

Introduction

The Xilinx LogiCORE™ Packet Queue is ideal for systems that require buffering packet-based data from multiple input streams with aggregation into a single output interface. Packet Queue implements a fully shared memory architecture, enabling a large storage capacity for the efficient use of FPGA block memory resources. External scheduling interfaces are provided for specifying scheduling and channel priorities. It also supports packet-level flow control, synchronous or asynchronous clocks, variable length packets, the ability to discard or retransmit packets, and easy integration into the Xilinx design flow using LocalLink interfaces. Configuration options include 1–32 input channels and configurable data depth and width (including optional bus-width conversion). The Xilinx CORE Generator™ system provides a tailored elastic memory buffer tuned for your specific application. Whether implementing bridges, aggregating data, or buffering streams of data, Packet Queue provides an optimized solution for highly efficient packet buffering.

Features

- Dynamic allocation of memory for efficient implementations
- Up to 32 configurable channels; each with a dedicated write interface
- Independent write and read clock domains
- Configurable interface data widths of 8 to 128 bits
- Operates at OC-48 and OC-192 line rates
- Enables automatic data width conversion between write and read interfaces
- Configurable buffer depths up to 16,384 words and packet lengths up to 16,384 bytes
- Write and read interfaces utilize LocalLink for seamless system integration
- Ability to discard and retransmit packets
- Two almost-full indicators
- Supports complete and programmable scheduling control for the write and read interfaces
- Provides system interface for system diagnostics and statistics

LogiCORE Facts	
Core Specifics	
Supported Device Family	Virtex™-5, Virtex-4, Virtex-II Pro, Virtex-II, Spartan™-3E, Spartan-3, Spartan-3A/3AN/3A DSP
Provided with Core	
Documentation	Product Specification Release Notes
Design File Formats	VHDL, Verilog
Instantiation Template	VHDL and Verilog Wrapper
Design Tool Requirements	
Xilinx Implementation Tools	Xilinx ISE™ v9.2i
Simulation	Model Technology ModelSim® v6.1e Cadence™ IUS v5.8
Synthesis	Xilinx Synthesis Tool (XST) Synplicity® Synplify®
Supporting Specification	
LocalLink interface specification: www.xilinx.com/xlnx/xbiz/designResources/ip_product_details.jsp?iLanguageID=1&sSecondaryNavPick=&key=LocalLink_UserInterface Note: You must register on the LocalLink website to access the LocalLink specification.	
Support	
Provided by Xilinx, Inc. www.xilinx.com/support	

Applications

Packet Queue is ideal for a system that needs to buffer packet data from multiple channels and aggregate the packet data into a single output interface. This type of data buffering and aggregation is common in many applications, including networking, data communications, telecommunications, and digital signal processing. Packet Queue also enables packet-level flow-control by supporting the ability to drop packets on the write interface and retransmit packets on the read interface. A typical Ethernet over SONET application is shown in Figure 1 with Packet Queue used to buffer data received by four 1-Gigabit Ethernet links and transmit them to a SONET framer over a SPI-4.2 interface.

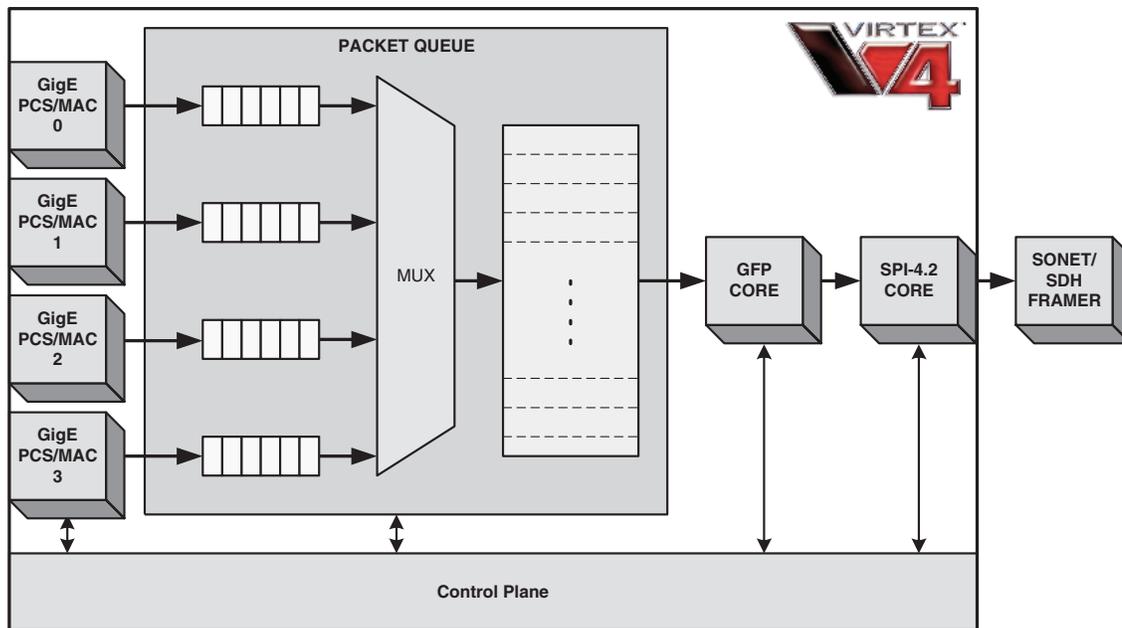


Figure 1: Ethernet over SONET Example Application

Core Overview

Packet Queue is a multi-channel, packet-oriented FIFO. A packet is defined as a group of data delineated by a Start-of-Frame (SOF) and an End-of-Frame (EOF). It accepts packets from several channels, each with its own independent write interface. Packet data is then accessed from a single read interface. Packet Queue allows packets being written to be discarded if errors are detected, and allows packets on the read interface to be transmitted and retransmitted as many times as required before they are purged from memory. Scheduling interfaces are provided to direct packet transfers.

Features

Dynamic Memory Allocation of Shared Resources

Packet Queue memory is dynamically allocated for each channel. Because the core does not automatically assign a fixed amount of memory per channel, the implementation enables an efficient use of block RAM for multi-channel applications. If the application requires memory to be allocated on a per-channel basis, it can be readily supported by restricting the flow of data on a user-controlled scheduler interface.

The depth of Packet Queue memory is configurable up to 16,384 words (words are relative to the width of the read interface bus). Individual packets of up to 16,384 bytes can be written into the core, provided that the memory is configured to be larger than the largest packet.

Supports Independent Channels

The Packet Queue core supports between 1–32 independent channels. Each channel writes data into the core on a dedicated interface. Each interface can have a unique write clock domain, which is fully asynchronous (no phase or frequency relationship) to every other channel and to the read interface. Support for independent clock domains enables synchronization to a single system clock.

Configuration Data Width

Packet Queue supports configurable read and write interfaces. Any byte-multiple from 8 to 128 bits is supported. The write interface also provides a remainder input that can be used to indicate the number of bytes valid during an end-of-frame transfer.

Packet Queue can be configured to transmit up to 32 bits of sideband data for each channel. It also supports the case of asymmetric data widths where the write bus is narrower than the read bus. If sideband data is enabled, it is width-converted as well. Asymmetric ports are supported when the channel specific shallow FIFOs (Front-End FIFOs) are configured to use block RAM.

Supports Retransmitting and Discarding Packets

Packet Queue supports the ability to retransmit packets on the read interface. When it is certain that a packet no longer needs to be transmitted, the user simply instructs the core to flush the packet, freeing up memory space. When packet errors are detected while writing data, the packet can be discarded mid-stream.

Debug and Monitoring

The Packet Queue core reports the number of packets currently stored, the number of segments (blocks of 64 bytes) used by each channel, and the number of unused segments remaining in memory. The packet count indicates the number of whole (readable) packets stored for a specific channel.

When a front-end FIFO fill-level surpasses a user-defined threshold, an almost-full indicator is triggered on the Write Scheduler Interface so that the Scheduler can take action to transfer that FIFO data into the Primary Data Buffer. Two thresholds are provided with two almost-full indicators for each channel.

Complete Scheduling Control

Packet Queue includes control and status interfaces required to aggregate and schedule multiple streams of data. The Write Scheduler Interface controls the transfer of packets from individual shallow FIFOs into the shared memory space. The Read Scheduler Interface is used to determine the sequence that packets are transmitted and retransmitted.

LocalLink User Interface

Packet Queue uses the Xilinx standard LocalLink interface for read and write access to the core. Local Link is a high performance, synchronous, point-to-point interface designed to rapidly enable the connection of multiple cores. The interface defines a set of protocol-transparent signals that allow for the transfer of generic packets.

