Crossbow ASIC Specification

Revision 12.0

Do not copy or distribute this document

Rob Martin

Reynold Leong

Justin Chueh

Jonah Alben

November 18, 1996

CHAPTER 1 Introduction 9

1.1 Part Name and Number 9
1.2 Related Documents 9
1.3 Terms 10
1.4 Signal Names 12
1.5 Crosstalk Bus Overview 12
1.6 Crossbow ASIC Overview 13
1.6.1 Physical Link Operation 16
1.6.2 Link Level Protocol 21
1.6.3 Crosstalk Packet Flow 23
1.6.4 Overview of ASIC functional blocks 25
1.6.5 Source Link Controller 27
1.6.6 Destination Link Controller 28
1.6.7 Widget 0 29
1.6.8 Crossbar Switch 29
1.7 Features 30

CHAPTER 2 Device Interface 31

2.1 Pin Diagram 31
2.2 Pin Descriptions 33
2.3 Package Pin Assignments 36

CHAPTER 3 Programmers Interface 63

3.1 Overview 63
3.1.1 Error Handling 64
3.1.2 Buffer Flush mechanism 70
3.1.3 Arbitration Control 71
3.2 Address Map 72
3.3 Register Definitions 77
3.3.1 Link (x) Input Buffer Flush 77
3.3.2 Link (x) Control Register 77
3.3.3 Link (x) Status Register 79
3.3.4 Link (x) Auxiliary Status 81
3.3.5 Link (x) Arbitration Upper 82
3.3.6 Link (x) Arbitration Lower 83
3.3.7 Link (x) Reset 84
3.3.8 Crossbow Identification 84
3.3.9 Crossbow Error Command Word 85
3.3.10 Crossbow Error Upper Address 86
3.3.11 Crossbow Error Lower Address 86
3.3.12 Crossbow Interrupt Destination Upper Address 87
3.3.13 Crossbow Interrupt Destination Lower Address 87
3.3.14 Crossbow Arbitration Reload 87
3.3.15 Crossbow Packet Time-out register 88
3.3.16 Crossbow Widget 0 Status 88

3.3.17 Crossbow Widget 0 Control 90
3.3.18 Crossbow LLP Control 90
3.3.19 Crossbow Performance Counter A 91
3.3.20 Crossbow Performance Counter B 92
3.3.21 Crossbow NIC register 92
3.4 Function Errata 93
3.4.1 Crossbow NIC register 93
3.4.2 Link (x) Status Register 93
3.4.3 Link (x) Reset 93
3.4.4 Double Overflow 93
3.4.5 Supported Crosstalk Packet Types 93

CHAPTER 4 Architectural Description 97

4.1 Overview 97
4.1.1 Clocking regimes 99
4.2 Crossbar Switch 99
4.3 Link Controller 100
4.3.1 Source Link Controller 101
4.3.2 Source Synchronous Receiver and Link Level Protocol Receiver 102
4.3.3 Input Packet Buffer 105
4.3.4 Packet Receive Control 106
4.3.5 Free Buffer Stack 106
4.3.6 Crossbar Request Manager 107
4.3.7 Packet Dispatch Control 112
4.3.8 Exception module/Error handling 114
4.3.9 Destination Link Controller 119
4.3.10 Crossbar Connection Arbitration 119
4.3.11 Free buffer issue logic and Crossbar Flow Control 123
4.3.12 Sideband Datapath 124
4.3.13 Link Level Protocol Transmitter and Source Synchronous Driver 124
4.4 Widget 0 127
4.4.1 Internal Register Access 127
4.4.2 Error Processing 128
4.4.3 Interrupt processing 129
4.4.4 Buffer Flush Handling 129
4.4.5 Timers 130
4.5 Reset of Device 132

Figure 1	Write Packet transfer path Widget F to WIdget B	13
Figure 2	Typical System Block Diagram	15
Figure 3	Crossbow Block Diagram	16
Figure 4	STL Driver	17
Figure 5	STL Receiver	17
Figure 6	Crosstalk Bus Connection	19
Figure 7	SSD detail	20
Figure 8	SSR detail	21
Figure 9	Micropacket Format	22
Figure 10	Crosstalk Receive/Transmit Pair	23
Figure 11	Crossbow Block Diagram (link controller detail)	26
Figure 12	Crossbow Pin Out (624 PIN CCGA)	32
Figure 13	Crossbow Block Diagram	98
Figure 14	Crossbar Switch	100
Figure 15	Source Link Controller	102
Figure 16		104
Figure 17	LLPr interface 16 bit mode, with packet error	105
Figure 18	XBAR Request Dispatch	108
Figure 19	Request Manager Detail	109
Figure 20	Packet Dispatch Control	114
Figure 21	Flow diagram packet time-out processing	118
Figure 22	Crossbar Connection Arbitration Diagram	122
Figure 23	General Ring Arbitration Implementation	123
Figure 24	LLPt transmitting	127
Figure 25	Widget 0 Datapath	131

Table 1	Write Request with Response Command Word	24
Table 2	Write Request Quarter Cache Line	25
Table 3	CCGA PIN ASSIGNMENTS	36
Table 4	Link Controller Error Summary	69
Table 5	Widget 0 Error Conditions	69
Table 6	Link(x) Control Register	77
Table 7	Link(x) Status Register	79
Table 8	Link(x) Auxiliary Status	81
Table 9	Link (x) Arbitration Register	83
Table 10	Link (x) Arbitration Register	84
Table 11	Crossbow Identification Register	85
Table 12	Crossbow Error Command Word Register	85
Table 13	Crossbow Error Upper Address Register	86
Table 14	Crossbow Error Lower Address Register	86
Table 15	Crossbow Interrupt Destination Upper Address Register	87
Table 16	Crossbow Interrupt Destination Lower Address Register	87
Table 17	Crossbow Arbitration Reload Register	88
Table 18	Crossbow Packet Time-out register	88
Table 19	Crossbow Widget 0 Status Register	89
Table 20	Crossbow Widget 0 Control Register	90
Table 21	Crossbow LLP Control Register	90
Table 22	Crossbow Performance Counter A	91
Table 23	Crossbow Performance Counter B	92
Table 24	Crossbow NIC register	92
Table 25	Supported Crosstalk Packet types	94
Table 26	Xbar Request Queue Entry	111

CHAPTER 1

Introduction

1.1 Part Name and Number

Part Name: Crossbow

SGI Part Number: TBD

Vendor: : IBM

Vendor Part Number:50H6512

Process Design Technology: CMOS5L

Package: 624 pin CCGA

Estimated Gate Count: 600k

1.2 Related Documents

- *Godzilla* Architecture Specification
  by Karim Abdalla, Steven Miller, James Tornes, Ross Werner, Daniel Yau
- ***Crosstalk*** System Interconnect Specification
  by Steven Miller & James Tornes

- *DSM* Router Chip and Fixture
  by Mike Galles and Ron Nikel

1.3 Terms

It is assumed the reader has read the Crosstalk Bus specification and has thorough knowledge of the Crosstalk Bus protocol, packet formats, and electrical characteristics. The following terms are discussed in the Crosstalk Bus specification and used throughout this document:

WIDGET Any ASIC connected to the Crossbow ASIC with *Crosstalk* interconnect links.

TARGET The destination widget of a Crosstalk packet.

INITIATOR The source widget of a packet.

LINK The physical connection from the Crossbow ASIC to any widget (device).

CROSSTALK PACKET

The minimum unit of data transfer on the Crosstalk bus. Data transfer on the Crosstalk Bus occurs only in several fixed data packet sizes ranging from a double word value (8 bytes), to a full cache line (128 bytes). plus header.

CROSSBAR Central routing agent for packets. (internal interconnect in Crossbow ASIC)

The following terms are specific to the Crossbow document:

SOURCE LINK

The portion of a link which is used to transmit data from a widget to the Crossbow. (please refer to Figure 2) Also referred to as an uplink.

DESTINATION LINK

The portion of a link which is used to transmit data from the Crossbow to a widget. (please refer to Figure 2) Also referred to as a downlink.

UPLINK,DOWNLINK

For a given Crossbow port data is transferred to the crossbow from the widget on the uplink (source link). Data flow control information is sent back to initiating widget on the downlink. (please refer to Figure 2).

CROSSBAR PORT

A connection to the internal crossbar interconnect. A source crossbar port receives data form a source link controller. Conversely, a destination crossbar port transmits data to a destination link controller.

LINK CONTROLLER

The portion of the Crossbow ASIC which controls movement of crosstalk packets from a crosstalk link to a crossbar port or controls the movement of a crosstalk packet from the crossbar port to a crosstalk link. A link controller which is connected between the source link and the crossbar is known as a source link controller. Conversely, a link controller which is connected between the crossbar and the destination link is known as a destination link controller.

