Interlaken 150G v1.6

LogiCORE IP Product Guide

Vivado Design Suite

PG212 October 4, 2017

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The Xilinx® Interlaken 150G LogiCORE™ IP core implements a scalable chip-to-chip interconnect protocol designed to enable transmission speeds up to 150 Gigabits per second (Gb/s). Using the latest serial transceiver technology and a flexible protocol layer, Interlaken minimizes the pin and power overhead of chip-to-chip interconnect and provides a scalable solution that can be used throughout an entire system. In addition, the Interlaken 150G core uses two levels of Cyclic Redundancy Check (CRC) and a self-synchronizing data scrambler to ensure data integrity and link robustness.

Features

- Support for any data rate up to 150 Gb/s
- Flexible serial transceiver interface to accommodate different I/O implementations
- Data striping and de-striping across 1 to n lanes. The number of lanes is limited only by the available FPGA resources.
- Programmable BurstMax, BurstShort and MetaFrameSize parameters
- 64/67 encoding and decoding
- Automatic word and lane alignment
- Self-synchronizing data scrambler
- Data bus width of 64, 128, 256, or 512 bits
- CRC24 generation and checking for burst data integrity

For more features, refer to Feature Summary in Chapter 1.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family	Virtex® UltraScale+™ Kintex® UltraScale+ Zynq® UltraScale+ Virtex UltraScale™ Kintex UltraScale Virtex-7 Kintex-7
Supported User Interfaces	LBUS and Segmented LBUS
Resources	Table 2-1
Provided with Core	
Design Files	Encrypted register transfer level (RTL)
Example Design	Verilog
Instantiation Template	Verilog
Test Bench	Verilog Test Bench
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Verilog
Supported S/W Driver(1)	Not Applicable
Tested Design Flows(1)	
Design Entry	Vivado® Design Suite
Simulation	Mentor Graphics Questa Advanced Simulator Synopsys VCS Cadence Incisive
Synthesis	Vivado
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The Interlaken 150G IP core is a high-performance, low-power, flexible implementation of the Interlaken Protocol. The Interlaken 150G IP core is based on the *Interlaken Protocol Definition, Revision 1.2* [Ref 1], and offers system designers a risk-free and quick path for adopting Interlaken as their chip-to-chip interconnect protocol. The Interlaken 150G IP core is available as an embedded soft macro targeted for Kintex®-7, Virtex®-7, Virtex UltraScale™, Kintex UltraScale, Zynq® UltraScale+™, Kintex UltraScale+, and Virtex UltraScale+ devices.

This product guide describes the Interlaken 150G IP core in detail and provides the information required to integrate the core into your designs. The document assumes familiarity with the Interlaken protocol and FPGA design methodology.

Feature Summary

Interlaken is a very flexible and customizable protocol. The following protocol features are compliant with the *Interlaken Protocol Definition, Revision 1.2* [Ref 1]. Certain features, such as the number of serial lanes, number of logical channels, and flow control can be modified depending on the application.

- Use of any number of serial lanes for an aggregate bandwidth of up to 150 Gb/s
- CRC32 generation and checking for lane data integrity
- Data scrambling and disparity tracking to minimize baseline wander and maintain Direct Current (DC) balance
- Support for all Synchronization, Scrambler State, Diagnostic, and Skip Word Block Types
- Programmable Rate Limiting circuitry
- Robust error condition detection and recovery
- Channel-level and link-level flow control mechanism
- Support for 256 different logical channels (More channels are supported through the Channel Extension feature.)
- Burst-interleaved and packet modes supported
- BurstMax size can be programmed up to 256 bytes
- In-band flow control

- Out-Of-Band flow control
- Rate matching
- Meta Frame Length programmable between 128 to 8K words

Unsupported Features

It is not designed to be used for *repeater* applications. The IP core solution does not support skip insertion and deletion.

Applications

Interlaken can be used in many applications:

- Framer/MAC to NPU or L2/L3 switch interface
- Line Card to Switch Fabric Interface

It can also run on multiple media: FR4 (PCB), backplanes, or over cable.

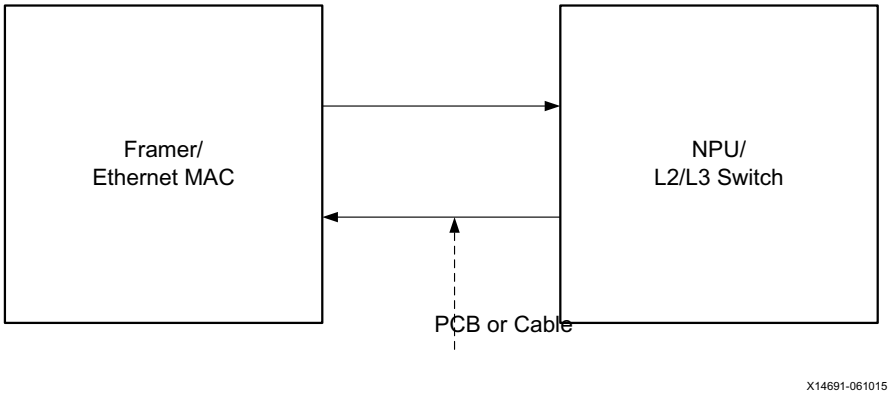

Figure 1-1: **Framer/MAC to NPU/L2 or L3 Switch**

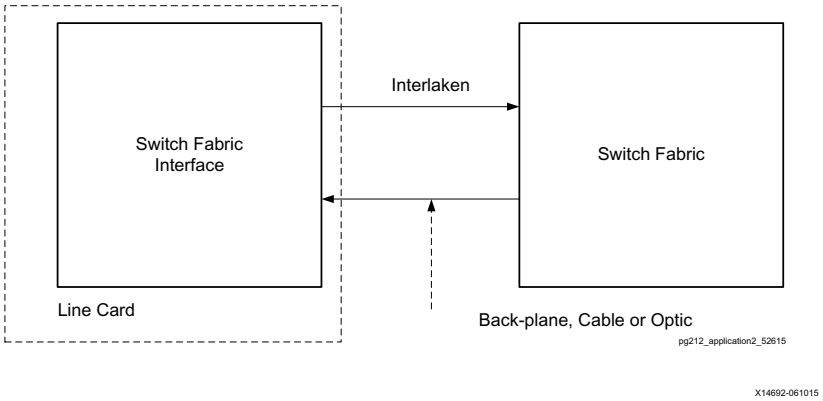

Figure 1-2: **Framer/MAC to NPU/L2 or L3 Switch**

Licensing and Ordering

The Interlaken 150G IP core is provided under the Xilinx® Project License, which must be executed for each design project. Upon execution of the Project License Agreement, the full functionality of the core is configured based on your selected part number (see the following list) and configuration settings and is provided to you.

Note: Interlaken core data rates are defined as equal to *# lanes × serial transceiver speed* (for example, 6 lanes × 6.25 G = 37.5 G).

Xilinx Part Number: EM-DI-ILN-40G-PROJ for Interlaken 10 G-40 G data rates

Xilinx Part Number: EM-DI-ILN-90G-PROJ for Interlaken 50 G-90 G data rates

Xilinx Part Number: EM-DI-ILN-150G-PROJ for Interlaken 100 G-150 G data rates

Contact your local Xilinx sales representative for the Xilinx Project License form, Interlaken core pricing, and availability. For more information, see the Interlaken Core home page.

Product Specification

The Interlaken 150G IP core is delivered as a soft macro block, which allows implementation of the core in Kintex®-7, Virtex®-7, Kintex UltraScale™, Virtex UltraScale, Zynq® UltraScale+™ Kintex UltraScale+, and Virtex UltraScale+ devices with minimal effort. "Minimal effort" means you can instantiate the core as a black box in your design. To implement the core, connect the Interlaken 150G IP core to the high-speed serial transceiver blocks, provide the appropriate clock signals, and connect to the Local Bus (LBUS) user-side interface.

The block diagram for the Interlaken 150G IP core is shown in Figure 2-1. The right side is the user interface and the left side is the external device interface.

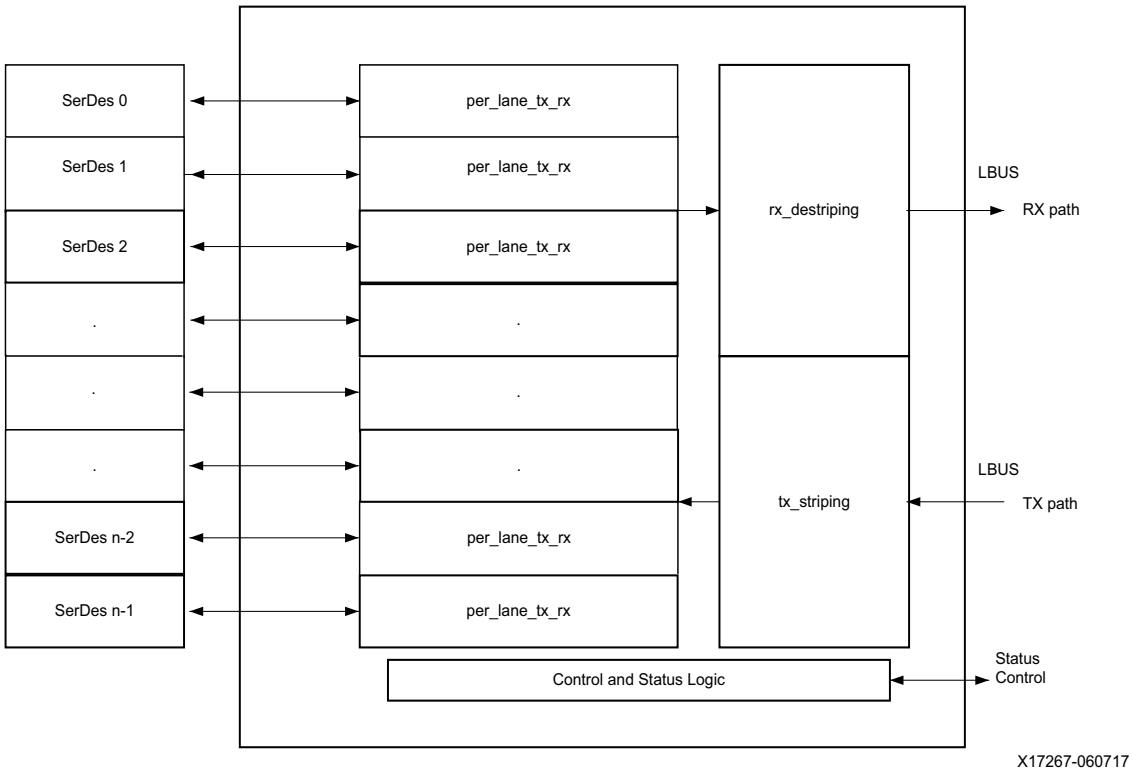

Figure 2-1: **Block Diagram**

Hierarchy

The Interlaken 150G IP core is delivered as a soft macro block. The upper levels of hierarchy are shown in Figure 2-2. The text in Figure 2-2 represents the name of the module at that level of hierarchy. The instance name is in parentheses.

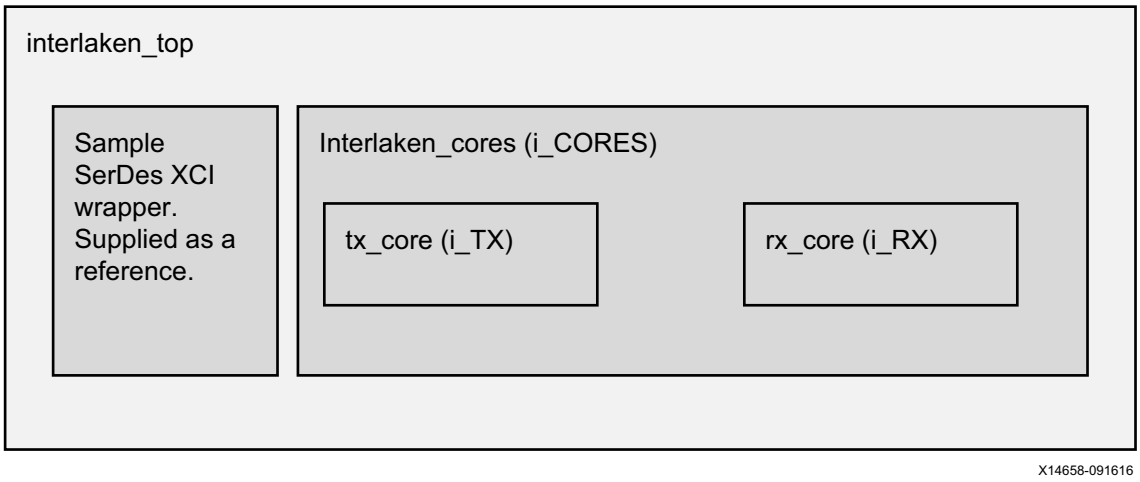

Figure 2-2: **Core Hierarchy**

The `interlaken_top` module contains instantiations of the receive and transmit serial transceiver macros to allow for testing and simulation of the Interlaken 150G IP core. The connections to/from the serial transceiver macros in the `interlaken_top` module must only be used as a reference. You must instantiate the appropriate serial transceiver and make the proper connections to the `interlaken_cores` modules.

The `interlaken_cores` module contains all of the logic for the Interlaken 150G IP core. It is divided into two submodules: `tx_core` and `rx_core`. Instantiate `interlaken_cores` when using the Interlaken 150G IP core in duplex mode, or instantiate the `tx_core` and `rx_core` separately when using the Interlaken 150G IP core in simplex mode. See References in Appendix F for a list of transceiver user guides.

Typical Operation

The Interlaken 150G IP core handles all protocol-related functions to communicate to the other devices Interlaken interface. All handshaking, synchronizing, and error checking are handled by the core. You can provide packet data through the LBUS TX interface and receive it from the LBUS RX interface. The LBUS is designed to match packet bus protocols made common by the SPI4.2 protocol; a detailed description is provided in TX LBUS Interface and RX LBUS Interface in Chapter 3, Designing with the Core.

The Interlaken 150G IP core is designed to be as flexible as possible and to be used in many different applications. As such, the core provides all of the flexibility offered by the Interlaken protocol. Flow control information is automatically extracted by the RX path of the core. You must monitor the flow control information and ensure proper data transmission through the core. Also, the Interlaken 150G IP core TX path consists of a single pipeline with a single memory buffer. You must build the scheduling mechanism external to the core to multiplex data from different logical channels.

Following are the steps after the Interlaken 150G IP core is powered up.

1. After the device is powered up and the reset procedure completed, the Interlaken 150G IP core TX path starts transmitting Control/Idle words to align and synchronize the receiving device Interlaken interface. Similarly, the core RX path receives Control/Idle words and completes its own synchronization procedure.
2. Set all of the flow control inputs to the Interlaken 150G IP core TX path to the XOFF state to prevent any real data transfer.
3. The RX path is eventually aligned and synchronized and signals the user logic that synchronization is complete. You can then turn the flow control information from XOFF to XON for any of the channels that are ready to accept data.
4. When the other device is ready to receive data, it sends XON information to the Interlaken 150G IP core. The core will signal the user logic which channels can be used for data transmission.

These steps provide a simple and easily implemented procedure for using the Interlaken 150G IP core. You build a scheduler to multiplex data among the different logical channels and use the flow control information output by the core to manage the scheduling function. Lower-level Interlaken protocol details do not affect the outcome of designs using this core.

Standards

The Interlaken 150G IP core solution is designed to be compliant with *Interlaken Protocol Definition 1.2* [Ref 1] for end-point transmitter and receiver applications. It is not designed to be used for repeater applications.

Performance and Resource Utilization

Table 2-1 provides approximate utilizations figures for some representative supported configurations when a single instance is instantiated in a 7 series device. These are estimated sizes only based on specific configurations and are subject to revision.

Contact your Xilinx sales representative with your specific requirements.

Table 2-1: **Device Utilization for Interlaken (7 Series)**

IP Bandwidth	GTX Lanes	Serial Transceiver Rate	LUTs in 1000s	# of ffs in 1000s	Block RAM	LBUS Width	Clk Frequency	Min. Required Speed Grade
Interlaken (150G)	24	6.25G	40.5	57.8	16	512	294 MHz	-2
Interlaken (123.75G)	12	10.3125G	32.7	46.2	16	512	300 MHz	-2
Interlaken (125G)	10	12.5G	34.9	44.4	16	512	250 MHz	-2
Interlaken (75G)	12	6.25G	29.7	41.3	16	512	300 MHz	-2
Interlaken (50G)	8	6.25G	28.8	36.5	5	512	200 MHz	-2

Port Descriptions

Table 2-2 shows the port list for the Interlaken IP core. More detail on the use of each signal is given in Chapter 3. The variable LANES represents the number of high-speed serial transceiver lanes used in the core, and the SERDES_WIDTH variable represents the width of the parallel bus to/from the serial transceiver macros. Some of these signals are input to the serial transceiver (GT) and only exist in the interlaken_top module.

IMPORTANT: *Static signals are control signals that can be changed only when the corresponding reset is asserted. Their values must be held static during the normal operation of the core.*

Table 2-2: **Port Description**

Name	Direction	Domain	Description
Transceiver I/O			
gt_refclk_p	Input		Differential reference clock for the transceiver, positive phase (P).
gt_refclk_n	Input		Differential reference clock for the transceiver, negative phase (N).
drpclk_p	Input		Positive Phase of DCLK (Dynamic Reconfiguration Port (DRP) Clock). This must be a convenient stable clock, for example 50 MHz. Refer to the current transceiver guide for up to date information.
drpclk_n	Input		Negative Phase of DCLK. This must be a convenient stable clock, for example 50 MHz. Refer to the current transceiver guide for up to date information.
gt_soft_reset	Input	async	Active-High asynchronous reset signal that is a primary input to the GT Transceiver wrapper. Within the transceiver wrapper, it is connected to all of the start-up FSMs. There can be two start-up FSMs for each lane, one for the transmitter and one for the receiver. The start-up FSMs are used to provide individual resets to the internal GT components in the correct sequence.
gt_tx_reset	Input	async	Active-High asynchronous reset signal that is a primary input to the transceiver wrapper, connected to GTTXRESET.
gt_rx_reset	Input	async	Active-High asynchronous reset signal that is a primary input to the transceiver wrapper, connected to GTRXRESET.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
clt_loopback	Input	async	Active-High signal to the GT Transceiver wrappers. Within the transceiver wrapper, it is connected to bit 1 (the middle bit) of the 3-bit 'loopback_in' pin for each lane. Bits 0 and 2 of the 'loopback_in' pins are tied Low. The behavior of the 'loopback_in' pin(s) are described in the *UltraScale Architecture GTH Transceivers Advance Specification User Guide* (UG576) [Ref 2]. ctl_loopback provides a local physical medium attachment (PMA) loopback function. When the signal ctl_loopback is Low, transmit data exits at the serial outputs, and received data enters at the serial inputs. When ctl_loopback is High, the local PMA loopback is enabled.
rx_serdes_data[SERDES_WIDTH-1:0]	Input	rx_serdes_clk	Data bus from the serial transceiver macros. There are LANES rx_serdes_data buses; one bus for each serial transceiver lane and each bus has SERDES_WIDTH bits. By definition, bit [SERDES_WIDTH-1] is the first bit received by the Interlaken 150G IP core. Bit [0] is the last bit received. The core can handle any even-width serial transceiver bus. Typical widths are 32 and 64. Note that when the 64/67 transceiver gearbox is used, additional signals will be present to indicate the data sequence. Refer to the appropriate transceiver guide.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
tx_serdes_data[SERDES_WIDTH-1:0]	Output	tx_serdes_refclk	Data bus to the serial transceiver macros. There are LANES tx_serdes_data buses; one bus for each serial transceiver lane and each bus has SERDES_WIDTH bits. By definition, bit [SERDES_WIDTH-1] is the first bit transmitted by the core. Bit [0] is the last bit transmitted. The core can handle any even width serial transceiver bus. Typical widths are 10, 16, 20 and 32. Note that when the 64/67 transceiver gearbox is used, additional signals will be present to indicate the data sequence. Refer to the appropriate transceiver guide.
rx_serdes_clk[LANES-1:0]	Input		Recovered clock of each serial transceiver lane. The rx_serdes_data bus for each lane is synchronized to the positive edge of the corresponding bit of this bus.
rx_serdes_reset[LANES-1:0]	Input	async	Asynchronous Reset for each RX serial transceiver lane. The recovered clock for each serial transceiver lane has associated with it an active-High reset. This signal should be asserted whenever the associated recovered clock is not operating at the correct frequency. The core handles synchronizing each rx_serdes_reset signal to the appropriate clock domain. Generally, this signal is connected to a Phase-Locked Loop (PLL) lock signal.
tx_serdes_refclk	Input		Reference clock for the TX lanes datapath. This clock must be connected to the same reference clock used to drive the TX serial transceivers. The tx_serdes_data bus for each lane is synchronized to the positive edge of the this clock.
tx_serdes_refclk_reset	Input	async	Asynchronous Reset for TX Reference clock. This active-High (1 = reset) signal should be asserted whenever the tx_serdes_refclk input is not operating at the correct frequency. The core handles synchronizing the tx_serdes_refclk_reset signal to the appropriate clock domain.
LBUS Interface – Clock/Reset Signals			