Features include the following:

- Interface for packet oriented data of any length
- Synchronous point-to-point communication
- Upstream and downstream flow control
- Efficient channel bandwidth utilization

Functional Overview

Figure 2 is a functional block diagram of the Packet Queue core. For each channel, packet data is written into a shallow FIFO (Front-End FIFO) that buffers incoming packets for storage. Packet data is then transferred from the Front-end FIFOs into storage (the Primary Data Buffer). A user-defined scheduler determines the order that packet data is transmitted from this buffer.

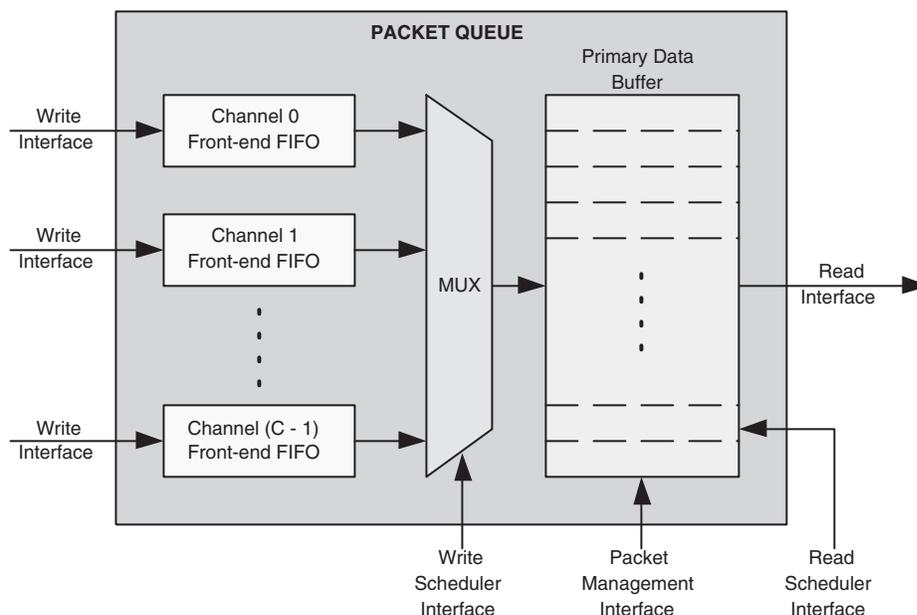


Figure 2: Packet Queue Functional Block Diagram

Writing Packet Data

The Write Interface for Packet Queue uses the Xilinx standard LocalLink interface for writing packets into the individual Front-End FIFOs. The write interface for each channel can have a unique clock domain, fully asynchronous to every other channel and to the read interface clock. If an error is detected while packet data is being written, the packet can be discarded by asserting a dedicated discontinue signal during the transfer.

If Packet Queue is configured to use block RAM for the Front-End FIFO, the core can also be configured to support asymmetric port widths where the write interface is narrower than the read interface.

Internal Packet Processing

The Write Scheduler Interface is used to control the transfer of packets from the individual Front-End FIFOs to the Primary Data Buffer. Custom user logic uses this interface to receive the FIFO status to determine the order that packets are transferred from the Front-End FIFOs to the Primary Data Buffer.

The Primary Data Buffer also provides status on the Write Scheduler Interface to indicate the number of packets stored per channel and the amount of available buffer space.

The Primary Data Buffer is the principal memory storage for Packet Queue and is implemented with block RAM. The size of the buffer is configurable and its storage capacity is dynamically allocated in 64-byte segments to incoming packets. Each packet consumes a minimum of one segment, and multiple packets cannot share a memory segment.

Reading Packet Data

The Read Interface for Packet Queue also uses the Xilinx standard LocalLink interface for transmitting data out of the core. The sequence of data transmission is controlled by the Read Scheduler Interface. This interface is driven by logic external to the core by using the Primary Data Buffer status and application-specific requirements to determine scheduling. The Packet Management Interface controls when packets are flushed or retransmitted from memory.

Core Interface

The abbreviations defined in [Table 1](#) are used to describe bus widths. For their values, see the instantiation template that is provided with the core. [Figure 3](#) illustrates the Packet Queue interfaces and signals. The Write, Read, Packet Management, and Read Scheduler Interfaces of Packet Queue conform to the LocalLink specification. All signals are active high.

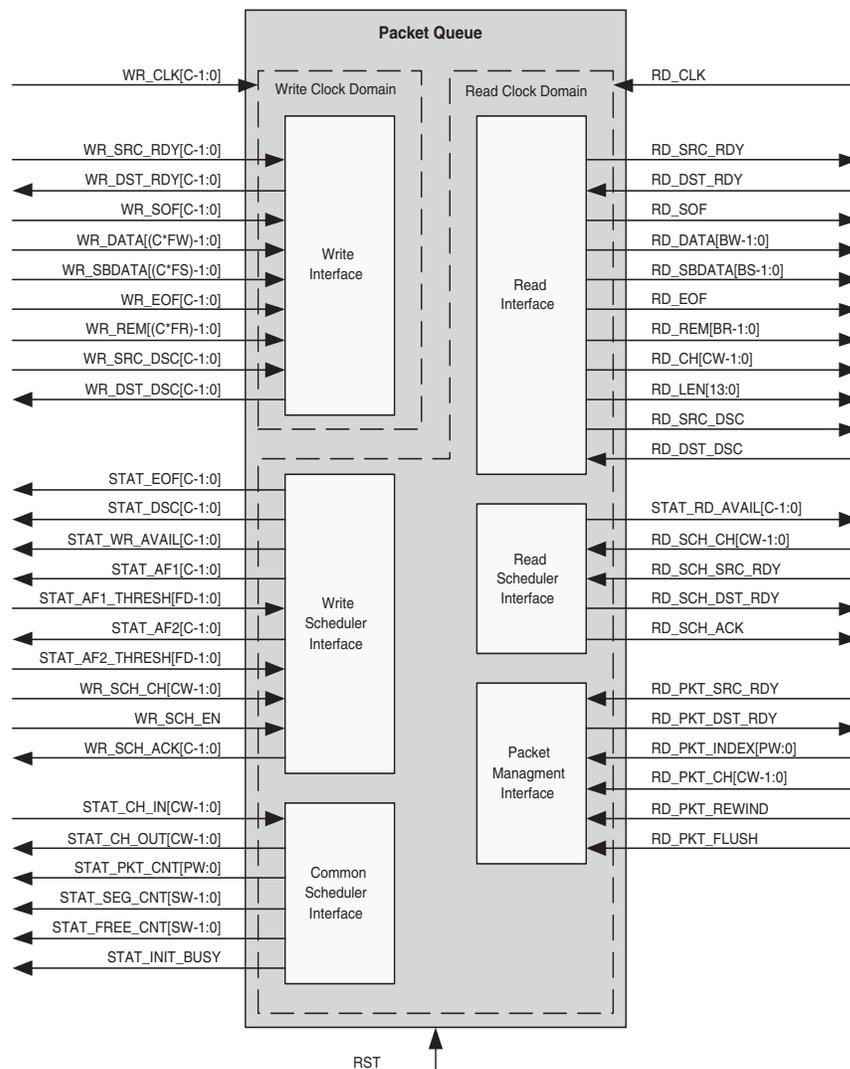


Figure 3: Packet Queue Interfaces

Table 1: Packet Queue Abbreviations

Parameter	Description
C	Channels: The number of channels supported by the core, as specified in the CORE Generator GUI.
FW	Front-end FIFO Data Width: The width of the Write Interface Data bus, as specified in the GUI.
FS	Front-end FIFO Sideband Width: The width of the Write Interface Sideband Data bus, as specified in the GUI.
FR	Front-end FIFO Remainder Width: The width of the Write Interface LocalLink Remainder bus, based on the value of FW.
FD	Front-end FIFO Depth: The number of bits required to represent the depth specified in the GUI for the Front-end FIFO.
CW	Channel Number Width: The number of bits required to represent a channel number (C).

Table 1: Packet Queue Abbreviations (Continued)

Parameter	Description
PW	Packet Count Width: The number of bits required to represent the number of packets (per channel) specified in the GUI.
SW	Segment Count Width: The number of bits required to represent the number of segments in the Primary Data Buffer, based on the Buffer depth specified in the GUI.
BW	Buffer Width: The width of the Read Interface Data bus, as specified in the GUI.
BS	Buffer Sideband Width: The width of the Read Interface Sideband Data bus, based on the value of FS, FW, and BW.
BR	Buffer Remainder Width: The width of the Read Interface LocalLink Remainder bus, based on the value of BW.

System Interface

Table 2 provides the signal name, direction, and a description of the System Interface.

Table 2: System Interface Ports

Signal Name	Direction	Description
RST	IN	Reset: Asynchronous signal that resets all logic internal to Packet Queue. The RST signal is latched asynchronously, sampled in the RD_CLK and each WR_CLK domain, and used to synchronously reset all internal logic. A RST pulse is not required after configuration. See "Reset Behavior" on page 15 for more information.