BAUD RATE

Indicates the number of symbols transmitted per second. As in the case of the Crosstalk bus a 16 bit crosstalk input link operating at 400 Mbaud would be transferring at a peak rate of 800 Mbyte/second.

1.4 Signal Names

Signal names that end in _N denote signals that are active low. The term "assert" means to drive to an active state, "de-assert" means to drive to an inactive state. Buses are denoted by **BUSNAME(MSB:LSB).**

1.5 Crosstalk Bus Overview

The Crosstalk Bus is a high speed, packet switched bus. All crosstalk connections are point to point unidirectional connections to provide a controlled impedance transmission line. Data is transmitted source synchronous (the clock is sent with the data) at data rates of up to 800 Mbytes/sec for 16 bit links and up to 400 Mbytes/sec for 8 bit links. A link is the term used for the actual crosstalk physical connection. Crosstalk devices communicate with each other by sending data packets through a crossbar switch (Reference Figure 1.). Packets of data are transferred between crosstalk devices ("widgets") in specific quanta of a double word (1 - 8 bytes), a quarter cache line (one to 32 bytes), or a full cache line (128 bytes). The Crosstalk Bus protocol uses split transactions. Crosstalk bus transactions are started by an initiator widget which sends a request packet (i.e. read command or write command plus data) to a target widget which then replies with a response packet (i.e. read data or optionally a write acknowledgment).

The crossbow is a switched packet router. A widget wishing to transfer a packet to another widget will transfer the packet to its associated input packet buffer in the crossbow. Once the packet routing information has been correctly received, arbitration begins for the destination port resource. The crosstalk packet will be stored until the source link controller can successfully obtain access to the destination port resource. As soon as access is granted, the packet is transferred through the crossbar to the destination widget.Please reference Figure 2.

This topology provides very high bandwidth physical connections and the added ability to perform multiple concurrent data transfers between widgets. Another feature of this bus architecture is to provide guaranteed bandwidth and latencies for real time widgets. The bus arbitration algorithm is designed to provide this.

As described in the Crosstalk Bus specification, arbitration among initiator packets wishing to route data to the same destination port is accomplished via a programmable weighted round-robin scheme which guarantees a minimum bandwidth for real time (Guaranteed Bandwidth Ring) widgets and distributes remaining bandwidth to packets from non-real time (Remainder Ring) widgets.

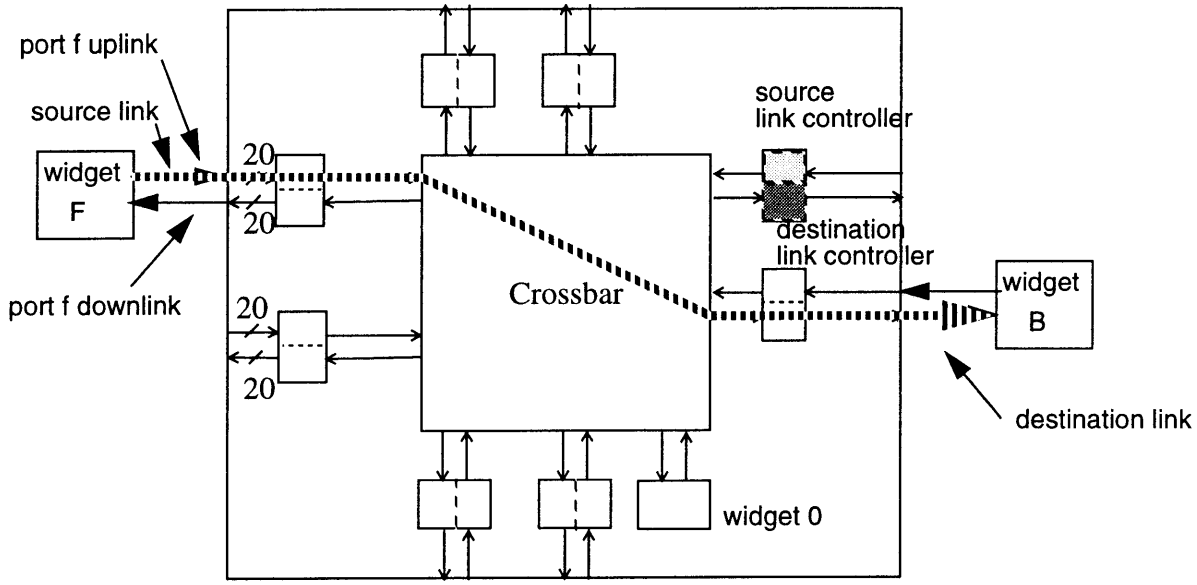


Figure 1 Write Packet transfer path Widget F to WIdget B

1.6 Crossbow ASIC Overview

The Crossbow ASIC provides the function of a crossbar switch which has the ability to connect eight 16 bit wide crosstalk ports. Each port has the ability to operate in either a 16 bit or 8 bit link.Each widget uses two crosstalk links, one for transmit (source link) and one for receive (desti-

nation link). The Crossbow behaves as a packet switched router, which routes packets received from an initiator widget and forwards them through the crossbar switch to a target widget. In the event that there is contention for the target from multiple initiators, the Crossbow provides buffering for up to four crosstalk packets per initiator link port. An example of a Crossbow in a typical system configuration is shown in Figure 2.

A representation of the Crossbow's major functional blocks are shown in Figure 2. These blocks include link controllers, widget 0, and the crossbar. The link controllers handle all crosstalk packet transfers on the Crosstalk link port between a widget and the Crossbow. The link controllers are comprised of two sub-blocks; the source link controller and the destination link controller. The source link controller controls all crosstalk packet movement from a source link to the internal crossbar switch. Conversely a destination link controller controls all crosstalk packet movement from the crossbar to the destination link. The crossbar is a 9 port crossbar switch which connects the source link controllers to destination link controllers. Additionally, one port on the crossbar is reserved for Widget 0. Widget 0 contains the interface to all Crossbow internal registers and also functions in conjunction with the link controllers during error handling.

