also keeps track of the burst boundary for cases where termination occurs due to Max_Burst levels and conclusion occurs at the proper TX_BLEN[] boundary. The valid range for TX_BLEN[] is from 4 - 64. In addition, when using the minimum burst mode, the TX_AFE_THRHD[] must be set to at least 1 greater than TX_BLEN[].

Lastly, TX_BLEN[] now appears as primary I/O in addition to previously only MACO SMI memory (see "SMI Memory Address Map & Bit Assignments"). The same value must be used in both places and is common to all channels.

Performance – Back-2-Back FIFO Service

- Back-to-back performance on burst termination (TX_BLEN[])
- Back-to-back performance on Max Burst termination (w/no EOP)
- One cycle response on empty (non-drop mode) and End-Of-Packet - cases that cannot be predicted easily through look-ahead means. Considering ~30% over-speed in the system data clock domain relative to the SPI4 line, this one cycle response does not adversely affect the throughput of the system.

Channel Mapper - Receiver

Because there is no scheduling function associated with the receiver, the design complexity required in the receive direction is minimized. The affected areas are limited to multiple channel buffering, channel identification, and status channel encoding. Listed below are some of the attributes of the design:

- The Receive Data Processor (RDP) continually reads from the "shared" buffer of the existing SPI4 IP core

(RxFIFO2), while it is not empty, and writes the data unconditionally to the appropriate physical FIFO based on the assignment of the receive channel map RAM. Internally managed flow control via the SPI4 status channel ensures that the buffers do not over-flow. In the event that the far-end sends more data then the buffers can handle, the Buffer Manager will drop over-flow packets cleanly as long as the packet checker function is equipped.

The RDP must keep track of the current channel address while reading since a channel switch can occur at any point while reading due to SPI4 segmentation. Writes to all physical buffers occurs with the same System Data Clock (SDCK) and data bus. Reads of the receive FIFOs (RxFIFOs) on the user side occurs with independent read clocks and data buses.

- A mapping of the incoming SPI4 channel to physical buffer occurs through the 64 x 4 Channel Map Ram (RXCHM) in the RDP allowing any channel or number of channels to be mapped to the same buffer in support of aggregation. When a connecting multiple SPI4 channels to the same buffer, either the SPI4 line must not be interleaved or the packet check/correct feature cannot be enabled.

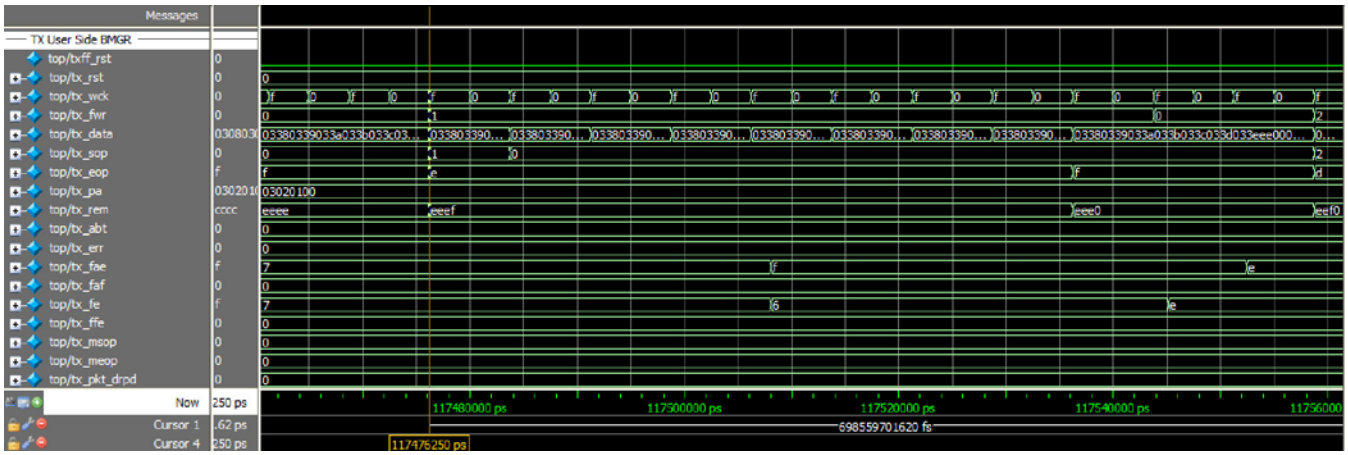
  A 4-bit bus and associated control signals are also provided to allow initialization of the RXCHM RAM.