Write LocalLink Interface

Each channel has an independent, LocalLink-compliant write interface. With the exception of WR_DATA, WR_SBDATA, and WR_REM, the ports defined in Table 3 are all busses of one bit per channel. For example, the WR_CLK[3] bit is the Write Clock for channel 3. However, WR_DATA, WR_SBDATA, and WR_REM are aggregations of all the individual data, sideband, and remainder busses (respectively) for all write channels. For example, if the FIFO Write Width is configured to 16 bits, then WR_DATA[31:16] is the slice of WR_DATA corresponding to channel 1. The general formulas for the slice locations of a channel's bus inside these aggregated busses are listed below. For more information, see the LocalLink specification.

$$\text{CHANNEL_DATA}[C] = \text{WR_DATA}[(C+1)*FW-1 : C*FW]$$

$$\text{CHANNEL_SBDATA}[C] = \text{WR_SBDATA}[(C+1)*FS-1 : C*FS]$$

$$\text{CHANNEL_REM}[C] = \text{WR_REM}[(C+1)*FR-1 : C*FR]$$

Table 3: Write LocalLink Interface Ports

Signal	Direction	Description
WR_CLK[C-1:0]	IN	Clock: All write interface signals are synchronous to the rising edge of this clock.
WR_SRC_RDY[C-1:0]	IN	Source Ready: Indicates that the source is ready to transfer a word of data.

Table 3: Write LocalLink Interface Ports (Continued)

Signal	Direction	Description
WR_DST_RDY[C-1:0]	OUT	Destination Ready: Indicates that the destination (Packet Queue) is ready to accept a word of data. Data is transferred only when WR_SRC_RDY and WR_DST_RDY are both asserted during the same clock cycle.
WR_SOF[C-1:0]	IN	Start-of-frame: Indicates that the data on WR_DATA represents the beginning of a packet.
WR_DATA [(C*FW)-1:0]	IN	Data: Data to be written into Packet Queue. Available bus widths (FW) are any power of 2, from 8 to 128 bits. Allowed widths are restricted as described in "Generating the Core" on page 13 .
WR_SBDATA [(C*FS)-1:0]	IN	Sideband Data: Sideband data to be written into Packet Queue. Configurable width from 0 to 32 bits.
WR_EOF[C-1:0]	IN	End-of-frame: Indicates that the data on WR_DATA represents the end of a packet.
WR_REM [(C*FR)-1:0]	IN	Remainder: Binary-encoded count of valid bytes in WR_DATA when WR_EOF is asserted. Note that the WR_REM signal is not present when the width of WR_DATA is 8 bits and that WR_REM refers only to the WR_DATA bus and not to WR_SBDATA. Refer to "LocalLink Remainder Signal" on page 12 for more information.
WR_SRC_DSC[C-1:0]	IN	Source Discontinue: Indicates that the source is aborting transfer of the current packet. WR_SRC_DSC is only valid when WR_SRC_RDY and WR_DST_RDY are asserted, and WR_EOF must be asserted with WR_SRC_DSC. The current packet will be overwritten by the next packet written to the core and will not be transmitted to the read interface.
WR_DST_DSC[C-1:0]	OUT	Destination Discontinue: Indicates that the destination is aborting transfer of the current packet. The source must assert WR_EOF the next time WR_SRC_RDY is asserted and should subsequently restart transmission of the current packet (if possible). At this time, use of the WR_DST_DSC is not implemented, but is reserved for future use. This signal will always be driven to a logical '0.'

Read LocalLink Interface

Packet Queue has a single LocalLink-compliant read interface. For additional information, see the LocalLink specification. The order in which stored packets are presented on the read interface is controlled by the Read Scheduler, described in ["Scheduler Interfaces" on page 10](#). The read interface ports are listed in [Table 4](#). All signals are synchronous to RD_CLK.

Table 4: Read LocalLink Interface Ports

Signal Name	Direction	Description
RD_CLK	IN	Clock: All read interface signals are synchronous to the rising edge of this clock.
RD_SRC_RDY	OUT	Source Ready: Indicates that the source (Packet Queue) is ready to transfer a word of data.

Table 4: Read LocalLink Interface Ports(Continued)

Signal Name	Direction	Description
RD_DST_RDY	IN	Destination Ready: Indicates that the destination is ready to accept a word of data. Data is transferred only when RD_SRC_RDY and RD_DST_RDY are both asserted during the same clock cycle.
RD_SOF	OUT	Start-of-Frame: indicates that the data on RD_DATA represents the beginning of a packet.
RD_DATA [BW-1:0]	OUT	Data: Data read from Packet Queue. Available bus widths (BW) are any power of 2, from 8 to 128 bits.
RD_SBDATA [BS-1:0]	OUT	Sideband Data: Sideband data read from Packet Queue.
RD_EOF	OUT	End-of-Frame: Indicates that the data on RD_DATA represents the end of a packet.
RD_REM [BR-1:0]	OUT	Remainder: Binary-encoded count of valid bytes in RD_DATA when RD_EOF is asserted. Note that the RD_REM signal is not present when the width of RD_DATA is 8 bits and that RD_REM refers only to the RD_DATA bus and not to RD_SBDATA. Refer to " LocalLink Remainder Signal " on page 12 for more information.
RD_CH [CW -1:0]	OUT	Read Channel: Indicates the channel that the current packet came from.
RD_LEN[13:0]	OUT	Read Length: The length, in bytes, of the current packet. Valid beginning at SOF.
RD_SRC_DSC	OUT	Source Discontinue: Indicates that the source is aborting transfer of the current packet. RD_EOF will be asserted when RD_SRC_DSC is asserted. At this time, use of the RD_SRC_DSC is not implemented, but is reserved for future use. This signal is always driven to a logical '0.'
RD_DST_DSC	IN	Destination Discontinue: Indicates that the destination is aborting transfer of the current packet. The source will assert RD_EOF the next time RD_SRC_RDY is asserted and will subsequently transmit the next packet indicated by the Read Scheduler.

Packet Management Interface Ports

The Packet Management Interface controls the retransmitting and flushing of packets in the Primary Data Buffer. Packets being read can be retransmitted as many times as necessary (see "[Packet Management](#)" on page 19). The Packet Management Interface Ports are listed in Table 5. All signals are synchronous to RD_CLK. Note that RD_PKT_INDEX, RD_PKT_CH, RD_PKT_REWIND, RD_PKT_FLUSH, and RD_PKT_SRC_RDY must be held constant between the assertion of RD_PKT_SRC_RDY the assertion of RD_PKT_DST_RDY. and that RD_PKT_SRC_RDY must be asserted before the core asserts RD_PKT_DST_RDY.

Table 5: Packet Management Interface Ports

Signal Name	Direction	Description
RD_PKT_INDEX [PW:0]	IN	Read Packet Index: Controls the number of packets rewound or flushed when RD_PKT_REWIND or RD_PKT_FLUSH are asserted, as described below in Packet Flow Management.
RD_PKT_CH [CW-1:0]	IN	Read Packet Channel: Specifies which channel will be affected by the Rewind/Flush command.
RD_PKT_REWIND	IN	Read Rewind: Sets the next packet to be read from the core to RD_PKT_INDEX after the oldest packet, as described in " Packet Management " on page 19.
RD_PKT_FLUSH	IN	Read Flush: Marks the RD_PKT_INDEX oldest packets as read, and frees space in the Primary Data Buffer for new data on the write interface. Unlike a normal FIFO, Packet Queue allows the ability to continually replay a packet until RD_PKT_FLUSH is used to remove it from memory.
RD_PKT_SRC_RDY	IN	Read Packet Source Ready: Indicates that the values on the above signals are valid. The Rewind/Flush operation will only be processed when RD_PKT_SRC_RDY and RD_PKT_DST_RDY are asserted simultaneously.
RD_PKT_DST_RDY	OUT	Read Packet Destination Ready: Indicates the core is ready to accept a Rewind/Flush command. The Rewind/Flush operation will only be processed when RD_PKT_SRC_RDY and RD_PKT_DST_RDY are both asserted during the same clock cycle. Note that the core will not assert RD_PKT_DST_RDY until after the user first asserts RD_PKT_SRC_RDY.

Scheduler Interfaces

Packet Queue requires an external scheduler module for both the write and read interfaces. This section describes connections between the core and each scheduler. All scheduler signals are synchronous to RD_CLK. [Table 6](#) defines core outputs common to both schedulers, [Table 7](#) defines Write Scheduler-specific ports, and [Table 8](#) defines Read Scheduler-specific ports.

Packet Queue provides status information needed by the scheduling modules to determine how to transfer packets. Each Front-end FIFO has two programmable almost-full indicators and a data available signal (STAT_WR_AVAIL). The Primary Data Buffer provides an indication of the number of free segments and a count of the number of packets and segments stored for each channel.