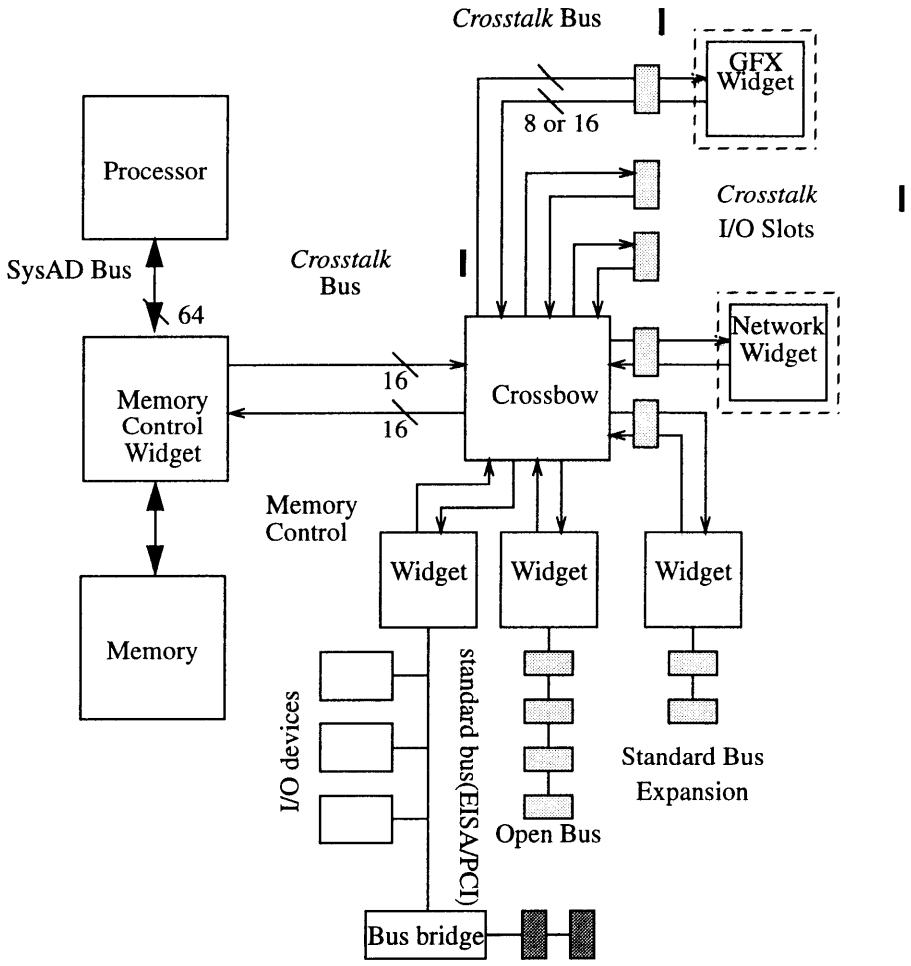


Figure 2 Typical System Block Diagram

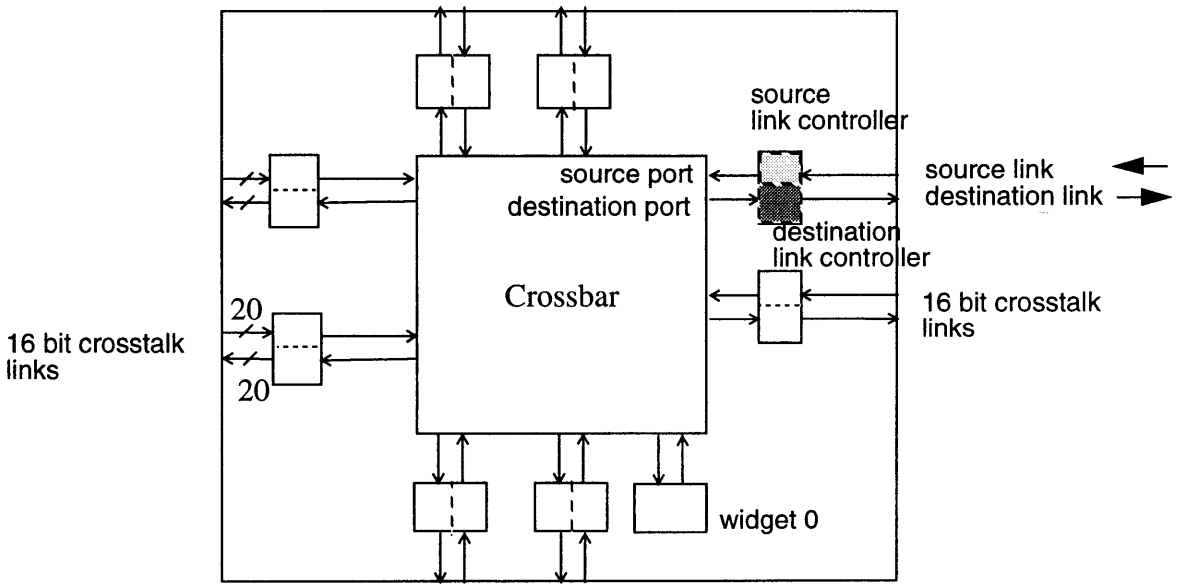


Figure 3 Crossbow Block Diagram

1.6.1 Physical Link Operation

For a complete and detailed specification of the physical link please refer to: DSM Router Chip and Fixture by Mike Galles and Ron Nikel

Data is transmitted on the physical links at a data rate of 400 Mbyte/sec for 8 bit links and at 800 Mbyte/sec for 16 bit links. These data rates are achieved by transmitting the data via a low voltage swing driver technology known as STL(SGI Transistor Logic). The STL circuit consists of a driver and a receiver. The driver is an open drain driver with low output resistance. The receiver is a differential receiver which employs a single or double current mirror to increase the gain of the receiver.

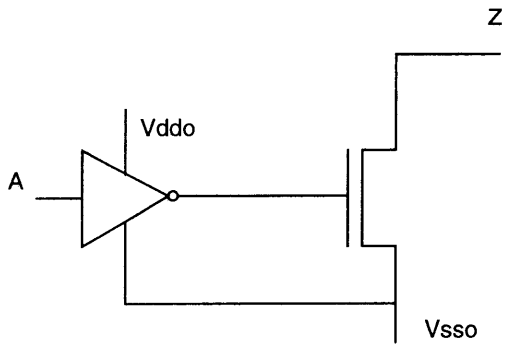


Figure 4 STL Driver

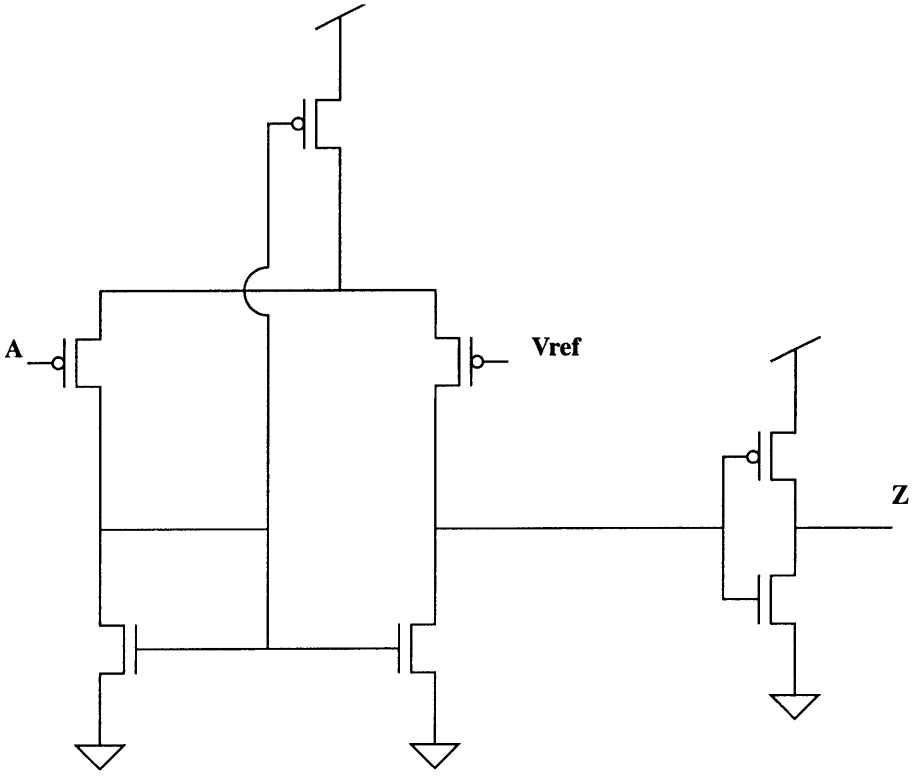


Figure 5 STL Receiver

Each Crosstalk connection consists of a pair of unidirectional links. A 16 bit link consists of

16	Data signals
4	Check bit, sequence number, and sideband signals
1	Data Ready framing signal
2	Differential clock signals
23	Total signals per unidirectional 16 bit data link

An 8 bit link consists of

8	Data signals
2	Check bit, sequence number, and sideband signals
1	Data Ready framing signal
2	Differential clock signals
13	Total signals per unidirectional 8 bit data link

In normal operation data may be simultaneously transmitted on both links hence, the cross-sectional bandwidth of a Crosstalk connection is 1600 Mbyte/sec for 16 bit connections and 800 Mbyte/sec for 8 bit connections. For the sake of discussion we will arbitrarily define the link which carries data from the widget to the Crossbow as the source link, and conversely we shall define the link which carries data from the Crossbow to the widget as the destination link.