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
clk	Input		Local bus clock. All signals between the Interlaken 150G IP core and the user side logic are synchronized to the positive edge of this signal.
rx_reset	Input	async	Asynchronous reset for the RX circuits. This signal is active-High (1= reset) and must be held High until all of the clocks for the RX path are fully active. These clocks are clk and rx_serdes_clk[LANES:0]. The core handles synchronizing the rx_reset input to the appropriate clock domains within the core.
tx_reset	Input	async	Asynchronous reset for the TX circuits. This signal is active-High (1= reset) and must be held High until all of the clocks for the TX path are fully active. These clocks are clk and tx_serdes_refclk. The core handles synchronizing the tx_reset input to the appropriate clock domains within the core.
LBUS Interface – RX Path Signals			
rx_dataout[511|255|127|63:0]	Output	clk	Receive LBUS Data. The value of the bus is only valid in cycles in which rx_enaout is sampled as 1.
rx_chanout[7:0]	Output	clk	Receive Channel Number. This bus indicates the channel number of the in-flight packet and is only valid in cycles in which rx_enaout is sampled as 1. When the channel extension feature is implemented, this bus will be wider as required.
rx_enaout	Output	clk	Receive LBUS Enable. This signal qualifies the other signal of the RX LBUS Interface. Signals of the RX LBUS interface are only valid in cycles in which rx_enaout is sampled as a 1.
rx_sopout	Output	clk	Receive LBUS Start Of Packet (SOP). This signal indicates the start of packet when it is sampled as a 1 and is only valid in cycles in which rx_enaout is sampled as a 1.
rx_eopout	Output	clk	Receive LBUS End Of Packet (EOP). This signal indicates the end of packet when it is sampled as a 1 and is only valid in cycles in which rx_enaout is sampled as a 1.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
rx_errout	Output	clk	Receive LBUS Error. This signal indicates that the current packet being received has an error when it is sampled as a 1. This signal is only valid in cycles when both rx_enaout and rx_eopout are sampled as a 1. When this signal is a value of 0, it indicates that there is no error in the packet being received.
rx_mtyout[5|4|3|2:0]	Output	clk	Receive LBUS Empty. This bus indicates how many bytes of the rx_dataout bus are **empty** or **invalid** for the last transfer of the current packet. This bus is only valid in cycles when both rx_enaout and rx_eopout are sampled as 1. When rx_errout and rx_enaout are sampled as 1, the value of rx_mtyout[2:0] is always 000. Other bits of rx_mtyout are as usual.
LBUS Interface – TX Path Signals			
tx_rdyout	Output	clk	Transmit LBUS Ready. This signal indicates whether the Interlaken 150G IP core TX path is ready to accept data and provides back-pressure to the user logic. A value of 1 means the user logic can pass data to the core. A value of 0 means the user logic must stop transferring data to the core. When tx_rdyout is asserted depends on the value of ctl_tx_rdyout_thresh[2:0].
tx_ovfout	Output	clk	Transmit LBUS Overflow. This signal indicates whether you have violated the back-pressure mechanism provided by the tx_rdyout signal. If tx_ovfout is sampled as a 1, a violation has occurred. You must design the rest of the user logic to prevent the overflow of the TX interface. If an overflow does occur, and the root cause is known, a reset is required on the TX core in order to resume operation.
tx_datain[511|255|127|63:0]	Input	clk	Transmit LBUS Data. This bus receives input data from the user logic. The value of the bus is captured in every cycle that tx_enain is sampled as 1.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
tx_chanin[7:0]	Input	clk	Transmit LBUS Channel Number. This bus receives the channel number for the packet being written. The value of the bus is captured in every cycle that tx_enain is sampled as 1. When the channel extension feature is implemented, this bus will be wider as required.
tx_enain	Input	clk	Transmit LBUS Enable. This signal is used to enable the TX LBUS Interface. All signals on this interface are sampled only in cycles in which tx_enain is sampled as a 1.
tx_sopin	Input	clk	Transmit LBUS Start Of Packet (SOP). This signal is used to indicate the start of packet (SOP) when it is sampled as a 1 and is 0 for all other transfers of the packet. This signal is sampled only in cycles in which tx_enain is sampled as a 1.
tx_eopin	Input	clk	Transmit LBUS End Of Packet (EOP). This signal is used to indicate the end of packet (EOP) when it is sampled as a 1 and is 0 for all other transfers of the packet. This signal is sampled only in cycles in which tx_enain is sampled as a 1.
tx_errin	Input	clk	Transmit LBUS Error. This signal is used to indicate a packet contains an error when it is sampled as a 1 and is 0 for all other transfers of the packet. This signal is sampled only in cycles in which tx_enain and tx_eopin are sampled as 1.
tx_mtyin[5|4|3|2:0]	Input	clk	Transmit LBUS Empty. This bus is used to indicate how many bytes of the tx_datain bus are **empty** or **invalid** for the last transfer of the current packet. This bus is sampled only in cycles that tx_enain and tx_eopin are sampled as 1. When tx_eopin and tx_errin are sampled as 1, the value of tx_mtyin[2:0] is ignored and treated as if it was 000. The other bits of tx_mtyin are used as usual.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
tx_bctlin	Input	clk	Transmit Force Insertion of Burst Control Word. This input is used to force the insertion of a Burst Control Word. When tx_bctlin and tx_enain, are sampled as 1, a Burst Control word is inserted *before* the data on the tx_datain bus is transmitted even if one is not required to observe the BurstMax parameter. This input is used by external schedulers that wish to reduce bandwidth lost due to observation of the BurstShort parameter. (See Use of tx_bctlin.) Use of this input is not a requirement for correct Interlaken 150G IP core operation. If this input is unused, it should be tied to 0.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
LBUS Interface – TX Path Control/Status Signals			
ctl_tx_enable	Input	clk	TX Enable. This signal is used to enable the transmission of data when it is sampled as a 1. When sampled as a 0, only Idle Control Words (and the Meta Frame Words) are transmitted by the Interlaken 150G IP core. This input should not be set to 1 until the receiver it is sending data to (the receiver in the other device) is fully aligned and ready to receive data. Otherwise, loss of data can occur. This input can be used for Link-Level flow control. For example, setting this input to 0 halts transmission of data and results in the entire link going into XOFF state.
ctl_tx_fc_stat[MAX_CALLEN-1:0]	Input	clk	TX In-Band Flow Control Input. These signals are used to set the status for each calendar position in the in-band-flow control mechanism (see *Interlaken Protocol Definition, Revision 1.2* [Ref 1]. A value of 1 means XON, a value of 0 means XOFF. These bits are transmitted in the Interlaken Control Word bits [55:40]. ***Note:*** MAX_CALLEN refers to the maximum number of calendar entries and is selected at the time of configuration.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
ctl_tx_fc_callen[3:0]	Input	static	TX Flow Control Calendar Length Input. This input controls the number of bits of ctl_tx_fc_stat that are actually used. The typical settings (MAX_CALLEN=256) for calendar length are as follows: 0x0 = 16 entries 0x1 = 32 entries 0x3 = 64 entries 0x7 = 128 entries 0xF = 256 entries All other values are reserved and must not be used. This input should only be changed when tx_reset is asserted. Additional values are possible if the channel extension feature is implemented. This bus will be wider as required when MAX_CALLEN exceeds 256.
ctl_tx_mubits[7:0]	Input	clk	TX Multiple-Use Control Bits. This bus contains the "Multi-Use" field of the Interlaken Control (see the *Interlaken Protocol Definition, Revision 1.2* [Ref 1]. The value of the bus is transmitted in the Interlaken Control Word bits[31:24]. You must define the function of this bus. If the bus is not used, set all bits to 0. ***Note:*** This bus is not present when the Look-Aside configuration is used.
ctl_tx_rlim_enable	Input	clk	TX Rate Limiter Enable. This signal is used to enable the Rate Limiter. A value of 1 turns on the Rate Limiter and a value of 0 turns off the Rate Limiter.
ctl_tx_rlim_max[11:0]	Input	static	TX Rate Limiter Maximum Token Count. This bus is used to set the maximum number of tokens in the bucket. A token is equal to 1 byte.
ctl_tx_rlim_delta[11:0]	Input	static	TX Rate Limiter Delta. This bus is used to set the number of tokens to add to the bucket each interval. A token is equal to 1 byte.
ctl_tx_rlim_intv[7:0]	Input	static	TX Rate Limiter Update Interval. This bus is used to set the number of Local bus clock cycles between additions of ctl_tx_rlim_delta tokens to the bucket.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
ctl_tx_burstmax[1:0]	Input	static	Interlaken TX BurstMax. This bus sets the BurstMax parameter for the TX as follows: 0x0 = 64 bytes 0x1 = 128 bytes 0x2 = 192 bytes 0x3 = 256 bytes The burst size selected by ctl_tx_burstmax must be greater than or equal to the burst size selected by ctl_tx_burstshort.
ctl_tx_burstshort[2:0]	Input	static	Interlaken TX BurstShort. This bus sets the BurstShort parameter for the TX for configuration with tx_datain[255|127|63:0] as follows: 0x0 = 32 bytes 0x1 = 64 bytes 0x2 = 96 bytes 0x3 = 128 bytes 0x4 = 160 bytes 0x5 = 192 bytes 0x6 = 224 bytes 0x7 = 256 bytes This bus sets the BurstShort parameter for the TX for configuration with tx_datain[511:0] as follows: 0x1 = 64 bytes 0x3 = 128 bytes 0x5 = 192 bytes 0x7 = 256 bytes The burst size selected by ctl_tx_burstshort must be less than or equal to the burst size selected by ctl_tx_burstmax.
ctl_tx_diagword_lanestat[LANES-1:0]	Input	async	Lane Status Messaging Inputs. This bus sets bit 33 in the Diagnostic Word for the respective lane. See Appendix A in the *Interlaken Protocol Definition, Revision* 1.2 [Ref 1].
ctl_tx_diagword_intfstat	Input	async	Interface Status Messaging Inputs. This signal sets bit 32 in the Diagnostic Word of each lane. See Appendix A in the *Interlaken Protocol Definition, Revision* 1.2 [Ref 1].

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
ctl_tx_rdyout_thresh[2:0]	Input	clk	Threshold Value for First In First Out (FIFO) Buffer in TX Path.
ctl_tx_disable_skipword	Input	static	Skip Word Injection Deletion. As required by the Interlaken specification, a skip word is inserted once per Meta Frame to permit clock compensation through a repeater function. If the Interlaken 150G IP core is not transmitting through an intermediary device, but is simply transmitting to an "ultimate" receiver such as another Interlaken 150G IP core, this input can be asserted to a value of 1. See Section 5.4.7 of the *Interlaken Protocol Definition, Revision* 1.2 [Ref 1]. This input should only be changed when tx_reset is asserted.
ctl_tx_mframelen_minus1[15:0]	Input	static	TX Meta Frame Length Minus One. This bus sets the MetaFrameLength parameter for the TX and should be set to the desired length minus 1. For example, to set the MetaFrame length to 2,048, set this bus to 2,047. MetaFrame length is defined as the number of Interlaken words. Each Interlaken word is 8 bytes (64 bits). This input should only be changed when tx_reset is asserted.
stat_tx_underflow_err	Output	clk	TX Underflow. This signal indicates if the LBUS interface is being clocked too slowly to properly fill the link with data. In normal operation, this signal is always sampled as 0. If this signal is sampled as 1, the clocks are not set to proper frequencies and must be fixed.
stat_tx_overflow_err	Output	clk	TX Overflow. This output should never be asserted and indicates a critical failure. If asserted, the TX core must be reset.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
stat_tx_burst_err	Output	clk	TX BurstShort Error. When this signal is a value of 1, a burst (that is, a sequence of Data Words between two Control Words) was shorter than the value specified by ctl_tx_burstshort. This signal is only asserted if the final Control Word did not contain an end of packet (EOP). When stat_tx_burst_err has been asserted, the TX core must be reset.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
LBUS Interface – RX Path Control/Status Signals			
ctl_rx_force_resync	Input	async	RX Force Resync Input. This signal is used to force the RX lane logic to reset, re-synchronize, and realign. A value of 1 forces the reset operation. A value of 0 allows normal operation. ***Note:*** This input should normally be Low and should only be pulsed (1 cycle minimum pulse) to force realignment.
ctl_rx_packet_mode	Input	static	RX Packet Mode Error Handling. This signal changes the way the error handler in the RX path processes errors. When this input is a value of 0, it assumes packets are arriving interleaved as segments (Burst-Interleaved Mode). When this input is a value of 1, it assumes packets are arriving as complete packets (Packet Mode). Use of this input ensures that packets delivered to the Local bus have the appropriate start of packet (SOP) and end of packet (EOP) pairing. ***Note:*** This signal is not present when the Interlaken 150G IP core has been configured for packet mode operation only.
ctl_rx_burstmax[1:0]	Input	static	Interlaken RX BurstMax. This bus sets the BurstMax parameter for the RX as follows: 0x0 = 64 bytes 0x1 = 128 bytes 0x2 = 192 bytes 0x3 = 256 bytes These inputs are only used in conjunction with stat_rx_burstmax_err.
ctl_rx_mframelen_minus1[15:0]	Input	static	RX Meta Frame Length Minus One. This bus sets the MetaFrameLength parameter for the RX and should be set to the desired length minus 1. For example, to set the MetaFrame length to 2,048, set this bus to 2,047. MetaFrame length is defined as the number of Interlaken words. Each Interlaken word is 8 bytes (64 bits). This input should only be changed when rx_reset or ctl_rx_force_resync are asserted.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
stat_rx_burstmax_err	Output	clk	RX BurstMax Error. When this signal is a value of 1, a burst (that is, a sequence of Data Words between two Control Words) was detected that was longer than the value of BurstMax specified by ctl_rx_burstmax. This signal is informational only and can be optionally ignored.
stat_rx_diagword_lanestat[LANES-1:0]	Output	clk	Lane Status Messaging Outputs. This bus reflects the most recent value in bit 33 of the Diagnostic Word received on the respective lane. These bits should only be considered valid if the respective bit in stat_rx_crc32_valid is a value of 1. See Appendix A in the *Interlaken Protocol Definition, Revision* 1.2 [Ref 1].
stat_rx_diagword_intfstat[LANES-1:0]	Output	clk	Lane Status Messaging Outputs. This bus reflects the most recent value in bit 32 of the Diagnostic Word received on the respective lane. These bits should only be considered valid if the respective bit in stat_rx_crc32_valid is a value of 1. See Appendix A in the *Interlaken Protocol Definition, Revision* 1.2 [Ref 1].
stat_rx_crc32_valid[LANES-1:0]	Output	clk	Diagnostic Word CRC32 Valid. This bus reflects the validity of the CRC32 in the most recently received Diagnostic Word for the respective lane. A value of 1 indicated the CRC32 was valid and a value of 0 indicated the CRC32 was invalid. See section 5.4.6 of the *Interlaken Protocol Definition, Revision* 1.2 [Ref 1].
stat_rx_crc32_err[LANES-1:0]	Output	clk	Diagnostic Word CRC32 Error/Invalid. This bus provides indication of an invalid CRC32 in the Diagnostic Word for the respective lane. These signals are asserted with a value of 1 for one LBUS clock cycle each time an error is detected.
stat_rx_fc_stat[MAX_CALLEN-1:0]	Output	clk	RX Flow Control Outputs. These signals indicate the flow control status for all of the calendar positions of the received data. A value of 1 means XON, a value of 0 means XOFF.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
stat_rx_mubits[7:0]	Output	clk	RX Multiple-Use Control Bits. This bus contains the "Multi-Use" field of the Interlaken Control (see *Interlaken Protocol Definition, Revision 1.2* [Ref 1]. The values of the bus are bits[31:24] of the most recently received Interlaken Control Word. ***Note:*** This bus is not present when the Look-Aside configuration is used.
stat_rx_synced[LANES-1:0]	Output	clk	Word Boundary Synchronized. These signals indicate whether a lane is word boundary synchronized. A value of 1 indicates the corresponding lane has achieved word boundary synchronization and that the metaframe and scrambler state words are correct.
stat_rx_synced_err[LANES-1:0]	Output	clk	Word Boundary Synchronization Error. These signals indicate whether an error occurred during word boundary synchronization in the respective lane. A value of 1 indicates the corresponding lane had a word boundary synchronization error.
stat_rx_mf_len_err[LANES-1:0]	Output	clk	Meta Frame Length Error. These signals indicate whether a Meta Frame length mismatch occurred in the respective lane. A value of 1 indicates the corresponding lane is receiving Meta Frame of wrong length.
stat_rx_mf_repeat_err[LANES-1:0]	Output	clk	Meta Frame Consecutive Error. These signals indicate whether consecutive Meta Frame errors occurred in the respective lane. A value of 1 indicates an error in the corresponding lane.
stat_rx_descram_err[LANES-1:0]	Output	clk	Scrambler State Control Word Error. These signals indicate a mismatch between the received Scrambler State Word and the expected value. A value of 1 indicates an error in the corresponding lane.
stat_rx_aligned	Output	clk	All Lanes Aligned/De-Skewed. This signal indicates whether or not all lanes are aligned and de-skewed. A value of 1 indicates all lanes are aligned and de-skewed. When this signal is a 1, the RX path is aligned and can receive packet data.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
stat_rx_aligned_err	Output	clk	Loss of Lane Alignment/De-Skew. This signal indicates an error occurred during lane alignment or lane alignment was lost. A value of 1 indicates an error occurred.
stat_rx_crc24_err	Output	clk	Control Word CRC24 Error. This signal indicates whether or not a mismatch occurred between the received and the expected CRC24 value. A value of 1 indicates a mismatch occurred.
stat_rx_overflow_err	Output	clk	RX FIFO Overflow Error. This signal indicates if the LBUS interface is being clocked too slowly to properly receive the data being transmitted across the link. A value of 1 indicates an error occurred. In normal operation, this signal is always sampled as 0. If this signal is sampled as 1, the clocks are not set to proper frequencies and must be fixed.
stat_rx_mf_err[LANES-1:0]	Output	clk	Meta Frame Synchronization Word Error. These signals indicate that an incorrectly formed Meta Frame Synchronization Word was detected in the respective lane. A value of 1 indicates an error occurred.
stat_rx_framing_err[LANES-1:0]	Output	clk	Framing Error. These signals indicate that an illegal framing pattern was detected in the respective lane. A value of 1 indicates an error occurred.
stat_rx_msop_err	Output	clk	Missing Start Of Packet (SOP) Error. This signal indicates that a missing start of packet (SOP) was detected (and corrected). This signal does not exist if the Interlaken core is built to operate in packet mode only.
stat_rx_meop_err	Output	clk	Missing End Of Packet (EOP) Error. This signal indicates that a missing end of packet (EOP) was detected (and corrected).
stat_rx_burst_err	Output	clk	Burst Error. This signal indicates that a BurstShort or a burst length error was detected.

Table 2-2: **Port Description *(Cont'd)***

Name	Direction	Domain	Description
stat_rx_misaligned	Output	clk	Alignment Error. This signal indicates that the lane aligner did not receive the expected Meta Frame Synchronization Word across all (active) lanes. This signal can be used to collect the statistic "RX_Alignment_Error" as described in Table 5-9 of the Interlaken specification. This signal is not asserted until the Meta Frame Synchronization Word has been received at least once across all lanes. A value of 1 indicates the error occurred.
stat_rx_bad_type_err[LANES-1:0]	Output	clk	Unexpected or Illegal Meta Frame Control Word Block Type. These signals indicate an unexpected or illegal Meta Frame Control Word Block Type was detected. These signals can be used to collect the statistic "RX_Bad_Control_Error" as described in Table 5-9 of the Interlaken specification. A value of 1 indicates an error in the corresponding lane.
stat_rx_mubits_updated	Output	clk	RX Multiple-Use/General Purpose Control Bits Updated. This output indicates that stat_rx_mubits has been updated and is asserted for one clock cycle. ***Note:*** This signal is not present when the Look-Aside configuration is used.
stat_rx_word_sync[LANES-1:0]	Output	clk	64B/67B Word Boundary Locked. These signals indicate whether a lane is 64B/67B word boundary locked. A 64B/67B word boundary lock occurs if a lane detects 64 consecutive valid framing patterns on bits[65:64] as per the Interlaken Specification 1.2 Section 5.4.2. These signals are independent of both the Meta Frame Synchronization Word and Scrambler State Control Word. A value of 1 indicates the corresponding lane has achieved 64B/67B word boundary lock.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

Clocking

The Interlaken 150G IP core has the following major clock domains:

- LBUS Interface

  This clock drives logic for both the RX and TX LBUS interfaces and the protocol layer processing.

- RX Serial Transceiver Domain

  Each serial transceiver lane has its own recovered clock. This clock is used for all of the logic for that serial transceiver lane. The Interlaken 150G IP core synchronizes the received data from all of the serial transceivers to the LBUS clock domain.

- TX Serial Transceiver Domain

  The TX serial transceiver domain consists of logic that is operated on the clock domain associated with each TX serial transceiver macro. All of the serial transceiver macros must be clocked using the same reference clock source to ensure frequency matching between the lanes.

  Figure 3-1 shows the different clock domains in the RX direction along with their associated clock inputs. Figure 3-2 shows the different clock domains in the TX direction along with their associated clock inputs. The core handles all clock domain crossings.