Common Scheduler Interface

The Common Scheduler Interface provides status that is shared by both the Write and Read Scheduler Interfaces. It reports information about one channel at a time. Status information for a given channel is requested by driving STAT_CH_IN with the channel number. After the request is processed by the core, STAT_PKT_CNT and STAT_SEG_CNT present the information for the requested channel, and STAT_CH_OUT reflects the channel number. Note that status information for different channels can continually be requested by driving a different STAT_CH_IN on every clock cycle; it is not necessary to wait for STAT_CH_OUT to indicate the channel status. All signals are synchronous to RD_CLK.

Table 6: Common Scheduler Interface Ports

Signal Name	Direction	Description
STAT_CH_IN [CW-1:0]	IN	Channel In: Specifies the channel that the Status values STAT_PKT_CNT and STAT_SEG_CNT will report on.
STAT_CH_OUT [CW-1:0]	OUT	Channel Out: Specifies the channel associated with counts being presented on STAT_PKT_CNT and STAT_SEG_CNT.
STAT_PKT_CNT [PW:0]	OUT	Packet Count: The number of packets stored in the Primary Data Buffer for the channel indicated by STAT_CH_OUT.
STAT_SEG_CNT [SW-1:0]	OUT	Segment Count: The number of segments in the Primary Data Buffer used for the channel indicated by STAT_CH_OUT.
STAT_FREE_CNT [SW-1:0]	OUT	Free Count: The number of unallocated memory segments in the Primary Data Buffer. Note that this will never be as large as the total number of segments in the Primary Data Buffer due to channel overhead.
STAT_INIT_BUSY	OUT	Initializing: Asserted while Packet Queue is initializing internal control logic. Refer to " Reset Behavior " on page 15 for details.

Write Scheduler Interface

The Write Scheduler Interface provides status outputs for use in the Write Scheduler, and also accepts input from the Write Scheduler to control transferring data from the Front-end FIFOs to the Primary Data Buffer. Data is transferred from the Front-end FIFOs on a word-by-word basis. All signals are synchronous to RD_CLK.

Table 7: Write Scheduler Interface Ports

Signal Name	Direction	Description
STAT_EOF[C-1:0]	OUT	Transferred End-of-Frame: Indicates that a packet EOF has been transferred from the Front-end FIFO (one per channel).
STAT_DSC[C-1:0]	OUT	Transferred Discontinue: Indicates that the last word of a discontinued packet has been transferred from the Front-end FIFO (one per channel).
STAT_WR_AVAIL [C-1:0]	OUT	Write Available: Indicates that the Front-end FIFO is ready to transfer a word of data (one per channel).
STAT_AF1[C-1:0]	OUT	Almost-Full 1: Asserted when the Front-end FIFO is at least as full as STAT_AF1_THRESH (one per channel). Because of the asynchronous nature of the Front-end FIFO, STAT_AF1 experiences a delay of several clock cycles.
STAT_AF1_THRESH [FD-1:0]	IN	Almost-Full Threshold 1: The level at which to assert STAT_AF1. Threshold applies to all Front-end FIFOs and are set relative to words on the Read Interface.
STAT_AF2[C-1:0]	OUT	Almost-Full 2: Asserted when the Front-end FIFO is at least as full as STAT_AF2_THRESH (one per channel). Because of the asynchronous nature of the Front-end FIFO, STAT_AF2 experiences a delay of several clock cycles.
STAT_AF2_THRESH [FD-1:0]	IN	Almost-Full Threshold 2: The level at which to assert STAT_AF2. Threshold applies to all Front-end FIFOs and are set relative to words on the Read Interface.

Table 7: Write Scheduler Interface Ports (Continued)

WR_SCH_CH [CW-1:0]	IN	Write Schedule Channel: The binary-encoded channel number indicating which Front-end FIFO to read from to transfer data to the Primary Data Buffer.
WR_SCH_EN	IN	Write Schedule Enable: Asserted to indicate that the value on WR_SCH_CH is valid and should be acted upon by the core.
WR_SCH_ACK [C-1:0]	OUT	Write Schedule Acknowledge: Asserted to indicate a word was transferred from the Front-end FIFO to the Primary Data Buffer. This is a one-hot bus, with each bit corresponding to one channel.

Read Scheduler Interface

The Read Scheduler Interface controls the output of the core on a packet-by-packet basis. It provides per-channel status indicating whether there is at least one packet available to read. It is used to control which packet gets read out by providing the core with the channel number (RD_SCH_CH). An output signal is provided indicating that a new channel address was accepted (RD_SCH_DST_RDY). A new channel can not be accepted by the core unless the user first asserts that the source is ready (RD_SCH_SRC_RDY). All signals are synchronous to RD_CLK.

Table 8: Read Scheduler Interface Ports

Signal Name	Direction	Description
STAT_RD_AVAIL [C-1:0]	OUT	Read Available: Indicates that at least one packet is stored in the channel (one per channel).
RD_SCH_CH [CW-1:0]	IN	Read Schedule Channel: The channel number to read from next.
RD_SCH_SRC_RDY	IN	Read Schedule Source Ready: Asserted to indicate that the value on RD_SCH_CH is valid and should be acted upon by the core.
RD_SCH_DST_RDY	OUT	Read Schedule Destination Ready: Asserted by the core to indicate that the value on RD_SCH_CH was accepted. RD_SCH_CH is not accepted by the core unless RD_SCH_SRC_RDY and RD_SCH_DST_RDY are asserted during the same clock cycle. Note that the core will not assert RD_SCH_DST_RDY until after the user first asserts RD_SCH_SRC_RDY.
RD_SCH_ACK	OUT	Read Schedule Acknowledge: Validates that the channel accepted when RD_SCH_DST_RDY asserted has data to transfer. This signal will assert two cycles after RD_SCH_DST_RDY if the channel has a packet available to read.

LocalLink Remainder Signal

Packet Queue does not require that packet lengths are even multiples of the interface width. Because of this, the LocalLink remainder signal (WR_REM or RD_REM) is used to specify which bytes are valid at EOF. The remainder is a binary-encoded value specifying the last valid byte as measured from the most-significant byte of the data word. [Table 9](#) shows an 8-byte example of an EOF word. In this example, the REM bus value of 5 indicates that the last byte of the packet is located at DATA [23 : 16] and that DATA [15 : 0] are not valid.

Table 9: EOF Word Example

REM bus value = 5	0	1	2	3	4	5	6	7
Data Bus (MSB)	63:56	55:48	47:40	39:32	31:24	23:16	15:8	7:0
	Payload Bytes						Not Valid	

Generating the Core

Use the following steps to generate the Packet Queue core.

1. Start the Xilinx CORE Generator.
2. After creating a new project or opening an existing one, the IP core functional categories appear at the left side of the window.
3. Choose Communication & Networking > Telecommunications; then choose Packet Queue.

The main window appears, from which you select project options.

Figure 4 illustrates the options available for configuring Packet Queue.