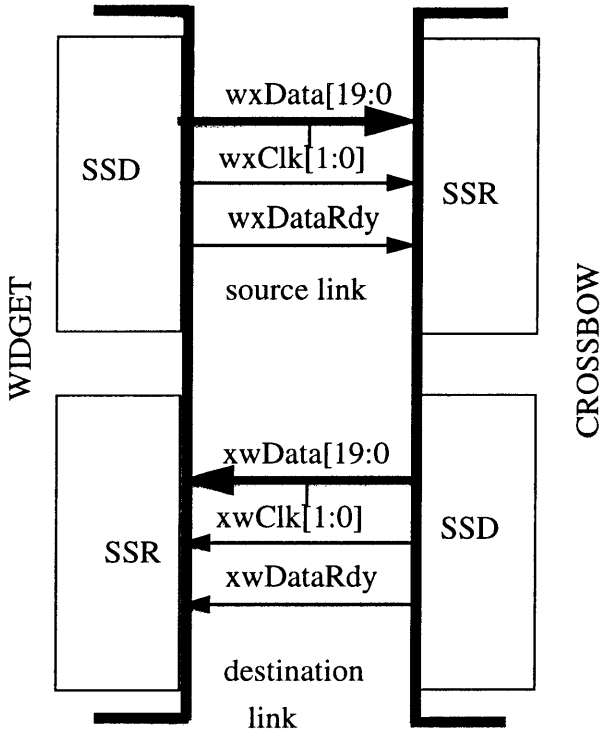
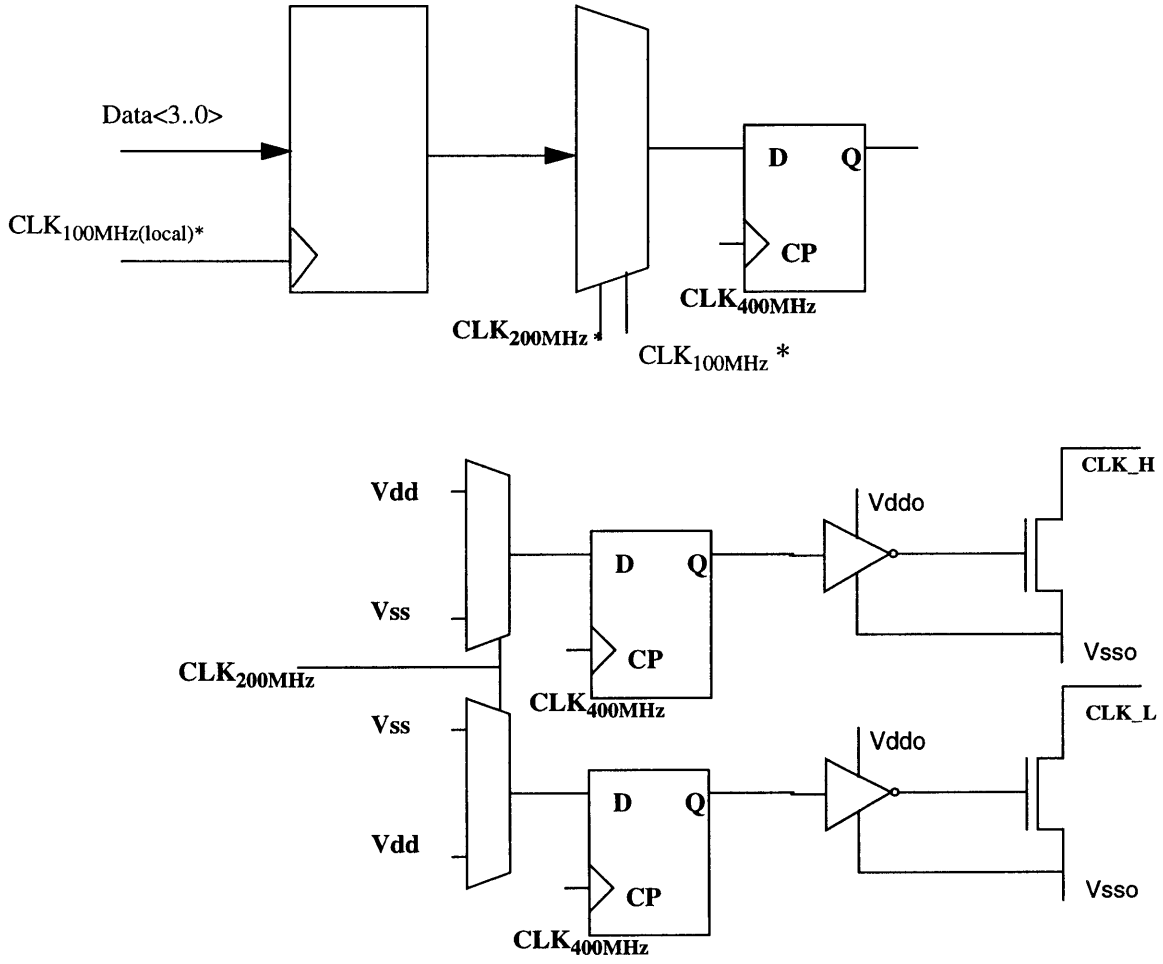


Figure 6 Crosstalk Bus Connection

The crossbow utilizes a specially designed macro known as the source synchronous driver (SSD) which converts a 64 or 32 bit wide CMOS data bus operating at 100 Mhz to a 16 or 8 bit STL data bus operating at an effective clock rate of 400 Mhz. The transmit clock is sent differentially with the data and used by receiver to clock the received data thus eliminating problems of clock skews between devices.

A macro known as the source synchronous receiver (SSR) receives the narrow, 400 mhz STL crosstalk data stream and the transmitted clock.It uses the clock to convert the stream back into the wide, 100 mhz CMOS data stream. Hence, the majority of the Crossbow logic is isolated from the high speed links and operates at the 100 mhz core clock frequency.



*clocks generated internal to SSD from CLK_{400MHz}

Figure 7 SSD detail

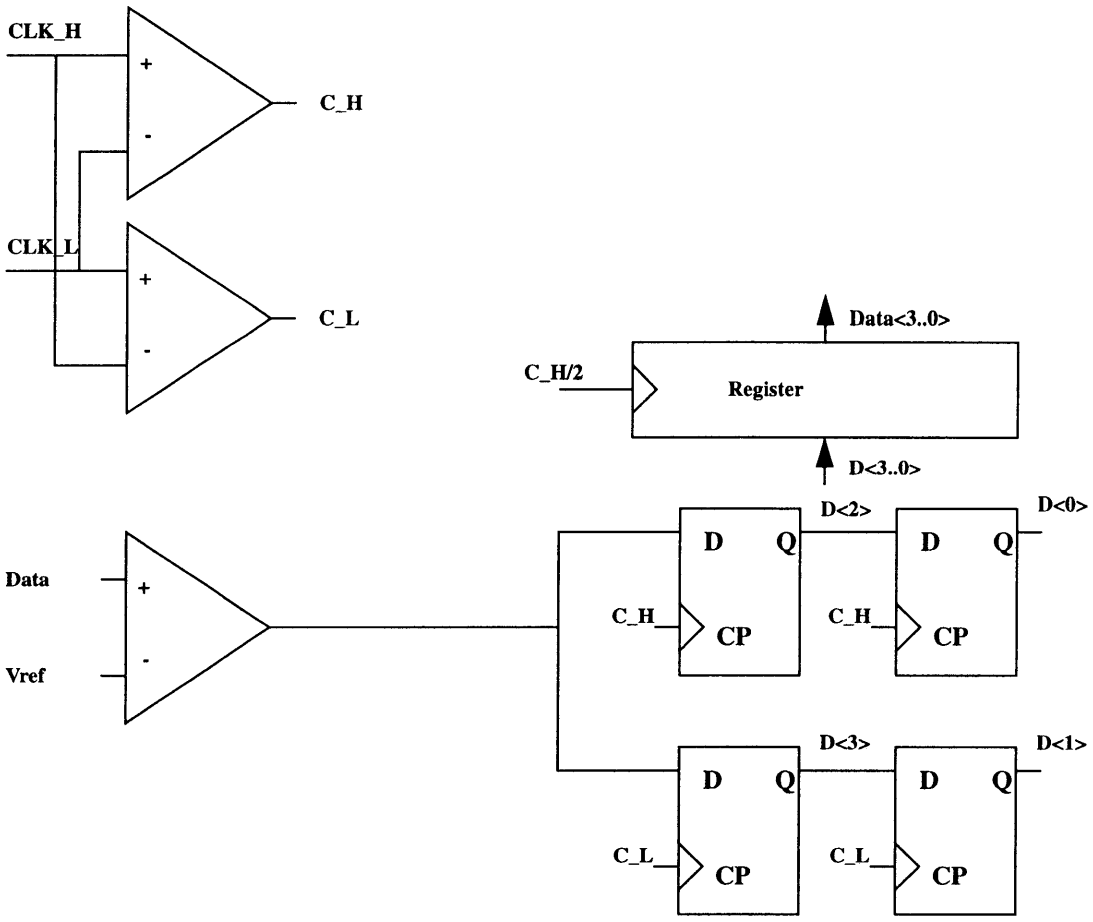


Figure 8 SSR detail

1.6.2 Link Level Protocol

To ensure error free transmission of data on the link, data is transferred in micropackets. Each micropacket contains 128 bits of data, 16 bits of check bits, 4 bits of transmit sequence number, 4 bits of receive sequence number, and 8 bits of sideband information.

When a micropacket is received the CRC is computed and checked against the check bits sent with the data. If the data is error free the trans-

mit sequence number is checked to ensure it matches the sequence number of the next expected packet. This ensures that micropackets are transmitted and received sequentially and no whole micropackets are dropped.

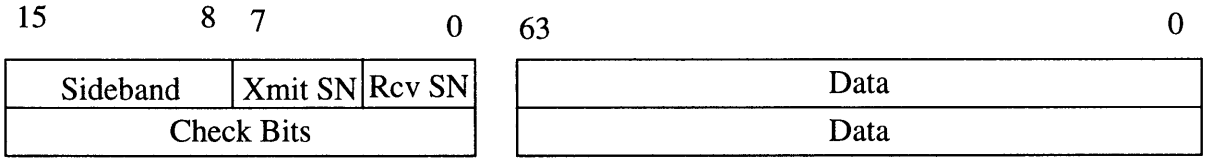


Figure 9 Micropacket Format

The link protocol consists of a sliding window protocol with a 'go back-N' retry policy. As the sender transmits micropackets to a destination it also buffers the data locally. When the data is correctly received at the destination an acknowledgment is sent back to the sender from the receiver. This acknowledgment consists of the receiver returning the sequence number of the last correctly received micropacket. In the case of the widget transmitting to the Crossbow when a packet is correctly received on the source link, the sequence number is placed in the receive sequence number field of a micropacket being transmitted on the destination link. When the sender receives this acknowledgment it will then free up all locations in the transmit buffer up to that sequence number. If the received packet is in error the receiver will not send the acknowledgment and cease to pass micropackets to the crossbow core. After a small time-out period if the sender does not receive an acknowledgment it will re-transmit its entire transmit buffer. Retry attempts will occur until the packet is acknowledged or a programmable maximum retry limit is reached. Conversely, the same sequence occurs for micropackets being transferred from Crossbow to widget. Data flows on the destination ink and receive sequence number acknowledgments flow back on the source link.