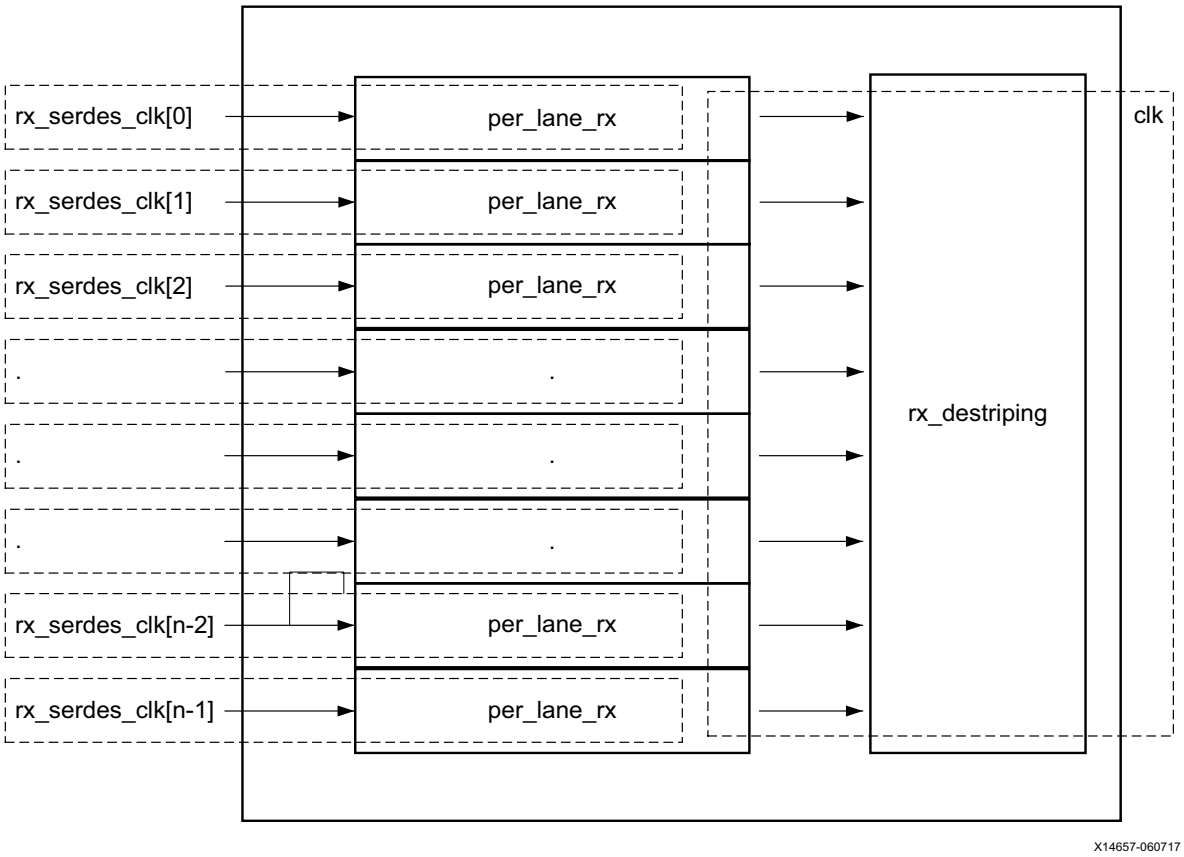

Figure 3-1: **RX Clock Domains**

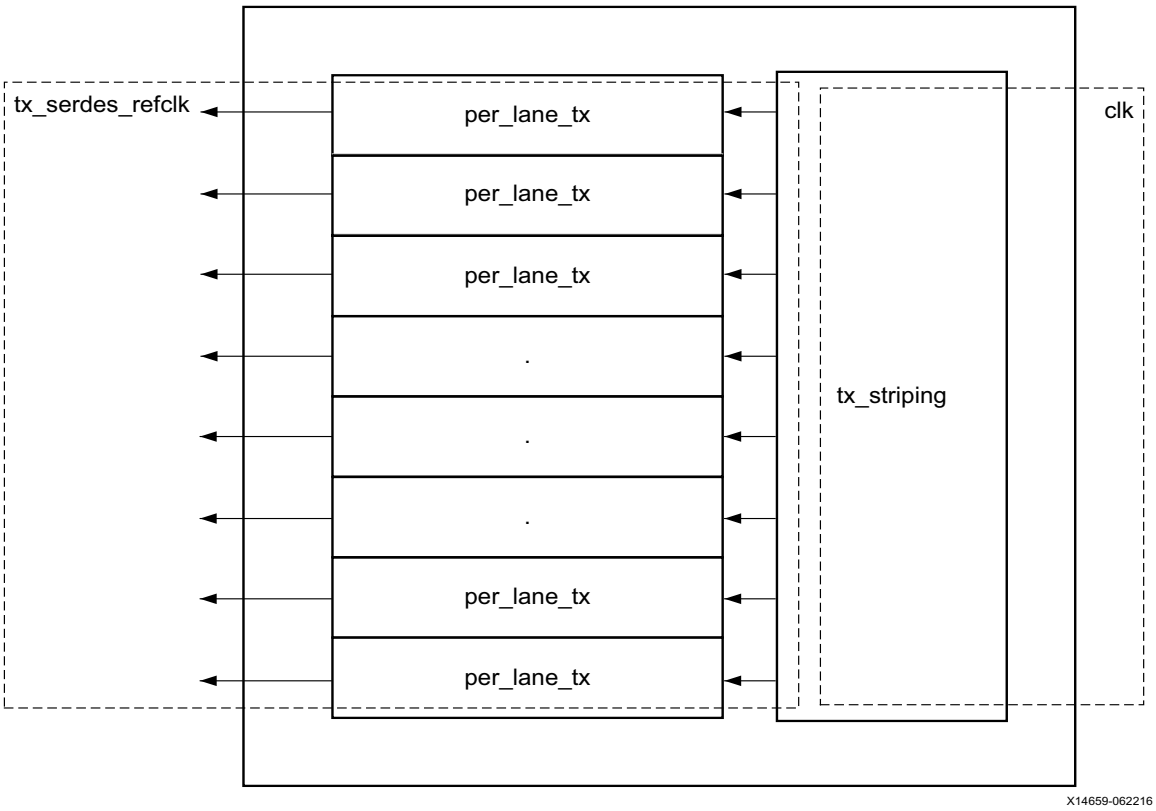

Figure 3-2: **TX Clock Domains**

The `clk` input port is used to clock the protocol processing of the Interlaken 150G IP core. This includes all logic in the TX and RX paths responsible for Control Word processing, Meta Frame processing, and the LBUS interface.

The frequency of the `clk` domain must be high enough to handle the overall bit rate of the serial link. The following equation is used to estimate the minimum `clk` frequency:

clk frequency > *(*`number_of_lanes` × `serial_bit_rate`*)/(*`internal_bus_width`*)*

Depending on the configuration of the core, `internal_bus_width` can be (*n* × 67) bits wide, where *n* is 1, 2, 3, 4, 5, 6, 7, or 8. Typically, the size of the `internal_bus_width` is chosen so that the core can operate at a desired frequency.

As an example, if there are 12 serial lanes with each lane running at 6.25 Gb/s and a 536-bit internal bus, the minimum `clk` frequency is (12 × 6.25 Gbps/536) = 140 MHz.

RECOMMENDED: *When calculating the minimum clock frequency for the LBUS, consider all maximum/minimum behavior of the different clock sources. Xilinx recommends using n×64 instead of n×67 for the internal_bus_width when calculating the minimum clock frequency to handle all possibilities. In the preceding example, the minimum clock frequency would therefore be 147 MHz.*

Note: The actual minimum frequency required for your release needs to be verified by the Xilinx tools and will be communicated in your release package.

rx_serdes_clk Domain

Each Serializer/Deserializer (SerDes) macro can provide its own recovered clock to the Interlaken 150G IP core. These clocks are connected to the `rx_serdes_clk[LANES-1:0]` input pins and are used to clock the per lane logic of each lane. It is possible to share the same clock among multiple lanes. The core handles crossing these domains to the `clk` domain.

The frequency of these domains is calculated by dividing the serial bit rate by the SerDes width for each serial transceiver block.

For example, if the serial bit rate is 6.25 Gb/s and the parallel bus is 32 bits wide, the `rx_serdes_clk[LANES-1:0]` will have a frequency of(6,250 Mbps/32) = 195 MHz.

tx_serdes_refclk Domain

The `tx_serdes_refclk` is the reference clock used to generate the high-speed clock for the transmitting serial transceiver. This clock is used by the Interlaken IP core to source the data going into each serial transceiver.

The frequency of the `tx_serdes_refclk` domains is calculated by dividing the serial bit rate by the parallel bus width into each serial transceiver block.

For example, if the serial bit rate is 12.5 Gb/s and the parallel bus is 64-bits wide, the `tx_serdes_refclk` has a frequency of (12,500 Mbps/64) = 195.3125 MHz.

Xilinx Transceiver Clocking

The example wrappers provided with the core can be generated to provide clocking to the Xilinx transceivers for either a synchronous system or an asynchronous system. In a synchronous system, the same reference clock is used for the Xilinx transceivers and for the downstream device. In an asynchronous system, a different reference clock is used for the Xilinx transceivers and for the downstream device.

Example Wrapper Clocking for a Synchronous System

In a synchronous system, `rx_serdes_clk` and `tx_serdes_refclk` can use the same clock. `TXOUTCLK` from one transceiver is sent to a Mixed-Mode Clock Manager (MMCM), which generates the RX and TX userclks for all of the transceivers and for the `rx_serdes_clk` and `tx_serdes_refclk` in the Interlaken core.

Example Wrapper Clocking for an Asynchronous System

In the asynchronous system, a separate clock is required for the RX and TX clocks. `TXOUTCLK` from one transceiver is sent to an MMCM, which generates the TX userclks for all of the transceivers and for `tx_serdes_refclk` in the Interlaken core. `RXRECCLK` from one transceiver is sent to an MMCM, which generates the RX `userclk` for all of the transceivers and for the `rx_serdes_clk` in the core.

Resets

The reset procedure for the Interlaken IP core is fairly straightforward. You are required to maintain the core in reset until the clocks are stable and glitch-free.

Certain control signals require that the core be held in reset while the value is being changed. Refer to the description for each signal to determine if this criterion applies.

User Side Interface

The Interlaken 150G IP core handles the intricate details of transporting data over an Interlaken link. The user interface is a simple packet interface designed to allow easy integration of the Interlaken 150G IP core into a system.

The LBUS consists of the following separate interfaces:

- Transmitter (TX) interface

  The transmitter accepts packet-oriented data, packages the data in accordance with the Interlaken specification, and sends that packaged data to the serial transceiver macros. The transmitter has control/configuration inputs to shape the data packaging to meet specific user requirements.

- Receiver (RX) interface

  The receiver accepts Interlaken bitstreams from the serial transceiver, removes the Interlaken packaging, and provides packet oriented data.

- Status/Control interface

  The Status/Control interface sets the characteristics of the interface and monitors its operation.

The following subsections describe the various LBUS interfaces and provide a detailed description of each individual port.

Note: In these subsections, *asserting* means "assigning a value of 1" and *negating* means "assigning a value of 0."

TX LBUS Interface

The synchronous TX Local bus interface accepts packet-oriented data of an arbitrary length. It accepts data in either Packet Mode or Burst-Interleaved Mode.

All signals are synchronous relative to the rising edge of the `clk` port. Figure 3-3 shows a sample waveform for a data transaction for a 65-byte packet.

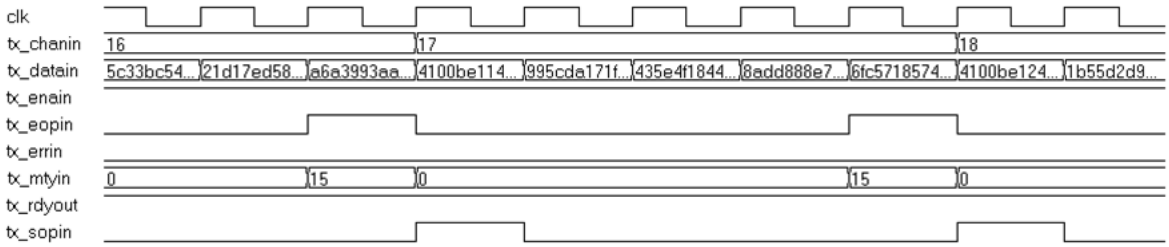

Figure 3-3: **Sample TX Waveform with a 128-bit Data Bus**

Data is written into the interface on every clock cycle when `tx_enain` is asserted. This signal qualifies the other inputs of the TX Local bus interface. This signal must be valid for every clock cycle.

The start of a packet is identified by asserting `tx_sopin` with `tx_enain`. The end of a packet is identified by asserting `tx_eopin` with `tx_enain`. Both `tx_sopin` and `tx_eopin` can be asserted during the same cycle. This is done for packets that are less than or equal to the bus width.

The channel number for a packet is presented on the `tx_chanin` inputs and must be valid for every cycle `tx_enain` that is asserted. After `tx_sopin` has been asserted with a certain channel number, it cannot be asserted again with that channel number until `tx_eopin` is asserted with the same channel number.

Data is presented on the `tx_datain` inputs. For a 512-bit wide bus, the first byte of the packet is written on bits [511:504], the second byte on bits [503:496], and so forth. For a 256-bit wide bus, the first byte of the packet is written on bits [255:248], the second byte on bits [247:240], and so forth. For a 128-bit wide bus, the first byte of the packet is written on bits [127:120], the second byte on bits [119:112], and so forth. For a 64-bit wide bus, the first byte of the packet is written on bits [63:56], the second byte on bits [55:48], and so forth.

For a 256-bit bus, the first 32 bytes of a packet are presented on the bus during the cycle that `tx_sopin` and `tx_enain` are asserted. Subsequent 32-byte chunks are written during successive cycles with `tx_sopin` negated. The last bytes of the packet are written with `tx_eopin` asserted. Unless `tx_eopin` is asserted, all 256 bits must be presented with valid data whenever `tx_enain` is asserted. 128-bit and 64-bit buses operate in the same manner as the 256-bit bus.

During the last cycle of a packet, the tx_mtyin signals can be asserted. These signals indicate how many byte lanes in the data bus are invalid (or empty). The tx_mtyin signals only have meaning during cycles when both tx_enain and tx_eopin are asserted. For a 256-bit wide bus, tx_mtyin is 5 bits. For a 128-bit wide bus, tx_mtyin is 4 bits. For a 64-bit wide bus, tx_mtyin is 3 bits.

If tx_mtyin has a value of 0x0, there are no empty byte lanes or all bits of the data bus are valid. If tx_mtyin has a value of 0x1, then one byte lane is empty, specifically bits [7:0] do not contain valid data. If tx_mtyin has a value of 0x2, then two byte lanes are empty, specifically bits [15:0] do not contain valid data. If tx_mtyin has a value of 0x3, then three byte lanes are empty, specifically bits [23:0] do not contain valid data, and so forth. The tx_mtyin signal works the same for 256-, 128-, and 64-bit buses.

During the last cycle of a packet, when tx_eopin is asserted with tx_enain, tx_errin can also be asserted. This marks the packet as being in error and this information is included in the final Interlaken Control Word associated with this packet. When tx_eopin and tx_errin are sampled as 1, the value of tx_mtyin[2:0] is ignored and treated as equal to 000. The other bits of tx_mtyin are used as usual.

The number of additional writes that can be performed after tx_rdyout is negated is determined by the signal ctl_tx_rdyout_thresh. If tx_ovfout is ever asserted, the TX core must be reset before resuming normal operation.

See Transmitter FIFO Threshold for more information on setting ctl_tx_rdyout_thresh.

Data can be safely written (tx_enain asserted) whenever tx_rdyout is asserted. After tx_rdyout is negated, additional writes, using tx_enain, can be safely performed provided tx_ovfout is never asserted (set the ctl_tx_rdyout_thresh bus to a value as low as possible to ensure tx_ovfout is never asserted). When tx_rdyout is asserted again, additional data can be written. If the back-pressure mechanism is violated, the tx_ovfout is asserted to indicate the violation.

Data Formatting

The transmit logic of the Interlaken 150G IP core segments inbound packets into bursts as described in the *Interlaken Protocol Definition, Revision 1.2* [Ref 1]. On the Interlaken interface, a burst is a sequence of 64-bit Data Words between two 64-bit Control Words. The first of those two Control Words must be a Burst Control Word and it indicates the channel number associated with the burst and whether the burst is the SOP.

The size of the bursts generated by the Interlaken 150G IP core is controlled by these factors:

- Inputs ctl_tx_burstmax and ctl_tx_burstshort
- How packets are written to the TX

The core operates in one of two modes, depending on how the data is written to the TX:

- Packet Mode
- Burst-Interleaved Mode

Packet Mode

Packet Mode occurs when a packet with a certain channel number is transferred in its entirety across the TX LBUS without interruption by a packet belonging to a different channel. The SOP data is transferred when `tx_sopin` is asserted. The final data belonging to the packet is transferred when `tx_eopin` is asserted.

Unless `tx_bctlin` is asserted (see Use of tx_bctlin) the size of all bursts (except the last) on the Interlaken interface will match the value specified by the `ctl_tx_burstmax` configuration input. If the size of the last burst containing data for a given packet is less than the BurstShort value specified by `ctl_tx_burstshort`, idle control words are added to maintain the BurstShort requirement.

Burst-Interleaved Mode

Burst-interleaved mode occurs when bursts from packets with different channel numbers are interleaved on the Interlaken interface. In other words, one burst contains data representing a portion of one packet, while the next burst contains data representing a portion of another packet (from a different channel). Ensure that the following are strictly observed:

- When `tx_sopin` has been asserted for a channel, it cannot be asserted again for that channel until a corresponding `tx_eopin` for that channel has been written.
- Unless `tx_eopin` is asserted, the full width of the data bus, `tx_datain`, must contain valid data as discussed in TX LBUS Interface.

The size of the bursts generated in burst-interleaved mode is governed by the `tx_bctlin` input (see Use of tx_bctlin), `ctl_tx_burstshort`, `ctl_tx_burstmax`, and the changing of the channel number `tx_chanin` as data from different packets is interleaved.

RECOMMENDED: *Xilinx strongly recommends implementing the Optional Scheduling Enhancement as described in section 5.3.2.1.1 of the Interlaken Protocol Definition, Revision 1.2 [Ref 1] to maximize throughput.*

Use of tx_bctlin

The `tx_bctlin` input operates in a similar manner to `tx_sopin`; both signals cause a Burst Control Word to be injected into the data stream.

The purpose of the `tx_bctlin` input is to permit the forcing of Burst Control Words that otherwise would not be transmitted. This is a necessary function for the creation of an external scheduler that implements the Optional Scheduling Enhancement described in section 5.3.2.1.1 of the *Interlaken Protocol Definition, Revision 1.2* [Ref 1]

The Interlaken 150G IP core strictly observes the programmed values for `ctl_tx_burstmax` and `ctl_tx_burstshort` and injects Burst and Idle Control Words where required. Consequently, the core can inject Idle Control Words that otherwise would not be required, which results in reducing the effective bandwidth.

For example, assume the `ctl_tx_burstmax` is set to 256 bytes and `ctl_tx_burstshort` is set to 32 bytes. A packet of 264 bytes written into the Interlaken 150G IP core, without the use of `tx_bctlin`, is followed by three undesirable Idle Control Words that are required to meet the `ctl_tx_burstshort` parameter. Specifically:

- 1 Burst Control Word (with start of packet) is sent
- 32 Data Words are sent
- 1 Burst Control Word (without start of packet) is sent
- 1 Data Word is sent
- 3 Idle Control Words are sent (to satisfy BurstShort)

for a total of 38 Words.

If `tx_bctlin` is asserted after 128 bytes are sent, the following occurs:

- 1 Burst Control Word (with start of packet) is sent
- 16 Data Words are sent
- 1 Burst Control Word (without start of packet) is sent
- 17 Data Words are sent
- 0 Idle Control Words is sent

for a total of 35 Words.

To ensure maximum bandwidth through the Interlaken core, make sure that all rules that govern Interlaken bursts, as defined in the *Interlaken Protocol Definition, Revision 1.2* [Ref 1], are followed when using `tx_bctlin`. In particular, ensure that the LBUS data corresponding to each Interlaken burst (which is not EOP or EOP-1) on a given channel equals BurstMax bytes. In other words, the data corresponding to all Interlaken bursts, except the last two for a given packet, should equal BurstMax. Furthermore, the data corresponding to a given Interlaken burst should be written to the LBUS in its entirety before changing channels or transferring the data that corresponds to the next burst.

RX LBUS Interface

The synchronous RX Local bus interface provides packet-oriented data much like the TX Local bus interface accepts. All signals are synchronous with the rising edge of the Local bus clock. Figure 3-4 shows a sample waveform for a data transaction for a 65-byte packet.

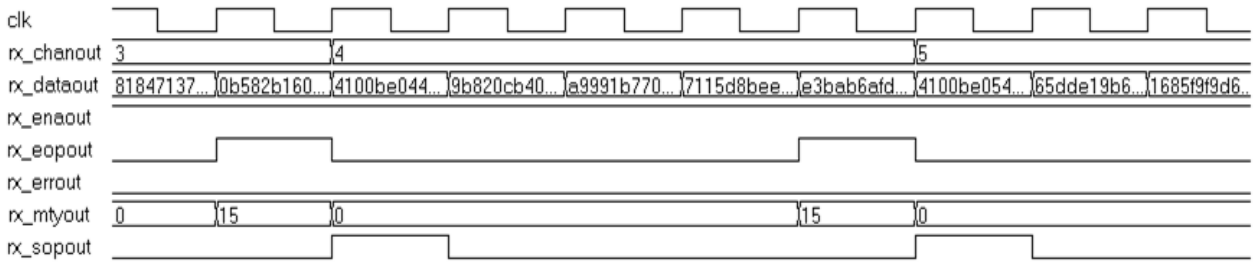

Figure 3-4: **Sample RX Waveform With a 128-bit Data Bus**

Data is supplied by the core on every `clk` clock cycle when `rx_enaout` is asserted. This signal qualifies the other outputs of the RX Local bus interface.

Similar to the TX Local bus interface, `rx_sopout` identifies the start of a packet and `rx_eopout` identifies the end of a packet. Both `rx_sopout` and `rx_eopout` are asserted during the same cycle for packets that are less than or equal to the bus width.

The channel number for a packet is presented on the `rx_chanout` outputs and is valid during every cycle `rx_enaout` is asserted.