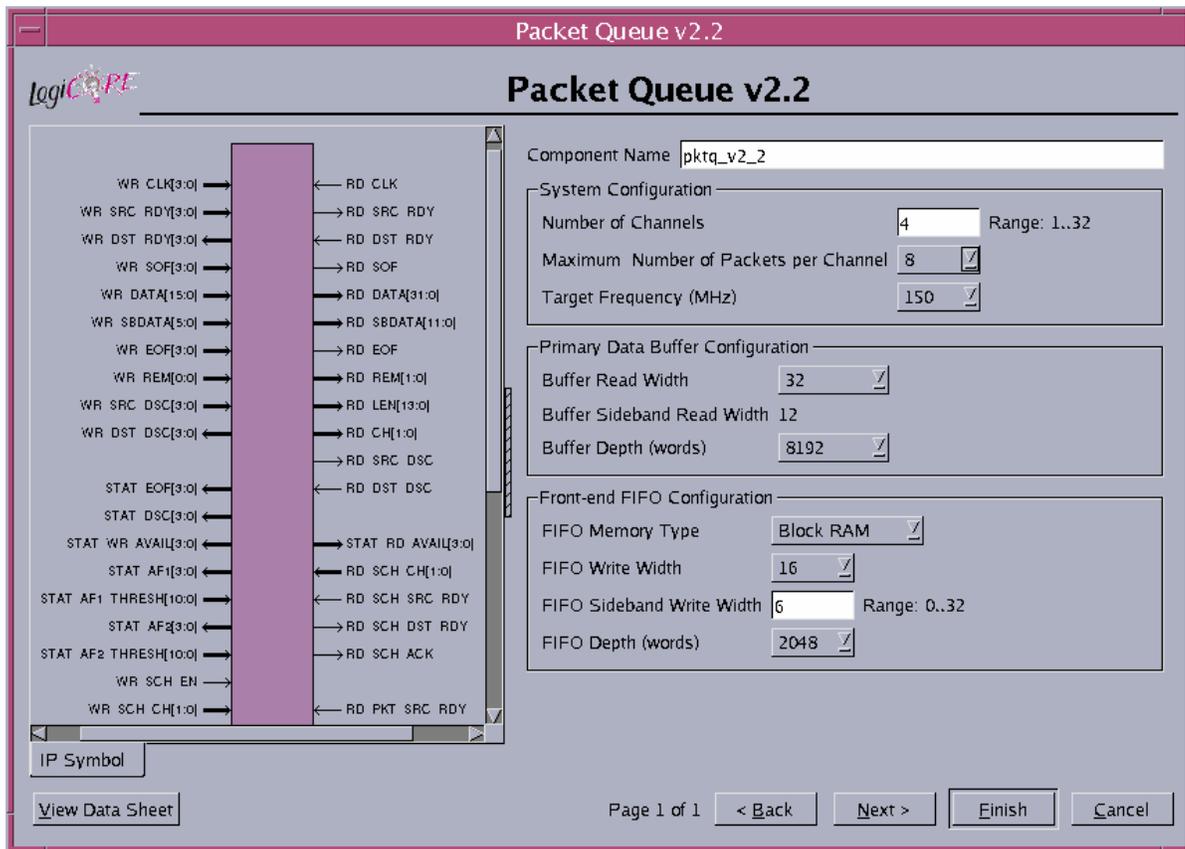


Figure 4: Packet Queue CORE Generator Main Screen

Component Name

The base name of the output files generated for the core. Names must begin with a letter and be composed of any of the following characters: a to z, 0 to 9, and "_." Names cannot be Verilog or VHDL reserved words.

Number of Channels

Enter the number of channels to be supported by the core. This determines the number of write interfaces and corresponding Front-end FIFOs included in the core.

Maximum Number of Packets per Channel

Enter the maximum number of channels to be stored at a time, per channel, in the Primary Data Buffer. This value should be based on the system that will include Packet Queue. A larger number allows more outstanding packets at a time, but also consumes more FPGA resources.

Target Frequency

The maximum clock frequency expected for the write and read interfaces. The CORE Generator will optimize Packet Queue for the frequency requested.

Buffer Read Width

The width of the Read Interface Data bus, in bits.

Buffer Depth

The depth of the Primary Data Buffer, in words. Words are in terms of the read interface width. The usable space in the Primary Data Buffer is reduced by one segment per channel due to overhead plus one segment for the core.

Buffer Sideband Read Width

The width of the Read Interface Sideband Data bus, in bits. This value is calculated from the FIFO Sideband Width.

FIFO Memory Type

The type of memory primitive to use for the Front-end FIFOs. If data-width conversion is required, select block RAM.

FIFO Write Width

The width of the Write Interface Data buses, in bits. This can be the same as the Buffer Read Width or smaller, if data-width conversion is required. For data-width conversion, FIFO memory type must be block RAM, and the FIFO Write Width must be less than the Buffer Read Width.

FIFO Sideband Width

The number of sideband bits to be supported by Packet Queue Write Interface. When the optional data-width conversion is used, the sideband bits will also be widened by the ratio of read data width to write data width. Otherwise, the width of the sideband bus on the read-side will match that of the write-side.

FIFO Depth

The depth of each Front-end FIFO, in words. The permitted values for depth depend on the FIFO Memory Type selected. For distributed RAM, allowed values are 16, 32, 64, 128, or 256 words, and for block RAM, allowed values are 512, 1024, 2048, or 4096 words. The FIFO depth is specified in words, which is the write interface width. Note that the Front-end FIFO may be smaller than a single packet if desired.

Generated Files

Table 10 describes the files created when Packet Queue is generated.

Table 10: Packet Queue Generated Files

Name	Description
<component_name>.ngc	Core netlist
<component_name>.v[hd]	Functional VHDL or Verilog simulation model
<component_name>.v[ho eo]	VHDL or Verilog instantiation template
<component_name>.xco	CORE Generator project specific option file
<component_name>_flist.txt	List of generated files
packet_queue_release_notes.txt	Packet Queue Release Notes
packet_queue_ds509.pdf	Packet Queue Product Specification

Designing with the Core

This section provides information for creating a fully functional design incorporating Packet Queue, including the design steps required to implement each feature. For successful use of Packet Queue, the design guidelines provided in this section must be observed.

Reset Behavior

Packet Queue reset input (RST) is latched asynchronously by the core. After it is latched, it is synchronized to each clock domain within the core and is used to synchronously reset all internal logic. After RST is asserted, the core will deassert its WR_DST_RDY outputs, assert its STAT_INIT_BUSY output, and perform a full reset of internal logic. The Front-end FIFOs reset quickly and may be written to after the WR_DST_RDY outputs are asserted. However, resetting the Primary Data Buffer takes many clock cycles to complete, as block RAM contents must be initialized to preset values. STAT_INIT_BUSY remains asserted until the Primary Data Buffer has completed its initialization sequence. The core will not accept any scheduling requests while STAT_INIT_BUSY is asserted, so it is recommended that STAT_INIT_BUSY be used as a synchronous reset input to the scheduler logic.

It is not necessary to assert RST to the core after power-up, as it is fully functional after the FPGA has been programmed. RST is only required to return the core to its power-up condition after operation has begun.

Clocking

As Packet Queue does not include clock buffers, any clock scheme required by the design should be implemented outside of Packet Queue. You can provide independent clocks (no phase or frequency

relationship) for the read and write interfaces (and also separate clocks for each write interface channel), or you can connect all clocks together for fully synchronous operation.

Basic Operation

The two primary interfaces on Packet Queue are the write and read interfaces. The write interface buffers up an entire packet until the EOF (end-of-frame/packet) is written in, at which point the packet becomes available for reading on the read interface. At any point up until the EOF is written to Packet Queue, the write logic can discontinue (abort) the packet (WR_SRC_DSC). This causes the entire packet to be dropped from memory as if it had never been written into Packet Queue. Individual packets may be larger than the Front-end FIFO without any negative impact on core operation.

On the read interface, packets are available to be read (and re-read) until they are flushed from the core by the read logic. To free up space for new packets, the read logic should read packets and flush them as soon as it determines they are no longer needed.

Figures 5 and 6 are examples of basic write and read operations for channel 0. In each example, a packet begins with SOF and ends with EOF. Data is stalled on cycles where either source ready or destination ready is de-asserted.

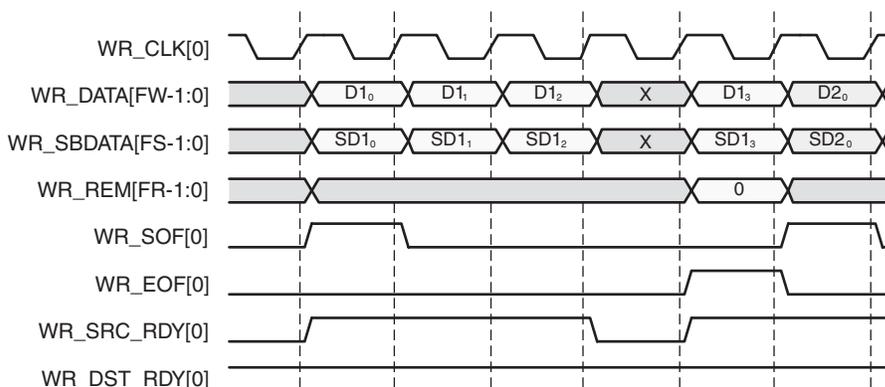


Figure 5: Basic Write Operation

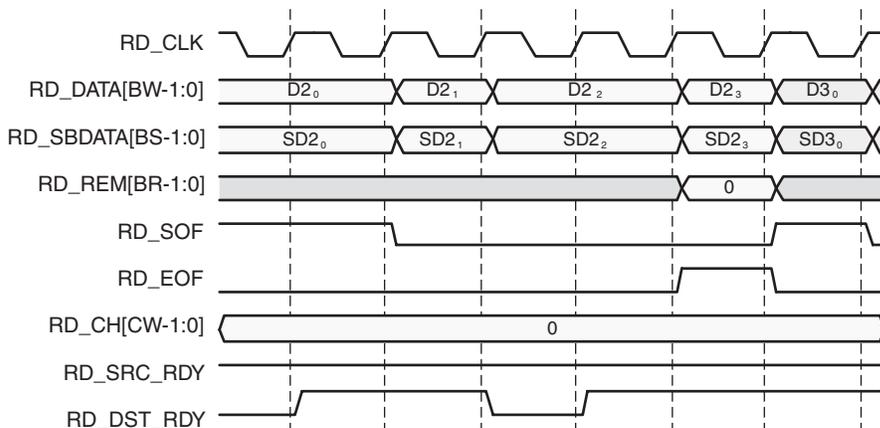


Figure 6: Basic Read Operation

Write Discontinue

Packets containing errors can be discarded at any point during a write operation. A write discontinue is invoked by asserting `WR_SRC_DSC` and `WR_EOF` when `WR_SRC_RDY` and `WR_DST_RDY` are asserted. The write discontinue causes the packet to be discarded as if it had never been written to the core. Although the Packet Queue core has a `WR_DST_DSC` port which would allow the core to terminate an incoming packet early, it is reserved for future use and is not asserted by the core in this version.