- Status encoding occurs via the receive FIFO fill levels and corresponds to the division of the buffer into three parts as defined in the OIF specification. Note that the almost empty and full thresholds are user programmable. Additionally, an input control is available to disable flow-control. When disabled, a constant "starved" status is sent to the far-end regardless of the FIFO fill levels.

Interface Waveforms

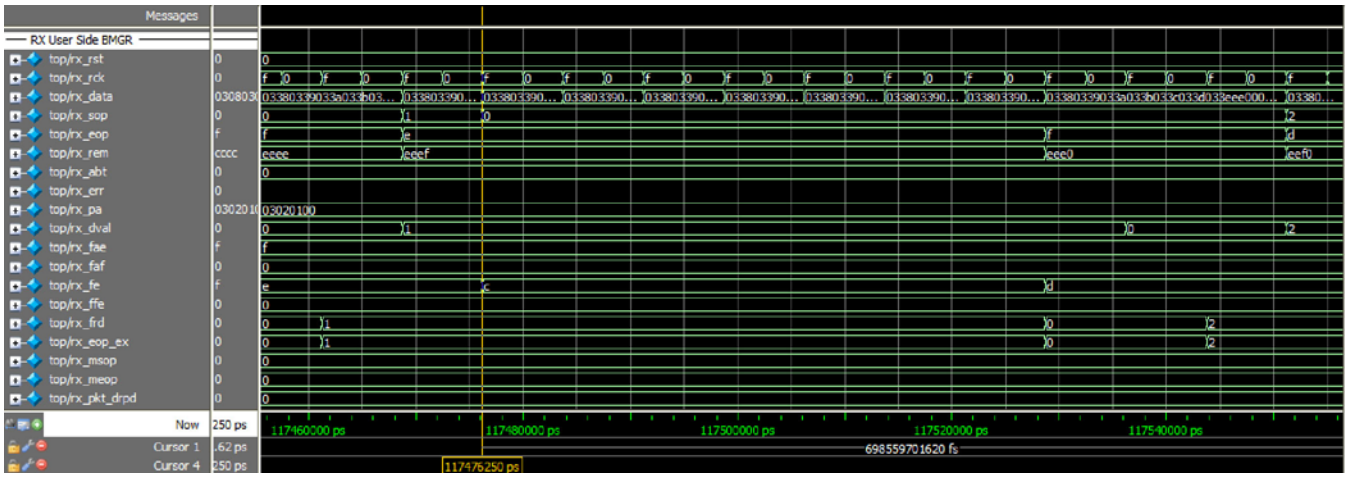
Transmit

Figure 31. Transmit Buffer Manager Timing



Receive

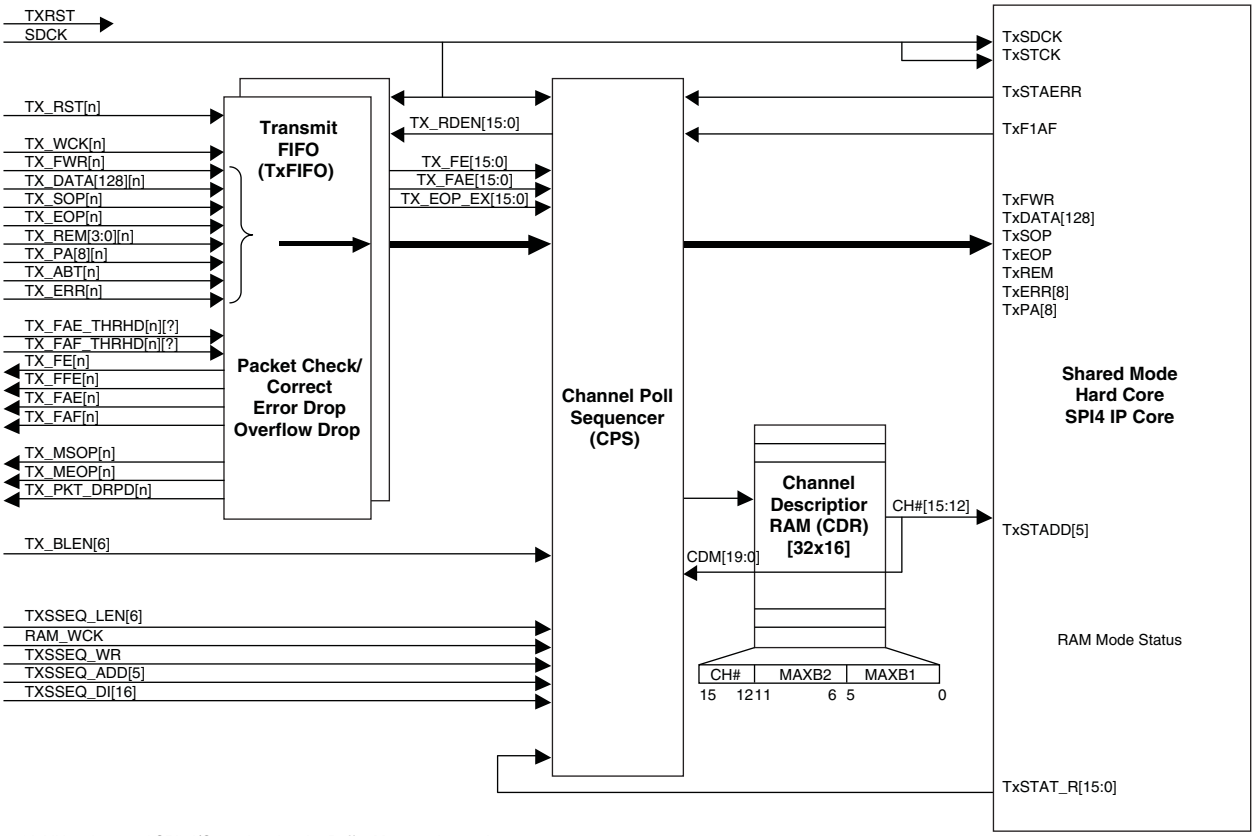
Figure 32. Receive Buffer Manager Timing