Similar to the TX Local bus interface, the first byte of a packet is supplied on the most significant bits of `rx_dataout`. For a 512-bit wide bus, the first byte of the packet is written on bits [511:504], the second byte on bits [503:496], and so forth. For a 256-bit wide bus, the first byte of the packet is written on bits [255:248], the second byte on bits [247:240], and so forth. For a 128-bit wide bus, the first byte of the packet is written on bits [127:120], the second byte on bits [119:112], and so forth. For a 64-bit wide bus, the first byte of the packet is written on bits [63:56], the second byte on bits [55:48], and so forth.

Similar to the TX Local bus interface, portions of packets are written on the bus in the full width of the bus unless `rx_eopout` is asserted. When `rx_eopout` is asserted, the `rx_mtyout` bus indicates how many byte lanes in the data bus are invalid. The encoding is the same as for `tx_mtyin`.

During the last cycle of a packet, when `rx_eopout` is asserted with `rx_enaout`, `rx_errout` can also be asserted, which indicates one of the following:

- The packet was sent with the "error flag" set.
- An error, such as a CRC24 error, was detected some time during the receipt of the packet.

When `rx_errout` is asserted, the value of `rx_mtyout[2:0]` is always 000. The other bits of `rx_mtyout` are as usual.

There is no mechanism to back pressure the RX Local bus interface. The user logic must be capable of receiving data when `rx_enaout` is asserted, and use the Interlaken flow control mechanism to stop the far device (the one sending data to the Interlaken 150G IP core) from sending more data if needed.

The data provided by the RX Local bus interface is in the same sequence as it is received from the Interlaken bus. Packets can be interleaved and are distinguished using the channel number presented on `rx_chanout`.

Interlaken Segmented LBUS Specification

This section describes the segmented LBUS protocol for the Interlaken system-side interface. Throughout this section, a segmented LBUS which consists of four "segments", each one 128-bits wide for a total of 512 bits, is used as an example.

Summary

The disadvantage of a wide non-segmented LBUS is the loss of potential bandwidth that occurs at the end of a packet when the size of the packet is not a multiple of the LBUS width. The segmented LBUS overcomes this problem to a large extent. The segmented LBUS is a collection of narrower LBUSes, each 128 bits wide, with multiple transfers presented in parallel during the same clock cycle. Each segment has all the control signals associated with a complete 128-bit LBUS. The 128-bit segments are ordered 0 to 3 for a 512-bit LBUS. Signals for each segment are listed in Table 3-1:

Table 3-1: **TX and RX Signals**

Segment Number	TX Signals	RX Signals
0	tx_datain0[127:0] tx_chanin0[7:0] tx_enain0 tx_sopin0 tx_eopin0 tx_errin0 tx_mtyin0[3:0] tx_bctlin0	rx_dataout0[127:0] rx_chanout0[7:0] rx_enaout0 rx_sopout0 rx_eopout0 rx_errout0 rx_mtyout0[3:0]
1	tx_datain1[127:0] tx_chanin1[7:0] tx_enain1 tx_sopin1 tx_eopin1 tx_errin1 tx_mtyin1[3:0] tx_bctlin1	rx_dataout1[127:0] rx_chanout1[7:0] rx_enaout1 rx_sopout1 rx_eopout1 rx_errout1 rx_mtyout1[3:0]
2	tx_datain2[127:0] tx_chanin2[7:0] tx_enain2 tx_sopin2 tx_eopin2 tx_errin2 tx_mtyin2[3:0] tx_bctlin2	rx_dataout2[127:0] rx_chanout2[7:0] rx_enaout2 rx_sopout2 rx_eopout2 rx_errout2 rx_mtyout2[3:0]
3	tx_datain3[127:0] tx_chanin3[7:0] tx_enain3 tx_sopin3 tx_eopin3 tx_errin3 tx_mtyin3[3:0] tx_bctlin3	rx_dataout3[127:0] rx_chanout3[7:0] rx_enaout3 rx_sopout3 rx_eopout3 rx_errout3 rx_mtyout3[3:0]

Following is a detailed description of the signals associated with segment 0. Signals associated with other segments are defined similarly.

tx_datain0[127:0]

Transmit Data for LBUS segment0. This port contains data from the user logic. This port is only sampled in cycles in which the corresponding `tx_enain0` is sampled as a 1.

tx_chanin0[7:0]

Transmit Channel Number for LBUS segment 0. This port indicates the channel number of the data that is present on `tx_datain0` segment. This port is only sampled in cycles in which `tx_enain0` is sampled as 1. When the channel extension feature is implemented, this bus will be wider as required.

In packet mode, the channel number remains the same for the duration of the packet transfer from SOP to EOP. In burst-interleaved mode, the channel number can change for each burst.

tx_enain0

Transmit Enable for LBUS segment 0. This signal qualifies the other signals of segment 0 of the TX LBUS Interface. Signals of segment 0 are only valid in cycles in which `tx_enain0` is sampled as a 1.

tx_sopin0

Transmit Start-Of-Packet (SOP) for LBUS segment 0. When this signal is sampled as a 1, it indicates that the SOP is present on `tx_datain0`. This signal is only valid in cycles in which `tx_enain0` is sampled as a 1.

tx_eopin0

Transmit End Of Packet (EOP) for LBUS segment 0. When this signal is sampled as a 1, it indicates that the EOP is present on `tx_datain0`. This signal is only valid in cycles in which `tx_enain0` is sampled as a 1.

tx_errin0

Transmit Error for LBUS segment 0. When this signal is sampled as a 1, it indicates an error in the packet whose data is present on `tx_datain0`. This signal is only valid in cycles when both `tx_enain0` and `tx_eopin0` are sampled as a 1.

tx_mtyin0[3:0]

Transmit Empty for LBUS segment 0. This bus is used to indicate how many bytes of the `tx_datain0` bus are empty or invalid for the last transfer of the current packet. This bus is sampled only in cycles that `tx_enain0` and `tx_eopin0` are sampled as 1.

When `tx_eopin0` and `tx_errin0` are sampled as 1, the value of `tx_mtyin0[2:0]` is ignored as treated as if it was 000, while `tx_mytin0[3]` is used as usual.

tx_bctlin0

Transmit force insertion of Burst Control word at LBUS segment 0. This input is used to force the insertion of a Burst Control Word into the outbound data stream. When `tx_bctlin0` and `tx_enain0`, are sampled as 1, a Burst Control word is inserted before the data on the `tx_datain0` bus is transmitted even if one is not required to observe the BurstMax parameter.

This input is used by the enhanced scheduling algorithm, external to the Interlaken IP Core. Enhanced Scheduling is required for the segmented LBUS.

This signal should be used carefully and sparingly. If incorrectly asserted, the result can be a conflict with an implied Burst Control Word, resulting in an error.

rx_dataout0[127:0]

Receive Data for LBUS segment 0. The value of the bus is only valid in cycles during which `rx_enaout0` is sampled as 1.

rx_chanout0[7:0]

Receive Channel Number for LBUS segment 0. This bus indicates the channel number of the data that is present on the `rx_dataout0` segment. This port is only valid in cycles in which `rx_enaout0` is sampled as 1. When the channel extension feature has been implemented, this bus will be wider.

rx_enaout0

Receive Enable for LBUS segment 0. This signal qualifies the other signals of segment 0 of the RX LBUS interface. Signals of segment 0 are only valid in cycles in which `rx_enaout0` is sampled as a 1.

rx_sopout0

Receive SOP for LBUS segment 0. When this signal is sampled as a 1, it indicates that the SOP is present on `rx_dataout0`. This signal is only valid in cycles in which `rx_enaout0` is sampled as a 1.

rx_eopout0

Receive EOP for LBUS segment 0. When this signal is sampled as a 1, it indicates that the EOP is present on `rx_dataout0`. This signal is only valid in cycles in which `rx_enaout0` is sampled as a 1.

rx_errout0

Receive Error for LBUS segment 0. When this signal is sampled as a 1, it indicates an error in the packet whose data is present on `rx_dataout0`. This signal is only valid in cycles when both the corresponding `rx_enaout0` and `rx_eopout0` are sampled as a 1.

rx_mtyout0[3:0]

Receive LBUS Empty. This bus indicates how many bytes of the `rx_dataout0` bus are empty or invalid for the last transfer of the current packet. This bus is only valid in cycles when both `rx_enaout0` and `rx_eopout0` are sampled as 1.

When `rx_errout0` and `rx_enaout0` are sampled as 1, the value of `rx_mtyout[2:0]` is always 000, while `rx_mtyout0[3]` is set as usual.

Segmented LBUS Protocol

The transmitter accepts packet-oriented data. The transmitter also has control/ configuration inputs to shape the data packaging to meet design-specific requirements.

The receiver accepts Interlaken bitstreams from the SerDes and provides packet-oriented data to the user-side segmented LBUS.

The transmit and receive segmented LBUS directions are not symmetrical. That means, for example, that there are some signals which have no corresponding signal in the other direction. In addition, the rules are not the same in both directions. For example, the transmit side inserts Burst Control Words with certain restrictions, but on the receive side the Burst Control Words are removed, which can sometimes allow packets to be more closely spaced on the LBUS. As a result of these differences, it is not advisable, or even possible, to directly connect an RX LBUS to a TX LBUS without a buffer and logic to handle the signal differences.

TX LBUS Interface

The synchronous TX Local bus interface accepts packet-oriented data of arbitrary length. It can accept data in either Packet Mode or Burst-Interleaved Mode.

All signals are synchronous relative to the rising-edge of the `clk` port. Figure 3-5 shows a sample waveform for data transactions for two consecutive 65-byte packets using a 512-bit segmented bus.

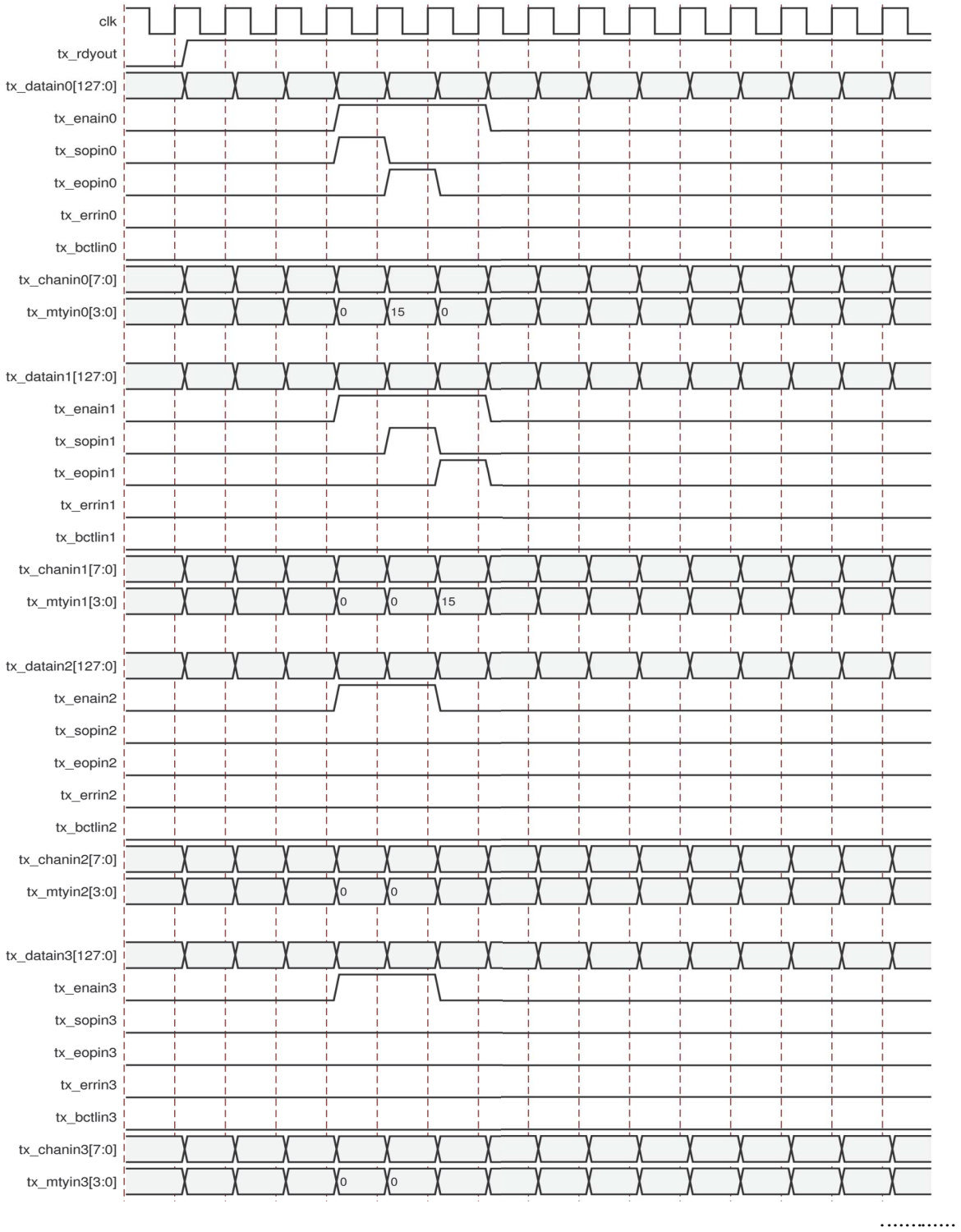

Figure 3-5: **Sample TX Waveform with a 512-bit Segmented LBUS Interface**

TX Transactions

Data is transferred on a given tx_datain<N> segment when the corresponding tx_enain<N> is asserted. The TX core logic stores incoming data and does not forward it until it has sufficient quantity for a complete burst. Consequently, it is acceptable to have clock cycles in which none of the tx_enain<N> signals are active. However, in any cycle where any tx_enain<N> is asserted, it is required for tx_enain0 to be asserted. Furthermore, segments must be filled in sequence with no gaps between active segments. This means that, for example for a 512-bit bus, if tx_enain<i> is set to 1, then tx_enain<i-1> must also be set to 1, for *i* ranging from 1 to 3.

The start of a packet is identified by the assertion of tx_sopin<N> with the corresponding tx_enain<N>. Similarly, the end of a packet is identified by the assertion of tx_eopin<N> with the corresponding tx_enain<N>. Both tx_sopin<N> and tx_eopin<N> can be asserted on a given cycle. This occurs for packets that are less than or equal to the LBUS width. Furthermore, both tx_sopin<N> and tx_eopin<N> can be asserted for a given segment on a given cycle. This occurs for packets that are less than or equal to 16 bytes (the segment size).

The channel number for a packet is presented on the tx_chanin<N> input of the corresponding segment and must be valid for every segment where tx_enain<N> is asserted. After SOP has been asserted for a certain channel number, it cannot be asserted again with that channel number until EOP has been asserted for the same channel number.

The first 16 bytes of a packet must be presented on a given tx_datain<N> segment during the cycle that the corresponding tx_sopin<N> and tx_enain<N> are asserted. In other words, the SOP is segment aligned. Subsequent 16-byte chunks of data are transferred during segments that follow. For each of those segments, the corresponding tx_sopin<N> must be negated. The first byte of the packet is written on bits [127:120] of the segment, the second byte on bits [119:112], and so forth.

The last bytes of the packet are transferred on the tx_datain<N> segment whose corresponding tx_eopin<N> is asserted. Unless tx_eopin<N> is asserted, all 16 bytes of tx_datain<N> must contain valid data whenever tx_enain<N> is asserted. Note that if burst-interleaved mode is employed, then segments containing data from other packets can be interleaved with segments containing data for a given packet. The tx_chanin<N> input identifies the packets from different channels.

During the segment containing the last bytes of a packet, the tx_mtyin<N> port reflects how many bytes of the corresponding tx_datain<N> are invalid (or empty). A given tx_mtyin<N> port only has meaning during cycles when both the corresponding tx_enain<N> and tx_eopin<N> are asserted. If tx_mtyin<N> has a value of 0x0, there are no empty byte lanes (that is, all bits of the segment are valid). If tx_mtyin<N> has a value of 0x1, then one byte lane is empty, specifically tx_datain<N>[7:0] does not contain valid data. If tx_mtyin<N> has a value of 0x2, then two byte lanes are empty — specifically tx_datain<N>[15:0] does not contain valid data. If tx_mtyin<N> has a value of 0x3, then three byte lanes are empty — specifically tx_datain<N>[23:0] does not contain valid data. And so forth for other possible values of tx_mtyin<N>.

During the segment containing the last bytes of a packet, when tx_eopin<N> is asserted with tx_enain<N>, the corresponding tx_errin<N> can also be asserted. This marks the packet as being in error and this information is included in the final Interlaken Control Word associated with this packet. When tx_eopin<N> and tx_errin<N> are sampled as 1, the value of tx_mtyin<N>[2:0] is ignored and treated as equal to 000, while tx_mtyin<N>[3] is used as usual.

Data can be safely written (one or more tx_enain<N> might be asserted) whenever tx_rdyout is asserted. After tx_rdyout is negated, additional writes, using tx_enain<N>, can be safely performed provided tx_ovfout is never asserted. When responding to back-pressure during a clock cycle, none of the tx_enain<N> can be active. When tx_rdyout is asserted again, additional data can be written. If the back-pressure mechanism is violated, tx_ovfout is asserted to indicate the violation.

The number of additional writes that can be performed after tx_rdyout is negated is determined by the signal ctl_tx_rdyout_thresh. If tx_ovfout is ever asserted, the TX core must be reset before resuming normal operation.

See Transmitter FIFO Threshold for more information on setting ctl_tx_rdyout_thresh.

RX LBUS Interface

The synchronous RX Local bus interface provides packet-oriented data much like the TX Local bus interface accepts. All signals are synchronous with the rising edge of the Local bus clock. Figure 3-6 shows a sample waveform for two data transactions for 65-byte packets using a 512-bit segmented LBUS.

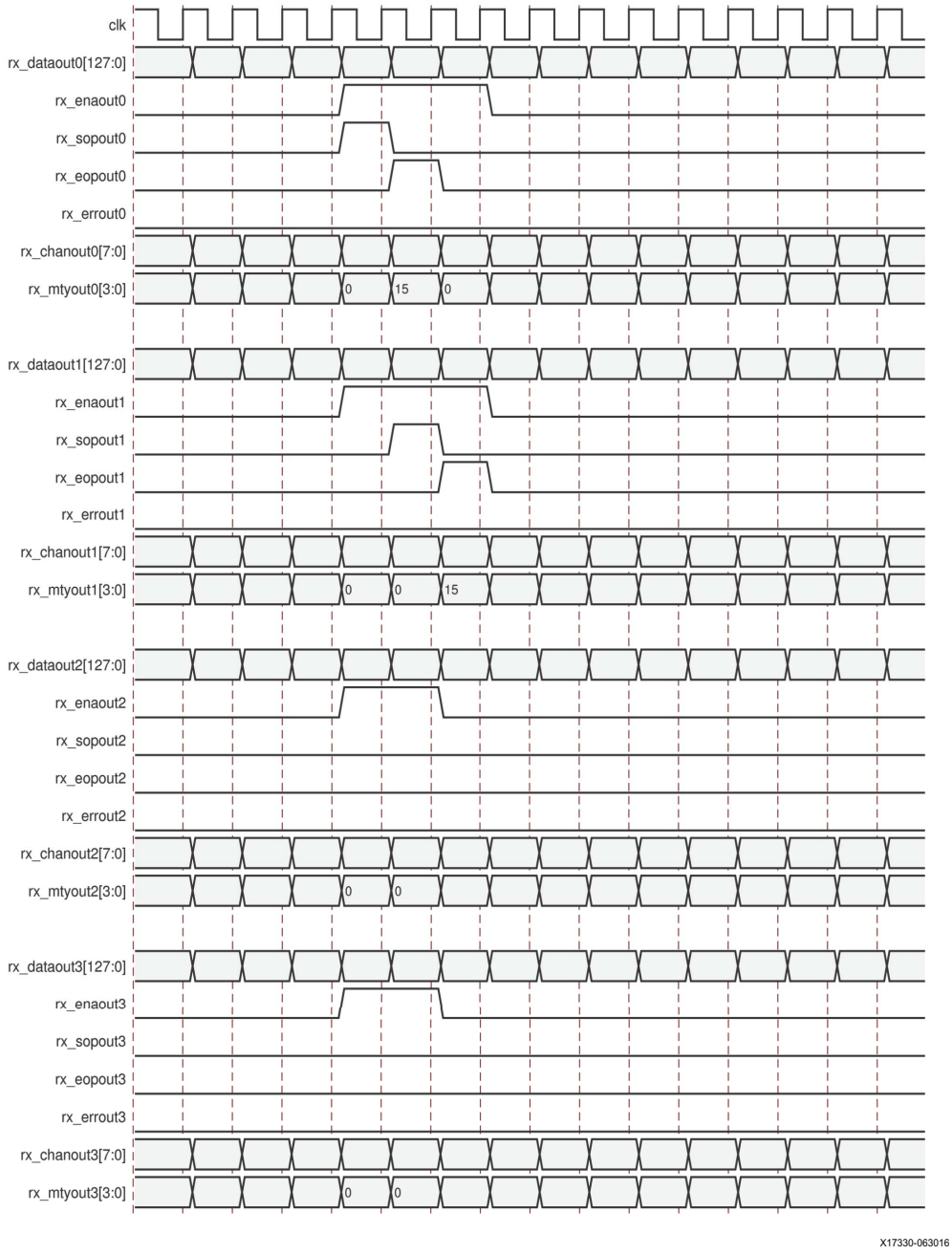

Figure 3-6: **Sample RX Waveform with a 512-bit Segmented LBUS Interface**

Similar to the TX Local bus interface, the RX LBUS is divided into 128-bit segments, with multiple transfers presented in parallel during the same clock cycle. The first of the 128-bit transfers occurs on segment 0 (that is, `rx_dataout0`), the second on segment 1 (that is, `rx_dataout1`), and so forth.