Figure 7 shows the Write Interface signals during a write discontinue. The upstream logic begins writing the first packet (beginning with `D10` and `SOF`). On the third word (`D12`), it issues a write discontinue by asserting `WR_SRC_DSC` and `WR_EOF` along with `WR_SRC_RDY`. Following the write discontinue, the upstream logic begins writing a new packet which then overwrites the aborted packet.

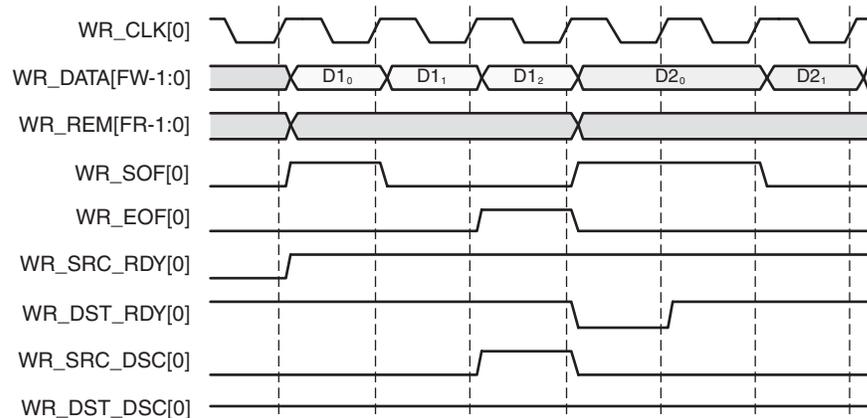


Figure 7: Write Discontinue

Write Scheduling

When packets are written to Packet Queue, they are initially stored in the Front-end FIFO corresponding to the channel from which the packet came. Packets must subsequently be transferred into the Primary Data Buffer before they can be transmitted on the core Read Interface. The scheduling of these transfers is controlled by the Write Scheduler Interface.

The core provides several signals on the Common Scheduler Interface and the Write Scheduler Interfaces for the scheduling process. These signals include indications of fullness of the Front-end FIFOs and Primary Data Buffer, and whether the word at the output of each Front-end FIFO is an EOF or discontinue. The scheduler must drive the core `WR_SCH_EN` and `WR_SCH_CH` inputs for each cycle that data is to be transferred. The `WR_SCH_EN` signal indicates that a word should be transferred to the Primary Data Buffer during that cycle. When `WR_SCH_EN` is asserted, `WR_SCH_CH` specifies the binary-encoded channel number that data should be transferred from, provided there is data available to transfer. For every word that is successfully transferred to the Primary Data Buffer (that is, if `WR_SCH_EN` is asserted and there is data available for the selected channel), the bit in `WR_SCH_ACK` corresponding to the selected channel is asserted some time later. It is not harmful to select a channel for which no data is available; the request will simply be ignored by Packet Queue.

Figure 8 illustrates the interaction between the Write Scheduler Interface and Packet Queue in a two-channel system. Although several signals are available to the scheduler, only a small subset are used in this example.

Data becomes available on channel 0 and it is selected by the scheduler. After a few cycles, data also becomes available on channel 1 and the scheduler alternates between the two channels. Finally, channel 0 empties and the scheduler selects channel 1 continuously.

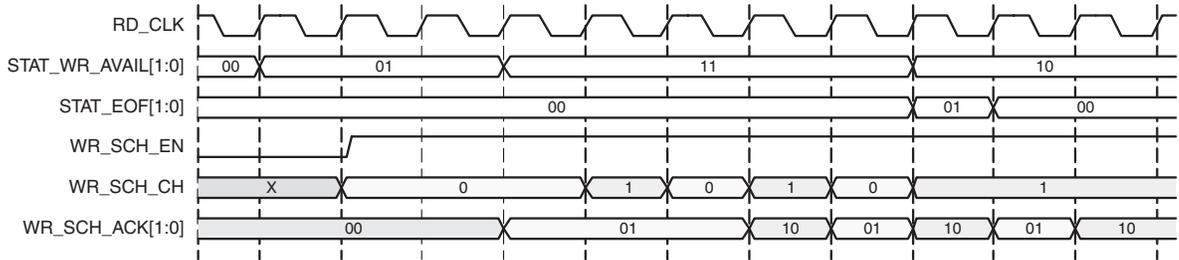


Figure 8: Write Scheduling Example

Read Scheduling

Packets must be scheduled to be transmitted to the Read Interface. As with the Write Scheduler Interface, this process is controlled by the Read Scheduler Interface. RD_SCH_CH and RD_SCH_SRC_RDY signals are driven by the scheduler to select the next channel to transmit from. The core drives RD_SCH_DST_RDY to indicate that the channel number was accepted. If a packet is available to be read from the selected channel, Packet Queue asserts RD_SCH_ACK two cycles after asserting RD_SCH_DST_RDY. If not, the request will be ignored—scheduling an empty channel has no negative impact on the functioning of the core.

Figure 9 shows an example of the Read Scheduler Interface in a two-channel system. In this example, STAT_RD_AVAIL indicates that a packet is available on channel 0. The Read Scheduler Interface signals the core to read the packet by setting RD_SCH_CH to 0, and asserting RD_SCH_SRC_RDY. The core responds by asserting RD_SCH_DST_RDY, indicating that it will read from channel 0 next. As the packet from channel 0 is being transmitted, a packet becomes available on channel 1, as indicated by the transition on STAT_RD_AVAIL. The Read Scheduler Interface signals to the core to read from channel 1 by setting RD_SCH_CH to 1 and asserting RD_SCH_SRC_RDY. The core asserts RD_SCH_DST_RDY in response to indicate that it will read from channel 1 next. After the core transmits the end of the packet from channel 0, it starts transmitting the packet from channel 1. Additionally, the core asserts RD_SCH_ACK, indicating that a packet has been successfully scheduled and will appear on the Read Interface.

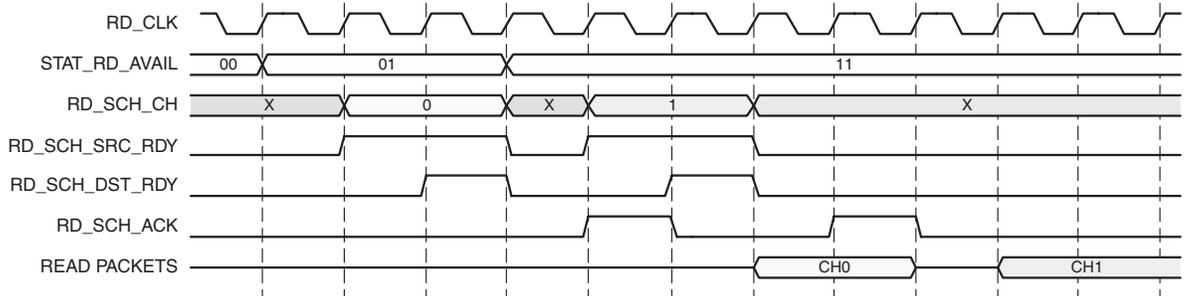


Figure 9: Read Scheduling

Figure 10 shows a situation when RD_SCH_DST_RDY asserts, but RD_SCH_ACK does not assert until two cycles later. There is a packet available for channel 0 because STAT_PKT_CNT is 1, while STAT_CH_OUT is set to 0 (for channel 0). The first assertion of RD_SCH_DST_RDY schedules the single

packet in channel 0, and two cycles later RD_SCH_ACK asserts to acknowledge that the schedule operation was accepted. However, the second assertion of RD_SCH_DST_RDY is not followed by a RD_SCH_ACK two cycles later, because the single packet has already been scheduled.