Block Diagrams

Transmit Buffer Manager

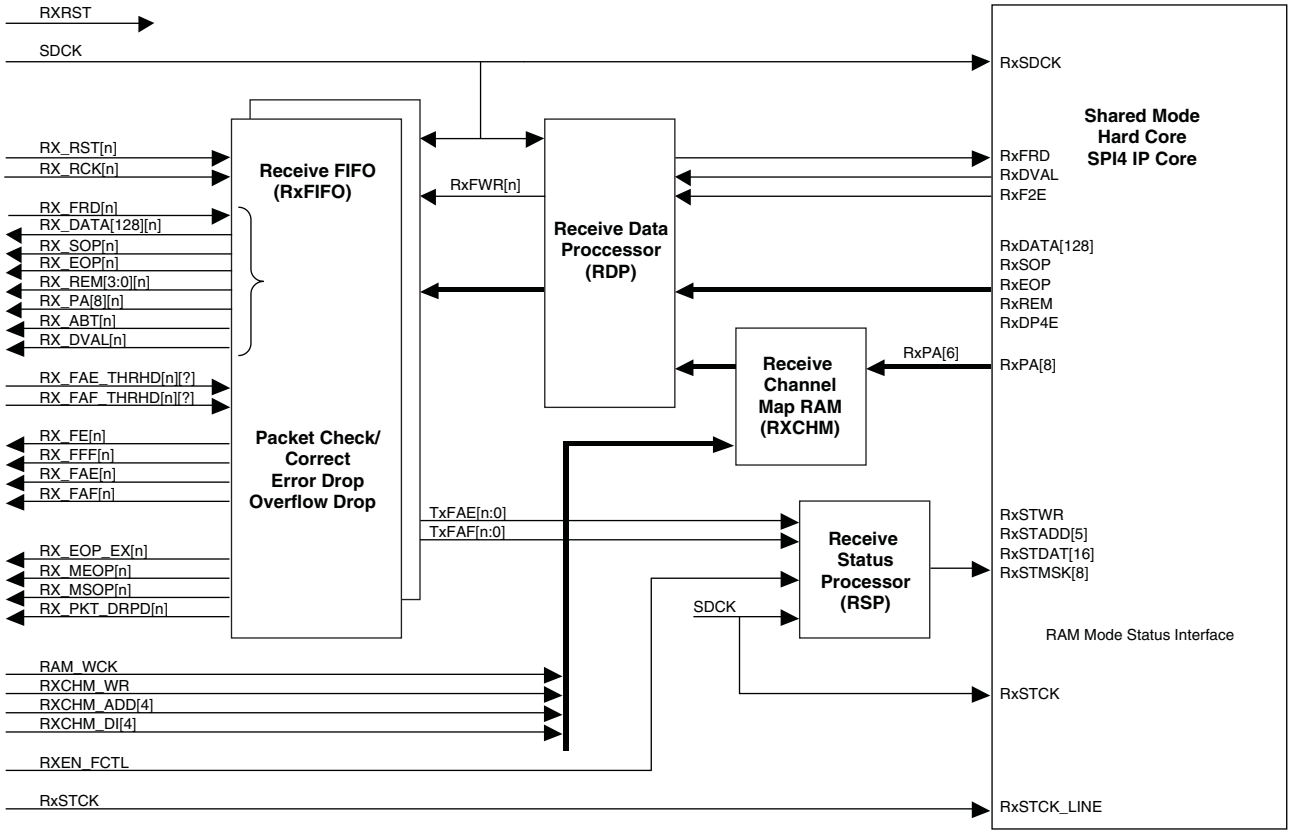
Figure 33. Transmit Buffer Manager



*Additional general SPI4 I/O unrelated to the Buffer Manager is not shown.

Receive Buffer Manager

Figure 34. Transmit Buffer Manager



* Additional general SPI4 I/O unrelated to the Buffer Manager is not shown.

Signal Descriptions

Table provides I/O descriptions for the SPI4 IP core when configured for use in the Buffer Manager mode. SPI4 line side I/O remains the same.

Table 12. User Side Signal Descriptions

Signal Name	Direction	Description
S4RX/S4TX Common Signals		
SDCK	Input	**System Data Clock** - Should have over-speed relative to the SPI4 receive line clock / 4 (i.e. RCLK/4). Also used in user domain logic to transfer data to and from the IP core. See the Special Clock Requirements section of this document for 'SDCK' and 'SDCK2' phase requirements.
SDCK2	Input	**System Data Clock Multiplied By 2** - Should have over-speed relative to the SPI4 receive line clock (i.e. RCLK/2). Not used in user domain logic due to high speed (325MHz maximum rate).
RAM_WCK	Input	**RAM Write Clock** - This clock signal is used to synchronously read/write the Receive Channel Map and Transmit Schedule Sequencer RAM. The frequency of this clock is not critical since once initialized, access is not expected.

Table 12. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
GRST_N	Input	**SPI4 MACO LatticeSCM Global Core Reset** (active low).
S4RX Common Signals		
RxRST	Input	**Receive Reset** - This signal when active causes a reset of the entire S4RX receiver.
RXF2AE	Output	**Receive FIFO Almost Empty** - This signal when active (=1), indicates RxFIFO2 is almost empty as determined using synthesis parameter RXF2AETHRSH[8:0] above. This signal is used by the S4RXS block but may be used by the user side interface. See also output signal RXF2AF below. This is a debug signal in this mode - connection is not required.
RXF2AF	Output	**Receive FIFO Almost Full** - This signal when active (=1), indicates RxFIFO2 is almost full as determined using synthesis parameter RXF2AFTHRSH[8:0] above. Also, when this signal is asserted (edge), the "Satisfied" condition can optionally be transmitted on the RXSTATUS[1:0] channel until the corresponding RxFAE signal is asserted (edge). This is a debug signal in this mode - connection is not required.