This protocol is maintained by two block which work in conjunction the SSD and SSR. These blocks are referred to as the Link Level Protocol Transmitter (LLPt) and the Link Level Protocol Receiver (LLPr). These blocks isolate all upper levels of logic in the Crossbow from the link level

protocol. The chip core simply provides data and sideband information to the LLPt and monitors a few status signals and receives data and sideband information along with a few status signals. Additionally during the reset sequence the LLP's will negotiate between 16 bit and 8 bit link mode operation and come up operating in the least common denominator of the two modes automatically.

For further details on the LLP blocks and the Link Level protocol please refer to: *Crosstalk System Interconnect Specification*

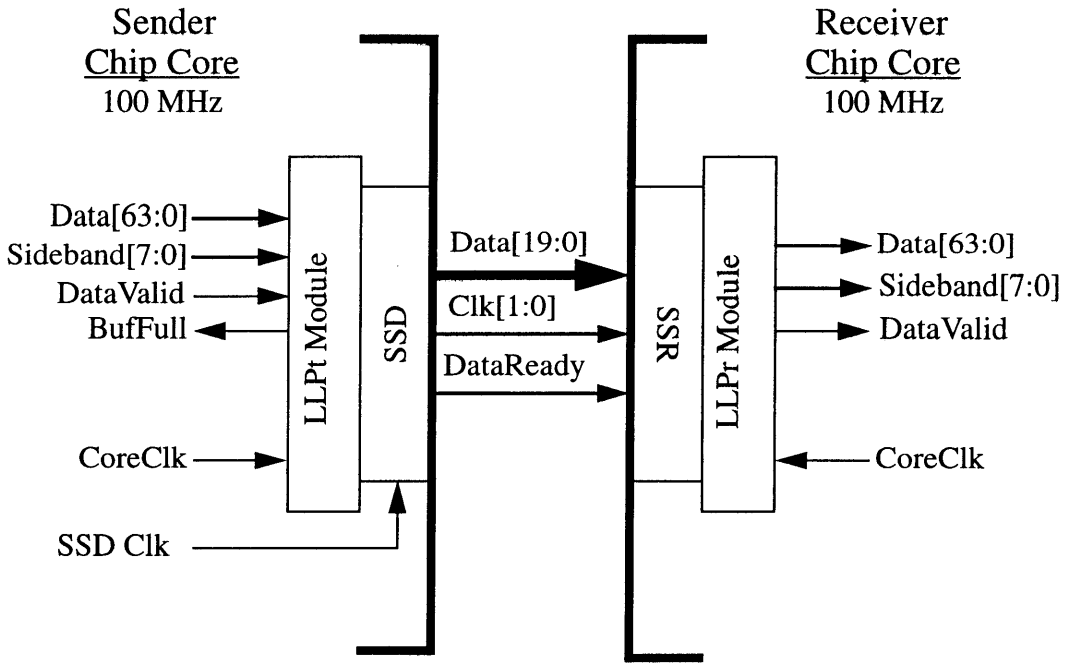


Figure 10 Crosstalk Receive/Transmit Pair

1.6.3 Crosstalk Packet Flow

The next level of protocol is the Crosstalk packet protocol which used to move data between widgets. The primary function of the Crossbow core is to support this protocol. As previously mentioned the Link Level protocol is mostly transparent to crossbow core. Crosstalk packets are de-

composed by the LLPt into a series of micropackets and reconstructed by the LLPr.

A Crosstalk packet consists of a command word and may contain a 48 bit address, a 16 bit remote map field and/or a data field consisting of 8, 32, or 128 bytes. The command word is used by the Crossbow to determine the destination, type, and priority of the packet.

Bits	Definition
31-28	Destination ID Number (DIDN)
27-24	Source ID Number (SIDN)
23-19	Transaction NUMber (TNUM)
18-15	Write Request w/ Response (PACTYP)
14	Coherent Transaction (CT)
13-12	Packet Data Size
11	Guaranteed Bandwidth Ring enable (GBR)
10	VBP Message (VBPM)
9	Error Occurred(ERROR)
8	Barrier Operation(BO)
7-0	*Reserved*

Table 1 **Write Request with Response Command Word**

63:56	55:48	47:40	39:32	31:24	23:16	15:8	7:0
Command Word				Remote Map Field		Address 48:32	
Address 31:0				Data Enables 31:0			
Data 63:0 Lowest Address Data							
Data 63:0							
Data 63:0							
Data 63:0							

Table 2 **Write Request Quarter Cache Line**

As a requirement of the Crosstalk packet protocol, the widgets and the crossbow adhere to rules concerning the number and types of packages that may be transferred. These rules are to avoid deadlock conditions inherent in crossbar configurations and to ensure the proper functioning of the guaranteed bandwidth arbitration scheme. These rules are described in great detail in the *Crosstalk Bus Specification, Chapter 4 Link Behavior.* It is assumed that the reader is familiar with the protocol as the following brief description of Crossbow core blocks is only concerned with the implementation of the protocol as it relates solely to the Crossbow.

1.6.4 Overview of ASIC functional blocks

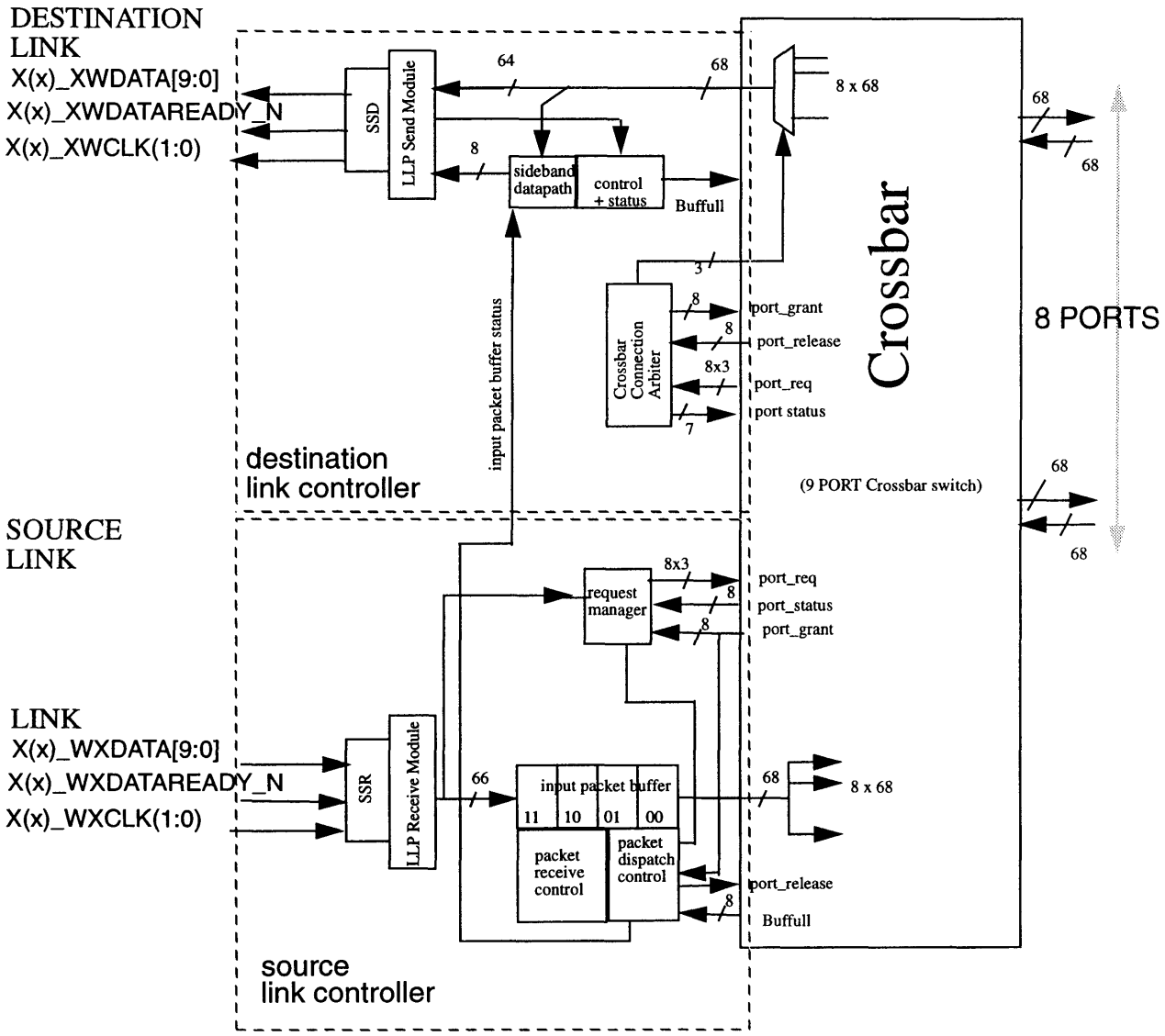


Figure 11 Crossbow Block Diagram (link controller detail)

1.6.5 Source Link Controller

The source link controller handles all traffic between the source link and the crossbar. As micropackets are transferred on the uplink and the data is received by the SSR/LLPr blocks, stripped of all Link Protocol information and passed to the next stages of logic.