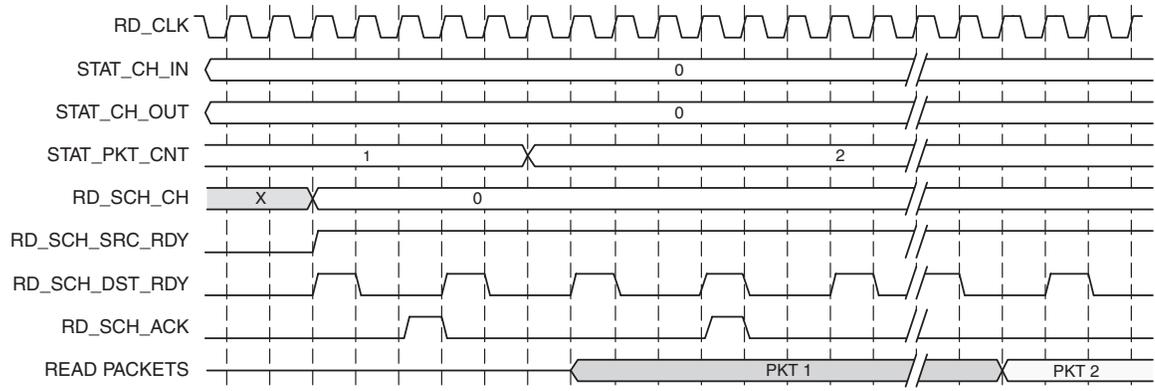


Figure 10: Read Schedule Acknowledge and Packet Response

Packet Management

The signals of the Packet Management Interface control two operations—rewind and flush. They operate on a single channel at a time and both a rewind and flush can be requested simultaneously. Packet Queue uses the Packet Management Interface signals RD_PKT_REWIND, RD_PKT_FLUSH, RD_PKT_INDEX, RD_PKT_CH, RD_PKT_SRC_RDY, and RD_PKT_DST_RDY to control packet ordering on the Read Interface. These signals direct the core to retransmit a packet if needed and to flush packets after it is determined that they are no longer required.

Flush

To flush a packet or packets, RD_PKT_FLUSH is asserted along with RD_PKT_SRC_RDY. Doing so causes the oldest RD_PKT_INDEX packets for the channel specified by RD_PKT_CH to be discarded. As shown in Figure 11, if RD_PKT_INDEX = 2, the two oldest packets are discarded.

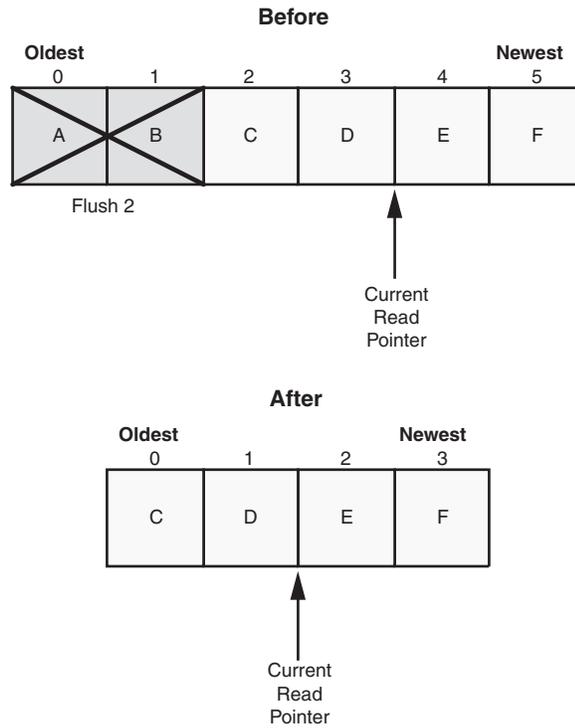


Figure 11: Flush with RD_PKT_INDEX = 2

Flushes may be done on a different channel than the one being read, or on the same channel, provided the packet currently being read is not flushed. You must instruct the core with the RD_PKT_FLUSH control to delete a packet from memory. This does not happen automatically after a packet is transmitted.

Rewind

Asserting RD_PKT_REWIND with RD_PKT_SRC_RDY causes the internal read pointer for the channel specified by RD_PKT_CH to be moved to the beginning of the packet which is RD_PKT_INDEX after the oldest stored packet. As shown in Figure 12 and Figure 13, if RD_PKT_INDEX = 2, then the next packet read from that channel is the third-oldest packet in memory; if RD_PKT_INDEX = 0, the next packet read is the oldest.

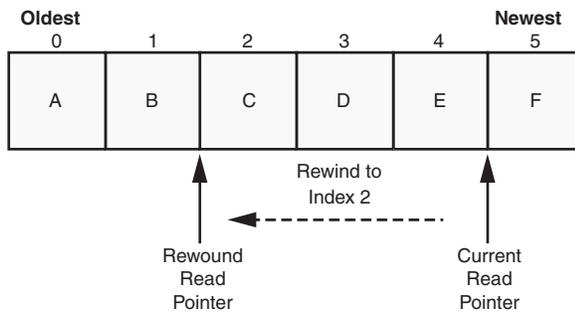


Figure 12: Rewind with RD_PKT_INDEX = 2

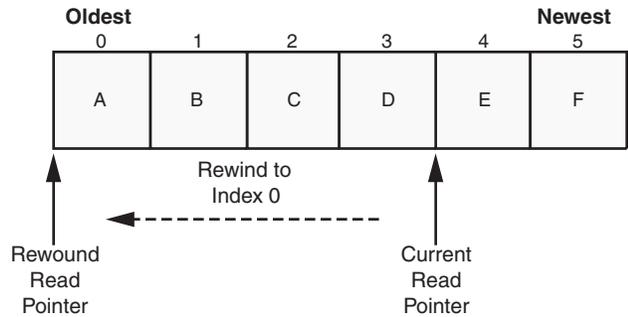


Figure 13: Rewind with RD_PKT_INDEX = 0

Rewinds may be executed on the current channel being read or on a different channel; however, if the rewind is executed on the current channel being read, the read pointer of the channel is not rewound until the end of the current packet is transmitted on the read interface.

Rewind and Flush

RD_PKT_FLUSH and RD_PKT_REWIND can be asserted simultaneously. In this case, they operate on the same channel in a related way. As shown in Figure 14, if RD_PKT_FLUSH and RD_PKT_REWIND are both asserted with RD_PKT_INDEX = 2, the two oldest packets is discarded and the third oldest packet is the next packet transmitted from the selected channel.

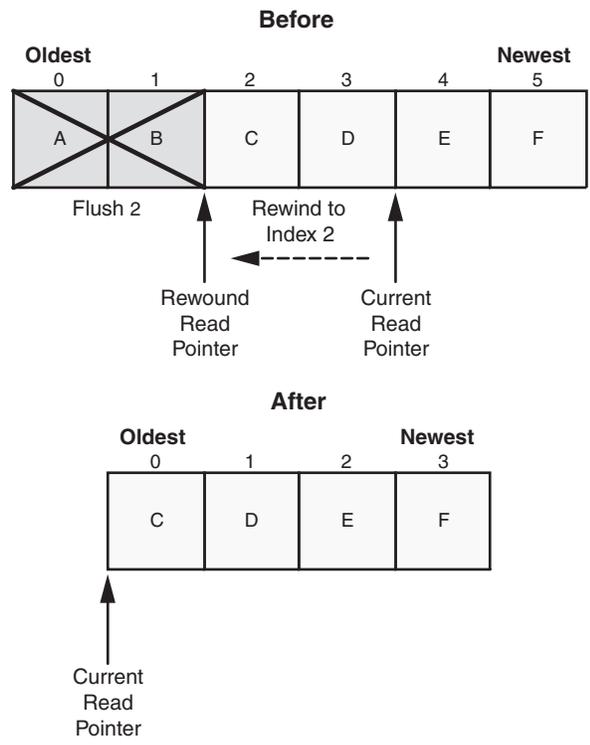


Figure 14: Rewind and Flush with RD_PKT_INDEX = 2

Systems Not Requiring Retransmit

If Packet Queue is to be implemented in a system that does not require the retransmit functionality, it is possible to configure the Packet Management Interface to flush every packet that is transmitted. In this configuration, drive RD_PKT_REWIND with a logical '0.' When an EOF occurs on the Read Interface

(RD_EOF = '1,' RD_SRC_RDY = '1,' and RD_DST_RDY = '1') the value of a user-implemented counter corresponding to that channel should be incremented. When that channel counter is non-zero, the channel should be latched onto RD_PKT_CH, RD_PKT_INDEX should be driven with the counter value, and RD_PKT_SRC_RDY driven to a logical '1.' RD_PKT_SRC_RDY should be cleared when RD_PKT_DST_RDY is asserted by the core and the channel counter should be decremented by RD_PKT_INDEX.

Packet Management Interface Requirements

The Packet Management Interface requires that after RD_PKT_SRC_RDY is asserted, RD_PKT_REWIND, RD_PKT_FLUSH, RD_PKT_CH, and RD_PKT_INDEX must be held constant until RD_PKT_DST_RDY is asserted. Packets must be flushed from memory when no longer needed; otherwise, the Primary Data Buffer fills up and Packet Queue will stop accepting packets on its Write Interfaces.

The following additional restrictions apply to the Packet Management Interface signals:

- RD_PKT_FLUSH must not be asserted when RD_PKT_INDEX = 0 or when RD_PKT_INDEX is greater than the number of stored packets for the given channel.
- RD_PKT_FLUSH must not be asserted if the value on RD_PKT_INDEX would cause a packet currently being read to be flushed.
- RD_PKT_REWIND must not be asserted when RD_PKT_INDEX is greater than or equal to the number of stored packets for the given channel.