RXF2E	Output	**Receive FIFO Empty** - This status signal when active (=1), indicates RxFIFO2 is empty and the user side should stop reading. This is a debug signal in this mode - connection is not required.
RXF2FE	Output	**Receive FIFO Full Error** - This error signal when active (=1), indicates RxFIFO2 is full and should be considered to have over-flowed.
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. (Clock=SDCK, at least 1 cycle active).
RXAERR	Output	**Receive Alignment Error** - This signal when active (=1), indicates the S4RXD block has not achieved alignment (consecutive number of correct DIP4 words). (Clock=SDCK, at least 2 cycles active)

Table 12. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
RS4ERR[4:0]	Output	**Receive SPI4 Line Protocol Violation Errors** - These signals when active (=1) indicate that the S4RXD block has received a sequence of data and control words that are in violation of the SPI4 protocol. All signals are synchronous to the SDCK clock domain and will be active for at least 1 full cycle. Note: RxS4ERR[4] must be sampled in the SDCK clock domain before being analyzed because it is not a direct output from a FF. The others are direct FF outputs and do not require sampling. RS4ERR[4] - A valid write to RxFIFO2 where less than 16 bytes are marked valid without an EOP active. SPI4 burst aligned and "or"ed into the 'RXDP4E' 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.
RXF1F	Output	**Receive FIFO1 Full Error** - This signal when active (=1) indicates that FIFO1 has become full and has over flowed. With an operational status channel, this error should never occur.
RXF1E	Output	Need to find out what this signal is in new design empty ? or err
RXF1_PARERR	Output	**Receive FIFO1 Parity Error** - This signal when active (=1) indicates that a parity error was detected on data read from RxFIFO1.
RxRSTAIL	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. (Clock=rxs4ls2_ck, 4 cycles active)
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. (Clock=rxs4ls2_ck, 4 cycles active)
RxDSKWD	Output	**Receive De-Skewed** - This signal when active indicates that the de-skew block has successfully de-skewed the input SPI4 line. (Clock=rxs4ls2_ck, at least 4 cycles active)