The packet receive control logic scans the sideband data for "a start of packet" code. If this is received the control logic will begin filling one of 4 input packet buffers. The input packet buffer serves two purposes, it provides a place to park a crosstalk packet when the packet destination is busy and it provides for rate matching between the data stream coming from LLP and the crossbar. The Crossbow is architected in such a way that the crossbar transfers data at 800 Mbytes/sec; however, the LLP's provide data at either 400 Mbytes/sec or 800 Mbytes/sec depending on whether they are associated with 8 bit or 16 bit links respectively.

The packet receive logic will also write pertinent information from the command word portions of the packet and place it in the crossbar request queue which is located in the request manager. The information written into the crossbar request queue defines the Crosstalk packet's destination, priority, and type (request or response). It is the request managers job to determine which packets are eligible for arbitration and from among the packets that are eligible for arbitration select the packet which has the highest priority and arbitrate for a connection to that packets destination crossbar port. While the crosstalk packet is being received and put into one of the input packet buffers, the request manager will check the status of the destination port (port_status) and the priority of the packets in the queue to determine which of the packets in the input packet buffer has the highest priority. Details of the request manager selection algorithm are left for the architectural description portions of this document.

If the packet which has just entered the queue has the highest priority of all packets currently in the queue, it will advance to the front of the queue and enter the crossbar connection arbitration phase. If there are higher priority crossbar connection requests already in the queue, it will wait until they are serviced.

During the crossbar arbitration phase the request manager sends a crossbar connection request (port_req) to the destination link controller associated with the Crosstalk packet's destination. The request manager then alerts the packet dispatch control that a crossbar connection arbitration is in progress.

When the Crosstalk packet wins arbitration a port_grant signal will be sent back from the destination link controller to the requesting source.

The dispatch controller will begin transferring the packet out of the input packet buffer and into the crossbar. The request manager will then retire the entry from the request queue. As the dispatch controller is transferring the packet it monitors the destination's buffer full flag (Buffull). This flag indicates to the source that the destination can currently accept no more data.

When the transfer of the packet nears completion the dispatch controller will release control of the destination port by asserting port_release. This frees the crossbar connection arbiter to start a new arbitration phase and establish a new crossbar connection.

Once the dispatch controller has finished transferring the packet, a "credit" to the initiator widget via the downlink sideband data stream to indicate an input packet buffer slot has been freed.

1.6.6 Destination Link Controller

The Destination Link Controller handles all packet traffic between the crossbar and the downlink. In addition it controls all access to the destination crossbar port via the crosstalk connection arbiter.

The crossbar connection arbiter is responsible for selecting from among all the source link controllers wishing to establish a crossbar connection to its destination crossbar port. The arbiter scans all current port_req signal and sends a port_grant signal back to winning link source controller. It then updates the status of the destination port (port_status). The arbitration algorithm consists of a weighted round robin scheme and is discussed in detail in the Architectural Description chapter.

As the port_grant acknowledge signal is sent, the crossbar connection arbiter also schedules switching the crossbar mux to coincide with first data arriving at the destination crossbar port from the source link controller. A new arbitration cycle will begin when the arbiter receives a port_release signal from the source link controller.

Data is streamed directly from the crossbar to the LLP. The LLP contains a internal buffer known as the retry buffer whose use is twofold. A portion of the buffer in the LLP is used for supporting the LLP sliding window protocol. As data is transferred over the link it also written into the LLP's retry buffer. If receipt of the data is acknowledged by the receiver, the buffer locations are freed. If an acknowledgment is not received the data is retransmitted. In normal operation with packets being received

correctly, only a portion of the buffer is used to support this protocol. The Crossbow uses the remaining locations in the buffer to rate match between the 800 Mbyte/sec crossbar and the 400 mbyte/sec 8 bit links. This buffering allows a 16 bit source link controller or an 8 bit source link controller that has accumulated a full crosstalk packet, to transfer at the full data rate to an 8 bit destination link and then go service another destination while the transfer on the link is occurring.

1.6.7 Widget 0

All access to internal registers in the crossbow is via widget 0. Widgets wishing to modify crossbow registers should direct their request packets to the widget 0 destination. Widget 0 behaves much the same as any set of link controllers. Source link controllers wishing to connect to widget 0 send a crossbar connection request to widget 0. The widget 0 crossbar connection arbiter will send an acknowledgment and then receive the packet. After widget 0 has received the packet it will perform the necessary operations on the Crossbow registers. If a response is required widget 0 will then form a response packet and transfer it back to the initiating widget via the crossbar.

1.6.8 Crossbar Switch

The central connection through which all link controllers pass data is the crossbar switch. The crossbar switch consists of 9 ports. Each crossbar connection consists of one 68 bit source port and one 68 bit destination port which utilizes a 68 bit wide by 8 mux to establish a connection to the other source ports. The 68 bits consist of 64 bits of data, 3 bits of sideband information and 1 bit of control (bit indicates valid data). The destination link controller's crossbar connection arbiter controls the mux. All data is registered in and registered out of the crossbar, hence it is a "one hop" interconnect.

1.7 Features

- Maximum bisectional transfer rate of 800 Mbyte/sec for 8 bit widgets (bisectional bandwidth implies the source link and destination link are operating concurrently)
- Maximum bisectional transfer rate of 1600 Mbyte/sec for 16 bit widgets
- Support for 8 16 bit CROSSTALK links
- 16 bit links may be configured to run in 8 bit mode
- Crossbar topology allows concurrent transfers between widgets
- Peer to Peer Communication
- Real Time Bandwidth Guarantees between Peers
- Programmable Bandwidth Control