Data is transferred on a given rx_dataout<N> segment when the corresponding rx_enaout<N> is asserted. The RX core logic stores incoming data and does not forward it until it has a sufficient quantity. Consequently, there can be clock cycles where none of the rx_enaout<N> signals are asserted.

The start of a packet is identified by the assertion of rx_sopout<N> with the corresponding rx_enaout<N>. The first byte of the packet is written on bits [127:120] of the segment, the second byte on bits [119:112], and so forth.

Similarly, the end of a packet is identified by the assertion of rx_eopout<N> with the corresponding rx_enaout<N>. Both rx_sopout<N> and rx_eopout<N> can be asserted on a given cycle. This occurs for packets that are less than or equal to the LBUS width. Furthermore, both rx_sopout<N> and rx_eopout<N> can be asserted for a given segment on a given cycle. This occurs for packets that are less than or equal to 16 bytes (the segment size). When rx_eopout<N> is asserted, the rx_mtyout<N> bus indicates how many byte lanes in the data bus are invalid. The encoding is the same as for tx_mtyin<N>.

During the last cycle of a packet, when rx_eopout<N> is asserted with rx_enaout<N>, rx_errout<N> can also be asserted to indicate an error in the packet ended on the corresponding segment.

Bus Rules

A number of rules govern the successful use of the segmented LBUS protocol.

Segment Ordering

The 128-bit segments are ordered 0 to 3 (for a 512-bit LBUS). The first of the 128-bit transfers occurs on segment 0, the second on segment 1, and so forth.

During each local bus clock cycle that data is transferred on the segmented LBUS, segment 0 must be active. The segmented bus is aligned such that the first bit of the incoming data is placed at the MSB of segment 0.

Active Segments

Data is transferred in a segment on the TX interface when the corresponding tx_enain<N> is a value of 1. The TX interface buffers data and does not forward until it has a sufficient quantity. Therefore, it is acceptable to have clock cycles in which none of the tx_enain<N> signals are active. However, during a clock cycle with tx_enain0 active, segments must be filled in sequence with no gaps between active segments. The following are some of the illegal combinations of enabled LBUS segments:

- tx_enain0=0, tx_enain1=1, tx_enain2=1, tx_enain3=1
- tx_enain0=1, tx_enain1=0, tx_enain2=1, tx_enain3=1
- tx_enain0=1, tx_enain1=1, tx_enain2=0, tx_enain3=1

Data is transferred in a segment on the RX interface when the corresponding `rx_enaout<N>` is a value of 1. Similarly, the RX interface buffers data and does not forward until it has a sufficient quantity. Therefore, there will be clock cycles in which none of the `rx_enaout<N>` signals are active.

TX Back Pressure

The optimal use of bandwidth requires that TX Local bus data be able to be written at a rate faster than can be delivered on the serial interface. This means that there must be back-pressure, or flow-control, on the TX segmented LBUS. The signals used to implement back-pressure are `tx_rdyout`, `tx_ovfout`, and `ctl_tx_rdyout_thresh`. These signals are common for all segments and operate in the same manner as with the regular LBUS. When responding to back-pressure during a clock cycle, none of the `tx_enain<N>` signals can be active.

Gaps

The purpose of the segmented LBUS is to provide a means to optimally use the data bus. Therefore, as discussed in the section Active Segments, segments must be filled in sequence with no gaps between used segments. However, if a segment has an EOP, the following segments might be inactive. For example, the following are permitted during a single clock cycle:

- tx_enain0=1 tx_eopin0=0 tx_enain1=1 tx_eopin1=0
- tx_enain2=1 tx_eopin2=1 tx_enain3=0 tx_eopin3=0

or

- tx_enain0=1 tx_eopin0=0 tx_enain1=1 tx_eopin1=1
- tx_enain2=0 tx_eopin2=0 tx_enain3=0 tx_eopin3=0

or

- tx_enain0=1 tx_eopin0=1 tx_enain1=0 tx_eopin1=0
- tx_enain2=0 tx_eopin2=0 tx_enain3=0 tx_eopin3=0

Further, when BurstMax is reached on a segment, the following segments can be inactive.

Interlaken Burst Rules for Segmented LBUS

Fundamentally, Interlaken transfers packets between devices by breaking them into bursts. The bursts have defined characteristics that are closely tied to the operation of the segmented LBUS. The TX segmented LBUS adapter circuit requires that certain rules be followed for proper operation to ensure that Interlaken bursts are generated correctly. The rules are as summarized as follows.

- The segmented LBUS requires that the Interlaken Enhanced Scheduling algorithm is observed. This impacts the last two bursts of a packet transfer (the EOP and EOP-1 bursts).
- BurstShort must be at least equal to the width of the entire LBUS. For a 512-bit segmented LBUS the minimum value of BurstShort is therefore 64 bytes.
- Only 1 SOP is permitted in a clock cycle (except in the case of a Look-Aside configuration).
- Only 1 Burst Control Word is permitted in a clock cycle. Burst Control Words are usually generated automatically, so care must be taken to understand what transaction sequence will result in a Burst Control Word being generated.
- Idle segments (no data of any kind) are only allowed if all four segments are idle, if they are after a segment with an EOP, or if they are after a segment where BurstMax is reached.
- No segments with data of any kind can follow idle segments when they occur.
- Burst Control Words can be forced if necessary using the `tx_bctlin` signal, for example to avoid the generation of two Burst Control Words in a particular cycle.

If these rules are not followed, the signal `stat_tx_burst_err` will be asserted.

The rules are described in more detail in the following subsections.

Enhanced Scheduling

The segmented LBUS must be used in conjunction with the Enhanced Scheduling algorithm described in the Interlaken Protocol Definition. Among the requirements of this algorithm are:

- All bursts except the last two must be equal to BurstMax.
- Bursts must be written to the LBUS in their entirety before changing channels or writing the next burst.
- The last two bursts of a packet are delineated using `bctlin<N>` and by knowing the value of BurstMin.

Burst Length

In Interlaken, a burst is defined as the number of 64-bit data words between two control words. Data for different channels can be interleaved between control words. The segmented LBUS requires that bursts, not ending with an EOP, be multiples of the full width of the segmented LBUS. Consequently, for a segmented LBUS with four segments, bursts, not ending with an EOP, must be 64-bytes, 128-bytes, 192-bytes, 256-bytes, and so forth.

The Interlaken specification describes an enhanced scheduling algorithm. The previous example is the same as a scheduler with the enhanced algorithm that has a BurstMin of 64-bytes.

Burst Control Words

Burst Control Words are either forced via a `tx_sopin<N>` and `tx_bctlin<N>` input, forced via a change of channel on `tx_chanin<N>`, or implied by the value of BurstMax. The segmented LBUS requires that there be only one Burst Control Word per clock cycle. Consequently, two bursts, implied or forced, cannot begin in the same clock cycle. The signal `stat_tx_burst_err` is asserted if there is more than one Burst Control Word that occurs in the same cycle.

BurstShort

BurstShort must be at least equal to the total LBUS width AND a multiple of 32 bytes. For a 512-bit segmented LBUS, BurstShort can be 64 bytes, 96 bytes, 128 bytes, and so on up to 256 bytes. BurstShort must always be less than or equal to BurstMax.

The transmit logic of the Interlaken IP core never violates the BurstShort value set by the `ctl_tx_burstshort` input bus. However, more than the absolute minimum required number of idle control words can be injected depending on the burst size and which segment the burst ends. Xilinx requires designing the transmitting scheduler so that burst sizes are always equal to BurstMax (see the next section) except for the last two transfers of a packet (EOP and EOP-1).

The required value of BurstShort must be matched by the link partner.

BurstMax Requirements

BurstMax must be greater than or equal to BurstShort. BurstMax must be a multiple of 64 bytes.

The required value of BurstMax must be matched by the link partner. You must consult the user guide applicable to the link partner to find out how to set BurstMax in that device. If BurstMax is fixed in the link partner, the Xilinx® Interlaken IP needs to be programmed to that value using the signals `ctl_tx_burstmax` and `ctl_rx_burstmax`.

BurstMin Requirements

BurstMin is a virtual value used for the Enhanced Scheduling Algorithm.

BurstMin must be less than or equal to half BurstMax AND a multiple of the LBUS width.

Note: It is possible to set BurstShort = 64 Bytes and BurstMax = 64 Bytes. In cases such as those, the calculation for BurstMin does not apply, because bursts are not allowed to be less than BurstShort. This means that the enhanced scheduling algorithm needs to be modified such that all bursts are 64 Bytes. Also there will be bandwidth wasted due to idle byte insertion.

Channel Changes

Channel changes are only permitted after a packet has been fully written to the LBUS. This is not the case when using Burst Interleave Mode.

Exceptions to the Rules

Interlaken Look-Aside

Look-Aside is a variation of Interlaken which permits short packets to be transferred. These are often just the header of a larger packet. Therefore it makes sense to allow shorter bursts and multiple Burst Control Words per cycle.

Burst-Interleaved Mode

Burst Interleave mode means that a packet transfer can be interrupted by the transfer of another packet with a different channel number. When changing channels, a Burst Control Word is automatically generated. Channel changes are only permitted after a burst has been fully written to the LBUS.

Examples

Due to the many possible combinations of SOP, EOP, data, and idle segments, it is instructive to look at a number of examples to see how the rules of the segmented LBUS can be followed or broken.

The following examples illustrate segmented LBUS cycles covering various combinations of SOP (Start of Packet), DAT (data in the middle of a packet), EOP (End of Packet), and IDLE (`tx_enain<N>`=0) segments. Valid and invalid cycles are shown.

The segmented LBUS is assumed to be 512-bits wide and each segment is 128-bits wide (16 bytes). The TX direction is illustrated. The RX direction has analogous behavior but there will be no invalid cycles on the receive segmented LBUS.

The RX segmented LBUS can contain more than one SOP and more than one EOP, in contrast to the way the TX must function. This is because there is no requirement to accommodate Burst Control Words on the receive side LBUS (they have already been processed) and therefore packets can be packed as efficiently as possible when they arrive.

As you look at the examples, observe where the Burst Control words have been inserted and understand why this is the case. When designing your scheduler and visualizing the location of Burst Control words, you can easily predict whether the transfer follows the segmented LBUS rules or not. You need to make sure that there are Burst Control Words between bursts and that there is never more than one Burst Control Word in a clock cycle.

Examples of Permitted Transfers

Figure 3-7 and Figure 3-8 show a number of possible valid TX segmented LBUS cycles. In these examples, BurstMax has been set to 256 and BurstShort to 64. The different shadings represent different bursts from different channels.

BUS Cycle:											
Seg 0	SOP	IDLE	SOP	SOP	IDLE	DAT	DAT	IDLE	DAT	SOP	IDLE
Seg 1	DAT	IDLE	DAT	DAT	IDLE	EOP	DAT	IDLE	EOP	DAT	IDLE
Seg 2	DAT	IDLE	EOP	DAT	IDLE	SOP	DAT	IDLE	S/EOP	DAT	IDLE
Seg 3	EOP	IDLE	IDLE	DAT	IDLE	DAT	DAT	IDLE		DAT	IDLE
tx_rdyout	1	1	1	1	1	1	1	0		0	0
tx_ovfout	0	0	0	0	0	0	0	0		0	0

X14694-042217

Figure 3-7: **Valid TX Segmented LBUS Cycles (Packet Mode)**

BUS Cycle	1	2	3	4	5	6	7	8	9	10	11
Seg 0	EOP	DAT	DAT	DAT	IDLE	DAT	DAT	DAT	SOP	DAT	IDLE
Seg 1	DAT	DAT	DAT	DAT	IDLE	DAT	DAT	IDLE	DAT	DAT	IDLE
Seg 2	DAT	DAT	DAT	DAT	IDLE	DAT	DAT	IDLE	DAT	EOP	IDLE
Seg 3	DAT	DAT	DAT	DAT	IDLE	DAT	DAT	IDLE	EOP	IDLE	IDLE
tx_rdyout	1	1	1	1	1	1	1	0	1	0	0
tx_ovfout	0	0	0	0	0	0	0	0	0	0	0

X14695-042217

Figure 3-8: **Valid TX Segmented LBUS Cycles (Burst-Interleaved Mode)**

Examples of Transfers with Errors

Figure 3-9 shows several invalid TX segmented LBUS cycles as indicated by the ovals. In these examples, BurstMax has been set to 256 and BurstShort to 64. It is also assumed that the link partner operates in packet mode.

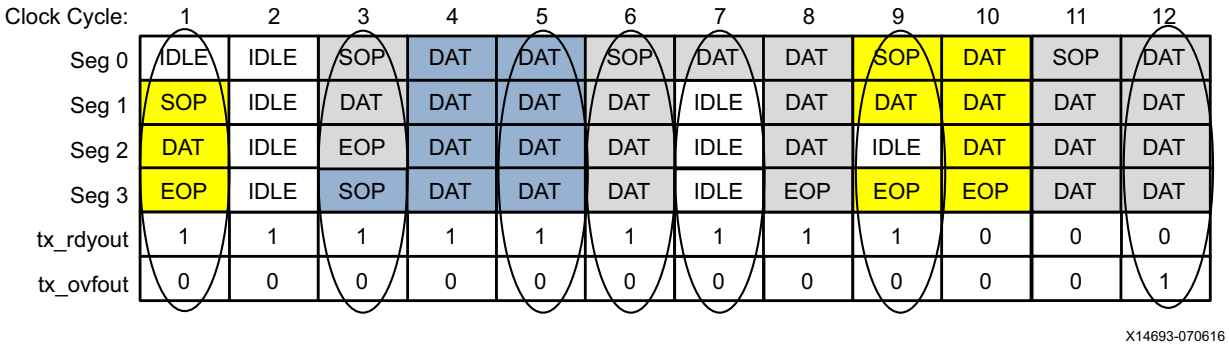

Figure 3-9: **Invalid TX Segmented LBUS Cycles**

- Cycle 1 is invalid as segment 0 is inactive while other segments are active.
- Cycle 3 is not valid because it contains two SOPs.
- Cycle 5 does not contain an EOP even though there is an SOP in the next cycle.
- Cycle 6 has an SOP even though the preceding packet was not closed with an EOP.
- Cycle 7 contains idles after an active segment even though there is no EOP or BurstMax.
- Cycle 9 contains an idle segment during a packet transfer which is not permitted by the segmented LBUS rules.
- Cycle 12 must never be performed as `tx_ovfout` is triggered due to `tx_rdyout` not being observed. In the event of `tx_ovfout` being asserted, the TX core should be reset.

Interlaken Look-Aside Examples

Interlaken Look-Aside is a variation of Interlaken which allows shorter bursts. This is illustrated in these examples.

Note that for Interlaken Look Aside, multiple Burst Control Words are possible in one cycle. Refer to your particular configuration for minimum BurstShort values. Up to 4 BCWs are possible for some configurations

Example: Interlaken LA TX LBUS transactions (A set of rotating colours denote the frames):														
BUSCYCLE:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Seg 0	SOP	S/EOP	DAT	DAT	DAT	DAT	DAT	SOP	EOP	SOP	S/EOP	S/EOP	DAT	DAT
Seg 1	EOP	S/EOP	DAT	DAT	DAT	DAT	DAT	DAT	S/EOP	DAT	S/EOP	SOP	DAT	DAT
Seg 2	S/EOP	SOP	DAT	DAT	DAT	DAT	EOP	DAT	SOP	DAT	S/EOP	EOP	DAT	EOP
Seg 3	S/EOP	DAT	DAT	DAT	DAT	DAT	S/EOP	DAT	EOP	EOP	S/EOP	SOP	DAT	S/EOP
Example: Interlaken LA RX LBUS transactions (A set of rotating colours denote the frames):														
BUSCYCLE:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Seg 0	SOP	IDLE	IDLE	S/EOP	IDLE	SOP	IDLE	DAT	DAT	IDLE	DAT	DAT	IDLE	EOP
Seg 1	EOP	IDLE	IDLE	S/EOP	IDLE	DAT	IDLE	DAT	DAT	IDLE	DAT	DAT	IDLE	S/EOP
Seg 2	S/EOP	IDLE	IDLE	IDLE	IDLE	DAT	IDLE	DAT	DAT	IDLE	DAT	DAT	IDLE	SOP
Seg 3	S/EOP	IDLE	IDLE	IDLE	IDLE	DAT	IDLE	DAT	DAT	IDLE	DAT	DAT	IDLE	DAT

X14717-062516

Figure 3-10: **Segmented LBUS Transactions for Interlaken Look-Aside**

Notice that cycle 10 in the top row shows the transfer of 4 packets. The minimum value for BurstShort is 16 bytes with the 512-bit segmented LBUS for Look-Aside mode. For packets smaller than 16 bytes, the remainder of the burst will be filled with idles.

Status/Control Interface

The Status/Control interface allows you to set up the core configuration and monitor the core status. The following subsections describe the various Status and Control signals.

Note: Most of the following status signal descriptions assume a good understanding of the Interlaken Protocol. See the *Interlaken Protocol Definition, Revision 1.2* [Ref 1] for more details.

RX Meta Frame Status

The Interlaken protocol requires that each lane align or synchronize to incoming words using the procedure described in the *Interlaken Protocol Definition Revision 1.2* [Ref 1]. The Interlaken 150G IP core provides status bits to indicate the state of word boundary synchronization and lane alignment. All signals are synchronous with the rising edge of `clk`, and a detailed description of each signal is included in this section.

stat_rx_synced[LANES-1:0]

When a bit of this bus is 0, it indicates that word boundary synchronization of the corresponding lane is not complete or that an error has occurred as identified by another status bit.

When a bit of this bus is 1, it indicates that the corresponding lane is word boundary synchronized and is receiving Meta Frame Synchronization Words and Scrambler State Control Words as expected.

stat_rx_synced_err[LANES-1:0]

When a bit of this bus is 1, it indicates one of several possible failures on the corresponding lane:

- Word boundary synchronization in the lane was not possible using Framing bits [65:64].
- After word boundary synchronization in the lane was achieved, errors were detected on Framing bits [65:64].
- After word boundary synchronization in the lane was achieved, a valid Meta Frame Synchronization Word was never received.

The bits of the bus remain asserted until word boundary synchronization occurs or until some other error or failure is signaled for the corresponding lane.

stat_rx_mf_len_err[LANES-1:0]

When a bit of this bus is 1, it indicates that Meta Frame Synchronization Words are being received but not at the expected rate in the corresponding lane. The transmitter and receiver must be re-configured with the same Meta Frame length The bits of the bus remain asserted until word boundary synchronization occurs or until some other error or failure is signaled for the corresponding lane.

stat_rx_mf_repeat_err[LANES-1:0]

After word boundary synchronization is achieved in a lane, if a bit of this bus is a 1, it indicates one of the following:

- Four consecutive invalid Meta Frame Synchronization Words were detected in the corresponding lane.
- Three consecutive invalid Scrambler State Control Words were detected in the corresponding lane.

The bits of the bus remain asserted until word boundary synchronization occurs or until some other error or failure is signaled for the corresponding lane.

stat_rx_descram_err[LANES-1:0]

When a bit of this bus is 1, it indicates that a Scrambler State Control Word with an unexpected value was received on the corresponding lane. This bit is only asserted after word boundary synchronization is achieved. This output is asserted for one clock period each time a descrambler error is detected.

stat_rx_mf_err[LANES-1:0]

When a bit of this bus is 1, it indicates that an invalid Meta Frame Synchronization Word was received on the corresponding lane. This bit is only asserted after word boundary synchronization is achieved. This output is asserted for one clock period each time an invalid Meta Frame Synchronization Word is detected.

stat_rx_aligned

When `stat_rx_aligned` is a value of 1, all of the lanes are aligned or de-skewed as explained in the Interlaken specification and the receiver is ready to receive packet data.

stat_rx_aligned_err

When `stat_rx_aligned_err` is a value of 1, one of the following occurs:

- Lane alignment fails after several attempts
- Lane alignment is lost (`stat_rx_aligned` is asserted and then it is negated)

stat_rx_framing_err[LANES-1:0]

When a bit of this bus is 1, an illegal framing pattern is detected on the corresponding lane after word boundary synchronization. If this error is detected after lane alignment, the error is treated like a CRC24 error (see Error Handling).

This output is asserted for one clock period each time an illegal framing pattern is detected.

RX Error Status

The Interlaken 150G IP core provides status signals to identify Interlaken data transmission protocol violations in sequences of Control and Data words. These are errors independent of the status of the Meta Frame. Generally, these signals do not indicate a failure on the part of the sending transmitter but some type of corruption during the transmission.

All signals are synchronous with the rising edge of `clk` and a detailed description of each signal follows.

stat_rx_err

This signal indicates whether an Interlaken protocol formatting error or a CRC24 error occurred in a Control Word. When operating in packet mode, this signal also indicates missing SOP or missing EOP errors. A value of 1 indicates an error occurred. Whenever this signal is asserted, all open packets are marked as containing errors as specified by the Interlaken Protocol Definition, Revision 1.2 [Ref 1]. By definition, there is no mechanism provided by Interlaken to associate a CRC24 or similar error with individual packets.

This signal is asserted for one clock period each time an error is detected.

stat_rx_msop_err

Packets received with a particular channel address must begin with a valid Start-of-Packet (SOP). If data is detected for a particular channel without a valid SOP, this signal is asserted for a single Local bus clock cycle. Additionally, the required SOP is inserted before the data and an error is signaled in the End-of-Packet (EOP) cycle via the `rx_errout` signal (or the corresponding `rx_errout<N>` signal in case of a segmented LBUS). This signal does not exist if the Interlaken core is built to operate in packet mode only.

This signal is available as a status signal to indicate a missing SOP error condition occurred. No indication is provided on the LBUS which packet had the missing SOP. The packet is simply marked as containing an error. This is because a missing SOP is almost always associated with other errors that cannot be associated with a particular packet.