Table 12. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
RxAIL_LOCK	Output	**Receive AIL Locked** - This signal when active (=1) indicates that all AIL circuits are locked. (Clock=rxs4hs_ck, at least 4 cycles active)
RxRUNAIL	Input	**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.
RxLOOP	Input	**Receive Loop** - Unused in this version of the core.
RxDCNTL[8:0]	Input	**Receive Delay Control** - This array specifies the data delay value relative to clock in Static mode - currently unused since static mode is not offered.
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.
RXS4LS2_CK	Output/(Input)	**Receive SPI4 Low-Speed Divide By 2 Clock** - This is the divide by 2 version of the SPI4 receive line clock. An output under almost all configurations. An input only when configured for SCM15 device with 256p package.
S4RX Buffer Manager Related Signals		
RX_RST[N:0]	Input	**Receive Reset** - This signal when active (=1) resets channel N FIFO receive controller and surrounding.
RX_RCK[N:0]	Input	**Receive Read Clock** - Channel N read clock. All interface signals associated with the receive side of the Buffer Manager are synchronous to this clock.
RX_FRD	Input	**Receive FIFO Read** - This signal when active (=1), causes a read of the channel N FIFO buffer.
RX_DATA[(N*128)-1:0]	Input	**Receive Data** - Channel N read data. Note packed array format - channel 1 assigned to bits 127:0, channel 2 assigned to bits 255:128, etc.
RX_SOP[N:0]	Input	**Receive Start Of Packet** - The corresponding data slice contains the start of a packet (a=1) for channel N.
RX_EOP[N:0]	Input	**Receive End Of Packet** - The corresponding data slice contains the end of a packet (a=1) for channel N.
RX_REM[(N*4)-1:0]	Input	**Receive Remainder** - Channel N remainder. Indicates the byte lane position of the last valid data byte (meaningful if corresponding data slice contains an EOP). A value of 0 = 1 byte, left justified MSB = b127-b120). A value of 15 = all bytes valid.
RX_ABT[N:0]	Input	**Receive Abort** - The packet was received with an SPI4 "Abort" encoding from the far-end. (a=1). Will be active at the end of a packet co-incident with an EOP.
RX_ERR[N:0]	Input	**Receive Error** - May be unconnected in version (ver1.3) of the IP core. A DIP4 or Invalid Burst (RS4ERR[4]) error was detected in the SPI4 IP core during packet reception (a=1). This error signal should be checked through packet reading qualified by rx_dval (due to SPI4 segmentation).
RX_PA[(N*8)-1:0]	Input	**Receive Port Address** - Channel number. Note packed array format - channel 1 assigned to bits 7:0, channel 2 assigned to bits 15:8, etc.