CHAPTER 2

Device Interface

2.1 Pin Diagram

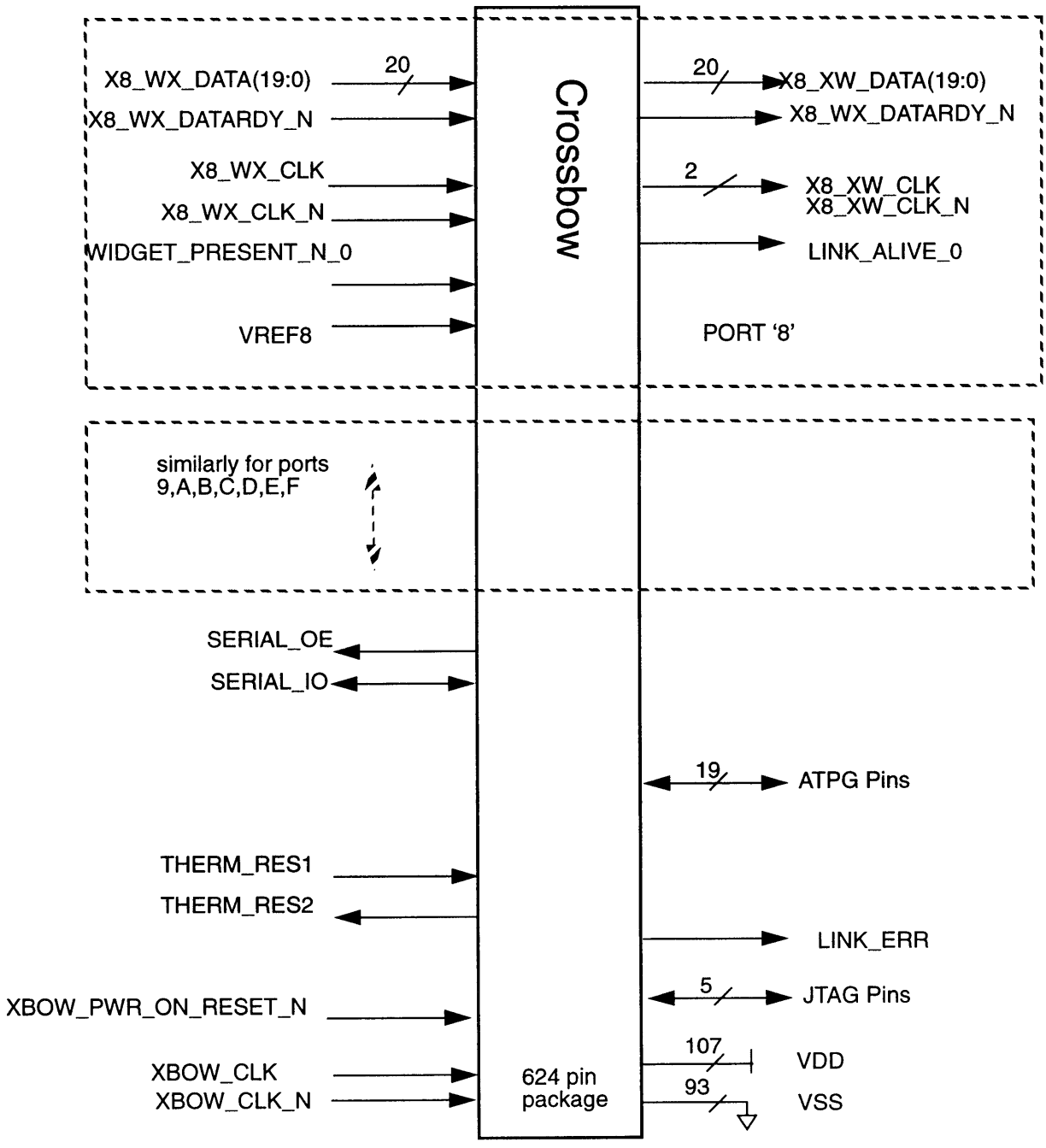


Figure 12 Crossbow Pin Out (624 PIN CCGA)

2.2 Pin Descriptions

The following tables list each Crossbow pin, its assertion level, direction (I, O) switching levels (LVTTL, STL, DIFFSTL or CMOS) and provide a brief functional description.

Pin Name	Type	Levels	# of pins	Function
X(8,9,A,B,C,D,E,F)_WX_CLK	I	DIFF STL	8	200 MHZ STL level clock from Widget connected to +ve input of differential STL receiver
X(8,9,A,B,C,D,E,F)_WX_CLK _N	I	DIFF STL	8	200 MHZ STL level clock from Widget connected to -ve input of differential STL receiver
X(8,9,A,B,C,D,E,F)_WX_DAT A(19:0)	I	STL	160	20 bit input data for 16 bit links
X(8,9,A,B,C,D,E,F)_WX_DAT ARDY_N	I	STL	8	data ready indicating valid data is being transported on the link, also used for data framing.
X(8,9,A,B,C,D,E,F)_XW_CLK	O	STL	8	200 MHZ output clock
X(8,9,A,B,C,D,E,F)_XW_CLK _N	O	STL	8	200 MHZ inverted output clock
X(8,9,A,B,C,D,E,F)_XW_DAT A(19:0)	O	STL	160	Output data for 16 bit ports
X(8,9,A,B,C,D,E,F)_XW_DAT ARDY_N	O	STL	8	data ready indicating valid data is being transported on the link, also used for data framing
LINK_ALIVE(0,1,2,3,4,5,6,7)	O	LVTTL	8	Indicates that the link has correctly initailized and is functional (0 is associated with link 8, 1 is associated with
XBOW_CLK	I	DIFF_STL	1	External 400 mhz differential transmit clock input (+)
XBOW_CLK_N	I	DIFF_STL	1	External 400 mhz differential transmit clock input (-)
W_PRESENT_N(0,1,2,3,4,5, 6,7)	I	LVTTL	8	When pulled down indicates to internal logic that a widget is physically attached to this port. Also enables DC power to the port's STL recievers.
XBOW_PWR_ON_RESET_N	I	LVTTL	1	Crossbow Reset

Pin Name	Type	Levels	# of pins	Function
SERIAL_IO	I/O	LVTTL	1	MicroLan Interface
SERIAL_OE	O	LVTTL	1	MicroLan Interface(indicates port is driving)
LINK_ERR	O	LVTTL	1	Error on any one of the ports (OR of the error bits from all ports for the bits that are not masked) This pin will be used for debugging purposes
VREF(8,9,A,B,C,E,F)	I	N/A	8	DC 1.0v for STL input Differential input receiver
TDI	I	LVTTL	1	JTAG interface
TMS	I	LVTTL	1	JTAG interface
TCK	I	LVTTL	1	JTAG interface
TRSTN	I	LVTTL	1	JTAG interface
TDO	O	LVTTL	1	JTAG interface
LSSD_RI_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_DI1_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_DI2_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_X8_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_X9_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_IOTEST_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_SCAN_GATE_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_RAM_BIST_OUT	O	LVTTL	1	Pin required for IBM ATPG
LSSD_A_CLOCK_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 0
LSSD_B_CLOCK_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_B2_CLOCK_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1

Pin Name	Type	Levels	# of pins	Function
LSSD_B3_CLOCK_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_B4_CLOCK_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_C_CLOCK_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_GRAM_C_CLOCK_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_GRAM_ABIST_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_TAP_C2_CLOCK_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_TESTMODE_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
LSSD_C2_L	I	LVTTL	1	Pin required for IBM ATPG Functional Value = 1
THERM_RES1	I	N/A	1	Thermal resistor.
THERM_RES2	O	N/A	1	Thermal resistor.
VDD	N/A	107	TBD	+3V supply
GND	N/A	93	TBD	0V
TOTAL PINS			616	t

2.3 Package Pin Assignments

SIGNAL NAME	PIN ASSIGNMENT
GND	A03
GND	A11
GND	A23
GND	A25
GND	AA04
GND	AA20
GND	AB09
GND	AB13
GND	AB21
GND	AC01
GND	AC03
GND	AC07
GND	AC11
GND	AC15
GND	AC19
GND	AC23
GND	AC25
GND	AD02

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
GND	AD08
GND	AD24
GND	AE01
GND	AE03
GND	AE23
GND	AE25
GND	B02
GND	B24
GND	C01
GND	C03
GND	C07
GND	C11
GND	C13
GND	C15
GND	C18
GND	C19
GND	C23
GND	C25
GND	D13
GND	E12
GND	F08
GND	F18
GND	G01
GND	G03

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
GND	G07
GND	G11
GND	G14
GND	G15
GND	G17
GND	G19
GND	G23
GND	H13
GND	H18
GND	H20
GND	J08
GND	J12
GND	J22
GND	K01
GND	L03
GND	L07
GND	L09
GND	L11
GND	L15
GND	L17
GND	L19
GND	L23
GND	M03
GND	N04

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
GND	N05
GND	N13
GND	N19
GND	N22
GND	N23
GND	R03
GND	R07
GND	R11
GND	R15
GND	R19
GND	R23
GND	T05
GND	T18
GND	U13
GND	U19
GND	V16
GND	V18
GND	V20
GND	V23
GND	W03
GND	W07
GND	W09
GND	W11
GND	W13

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
GND	W15
GND	W19
GND	W23
LSSD_A_CLOCK_L	Y14
LSSD_B2_CLOCK_L	AB12
LSSD_B3_CLOCK_L	AD10
LSSD_B4_CLOCK_L	P08
LSSD_B_CLOCK_L	T14
LSSD_C2_L	K04
LSSD_C_CLOCK_L	T04
LSSD_DI1_L	K20
LSSD_DI2_L	M18
LSSD_GRAM_ABIST_L	K02
LSSD_GRAM_C_CLOCK_L	M08
LSSD_IOTEST_L	P18
LSSD_RAM_BIST_OUT	Y16
LSSD_RI_L	M22
LSSD_SCAN_GATE_L	AB16
LSSD_TAP_C2_CLOCK_L	K06
LSSD_TESTMODE_N	F10
LSSD_X8_L	K24
LSSD_X9_L	T20
TCK	B16
TDI	F12

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
TDO	D14
THERM_RES1	Y07
THERM_RES2	Y06
TMS	F14
TRSTN	H14
VDD	A02
VDD	A18
VDD	A24
VDD	AA02
VDD	AA05
VDD	AA09
VDD	AA17
VDD	AA21
VDD	AA24
VDD	AB20
VDD	AC08
VDD	AC13
VDD	AC22
VDD	AD01
VDD	AD05
VDD	AD09
VDD	AD13
VDD	AD17
VDD	AD21

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
VDD	AD25
VDD	AE02
VDD	AE07
VDD	AE13
VDD	AE24
VDD	B01
VDD	B05
VDD	B09
VDD	B13
VDD	B17
VDD	B21
VDD	B25
VDD	D01
VDD	E02
VDD	E05
VDD	E09
VDD	E16
VDD	E17
VDD	E21
VDD	E24
VDD	F13
VDD	F17
VDD	G02
VDD	G09

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
VDD	G12
VDD	G13
VDD	G18
VDD	H16
VDD	H17
VDD	H19
VDD	J02
VDD	J05
VDD	J09
VDD	J13
VDD	J17
VDD	J21
VDD	J24
VDD	K08
VDD	K13
VDD	L06
VDD	M12
VDD	M14
VDD	M17
VDD	M19
VDD	N01
VDD	N02
VDD	N03
VDD	N06