The purpose of SOP insertion is to ensure that packets for a particular channel are always delivered on the RX LBUS with the beginning with an SOP and ending with an EOP to remove the need for user logic to perform bus protocol checking. The `stat_rx_msop_err` status signal indicates that this function is being performed and for most applications can be ignored.

stat_rx_meop_err

Packets received with a particular channel address must begin with a valid SOP and end with a valid EOP. If an SOP is detected without receiving an EOP for the previous packet, this signal is asserted for a single Local bus clock cycle. Additionally, the extra SOP is deleted, the packets are merged together, and an error is signaled with the EOP by the `rx_errout` signal (or the corresponding `rx_errout<N>` signal in case of a segmented LBUS).

This signal is available as a status signal to indicate that a missing EOP error condition occurred and that SOP deletion occurred. No indication is provided on the local bus about which packet is actually a merged packet. The packet is marked as containing an error. This is because a missing EOP is almost always associated with other errors that cannot be associated with a particular packet.

The purpose of SOP deletion is to ensure that packets for a particular channel are always delivered on the RX Local bus beginning with an SOP and ending with an EOP to remove the need for user logic to perform bus protocol checking. The `stat_rx_meop_err` status signal indicates that this function is being performed and for most applications can be ignored.

stat_rx_burst_err

This signal is asserted if:

- BurstShort violation is detected
- Burst length violation is detected

When this signal has a value of 1, it indicates one of the preceding burst errors has been detected. These errors are treated as CRC24 errors and all open packets are treated as being in error.

This signal is asserted for one clock period each time an error is detected.

A BurstShort error occurs when the spacing between Burst Control Words is less than the minimum BurstShort the RX can handle which depends on the variant configuration. A burst length violation occurs when the length of a received burst, other than that ending with an EOP, is not a multiple of the LBUS width.

TX Rate Limiting

The Interlaken 150G IP core rate limiter can be used to reduce the overall Data Word transmission rate. This is achieved by transmitting Idle Control Words in between packet segments to limit the effective data transfer rate. The purpose of transmitter rate limiting is to reduce buffering requirements by the receiving device and reduce the amount of flow control stalling that can otherwise be required.

Rate limiting is not a substitute for flow control but something that should be used in conjunction with flow control when a receiver cannot continuously accept Data Words at the full rate.

The rate limiter uses a token bucket scheme. A token represents a single byte. When the token bucket contains at least BurstMax number of tokens, up to BurstMax bytes are sent. When that has completed, the transmitter waits until there are at least BurstMax number of tokens in the bucket again before sending more data. The token count goes negative if it is necessary to send a burst of data that cannot be interrupted.

The token bucket is refilled at a specified interval with some number of tokens. This interval is specified in terms of local bus clock cycles.

During each local bus clock cycle, eight tokens are drained for each Interlaken Data Word that is forwarded. This is true even for EOP Data Words that contain less than eight valid bytes.

A description of the signals that set the characteristics of the rate limiter follows. All signals are synchronous with the rising edge of `clk`.

ctl_tx_rlim_enable

When this input is a value of 1, the rate limiter is enabled. When this input is a value of 0, the rate limiter is disabled.

This input should only be changed from a 0 to a 1 after appropriate values have been put on `ctl_tx_rlim_max`, `ctl_tx_rlim_delta`, and `ctl_tx_rlim_intv`.

ctl_tx_rlim_max[11:0]

This input defines the maximum number of tokens in the bucket in terms of bytes (a value of 1, means 1 byte). The number of tokens in the bucket never exceeds this value. This value must be at least BurstMax. (For example, if BurstMax is set for 256 bytes, this value should be at least 256).

The value of this input should not be changed when `ctl_tx_rlim_enable` is a value of 1.

Note: Xilinx has observed that rates closest to the expected rates are observed when `ctl_tx_rlim_max` is set to a value between 1 and 2 times the value of BurstMax.

ctl_tx_rlim_intv[7:0]

This input specifies the update interval which is the number of Local bus clock cycle between additions to the token bucket. The value of this input should not be changed when `ctl_tx_rlim_enable` is a value of 1.

RECOMMENDED: *Xilinx recommends values between 8 and 32 for this input.*

ctl_tx_rlim_delta[11:0]

This input specifies how many tokens are to be added to the bucket after each interval. A token is equal to 1 byte. This value must be greater than 0. The value of this input should not be changed when `ctl_tx_rlim_enable` is a value of 1.

Note: This value should be calculated based on the desired rate and the value in `ctl_tx_rlim_intv`.

Example: Programming the rate limiter assuming the following:

- The local bus clock frequency is 200 MHz
- BurstMax is 256 bytes
- The transmission rate is to be limited to 16 Gbits/s (or 2 Gbytes/s)
- The interval is arbitrarily chosen to be 16 local bus clock cycles

The value for `ctl_tx_rlim_delta` is:

= (Byte Rate × Interval) / Local bus Frequency

= (2e9 × 16) / (200e6)

= 160 bytes

The value for `ctl_tx_rlim_max` must be 256 or greater. Different values result in different shaping of traffic. Simulations must be done to select the proper value to get the desired packet rate.

CRC32 Diagnostics Checking

Interlaken implements a CRC32 check for each lane of the interface for monitoring the health of each lane. The Interlaken 150G IP core uses the following two signals for this function. All signal are synchronous with the rising edge of `clk`.

stat_rx_crc32_valid[LANES-1:0]

When a bit of this bus is 1, it indicates:

- The CRC32 in the most recently received Diagnostic Word on the corresponding lane was valid.
- The corresponding lane is word boundary synchronized.

When this bit is a value of 0, it indicates that a CRC32 error was detected or the corresponding lane is not word boundary synchronized.

stat_rx_crc32_err[LANES-1:0]

When a bit in this bus is 1, it indicates that after the corresponding lane was word boundary synchronized, a CRC32 error was detected. This output is asserted for one clock period each time a CRC32 error is detected.

Note: CRC32 errors do not affect word boundary synchronization. They are only reported as status indicators. Keep a count of how many CRC32 errors were detected for each lane to examine the health of each individual lane over a period of time. The checking of the Diagnostic Word only checks the CRC32 and does not check whether or not the unused bits of the Diagnostic Word, 57:34, are 0s as described in the specification.

Interlaken Status Messaging for the Receiver

The Meta Frame Diagnostic words calculate a CRC32 over all the data within the Meta Frame in a lane to help diagnose errors. The Interlaken protocol provides optional status messaging within these Diagnostic Words. This mechanism allows a receiver to communicate, through the adjacent transmitter or an out-of-band flow control interface, the health of each (received) lane and the overall health of the receiver interface to the other device.

The results of received Diagnostic Words are described in the following subsections. All signals are synchronous with the rising edge of `clk`.

stat_rx_diagword_intfstat[LANES-1:0]

Each bit of this bus reflects the value of bit[32], the interface health (Status Bit 0), in the most recently received Diagnostic Word on the corresponding lane. The value of this bit should be considered invalid and ignored if the corresponding bit in `stat_rx_crc32_valid[LANES-1:0]` is a value of 0.

stat_rx_diagword_lanestat[LANES-1:0]

Each bit of this bus reflects the value of bit[33], the lane health (Status Bit 1), in the most recently received Diagnostic Word on the corresponding lane. The value of this bit should be considered invalid and ignored if the corresponding bit in `stat_rx_crc32_valid[LANES-1:0]` is a value of 0.

Interlaken Status Messaging for the Transmitter

The transmitter is capable of inserting the status messaging as described in the Interlaken Protocol into the Meta Frame Diagnostics words. Feed these inputs based on the health of the receiver.

All signals are synchronous with the rising edge of `clk` and a detailed description of each signal follows.

ctl_tx_diagword_intfstat

This input is transmitted on bit[32], the interface health (Status Bit 0), of every Diagnostic Word on all of the lanes. A value of 1 is defined as a healthy condition.

You must drive proper data for this input. In typical applications, connect this input to the `stat_rx_aligned` output of the receiver block.

ctl_tx_diagword_lanestat[LANES-1:0]

Each bit of this bus is transmitted on bit[33], the lane health (Status Bit 1), of every Diagnostic Word for the corresponding lane. A value of 1 is defined to mean a healthy condition.

You must drive proper data for this input. In typical applications, connect this input to the `stat_rx_synced[LANES-1:0]` output of the receiver block.

Transmitter Multiple-Use Bits

Interlaken defines an 8-bit field in each Control Word as "Multiple-Use" bits. These bits are transmitted with every Control Word that is sent and can be used to transmit any information. For example, one of the bits can be used to represent a link-level flow control status.

The Interlaken 150G IP core provides a mechanism to set these bits to any desired value. All signals are synchronous with the rising edge of `clk` and a detailed description of each signal follows.

ctl_tx_mubits[7:0]

These inputs control the information contained in bits [31:24] of the Control words generated by the transmitter. The value of `ctl_tx_mubits[0]` appears in bit 24 of the next Control Word generated by the TX. The value of `ctl_tx_mubits[1]` appears in bit 25, and so forth.

Note: This bus is not present if the core is configured in Look-Aside mode. Instead, the `tx_chanin` bus is expanded for Look-Aside mode.

Receiver Multiple-Use Bits

Similar to the transmitter, the Interlaken 150G IP core extracts the "Multiple-Use" field from every received Control Word and outputs the information for your interpretation. All signals are synchronous with the rising edge of `clk` and a detailed description of each signal follows.

stat_rx_mubits[7:0]

These outputs contain the information in bits [31:24] of the Control words received by the receiver. The value of Control Word bit [24] appears on `stat_rx_mubits[0]`. The value of Control Word bit [25] appears on `stat_rx_mubits[1]`, and so forth.

Transmitter Flow Control Inputs

The core implements the Interlaken in-band flow control mechanism as described in section 5.3.4 of the *Interlaken Protocol Definition Revision 1.2* [Ref 1]. This mechanism communicates XON/XOFF (for example, for different channels) using the In-Band Flow Control bits of Control words. Additionally, the Multiple Use bits of Control Words can be used in a similar manner.

Inside each Interlaken Control Word are 16 bits of In-Band Flow Control information, bits[55:40], and a Reset Calendar bit, bit[56]. These bits are shared over the calendar length as described in the following subsections. The core has a fixed calendar length and provides one transmit bit and one receive bit for each calendar entry.

Note: The MAX_CALLEN parameter represents the supported calendar length for each specific Interlaken 150G IP core configuration.

By definition, XON is represented by 1, and XOFF is represented by 0 for both the transmitter and the receiver. All signals are synchronous with the rising edge of `clk` and a detailed description of each signal follows.

ctl_tx_fc_stat[MAX_CALLEN-1:0]

There is full flexibility to implement any mechanism to handle system-wide flow control. The Interlaken 150G IP core transmitter inputs the supplied calendar information and packs it into the Interlaken Control words and transmits it over the link. This mechanism allows you to take the system-wide parameters into account and optimize the buffering by implementing the most optimum flow control mechanism.

The operation is as described in the *Interlaken Protocol Definition, Revision 1.2* [Ref 1]. The first calendar entry, `ctl_tx_fc_stat[0]`, is sent in bit[55] of a Control Word with the Reset Calendar bit, bit[56], set to a value of 1. The next calendar entry, `ctl_tx_fc_stat[1]`, is sent in bit[54] of the same Control Word and so on to bit[40]. The seventeenth calendar entry, `ctl_tx_fc_stat[16]`, is sent in bit[55] of the next Control Word that has the Reset Calendar bit, bit[56], set to a value of 0, and so forth.

ctl_tx_fc_callen[3:0]

The flow control calendar length can be shorter than the MAX_CALLEN of the Interlaken 150G IP core configuration. When `ctl_tx_fc_callen` is a value of 0, the calendar length becomes 16 and only `ctl_tx_fc_stat[15:0]` signals are used. When `ctl_tx_fc_callen` is a value of 1, the calendar length becomes 32 and only `ctl_tx_fc_stat[31:0]` signals are used. And so forth. The typical valid settings (MAX_CALLEN=256) for calendar length are as follows:

- 0x0 = 16 entries
- 0x1 = 32 entries
- 0x3 = 64 entries
- 0x7 = 128 entries
- 0xF = 256 entries

All other values are reserved.

Note: The value selected for `ctl_tx_fc_callen` must be less than or equal to the value of MAX_CALLEN.This input should be static and must only be changed during reset.

Receiver Flow Control Outputs

The Interlaken 150G IP core receiver automatically extracts the flow control information received over the link and outputs the information. You can then interpret the flow control status and take the appropriate action if required.

Note: The MAX_CALLEN parameter represents the supported calendar length for each specific Interlaken 150G IP core configuration.

By definition, XON is represented by 1, and XOFF is represented by 0 for both the transmitter and the receiver. All signals are synchronous with the rising edge of `clk` and a detailed description of each signal follows.

stat_rx_fc_stat[MAX_CALLEN -1:0]

The operation is as described in the *Interlaken Protocol Definition Revision 1.2* [Ref 1]. The first calendar entry, `stat_rx_fc_stat[0]`, is received from bit[55] of a Control Word with the Reset Calendar bit, bit[56], set to a value of 1. The next calendar entry, `stat_rx_fc_stat[1]`, is received from bit[54] of the same Control Word and so on. The 17th calendar entry, `stat_rx_fc_stat[16]`, is received from bit[55] of the next Control Word that has the Reset Calendar bit, bit[56], set to a value of 0, and so forth.

Transmitter FIFO Threshold

The Interlaken 150G IP core has a FIFO in the TX path that holds enough data to properly format Interlaken bursts. The status of this FIFO is used to drive the `tx_rdyout` signal to inform the user logic whether the TX path can accept data. The threshold at which point `tx_rdyout` is asserted/deasserted is programmable through the `ctl_tx_rdyout_thresh[2:0]` input bus. This flexible mechanism allows you to optimize the logic and pipeline datapath that feeds the Interlaken 150G IP core transmitter.

To determine the proper value for `ctl_tx_rdyout_thresh[2:0]`, follow these steps:

1. Set `ctl_tx_rdyout_thresh[2:0]` to `3'h0`.
2. Simulate sending packets and observe the `tx_ovfout` signal.
3. If `tx_ovfout` is never asserted, the minimum value for `ctl_tx_rdyout_thresh[2:0]`is found.
4. If `tx_ovfout` is asserted, increase `ctl_tx_rdyout_thresh[2:0]` by one and repeat from step 2.

Link Level Flow Control

The Interlaken Protocol does not restrict the use of the calendar entries to a particular flow control implementation. As such, you can decide if one or more of the calendar entries should be designated to control the flow control of the entire interface (that is, link level flow control). For example, if the calendar length is 32, calendar slots 0 and 16 can be selected as link level flow control bits. This transmits the link level information with every burst control word.

The Interlaken 150G IP core implements the in-band flow control mechanism as described in the Interlaken *Protocol Definition Revision 1.2* [Ref 1]. The core is configured to have a fixed calendar length. You must connect the link level flow control information to the appropriate input pins of the RX interface and interpret the appropriate output pins of the TX interface.

Error Handling

The Interlaken 150G IP core performs robust checking of all possible error conditions as described in the *Interlaken Protocol Definition, Revision 1.2* [Ref 1], including the following errors:

- Loss of lane alignment
- CRC24 error
- BurstShort violation
- Illegal Control Word Type
- Illegal framing pattern is detected

If any of the preceding conditions are detected, the Interlaken 150G IP core takes the following actions:

- All open channels are marked as being in error and the packet in flight for these channels indicates the error condition by having `rx_errout` (`rx_errout<N>` in segmented LBUS) set to 1 when the corresponding `rx_eopout` (`rx_eopout<N>` in segmented LBUS) is asserted.
- All bits in the bus `stat_rx_fc_stat` become a value of 0 indicating an XOFF condition.

Additionally, in case of losing lane alignment, all data that is in the RX pipeline is lost.

Note: According to the Interlaken *Protocol Definition Revision 1.2*, [Ref 1] CRC32 errors do not affect open channels or flow control. Additionally, bits on the `stat_rx_mubits` bus are unaffected by the preceding errors and maintain their previous state.

The error handling circuits in the RX path perform EOP/SOP checks for packet or burst-interleaved modes of operation. The `ctl_rx_packet_mode` signal is used to select what type of error checking is to be performed.

The Interlaken 150G IP core is designed to handle packets that arrive as interleaved bursts. The core ensures that packets for each channel have appropriate SOP and EOP pairings. For applications that only send complete packets, an SOP must be paired with the next EOP. To ensure this kind of checking, packet mode, the `ctl_rx_packet_mode` should be assigned a value of 1. In packet mode, all missing SOP and EOP errors are treated as CRC24 errors. For burst-interleaved mode, `ctl_rx_packet_mode` should be assigned a value of 0.

Note: The `ctl_rx_packet_mode` input should be static and must only be changed during reset.

Design Flow Steps

This core is delivered asynchronously and is not part of the Vivado® Design Suite. It can be requested through sales and is generated by Xilinx. See Licensing and Ordering in Chapter 1.

Example Design

This chapter contains information about the example design provided with the Interlaken 150G IP core.

Quick Start Example Design

The Interlaken 150G IP core example design is shown in Figure 5-1.

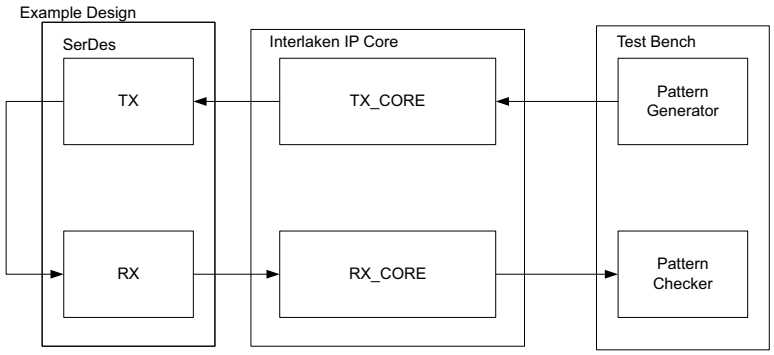

Figure 5-1: **Example Design**

The example design consists of the following:

- The Interlaken 150G IP core netlist
- A reference serial transceiver (GT) wrapper
- An example top-level illustrating the connection between the Interlaken 150G IP core and the GTs
- A demonstration test bench to exercise the example design
- An example simulation and compile scripts

The Interlaken 150G IP core example design is tested with Mentor Graphics Questa Advanced Simulator, Cadence Incisive Enterprise Simulator (NC-Verilog) and Synopsys VCS. For the supported version of the Xilinx tools, see the README file provided with the Interlaken 150G IP core release.

Implementation

The netlist and the example design HDL wrapper can be processed through the Xilinx implementation toolset. Included in the core drop are several scripts to assist in this processing.

To implement the example design:

Open a command prompt or shell in your project directory, then enter these commands for a Linux platform:

Linux

```
% cd {core_name}/compile/xilinx/interlaken_top/vivado/
% ./do_compile.sh
```

This starts a script that synthesizes the example design HDL wrapper, builds, maps, and places-and-routes the example design.

Simulation

The example design provided with the core provides a complete environment which allows you to simulate the core and view the outputs.

The Xilinx® UNISIM and SIMPRIM libraries must be mapped into the simulator. If the UNISIM and SIMPRIM libraries are not set up for your environment, see the *Xilinx Synthesis and Simulation Design Guide* (UG626) [Ref 3] for help on compiling Xilinx simulation models and setting up the simulator environment.

To simulate the example design:

1. Open a shell in your project directory, then set the current directory to: {*core_name*}/sim/
2. Launch the simulation script:

```
ModelSim: ./run_sim_msim.sh
NC-Sim: ./run_sim_ncv.sh
VCS: ./run_sim_vcs.sh
```

Demonstration Test Bench

Each Interlaken 150G IP core release includes a sample simulation test bench. This typically consists of a loopback from the TX side of the user interface, through the TX circuit, looping back to the RX circuit, and checking the received packets at the RX side of the user interface.

Directory and File Contents

Each configuration is referred to by the name in the brackets. The directory structure of the release for each configuration is the same and is described in the following subsections.

The files in this archive are listed in Figure 5-2.

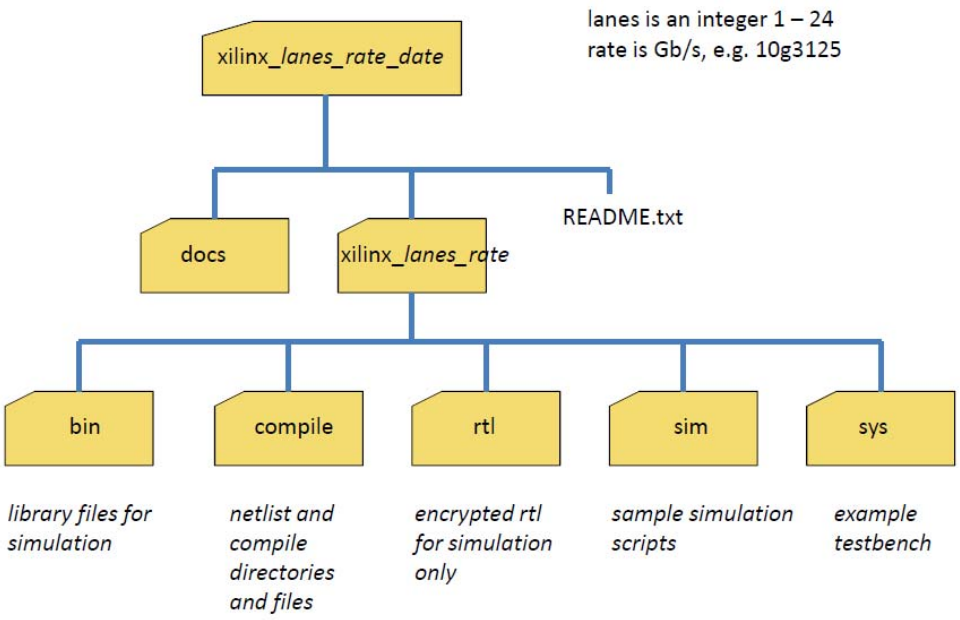

Figure 5-2: **Directory and File Contents**

<Project Directory>

The project directory is the top-level folder containing the Interlaken 150G IP core, product guide, and a README file.