Read Flush

Figure 15 shows flushing of packets that are no longer needed from Packet Queue. In this case, the four oldest packets for channel 2 are flushed. Flushing packets does not interrupt the flow of data on the Read Interface.

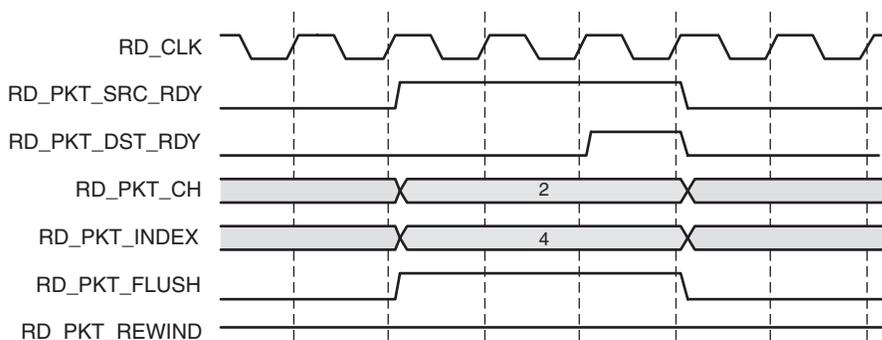


Figure 15: Read Flush Operation

Read Rewind and Flush

Figure 16 shows rewinding of the Packet Queue read pointer. In this case, the rewind selects the second-oldest packet and flushes the oldest for channel 3. The operation does not interrupt the flow of data on the Read Interface.

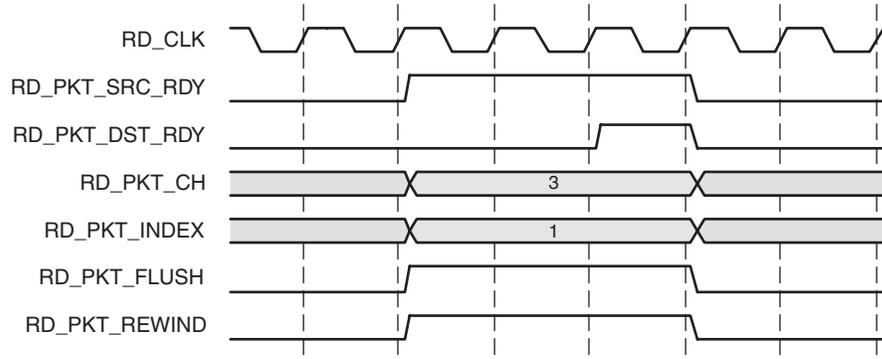


Figure 16: Read Rewind and Flush Operation

Read Discontinue

Packet Queue provides a mechanism to abort reading of the current packet on the Read Interface. For example, this could be used if it is detected that there was an error on the transmission interface and the current packet was not received correctly. Asserting RD_DST_DSC during a read causes Packet Queue to terminate the current packet prematurely, return the read pointer to the beginning of the current packet, and begin transmitting the next packet selected by the Read Scheduler Interface.

Figure 17 illustrates a read discontinue on the Read Interface. As word D1₂ is read, the downstream logic signals a discontinue by asserting RD_DST_DSC along with RD_DST_RDY. The core responds by asserting RD_EOF on the following clock cycle, and then subsequently begins transmitting the next packet selected by the Read Scheduler Interface.

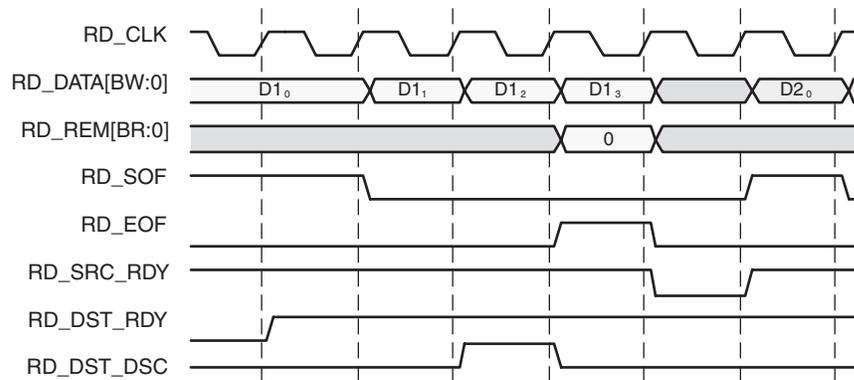


Figure 17: Read Discontinue

Input Validation

Packet Queue expects well-formed packets on its input. To insure consistent behavior at startup and after a reset, it silently drops all input data until it receives a SOF. After the SOF is received, it will ignore subsequent SOF words until an EOF is received. Following the EOF, data is again dropped

silently until the next SOF is received. In other words, only data between the first SOF and the first EOF signals is actually written to the internal memory—everything else is silently discarded.

Packet Queue supports packet sizes up to 16 kB; however, it does not verify that input packets meet this restriction. If oversize packets are written to the core, behavior is undefined. If an input packet was excessively long and filled the entire Primary Data Buffer, the core could enter a deadlock state and stop responding to input. To prevent a deadlock, input data must not violate maximum packet length.

Packet Flow Example

Figure 18 illustrates packet flow through Packet Queue. As shown, the read side of the core is connected to external interface logic that transmits packets across an external interface and determines whether or not packets were received successfully. The Write Interface is not shown.

Figure 18 shows signals that represent the Read, Read Scheduler, and Packet Management Interfaces, but does not show cycle-accurate representations, and is provided only as an illustration. In the example all the interfaces are operating on a single channel. Pulses on the “Schedule Channel” signal represent a packet being scheduled for the channel. For the Packet Management Interface, the SRC_RDY and DST_RDY signals are implied. The following events take place in the example:

1. The first packet (P1) becomes available for reading; the Read Scheduler Interface schedules it; the Read Interface begins reading it out.
2. A second packet (P2) becomes available and gets scheduled. After the Read Interface finishes reading the first packet, the second packet starts.
3. User logic determines that the first packet was received successfully and issues a flush command to discard it from Packet Queue.
4. The Read Scheduler Interface schedules the third packet (P3), which became available earlier. It shortly emerges on the Read Interface.
5. User logic determines that the second packet was not received successfully. It issues a rewind command to roll back the read pointer and schedules the channel so the packet is read out. Since P1 was flushed in (3), RD_PKT_INDEX=0 caused the read pointer to point at P2, as it is now the oldest packet in memory.

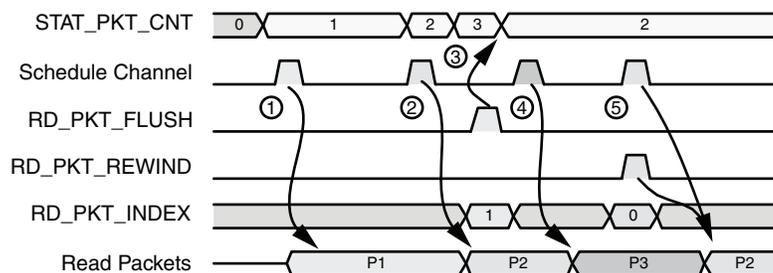


Figure 18: Packet Flow Example

References

1. *LocalLink Interface Specification v2.0*, July 25, 2005.

Support

Please visit [Xilinx Support](http://www.xilinx.com/support) for technical support. Xilinx provides technical support for this LogiCORE product when used as described in product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in unsupported devices, if customized beyond that speci-

fied in the product documentation, or if any changes are made in the sections marked *DO NOT MODIFY*.

Ordering Information

The core may be downloaded from the Xilinx [IP Center](#) for use with the Xilinx CORE Generator. The Xilinx CORE Generator is bundled with ISE Foundation software v9.2i at no additional charge. To order Xilinx software, please contact your local Xilinx sales representative. Information about Xilinx LogiCORE modules is available on the Xilinx [IP Center](#).

Related Information

Xilinx products are not intended for use in life-support appliances, devices, or systems. Use of a Xilinx product in such application without the written consent of the appropriate Xilinx officer is prohibited.

Document Revision History

Date	Version	Revision
1/18/06	1.0	Initial Xilinx release.
1/20/06	1.1	Updates to independent channel sections; minor modifications.
2/17/06	1.2	Updated to version 1.2.
2/28/06	1.3	Updated features list, GUI, and date.
2/15/07	1.4	Updated design tool requirements, core version, formatted signals, updated LocalLink URL.
8/08/07	2.0	Updated supported design tool requirements, core version, ISE to v9.2i, Spartan-3A, Spartan-3AN and Spartan-3A DSP support.

09/29/09 - This is the final publication. No content was changed.