Table 12. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
RX_DVAL[N:0]	Output	**Receive Data Valid** - This signal when active (=1) qualifies RX_DATA[] as being valid. There is no fixed relationship to the RxFRD input and this RxDVAL signal. RxDATA[] should be ignored when this is signal is inactive.
RX_FAE[N:0]	Output	**Receive FIFO Almost Empty** - This signal when active (=1), indicates that the FIFO buffer for channel N is almost empty as determined using synthesis parameter RXFIFO_AE_THRD [] obtained during GUI capture.
RX_FAF[N:0]	Output	**Receive FIFO Almost Full** - This signal when active (=1), indicates that the FIFO buffer for channel N is almost full as determined using synthesis parameter RXFIFO_AF_THRD [] obtained during GUI capture.
RX_FE[N:0]	Output	**Receive FIFO Empty** - This status signal when active (=1), indicates that the FIFO buffer for channel N is empty.
RX_FFE[N:0]	Output	**Receive FIFO Full Error** - This status signal when active (=1), indicates that the FIFO buffer for channel N is full. This is an error condition
RX_EOP_EX[N:0]	Output	**Receive End Of Packet Exists** - This status signal when active (=1), indicates that the FIFO buffer for channel N has at least one full packet inside it.
RX_MSOP[N:0]	Output	**Receive Missing Start Of Packet** - This error signal when active (=1), indicates that the packet checker logic for channel N has declared a missing EOP error.
RX_MEOP[N:0]	Output	**Receive Missing End Of Packet** - This error signal when active (=1), indicates that the packet checker logic for channel N has declared a missing SOP error.
RX_PKT_DRPD[N:0]	Output	**Receive Packet Dropped** - This error signal when active (=1), indicates that the packet checker logic for channel N has dropped a receive packet due to error.
RX_FAE_THRHD[(N*LOG2(FIFO_SIZE))-1:0]	Input	**Receive FIFO Almost Empty Threshold** - Channel N FIFO buffer almost empty threshold value in 16-byte increments. Note packed array format where lengths depend on number of channels and size of the FIFOs.
RX_FAF_THRHD[(N*LOG2(FIFO_SIZE))-1:0]	Input	**Receive FIFO Almost Full Threshold** - Channel N FIFO buffer almost full threshold value in 16-byte increments. Note packed array format where lengths depend on number of channels and size of the FIFOs.
RXCHM_WR	Input	**Receive Channel Map Write** - This signal when active (=1) causes synchronous write operations of the receive channel map RAM using input clock signal RAM_WCK. Write are allowed back-to-back on consecutive clock cycles.
RXCHM_ADD[3:0]	Input	**Receive Channel Map Address** - Address index into the 64x4 receive channel map RAM.
RXCHM_DI[3:0]	Input	**Receive Channel Map Data In** - Four bit data bus into the 64x4 receive channel map RAM.

Table 12. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
RXEN_FCTL	Input	**Receive Enable Flow Control** - This signal when active (=1) enables flow control on the output SPI4 receive status interface. When inactive, the status for all channels is forced to starved regardless of the fill level of the associated FIFO buffers.
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 = 125MHz).
RXSTCK_LINE	Input	**Receive Status Line Clock** - This clock signal is used to clock status off-chip onto the SPI4 line. See also signal description 'rxstck'.
RXFDIP2E	Input	**Transmit Force DIP2 Error** - This signal when active (=1) causes the S4RXS to insert DIP2 parity errors. Framing remains valid.
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.
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 RXSTCK 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.
S4TX Common Signals		
TxRST	Input	**Transmit Reset** - This signal when active (=1) causes a reset of the entire S4TX transmitter.
TXF1FE	Output	**Transmit FIFO1 Full Error** - This error signal when active (=1), indicates TxFIFO1 is full.
TXF1AE	Output	**Transmit FIFO1 Full Error** - This error signal when active (=1), indicates TxFIFO1 is full. This is a debug signal in this mode - connection is not required.
TXF1AF	Output	**Transmit FIFO1 Almost Full** - This signal when active (=1), indicates TxFIFO1 is almost full as determined using synthesis parameter TxAFTHRSH[] below. This is a debug signal in this mode - connection is not required.

Table 12. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
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.
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.
TxFIDLE	Input	**Transmit Force Idles** - This signal when active causes the S4TX transmitter to insert idle control words at the soonest available boundary.
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 DIP4 parity
TXREM_ERR	Output	**Transmitter Remainder Error** - This signal when active (a=1) indicates that TxFIFO1 was written with a remainder of other than 4'b1111 during a cycle when 'TXEOP' was not active; a clear error situation. (Clock=SDCK, 1 cycle active)
TXBURST_ERR	Output	**Transmitter Remainder Error** - This signal when active (a=1) indicates that TxFIFO1 was written with a remainder of other than 4'b1111 during a cycle when 'TXEOP' was not active; a clear error situation. (Clock=SDCK, 1 cycle active)
TXF2_PARERR	Output	**Transmitter FIFO2 Parity Error** - This signal when active (a=1) indicates that a parity error has been detected on a read operation of TxFIFO2.
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	Output	**Transmit SPI4 Low Speed Divide By 2 Clock** - This is the internal divide by 2 version of S4TXHS_CK. General purpose output.
S4TX Buffer & Transmit Scheduler Related Signals		
TX_RST[N:0]	Input	**Transmit Reset** - This signal when active (=1) resets channel N transmit FIFO controller and surrounding.
TX_WCK[N:0]	Input	**Transmit Write Clock** - Channel N write clock. All interface signals associated with transmit side of the Buffer Manager are synchronous to this clock.
TX_DATA[(N*128)-1:0]	Input	**Transmit Data** - Channel N write data. Note packed array format - channel 1 assigned to bits 127:0, channel 2 assigned to bits 255:128, etc.