Table 5-1: **Project Directory**

Name	Description
README.txt	Interlaken 150G IP core release notes

<Project Directory>/docs

The docs directory contains the PDF documentation provided with the core release.

Table 5-2: **docs Directory**

Name	Description
pg212-interlaken-150g.pdf	Interlaken 150G IP Product Guide

<Project Directory>/{name}

The {name} directory contains the core netlist, sample GT wrappers, example design, and the scripts provided for simulation and implementation organized in the various subdirectories.

{name}/bin

The bin directory contains the manifest files for running Verilog simulations. There are two manifest files for each of NC-Verilog, VCS, and ModelSim.

Table 5-3: **bin Directory**

Name	Description
loopback_simple_ncv.f	NC-Verilog commands file. This is called by the run_sim_ncv.sh script.
liblist_ncv.f	This file calls Xilinx Library files for the NC-Verilog simulator. If needed, the liblist file should be changed to match the location of the Xilinx library files.
loopback_simple_vcs.f	VCS commands file. This is called by the run_sim_vcs.sh script.
liblist_vcs.f	This file calls Xilinx Library files for the VCS simulator. If needed, the liblist file should be changed to match the location of the Xilinx library files.
loopback_simple_msim.f	ModelSim commands file. This is called by the run_sim_msim.sh script.

Table 5-3: **bin Directory *(Cont'd)***

Name	Description
liblist_msim.f	This file calls Xilinx Library files for the ModelSim simulator. If needed, the liblist file should be changed to match the location of the Xilinx library files.
loopback_simple.tcl	

{name}/compile

The compile directory contains the EDIF netlist along with sample scripts and constraints to implement the core.

Table 5-4: **compile Directory**

Name	Description
{name}/compile/xilinx/interlaken_top/edif/rev_1/{name}_interlaken_cores.edf	This is the EDIF netlist file that must be used during synthesis and implementation of the FPGA.
{name}/compile/xilinx/interlaken_top/vivado/*	This directory contains an XDC file with timing constraints for the core, a sample compile script that uses Vivado® design tools to compile the provided IP/RTL and sample log files from the compilation. ***Note:*** This directory is provided in the build when Vivado Design Suite is selected at the time of configuration.

{name}/rtl

This directory contains the simulation register transfer level (RTL) files for both Interlaken 150G IP, the Out-of-band flow control modules, and a sample instantiation of the transceiver.

Table 5-5: **rtl Directory**

Name	Description
{name}_interlaken_top.v	This is a REFERENCE top level file used to show the connectivity between the {name}_interlaken_cores module and the Xilinx transceiver.
{name}_interlaken_cores_int_bb.v	This is the clear text port list for the {name}_interlaken_cores_int module. This file can be used to instantiate {name}_interlaken_cores_int.
cores.vp	This is the encrypted simulation model for the IP.
{name}_interlaken_cores.v	This is the clear text port list for the {name}_interlaken_cores module. The port list also has detailed descriptions for each port that can be used as a quick reference guide. This module contains an instantiation of the {name}_interlaken_cores_int module.

Table 5-5: **rtl Directory *(Cont'd)***

Name	Description
{name}_rx_oobfc*.v	These are the RX out-of-band flow control modules. The RTL is clear-text.
{name}_tx_oobfc*.v	These are the TX out-of-band flow control modules. The RTL is clear text.
{name}_TRANSCEIVER_WRAPPER.v	This file comprises a reference instantiation of the Xilinx transceiver that can be used with the Interlaken 150G IP core. This is meant only as a reference to help with integration.
{name}_syncer*.v	These clear text files are used by the out-of-band flow control and interlaken_top modules.

{name}/sim

The sim directory contains the run scripts to perform simulations.

Table 5-6: **sim Directory**

Name	Description
loopback_simple.log	This is a sample simulation log provided as reference.
run_sim_ncv.sh	This is a run script to perform simulations with NC-Verilog.
run_sim_vcs.sh	This is a run script to perform simulations with VCS.
run_sim_msim.sh	This is a run script to perform simulations with ModelSim.

{name}/sys

The sys/tests directory contains a sample test bench and the sys/common directory contains the associated environment for running the sample test bench. The `run_sim` file automatically calls in the required modules.

Table 5-7: **sys Directory**

Name	Description
tests/loopback_simple/tb.v	Sample test bench
common/*.v	Files to set up the test environment

Test Bench

Each release includes a demonstration test bench that performs a loopback test on the complete core. For convenience, scripts are provided to launch the test bench from several popular simulators. The test program exercises the datapath to check that the transmitted frames are received correctly. RTL simulation models for the core are included. You must provide the correct path for the transceiver simulation model according to the latest simulation environment settings in your version of the Vivado® Design Suite.

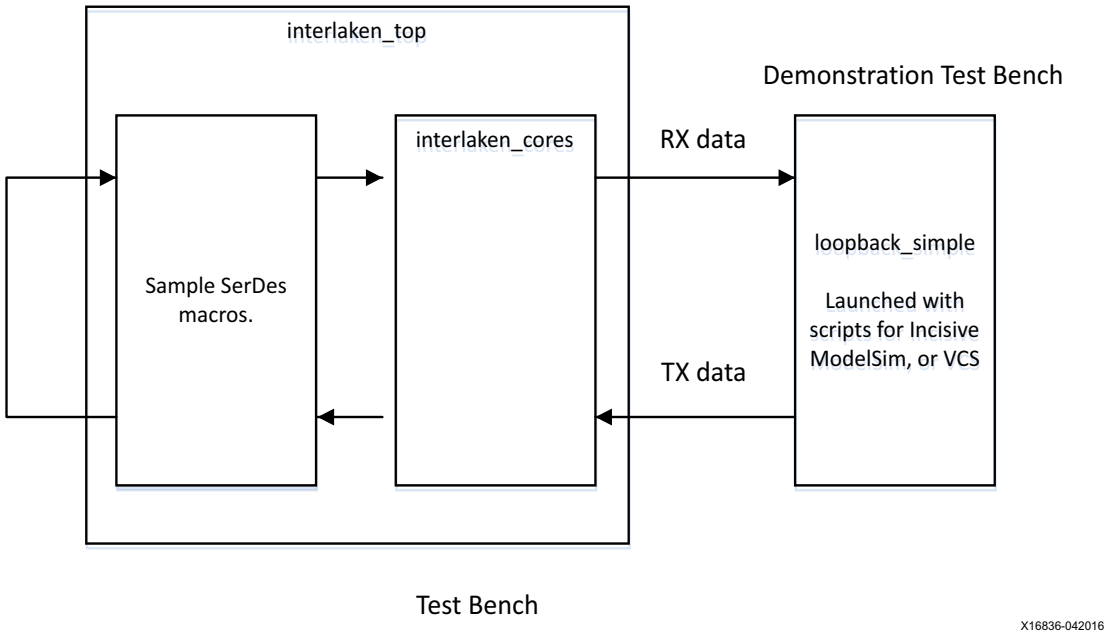

Figure 6-1: **Test Bench**

Verification, Compliance, and Interoperability

The Interlaken IP core has been designed to the requirements of the Interlaken Protocol Definition 1.2.

Simulation

This Interlaken IP Core has been thoroughly tested using a constrained random test environment that verifies the correct operation of each feature of the core.

Hardware Testing

This core is tested in Xilinx silicon for each major increment in functionality.

Upgrading

This section provides information about any changes to the user logic or port designations between core versions.

Changes from v1.5 to v1.6

- Added two ports to Table 2-2: rx_serdes_reset[LANES-1:0] and tx_serdes_refclk_reset.
- Changed names of the following ports:
  - tx_enainX to tx_enain<N>
  - tx_sopinX to tx_sopin<N>
  - tx_chaninX to tx_chanin<N>
  - tx_bctlinX to tx_bctlin<N>

Interlaken Look-Aside Port List

If the Interlaken 150G IP core is configured to operate according to the Interlaken Look-Aside (LA) specifications, the functionality of some of the ports is changed from what is described in Table 2-2. These changes are shown in Interlaken LA Ports (Table C-1). See the *Interlaken Look-Aside Protocol Definition Revision 1.1* [Ref 5] for more details on the Interlaken LA protocol.

Note: The Look-Aside version of Interlaken allows for small packets, which are typically the headers of larger packets. Therefore, up to four bursts are allowed in one cycle for Look-Aside. This is different from the Interlaken IP, which can only accept 1 burst per cycle on the TX.

Table C-1: **Interlaken LA Ports**

Name	Direction	Domain	Description
rx_chanout[29:0]	Output	clk	Interlaken LA only uses two channels; rx_chanout[0] indicates to which channel the incoming packet belongs. Additionally, Interlaken LA defines three Application Specific fields as part of the Interlaken Control Words. These three fields are mapped to the rx_chanout bus as follows: • Application Specific 2: rx_chanout[8:1] • Application Specific 1: rx_chanout[14:9] • Application Specific 0: rx_chanout[29:15] Application Specific 2 is defined as bits [31:24] of the Interlaken Control Word. Application Specific 1 is defined as bits [38:33] of the Interlaken Control Word. Application Specific 0 is defined as bits [56:42] of the Interlaken Control Word.

Table C-1: **Interlaken LA Ports *(Cont'd)***

tx_chanin[29:0]	Input	clk	Interlaken LA only uses two channels. tx_chanout[0] is used to indicate the channel of the outgoing packet. Additionally, Interlaken LA defines three Application Specific fields as part of the Interlaken control words. These three fields are mapped to the tx_chanout bus as follows: • Application Specific 2: tx_chanout[8:1] • Application Specific 1: tx_chanout[14:9] • Application Specific 0: tx_chanout[29:15] Application Specific 2 is defined as bits [31:24] of the Interlaken Control Word. Application Specific 1 is defined as bits [38:33] of the Interlaken Control Word. Application Specific 0 is defined as bits [56:42] of the Interlaken Control Word.
ctl_tx_fc_stat[1:0]	Input	clk	TX In-Band Flow Control Input. These signals are used to set the status for each calendar position in the in-band-flow control mechanism. A value of 1 means XON, a value of 0 means XOFF. ctl_tx_fc_stat[0] corresponds to channel 0 (Control Word bit [40]), and tx_fc_stat[1] corresponds to channel 1 (Control Word bit [41]).
ctl_tx_burstshort[2:0]	Input	static	Interlaken TX BurstShort. This bus sets the BurstShort parameter for the TX for configuration as follows: 0x0 = 8 bytes 0x1 = 16 bytes All other values are reserved and must not be used.
stat_rx_fc_stat[1:0]	Output	clk	RX Flow control Outputs. These signals indicate the flow control status for the two channels. A value of 1 means XON, a value of 0 means XOFF. stat_rx_fc_stat[0] corresponds to channel 0 (Control Word bit [40]), and stat_rx_fc_stat[1] corresponds to channel 1 (Control Word bit [41]).

Notes:

1. The signals associated with mubits are not present in Look-Aside mode.

Appendix D

Out-Of-Band Flow Control

Introduction

Interlaken is a scalable chip-to-chip interconnect protocol designed to support a wide range of transmission rates. Using the latest SerDes technology and a flexible protocol layer, Interlaken minimizes the pin and power overhead of chip-to-chip interconnect and provides a scalable solution that can be used throughout an entire system. In addition, Interlaken uses two levels of CRC checking and a self-synchronizing data scrambler to ensure data integrity and link robustness.

Interlaken defines two mechanisms for handling flow control across the interface:

- In-band flow control
- Out-of-band flow control

The choice of using in-band or out-of-band mechanisms depends on system considerations that are beyond the scope of this document. For further information or feedback, contact Xilinx.

This appendix describes the out-of-band flow control (OOBFC) implemented along with the Interlaken 150G IP core.

Overview

Interlaken employs a simple and flexible XON/XOFF mechanism to communicate flow control information across the interface. The XON/XOFF information is transmitted for all supported logical channels and provides you with a flexible means for implementing any scheduling algorithm.

Xilinx provides two independent modules for handling out-of-band flow control:

- TX_OOBFC
- RX_OOBFC

The module TX_OOBFC transmits out-of-band flow control information and the RX_OOBFC module receives out-of-band flow control information.

Logically, these modules should be instantiated at the same level of hierarchy as the core and operate on their own to transmit and receive flow control information from the corresponding interface.

Figure D-1 shows a typical instantiation of the Interlaken core and the OOBFC modules. You handle packet scheduling and respond to flow control information; it is represented by the User Logic block in this diagram.

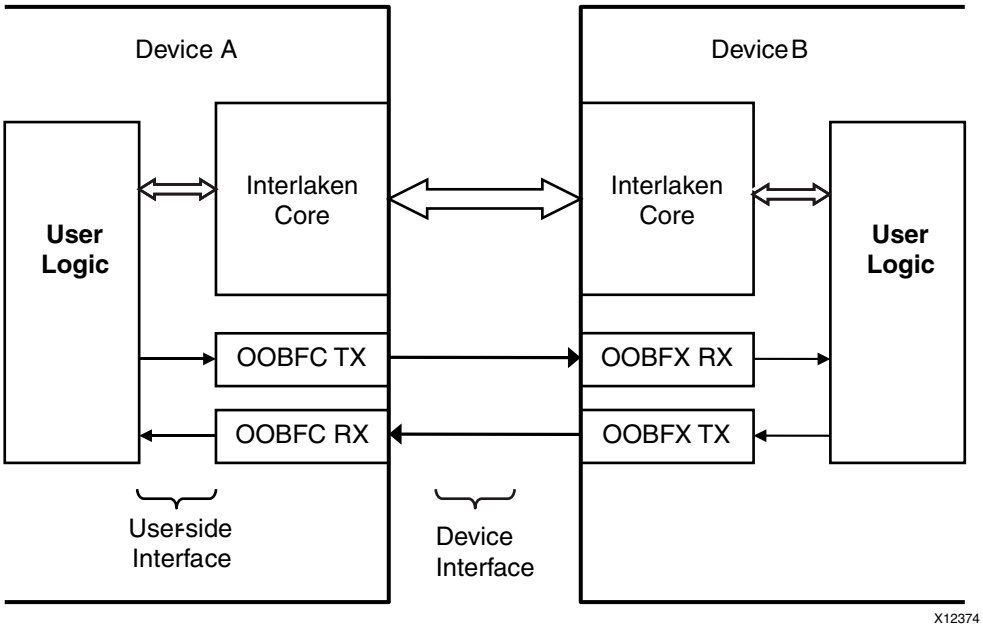

Figure D-1: **Typical Instantiation of the IFC and OOBFC Modules**

Port List

TX_OOBFC Pin List (Table D-1) and RX_OOBFC Pin List (Table D-2) show the port list of the TX_OOBFC and RX_OOBFC modules along with a description of each pin respectively. The LANES parameter is equal to the number of SerDes lanes. The LANE_WIDTH parameter represents the number of bits required to express the number of SerDes lanes in binary. The MAX_CALLEN parameter is set to the maximum number of calendar entries supported by the core and is a power of two from 32 to 2048. The UPDATE_WIDTH parameter is equal to max {1,MAX_CALLEN/64}.

Table D-1: **TX_OOBFC Pin List**

Name	Direction	Domain	Description
Clocking and Resets			
clk	Input		All signals between the TX_OOBFC and the user logic are synchronized to the positive edge of clk. In typical applications, this clock should be tied to the same clock that is used to run the user-side interface of the core.
clk_tx_ref	Input		This clock is used to generate the flow control signals. You are required to supply a clock with a maximum frequency of 200 MHz. The 200 MHz limitation (which is twice the OOBFC clock) is defined by the *Interlaken Protocol Definition, Revision 1.2* [Ref 1]
reset	Input	async	Active-High, asynchronous reset input. This signal is automatically synchronized to the appropriate clock domain by the TX_OOBFC. All circuits in the module are reset while this input has a value of 1. This signal must remain asserted until after several clock cycles on both the clk and clk_tx_ref inputs.
User-Side Interface – TX_OOBFC Signals			
tx_fc[MAX_CALLEN-1:0]	Input	clk	Input data bus containing the flow control information to be transmitted by the TX_OOBFC. The width of the bus is set by the MAX_CALLEN parameter. Unused bits, as defined by tx_callen_minus1[7:0], must be tied to 0. Interlaken defines a value of 1 as XON for the corresponding channel, and a value of 0 as XOFF for the corresponding channel.
tx_callen_minus1[10|9|8|7:0]	Input	clk	This input sets the calendar length for flow control information for the TX_OOBFC. Allowed values are 0 to MAX_CALLEN-1. For example, when MAX_CALLEN is set to 32, tx_callen_minus1[7:0] can be set to any value between 0 and 31. It is up to you to ensure that this input is set correctly. Incorrect setting results in undetermined behavior. This input should only be changed when reset is asserted (that is, set to 1). Changing the value on this input when not in reset can result in incorrectly transmitted flow-control or status information.
tx_lanes_minus1[LANE_WIDTH-1:0]	Input	clk	This input sets the number of lanes to be included in the Status Message. Allowed values are 0 to LANES-1. This value should be set to the number of lanes (minus 1) expected by the receiver. An incorrect setting can result in undetermined behavior. This input should only be changed when reset is asserted (that is, set to 1).

Table D-1: **TX_OOBFC Pin List *(Cont'd)***

Name	Direction	Domain	Description
tx_intf_status	Input	clk	Indicates the health of the interface to be transmitted to the other device as described in the Interlaken specification. A value of 1 indicates the interface is healthy. If unused, this input should be a set to a value of 1.
tx_lane_status[LANES-1:0]	Input	clk	Indicates the health of each lane to be transmitted to the other device as described in the Interlaken specification. A value of 1 indicates the corresponding lane is healthy. Bit 0 corresponds to lane 0, bit 1 corresponds to lane 1, and so on. If unused, all inputs should be a set to a value of 1.
tx_update[UPDATE_WIDTH-1:0]	Output	clk	This output bus indicates when the status and flow control information are latched for transmission. If tx_update[0] is asserted, tx_fc[63:0], tx_lane_status, and tx_intf_status bits will be latched at the completion of the current clock cycle; if tx_update[1] is asserted, tx_fc[127:64] will be latched at the end of the current clock cycle; if tx_update[2] is asserted, tx_fc[191:128] will be latched at the end of the current clock cycle and so on.
tx_status_enable	Input	clk	This input can be used to enable and disable the transmission of the interface and lane status. When this input is a value of 1, tx_intf_status and tx_lane_status operate according to the descriptions. When this input is a value of 0, the values on tx_intf_status and tx_lane_status are ignored and the interface and lane status are not transmitted. If the transmission of the interface and lane status are not required, this value should be tied to a value of 0.
tx_err	Output	clk	If the clock rates are incorrect for the selected calendar length, the signal tx_err can be asserted to notify you of this configuration error.
Device Interface – TX_OOBFC Signals(1)			
TX_FC_CLK	Output (LVCMOS)		This is the source synchronous clock generated by TX_OOBFC as defined by the Interlaken protocol. The frequency of this clock is one-half the frequency of clk_tx_ref. For example, if clk_tx_ref is 200 MHz, this clock is 100 MHz. This signal must be connected directly to a device output pin.
TX_FC_DATA	Output (LVCMOS)		The flow control information is transmitted by the TX_OOBFC with this signal as defined by the Interlaken protocol. This signal must be connected directly to a device output pin.

Table D-1: **TX_OOBFC Pin List *(Cont'd)***

Name	Direction	Domain	Description
TX_FC_SYNC	Output (LVCMOS)		This signal is used to synchronize the transmitted flow control information as defined by the Interlaken protocol. This signal must be connected directly to a device output pin.

Notes:
1. further details, the electrical and timing specifications of these signals, see the *Interlaken Protocol Definition Revision 1.2*.

Table D-2: **RX_OOBFC Pin List**

Name	Direction	Domain	Description
Clocking and Resets			
clk	Input		All signals between the RX_OOBFC and the user logic are synchronized to the positive edge of clk. In typical applications, this clock should be tied to the same clock used to run the user-side interface of the core. In order for correct operation in the RX_OOBFC, the frequency of this clock must be at least 33% faster than the frequency of RX_FC_CLK. For example, if the frequency of RX_FC_CLK is 100 MHz, the frequency of clk must be at least 133 MHz.
reset	Input	async	Active-High, asynchronous reset input. This signal is automatically synchronized to the appropriate clock domain by the RX_OOBFC. All circuits in the module are reset while this input is a value of 1. This signal must remain asserted until after several clock cycles on both the clk and RX_FC_CLK inputs.
reset _RX_FC_CLK	Input		Active-High, synchronous reset input. This signal only appears on the port list if the RX_OOBFC is implemented with flip-flops with synchronous reset inputs. This signal must remain asserted until after several clock cycles the RX_FC_CLK input.

Table D-2: **RX_OOBFC Pin List *(Cont'd)***

Name	Direction	Domain	Description
User-Side Interface – RX_OOBFC Signals			
rx_fc[MAX_CALLEN-1:0]	Output	clk	Output data bus to the user logic containing the flow control information received by the RX_OOBFC. The width of the bus is set by the MAX_CALLEN parameter. Bit 0 corresponds to channel 0, bit 1 corresponds to channel 1, and so on. Interlaken defines a value of 1 as XON and a value of 0 as XOFF. If the calendar length for the received flow control information has less than MAX_CALLEN entries, the unused bits are undefined and should be ignored. It is up to you to monitor this bus and take appropriate action as flow control information is changed. When an unhealthy interface status is received, as indicated by rx_intf_status being a value of 0, or an unhealthy lane status is received, as indicated by a bit of rx_lane_status being a value of 0, then all bits of rx_fc are XOFF or a value of 0. When a CRC error is detected, the outputs rx_fc are unchanged.
rx_crcerr	Output	clk	Indicates if an error was observed in the CRC field of the incoming status or flow control information. A value of 1 indicates a CRC error was detected. When this bit is a value of 1, the outputs rx_fc, rx_intf_status, and rx_lane_status are not updated and can be ignored. The cause of a CRC error can be due to clk not being sufficiently faster than RX_FC_CLK as required for correct operation.
rx_overflow	Output	clk	Indicates that clk is not faster than RX_FC_CLK as required for correct operation.
rx_intf_status	Output	clk	Indicates the health of the receive interface as described in the Interlaken specification. When this output has a value of 0, all flow-control bits in the rx_fc bus will be XOFF (that is, 0).
rx_lane_status[LANES-1:0]	Output	clk	Indicates the health of the corresponding receive lanes as described in the Interlaken specification. Bit 0 corresponds to lane 0, bit 1 corresponds to lane 1, and so on. When any bit of this bus has a value of 0, all flow-control bits in the rx_fc bus will be XOFF (that is, 0).
rx_update[UPDATE_WIDTH-1:0]	Output	clk	Indicates a valid CRC4 was detected and rx_fc bits were updated. If rx_update[0] is asserted, the CRC4 associated with rx_fc[63:0] was valid and rx_fc[63:0]were updated; if rx_update[1] is asserted, the CRC4 associated with rx_fc[127:64] was valid and rx_fc[127:64]were updated, and so on.
rx_callen_minus1[10|9|8|7:0]	Output	clk	Indicates the length of the most recently received calendar.

Table D-2: **RX_OOBFC Pin List *(Cont'd)***

Name	Direction	Domain	Description
rx_force_xoff_if_crcerr	Input	clk	When this input is a value of 1, all bits of rx_fc are forced to XOFF if an CRC error is detected.
Device Interface RX_OOBFC Signals(1)			
RX_FC_CLK	Input (LVCMOS)		This the source-synchronous clock used by RX_OOBFC as defined by the Interlaken protocol. This signal must be connected directly to a device input pin.
RX_FC_DATA	Input (LVCMOS)		The flow control information is received by the RX_OOBFC with this signal as defined by the Interlaken protocol. This signal must be connected directly to a device input pin.
RX_FC_SYNC	Input (LVCMOS)		This signal is used to synchronize the received flow control information as defined by the Interlaken protocol. This signal must be connected directly to a device input pin.

Notes:

1. Further details of the electrical and timing specifications for these signals are found in the *Interlaken Protocol Definition, Revision 1.2*.

General Operation

The OOBFC implements the protocol as described in the *Interlaken Protocol Definition, Revision 1.2* [Ref 1]. The following sections describe the operation in more detail and are intended to augment the Interlaken specification.

TX_OOBFC

The TX portion of the OOBFC starts transmitting information as soon as the `reset` input is deasserted. The `clk_tx_ref` input must be stable and running correctly before the `reset` input is deasserted.

The health status inputs (`tx_intf_status` and `tx_lane_status`) are transmitted only when an unhealthy condition exists. An unhealthy condition occurs when `tx_intf_status` is 0, or any bit of `tx_lane_status` is 0. The transmission of the status inputs is alternated with the transmission of the flow control information when an unhealthy status exists.

If the health status inputs (`tx_intf_status` and `tx_lane_status`) are all 1, meaning the interface is healthy, only flow control information is transmitted.

The length of the flow control calendar is programmed through `tx_callen_minus1`. The calendar is transmitted with the value of `tx_fc[0]` sent first, `tx_fc[1]` sent second, `tx_fc[2]` sent third and so on.

The `tx_fc` input is latched for transmission in 64-bit groups. The input `tx_fc[63:0]`, `tx_intf_status`, and `tx_lane_status` are latched at the completion of the clock cycle when `tx_update[0]` is asserted; the input `tx_fc[127:64]` is latched at the completion of the clock cycle when `tx_update[1]` is asserted; the input `tx_fc[191:128]` is latched at the completion of the clock when `tx_update[2]` is asserted and so on.

RX_OOBFC

In order for correct operation of the RX_OOBFC to occur, two things are essential:

- The user-side clock, `clk`, must be faster than the receiving clock, `RX_FC_CLK`, by at least 33%. For example, if the frequency of `RX_FC_CLK` is 100 MHz, then the frequency of `clk` must be at least 133 MHz.
- The reset input, `reset`, must be asserted until after several cycles of both input clocks, `clk` and `RX_FC_CLK`.

Reception of flow control information starts as soon as the `reset` input is deasserted. However, the RX_OOBFC does not consider itself initialized and does not supply the received information until it has seen several correct transitions on the `RX_FC_SYNC` input. During this initialization procedure, the `rx_fc` output bus is set to XOFF (that is, 0).

After the initialization procedure is completed, the RX_OOBFC supplies status and flow-control information as received through the interface.

The Interlaken protocol does not require the transmission of status information if the interface is healthy. As a result, RX_OOBFC assumes the interface is healthy if it receives two back-to-back valid calendars of flow control information. In healthy conditions, the `rx_intf_status` and `rx_lane_status` outputs are all set to 1.

Additionally, in healthy conditions, the received flow control information is presented on the `rx_fc` output bus. The first calendar value received is presented on `rx_fc[0]`, the second on `rx_fc[1]`, the third on `rx_fc[2]`, and so on. If the received calendar has fewer entries than the total width of `rx_fc` bus, the "unreceived" bits in `rx_fc` will be undetermined and must be ignored by the user logic.

Whenever unhealthy status information is received, all the bits of `rx_fc` are forced to XOFF (that is, 0). When healthy status information is determined, `rx_fc` is updated to reflect the most recently received values of flow control information.

If a CRC4 error is ever detected during the receipt of a calendar of flow control information or during the receipt of status information, the `rx_crcerr` output is asserted (set to 1). Whenever `rx_crcerr` has a value of 1, the `rx_fc`, `rx_intf_status`, and `rx_lane_status` outputs are not updated and should be considered invalid (the Interlaken protocol requires you to take appropriate action when `rx_crcerr` is asserted). The `rx_fc`, `rx_intf_status`, and `rx_lane_status` outputs should be considered valid only when `rx_crcerr` is negated (that is, 0).

The `rx_overflow` output is an aid to help identify incorrect clock rates. It is asserted (set to 1) when the `clk` input is not fast enough relative to `RX_FC_CLK`.

The RX_OOBFC accepts any calendar length up to MAX_CALLEN. The most recently received calendar length is reported on the output `rx_callen_minus1`.

The `rx_fc` output is updated in groups of 64 bits. When `rx_fc[63:0]` is updated, `rx_update[0]` is asserted. When `rx_fc[127:64]` is updated, `rx_update[1]` is asserted, and so forth.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. Also see the Interlaken Core home page for product information and resources.

Documentation

This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx® Documentation Navigator. Download the Xilinx Documentation Navigator on the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Technical Support

Xilinx provides technical support at Xilinx support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx support web page.

Debug Tools

There are many tools available to address Interlaken IP core design issues. It is important to know which tools are useful for debugging various situations

Example Design

See Chapter 5, Example Design.

The Interlaken 150G IP core is delivered with an example design netlist complete with functional test benches. The design includes example transceivers and loopback tests for common simulator programs.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 6]

Reference Boards

Various Xilinx development boards support the Interlaken IP core. These boards can be used to prototype designs and establish that the core can communicate with the system.

7 series FPGAs are recommended for optimum performance. Ensure that the board transceivers support the required Interlaken bit rate. For example, the VC7203 FPGA evaluation board is suitable for many Virtex®-7 FPGA7 series implementations.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado design tools has several license check points for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error.

License checkpoints are enforced by the following tools in the Vivado flow: Vivado synthesis, Vivado implementation, `write_bitstream` (Tcl Command)

IMPORTANT: *The IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Simulation Debug

Each Interlaken 150G IP core release includes a sample simulation test bench. This typically consists of a loopback from the TX side of the user interface, through the TX circuit, looping back to the RX circuit, and checking the received packets at the RX side of the user interface. Interlaken Look-Aside is available as an asymmetric configuration and therefore a loopback test is not appropriate. You might find that an Interlaken Look-Aside IP core includes sample unidirectional tests.

Each release usually includes a sample instantiation of a Xilinx transceiver corresponding to the device selected by the customer. The loopback simulation includes a path through the transceiver. The simulation is run using provided scripts for several common industry-standard simulators. If the simulation does not run properly from the scripts, the following items should be checked.

Simulator License Availability

If the simulator does not launch, you might not have a valid license. Ensure that the license is up-to-date. It is also possible that your organization has a license available for one of the other simulators, so try all the provided scripts.

Library File Location

Each simulation script calls up the required Xilinx library files. These are called by the corresponding liblist* file in the bin directory of each release.

There might be an error message indicating that the simulator is unable to find certain library files. In this case, the path to the library files might have to be modified. Check with your IT administrator to ensure that the paths are correct.

Version Compatibility

Each release has been tested according the Xilinx tools version requested by you, the customer. If the simulation does not complete successfully, first ensure that a properly up-to-date version of the Xilinx tools is used. The preferred version is indicated in the README file of the release, and is also indicated in the simulation sample log file included with the release.

Slow Simulation

Simulations can appear to run slowly under some circumstances. If a simulation is unacceptably slow (greater than one hour for the provided sample simulation to reach successful conclusion), the following suggestions might improve the run-time performance.

- Use a faster computer with more memory.
- Make use of a Platform Load Sharing Facility (LSF) if available in your organization.
- Bypass the Xilinx transceiver (this might require that you create your own test bench).
- Send fewer packets. This can be accomplished by modifying the appropriate parameter in the provided sample testable.
- Specify a shorter metaframe length. This should result in a shorter lane alignment phase, at the expense of more overhead. However, when the Interlaken IP core is finally implemented in hardware, the metaframe length should follow the specification recommendations. Contact Xilinx technical support for assistance if required.

Simulation Fails Before Completion

If the sample simulation fails or hangs before successfully completing, it is possible that a timeout has occurred. Ensure that the simulator timeouts are long enough to accommodate the waiting periods in the simulation, for example during the lane alignment phase.

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado Design Suite debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the debug feature for debugging the specific problems. Many of these common issues can also be applied to debugging design simulations.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean
- If using MMCMs in the design, ensure that all MMCMs have obtained a lock by monitoring the `LOCKED` port
- If your outputs go to 0, check your licensing.

Interlaken Specific Checks

Several issues can commonly occur during the first hardware test of an Interlaken IP core. These should be checked as indicated in the following steps.

It is assumed that the Interlaken 150G IP core has already passed all simulation testing which is being implemented in hardware. This is a pre-requisite for any kind of hardware debug

The usual sequence of debugging is to proceed in the following order:

1. Clean up signal integrity.
2. Ensure that each SerDes achieves clock data recovery (CDR) lock.
3. Check that each lane has achieved word alignment.
4. Check that lane alignment has been achieved.
5. Proceed to Interface and Protocol debug.

Signal Integrity

When bringing up a board for the first time and the Interlaken IP core does not seem to be achieving lane alignment, the most likely problem is related to signal integrity. Signal integrity issues must be addressed before any other debugging can take place.

Even if lane alignment is achieved, if there are periodic CRC32 errors, then signal integrity issues are indicated. Check the `stat_rx_crc32_err` signals to assist with debug.

Signal integrity should be debugged independently from the Interlaken IP core. The following procedures should be carried out.

Note: It is assumed that the PCB itself has been designed and manufactured in accordance with the required trace impedances and trace lengths.

- Transceiver Settings
- Checking For Noise
- Bit Error Rate Testing

If assistance is required for transceiver and signal integrity debugging, contact Xilinx technical support.

Lane Swapping

Unlike Ethernet, Interlaken requires that TX and RX lanes maintain the correct ordering. Note that lane alignment can still be achieved even if lanes are swapped. The reason is that the alignment marker is the same for each lane. Furthermore, even with lanes swapped, the CRC32 might be correct provided that signal integrity is good.

Lane swapping is often indicated by a CRC24 error while data is being sent.

N/P Swapping

If the positive and negative signals of a differential pair are swapped, no data will be correctly received on that lane. Verify that each link has the correct polarity of each differential pair.

Clocking and Resets

See the clocking and reset sections (Clocking and Resets in Chapter 3) of this product guide for these requirements.

Ensure that the clock frequencies for both the Interlaken 150G IP core as well as the Xilinx transceiver reference clock match the configuration requested when the IP core was ordered. The core clock has a minimum frequency associated with it. The maximum core clock frequency is determined by timing constraints.

The minimum core clock frequency is derived from the required Interlaken bandwidth plus some margin reserved for clock tolerance, wander, and jitter.

The first thing to verify during debugging is to ensure that resets remain asserted until the clock is stable. It must be frequency-stable as well as free from glitches before the Interlaken 150G IP core is taken out of reset. This applies to both the SerDes clock as well as the IP core clock.

If any subsequent instability is detected in a clock, the Interlaken 150G IP core must be reset. One example of such instability is a loss of CDR lock. The user logic should determine all external conditions that would require a reset (for example, clock glitches, loss of CDR lock, power supply glitches, etc.).

Configuration changes cannot be made unless the IP core is reset. An example of a configuration change is a change in BurstMax. Check the description for the particular signal on the port list to determine if this requirement applies to the parameter that is being changed.

Interface Debug

LBUS Interface

The Interlaken 150G IP core user interface is called the LBUS. There are two versions used — regular LBUS and segmented LBUS. See User Side Interface in Chapter 3 for a detailed description.

TX Debug

TX debug is assisted by using several diagnostic signals

Buffer Errors

Data must be written to the TX LBUS such that there are no overflow or underflow conditions. LBUS bandwidth must always be greater than the Interlaken bandwidth to guarantee that bursts can be sent without interruption

When writing data to the LBUS, the `tx_rdyout` signal must always be observed. This signal indicates whether the fill level of the TX buffer is within an acceptable range or not. If this signal is ever asserted, you must stop writing to the TX LBUS until the signal is deasserted. Because the TX LBUS has greater bandwidth than the TX Interlaken interface, it is not unusual to see this signal being frequently asserted and this is not a cause for concern. You must ensure that TX writes are stopped when `tx_rdyout` is asserted.

The level at which `tx_rdyout` becomes asserted is determined by the bus `ctl_tx_rdyout_thresh`. It might be necessary to adjust this bus as required to ensure that your transmitting protocol is able to handle the `tx_rdyout` requirement.

RECOMMENDED: *If `tx_rdyout` is ignored, the signal `tx_ovfout` might be asserted, indicating a buffer overflow. This must not be allowed to occur. Xilinx recommends that the Interlaken IP core be reset if `tx_ovfout` is ever asserted. Do not attempt to continue debugging after `tx_ovfout` has been asserted until the cause of the overflow has been addressed.*

If `stat_tx_underflow_err` is ever asserted, debugging must stop until the condition that caused the underflow is addressed. This can happen if the core clock is not fast enough to supply the transceiver with data, and you should ensure that the minimum core clock frequency is being observed.

Burst Errors

The TX must observe the LBUS rules that are required for compliance to the *Interlaken Protocol Definition, Revision 1.2* [Ref 1] as well as Xilinx implementation-specific requirements.

If a burst rule violation has occurred, the signal `stat_tx_burst_err` is asserted. You must ensure that the rules governing the scheduling of bursts are observed according to the LBUS width in the particular implementation. In particular, the segmented LBUS protocol should be given careful reading.

One common problem is that you have written data in such a way that more than one Burst Control Word generated during one clock cycle. Carefully review the segmented LBUS rules. Take care that an SOP does not occur in the same cycle as an implied Burst Control Word resulting from BurstMax having been reached.

This can be avoided by implementing the Enhanced Scheduling Algorithm, which makes use of the signal `tx_bctlin` to force Burst Control Words in such a way as to avoid in one clock cycle.

In summary, Burst Control Words can be implied or generated by the following cases:

- BurstMax reached (implied)
- SOP (implied)
- Channel Change (implied)
- bctlin assertion (forced)

It is up to you to ensure that these cases cannot occur during the same clock cycle.

RECOMMENDED: *Xilinx recommends (and required for segmented LBUS) implementing Enhanced Scheduling as described in this product guide and Interlaken Protocol Definition, Revision 1.2 [Ref 1], as a means to avoid some of the implied Burst Control Words (BCWs) so that the two cannot occur during the same cycle.*

RX Debug

The RX User Side interface indicates an error condition using the `rx_errout` signal.

If `rx_errout` is asserted, it usually indicates that the packet being received contains an error. One possible error condition that must be examined carefully is a missing SOP or EOP. In this event, the Interlaken 150G IP core attempts to recover the data by merging packets and closing them, presenting the data on the RX LBUS with an `rx_errout` indication. You must ensure that the transmitting device is sending proper SOP and EOP indications.

Ensure that the `stat_rx_overflow_err` signal is not asserted. If it is asserted, the RX LBUS is not being clocked fast enough to empty the RX buffer. The LBUS must have more bandwidth than the Interlaken interface.

Protocol Debug

To achieve error-free data transfers with the Interlaken 150G IP core, the protocol parameters need to be set correctly. This section details some common protocol problems which can occur. It assumed that the number of lanes and the bit rates are matched. It is also assumed that the signal integrity and lane ordering has been verified.

Configuration Match

In order for the Interlaken protocol to function correctly, both devices must have matching Interlaken protocol parameters. Ensure that the following parameters are the same for each end of the link:

- Meta Frame Length
- BurstMax
- BurstShort

Diagnostic Signals

There are many error indicators available to check for protocol violations. Carefully read the description of each one to see if it is useful for a particular debugging issue.

The following is a suggested debug sequence.

1. Ensure that Word sync has been achieved.
2. Ensure that lane alignment has been achieved.
3. Verify that the Metaframe indicators are clean.
4. Make sure there are no descrambler state errors.
5. Eliminate CRC24 errors, if any.
6. Make sure there are no burst errors (BurstMax, BurstShort mismatch).
7. Look for SOP and EOP errors and eliminate those, if any occur.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx® Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this product guide:

1. *Interlaken Protocol Definition, Revision* 1.2
2. *UltraScale Architecture GTH Transceivers Advance Specification User Guide* (UG576)
3. *Xilinx Synthesis and Simulation Design Guide* (UG626)
4. *ISE to Vivado Design Suite Migration Guide* (UG911)
5. *Interlaken Look-Aside Protocol Definition Revision* 1.1
6. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
7. *LogiCORE IP 7 Series FPGAs Transceivers Wizard User Guide* (UG769)
8. *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)
9. *UltraScale Architecture GTY Transceivers User Guide* (UG578)
10. *Integrated Interlaken up to 150G LogiCORE IP Product Guide* (PG169)
11. *Interlaken 600G LogiCORE IP Product Guide* (PG209) Registration required

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/04/2017	1.6	• Updated transmission speeds in the Introduction in the IP Facts page. • Updated the Standards section in Chapter 2, Product Specification. • Added Unsupported Features section in Chapter 1, Overview.
06/07/2017	1.6	• Included IDLE cycles as part of valid packet transfers in Figures 3-7 and 3-8. • Added Documentation Navigator and Design Hubs section.
11/30/2016	1.6	• Updated the list of devices to the first paragraph in the Overview and Product Specification chapters. • Added a description of "minimal effort" to the first paragraph in Product Specification chapter. • Added two ports and modified the names of several ports. See the Migrating and Updating appendix.
10/05/2016	1.6	• Added Updated BurstShort error description in the stat_rx_burst_err subsectio.n. • Added Important note about static signal at the beginning of the Port Descriptions section. • Updated Figure 2-2. • Changed "path" to "lane logic" for the port description of ctl_rx_force_resync.
06/08/2016	1.5	• Added support for Virtex UltraScale+, Kintex UltraScale+, and Zynq UltraScale+ devices. • Updated the descriptions of the following ports: drpclk_p, drpclk_n, tx_serdes_refclk, tx_reset, rx_chanout[7:0], tx_chanin[7:0], tx_bctlin, ctl_tx_fc_callen[3:0], stat_tx_overflow_err, and stat_rx_msop_err • Updated the descriptions of the following sections: Segment Ordering, Data Formatting, Interlaken Segmented LBUS Specification, TX LBUS Interface, TX Transactions. and Examples of Transfers with Errors. • Changed tx_serdes_clk and tx_serdes_refclk Domains section name to tx_serdes_clk Domain and updated the description. • Removed tx_serdes_clk from many of the descriptions.

Date	Version	Revision
06/08/2016 (continued)	1.5	• Changed "tx_enain" to "tx_enain0", "tx_eopin" to "tx_eopin0", "tx_datain" to "tx_datain0", "tx_mtyin[2:0]" to "tx_mtyin0[2:0]", "tx_errin" to "tx_errin0", "tx_bctlin" to "tx_bctlin0", "rx_enaout" to "rx_enaout0", "tx_bctlin" to "tx_bctlin0", "rx_eopout" to "rx_eopout0", "rx_dataout" to "rx_dataout0", "rx_errout" to "rx_errout0", "tx_ctl_burstshort" to "ctl_tx_burstshort", "rx_force_resync" to "ctl_rx_force_resync", "tx_rdyout_thresh" to "ctl_tx_rdyout_thresh". • Added stat_rx_err section. • Changed "Segment Mode" to "Burst-Interleaved Mode" throughout. • Changed many of the buses to 7 from 10. • Updated Figure 3-5, Figure 3-6, Figure 3-7, Figure 3-8, Figure 3-10.
07/24/2015	1.4	Initial Xilinx release of this product guide. The guide is based on UG785.

Please Read: Important Legal Notices