Table 12. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
TX_SOP[N:0]	Input	**Transmit Start Of Packet** - The corresponding data slice contains the start of a packet (a=1) for channel N.
TX_EOP[N:0]	Input	**Transmit End Of Packet** - The corresponding data slice contains the end of a packet (a=1) for channel N.
TX_REM[(N*4)-1:0]	Input	**Transmit Remainder** - Channel N remainder. Indicates the byte lane position of the last valid data byte (meaningful if corresponding data slice contains an EOP). A value of 0 = 1 byte, left justified MSB = b127-b120). A value of 15 = all bytes valid.
TX_ABT[N:0]	Input	**Transmit Abort** - Transmit packet abort control ending the packet and causing the corresponding packet to be transmitted with an SPI4 "Abort" encoding (a=1) when the "packet drop" feature is Not enabled. If enabled, the entire packet will be dropped before the SPI4 line.
TX_ERR[N:0]	Input	**Transmit Error** - Unconnected in version (ver1.3) of the IP core. Transmit packet error control causing the corresponding packet to be dropped when the "packet drop" feature is enabled. This signal can go active anywhere during the time when the packet is being written into the FIFO buffer. It will be latched and the packet dropped when the EOP arrives. If the "packet drop" feature is not enabled, the input does nothing.
TX_PA[(N*8)-1:0]	Input	**Transmit Port Address** - Channel number. Note packed array format - channel 1 assigned to bits 7:0, channel 2 assigned to bits 15:8, etc.
TX_FWR[N:0]	Output	**Transmit FIFO Write** - When active (=1), the corresponding 8-word data slice associated with TX_DATA[] is written into the FIFO buffer for channel N. If inactive, TX_DAT[] is ignored.
TX_FAE[N:0]	Output	**Transmit FIFO Almost Empty** - This signal when active (=1), indicates that the FIFO buffer for channel N is almost empty as determined using synthesis parameter TXFIFO_AE_THRD [] obtained during GUI capture.
TX_FAF[N:0]	Output	**Transmit FIFO Almost Full** - This signal when active (=1), indicates that the FIFO buffer for channel N is almost full as determined using synthesis parameter TXFIFO_AF_THRD [] obtained during GUI capture.
TX_FE[N:0]	Output	**Transmit FIFO Empty** - This status signal when active (=1), indicates that the FIFO buffer for channel N is empty.
TX_FFE[N:0]	Output	**Transmit FIFO Full Error** - This status signal when active (=1), indicates that the FIFO buffer for channel N is full. This is an error condition
TX_MSOP[N:0]	Output	**Transmit Missing Start Of Packet** - This error signal when active (=1), indicates that the packet checker logic for channel N has declared a missing EOP error.

Table 12. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
TX_MEOP[N:0]	Output	**Transmit Missing End Of Packet** - This error signal when active (=1), indicates that the packet checker logic for channel N has declared a missing SOP error.
TX_PKT_DRPD[N:0]	Output	**Transmit Packet Dropped** - This error signal when active (=1), indicates that the packet checker logic for channel N has dropped a Transmit packet due to error.
TX_FAE_THRHD[(N*LOG2(FIFO_SIZE))-1:0]	Input	**Transmit FIFO Almost Empty Threshold** - Channel N FIFO buffer almost empty threshold value in 16-byte increments. Note packed array format where lengths depend on number of channels and size of the FIFOs. Must be set to at least 1 greater than TX_BLEN[] when Min_Burst mode is used.
TX_FAF_THRHD[(N*LOG2(FIFO_SIZE))-1:0]	Input	**Transmit FIFO Almost Full Threshold** - Channel N FIFO buffer almost full threshold value in 16-byte increments. Note packed array format where lengths depend on number of channels and size of the FIFOs.
TXSSEQ_LEN[5:0]	Input	**Transmit Schedule Sequence Length** - The array value controls the length of the RAM sequence used to schedule channels for service.
TXSSEQ_WR	Input	**Transmit Schedule Sequence Write** - This signal when active (=1) causes synchronous write operations of the Transmit Scheduler Sequence RAM using input clock signal RAM_WCK. Write are allowed back-to-back on consecutive clock cycles.
TXSSEQ_ADD[4:0]	Input	**Transmit Schedule Sequence Address** - Address index into the 32x16 Transmit Scheduler Sequence RAM.
TXSSEQ_DI[15:0]	Input	**Transmit Schedule Sequence Data In** - 16-bit data bus into the 32x16 Transmit Scheduler Sequence RAM.
TXBLEN[5:0]	Input	**Transmit Burst Length** - Maximum number (4-64) of 16-byte blocks that are transmitted for a given channel before segmentation is allowed (i.e. Training & Control Insertion). Must match the value programmed into the SMI memory for MACO hard-core. Applies to all channels.
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 125MHz).
TXSTEN	Input	**Transmit Status Enable** - This signal when active (=1), status channel processing is enabled. When inactive, signal TxSTAERR below is forced active.
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. (Clock=I/O port tstatus_ck, minimum 4 cycle active)

Table 12. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
TXSTAT_T[1:0]	Output (TxSTATUS_CK)	**Transmit Status [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. This is a debug signal in this mode - connection is not required since status is handled internally in this mode.
TXSTPA[7:0]	Output (TxSTATUS_CK)	**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 S4TXS logic. Available in both RAM and Transparent status modes.
TXSTPA_VAL	Output (TxSTATUS_CK)	**Transmit Status Port Address Valid** - This signal when active (=1) indicates that the value on TxSTPA[] & 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). This is a debug signal in this mode - connection is not required since status is handled internally in this mode.
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]	Static Input	**Transmit Calendar Repetition** - Alpha.
TXCAL_LEN[8:0]	Static 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. (Clock=I/O port tstatus_ck, 1 cycle active)
TXNUMDIP2[2:0]	Static 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]	Static 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.
SMI Interface Signals		
SMI_RSTN	Input	**SMI Reset** - This active low reset affects the internal SMI controller.

Table 12. User Side Signal Descriptions (Continued)

Signal Name	Direction	Description
SMI_CLK	Input	**SMI Clock** - SMI Clock
SMI_RD	Input	**SMI Read** - Active high read strobe
SMI_WR	Input	**SMI Write** - Active High write strobe
SMI_WDATA	Input	**SMI Write Data** - Serial write data
SMI_ADDR[9:0]	Input	**SMI Address** - SMI address
SMI_RDATA	Output	**SMI Read Data** - SMI Read data

Appendix for LatticeSCM FPGAs

Table 13. Resource Utilization[1]

Configuration			Utilization			
Device	**Package**	**Status Mode**	**Slices**	**LUTs**	**Registers**	**EBRs**
SCM40	1020	Transparent	837	940	1200	14
SCM40	1152	RAM	1011	1186	1297	14

1. Performance and utilization characteristics using Lattice ispLEVER 7.1 software. When using this IP core with different software or in a different speed grade, performance may vary. Performance and utilization characteristics have been verified on the latest version of Lattice ispLEVER software to be within ± 5% of the above numbers.

Table 14. Buffer Manager Mode Resource Utilization[1, 2, 3]

Configuration					Utilization			
Device	**Chan**	**Buffer Size**	**Mode**	**Package**	**Slices[2]**	**LUTs[2]**	**Registers**	**EBRs**
SCM15-6	1	32k(R/T)	Drop	256	2038	2638	2432	46
SCM25-6	4	16K(R/T)	Drop	900	4252	6610	5310	78
SCM40-6	8	16K(R/T)	Drop	1152	7373	11,922	9163	142
SCM115-6	16	32K-R, 16K-T	Drop	1704	13,119	22,582	17,019	398

1. Performance and utilization characteristics were obtained using Lattice ispLEVER 7.1 software. When using this IP core with different software, performance may vary. Performance and utilization characteristics have been verified on the latest version of Lattice ispLEVER software to be within ± 5% of the above numbers.
2. For cases where a multi-channel interleaved SPI4 line (configured via the GUI) is received into the SPI4 IP core, add ~1250 slices and ~1700 luts to the utilization numbers above.
3. The utilization numbers above were obtained through place and route of the "core only" design that is automatically created during IP core generation. For these cases, place and route was run using a -6 speed grade device and constraints consistent with a 16Gbps SPI4 line (500 MHz DDR). Depending line rate required and complexity of user logic functions, a slower speed grade (-5) may be used or a faster speed grade (-7) device may be required.

Ordering Part Number

All MACO IP, including the SPI4 MACO Core, is pre-engineered and hardwired into the MACO structured ASIC blocks of the LatticeSCM family of parts. Each LatticeSCM device contains a different collection of MACO IP. Larger FPGA devices will have more instances of MACO IP. Please refer to the Lattice web pages on LatticeSCM and MACO IP or see your local Lattice sales office for more information.

All MACO IP is licensed free of charge, however a license key is required. See your local Lattice sales office for the license key.