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
VDD	N07
VDD	N09
VDD	N10
VDD	N12
VDD	N16
VDD	N17
VDD	N18
VDD	N20
VDD	N21
VDD	N24
VDD	P07
VDD	P09
VDD	P12
VDD	P13
VDD	P14
VDD	P19
VDD	R06
VDD	R08
VDD	T08
VDD	T13
VDD	U02
VDD	U04
VDD	U05
VDD	U09

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
VDD	U17
VDD	U18
VDD	U20
VDD	U21
VDD	U24
VDD	V08
VDD	V10
VDD	V13
VDD	V15
VDD	V19
VDD	W02
VDD	W16
VDD	W20
VDD	Y04
VDD	Y13
VDD	Y23
Vref8	J18
Vref9	R18
VrefA	V17
VrefB	U12
VrefC	U08
VrefD	M07
VrefE	H09
VrefF	H15

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
link_alive_0	K22
link_alive_1	P20
link_alive_2	AD16
link_alive_3	Y12
link_alive_4	P10
link_alive_5	T06
link_alive_6	B10
link_alive_7	F16
link_err	N08
serial_io	Y20
serial_oe	Y21
widget_present_n_0	M20
widget_present_n_1	T22
widget_present_n_2	V14
widget_present_n_3	V12
widget_present_n_4	P06
widget_present_n_5	M06
widget_present_n_6	D10
widget_present_n_7	D16
x8_wx_clk	K17
x8_wx_clk_n	K18
x8_wx_data_0	G24
x8_wx_data_1	H22
x8_wx_data_10	F22

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
x8_wx_data_11	G20
x8_wx_data_12	H21
x8_wx_data_13	G21
x8_wx_data_14	D25
x8_wx_data_15	E23
x8_wx_data_16	D24
x8_wx_data_17	E22
x8_wx_data_18	C24
x8_wx_data_19	D23
x8_wx_data_2	K16
x8_wx_data_3	J19
x8_wx_data_4	F25
x8_wx_data_5	G22
x8_wx_data_6	F21
x8_wx_data_7	F23
x8_wx_data_8	J20
x8_wx_data_9	F24
x8_wx_datardy_n	E25
x8_xw_clk	E19
x8_xw_clk_n	D18
x8_xw_data_0	C22
x8_xw_data_1	D22
x8_xw_data_10	A22
x8_xw_data_11	D19

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
x8_xw_data_12	E18
x8_xw_data_13	C20
x8_xw_data_14	A21
x8_xw_data_15	B19
x8_xw_data_16	B18
x8_xw_data_17	B20
x8_xw_data_18	A20
x8_xw_data_19	A19
x8_xw_data_2	D21
x8_xw_data_3	B23
x8_xw_data_4	B22
x8_xw_data_5	E20
x8_xw_data_6	F20
x8_xw_data_7	J16
x8_xw_data_8	C21
x8_xw_data_9	F19
x8_xw_datardy_n	D20
x9_wx_clk	P17
x9_wx_clk_n	R17
x9_wx_data_0	T21
x9_wx_data_1	R20
x9_wx_data_10	P22
x9_wx_data_11	P16
x9_wx_data_12	T23

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
x9_wx_data_13	P21
x9_wx_data_14	R25
x9_wx_data_15	P24
x9_wx_data_16	T24
x9_wx_data_17	P25
x9_wx_data_18	N15
x9_wx_data_19	N25
x9_wx_data_2	P15
x9_wx_data_3	U25
x9_wx_data_4	T25
x9_wx_data_5	V22
x9_wx_data_6	R21
x9_wx_data_7	R24
x9_wx_data_8	R22
x9_wx_data_9	N14
x9_wx_datardy_n	P23
x9_xw_clk	L21
x9_xw_clk_n	L18
x9_xw_data_0	M25
x9_xw_data_1	M15
x9_xw_data_10	H24
x9_xw_data_11	L22
x9_xw_data_12	J25
x9_xw_data_13	L20

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
x9_xw_data_14	L16
x9_xw_data_15	H23
x9_xw_data_16	K19
x9_xw_data_17	H25
x9_xw_data_18	J23
x9_xw_data_19	K21
x9_xw_data_2	L25
x9_xw_data_3	M16
x9_xw_data_4	M24
x9_xw_data_5	L24
x9_xw_data_6	M21
x9_xw_data_7	M23
x9_xw_data_8	K23
x9_xw_data_9	G25
x9_xw_datardy_n	K25
xa_wx_clk	U16
xa_wx_clk_n	W17
xa_wx_data_0	Y17
xa_wx_data_1	AD18
xa_wx_data_10	AE19
xa_wx_data_11	AB19
xa_wx_data_12	AA18
xa_wx_data_13	W18
xa_wx_data_14	AC21

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xa_wx_data_15	AD19
xa_wx_data_16	AE22
xa_wx_data_17	AE20
xa_wx_data_18	AD23
xa_wx_data_19	AD22
xa_wx_data_2	Y19
xa_wx_data_3	Y18
xa_wx_data_4	AD20
xa_wx_data_5	AB18
xa_wx_data_6	T16
xa_wx_data_7	AC18
xa_wx_data_8	AA19
xa_wx_data_9	AC20
xa_wx_datardy_n	AE21
xa_xw_clk	W21
xa_xw_clk_n	AB25
xa_xw_data_0	AA22
xa_xw_data_1	AB23
xa_xw_data_10	W24
xa_xw_data_11	U22
xa_xw_data_12	AA25
xa_xw_data_13	W22
xa_xw_data_14	T19
xa_xw_data_15	W25

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xa_xw_data_16	Y24
xa_xw_data_17	R16
xa_xw_data_18	U23
xa_xw_data_19	V25
xa_xw_data_2	AB22
xa_xw_data_3	AA23
xa_xw_data_4	AB24
xa_xw_data_5	Y22
xa_xw_data_6	Y25
xa_xw_data_7	AC24
xa_xw_data_8	V21
xa_xw_data_9	T17
xa_xw_datardy_n	V24
xb_wx_clk	Y10
xb_wx_clk_n	AB10
xb_wx_data_0	Y11
xb_wx_data_1	AE08
xb_wx_data_10	AC10
xb_wx_data_11	AE09
xb_wx_data_12	AD12
xb_wx_data_13	T12
xb_wx_data_14	W12
xb_wx_data_15	AA12
xb_wx_data_16	AC12

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xb_wx_data_17	AE11
xb_wx_data_18	AA13
xb_wx_data_19	AE12
xb_wx_data_2	T11
xb_wx_data_3	AA11
xb_wx_data_4	AE10
xb_wx_data_5	U11
xb_wx_data_6	V11
xb_wx_data_7	AC09
xb_wx_data_8	AB11
xb_wx_data_9	AA10
xb_wx_datardy_n	AD11
xb_xw_clk	Y15
xb_xw_clk_n	U14
xb_xw_data_0	AB14
xb_xw_data_1	AD14
xb_xw_data_10	AA15
xb_xw_data_11	R14
xb_xw_data_12	AC16
xb_xw_data_13	AE17
xb_xw_data_14	AC17
xb_xw_data_15	AB17
xb_xw_data_16	U15
xb_xw_data_17	AE18

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xb_xw_data_18	AA16
xb_xw_data_19	T15
xb_xw_data_2	AE14
xb_xw_data_3	R13
xb_xw_data_4	AC14
xb_xw_data_5	AA14
xb_xw_data_6	W14
xb_xw_data_7	AE15
xb_xw_data_8	AB15
xb_xw_data_9	AD15
xb_xw_datardy_n	AE16
xbow_clk	F07
xbow_clk_n	G08
xbow_pwr_on_reset_n	H08
xc_wx_clk	U07
xc_wx_clk_n	V07
xc_wx_data_0	T07
xc_wx_data_1	V05
xc_wx_data_10	W04
xc_wx_data_11	W05
xc_wx_data_12	Y02
xc_wx_data_13	Y05
xc_wx_data_14	AB02
xc_wx_data_15	Y03

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xc_wx_data_16	AB03
xc_wx_data_17	W06
xc_wx_data_18	AB04
xc_wx_data_19	AC02
xc_wx_data_2	U06
xc_wx_data_3	T09
xc_wx_data_4	AA01
xc_wx_data_5	V06
xc_wx_data_6	T10
xc_wx_data_7	W01
xc_wx_data_8	Y01
xc_wx_data_9	AB01
xc_wx_datardy_n	AA03
xc_xw_clk	AE04
xc_xw_clk_n	AE06
xc_xw_data_0	AC05
xc_xw_data_1	AD03
xc_xw_data_10	Y08
xc_xw_data_11	AA08
xc_xw_data_12	AC06
xc_xw_data_13	U10
xc_xw_data_14	AB07
xc_xw_data_15	AD07
xc_xw_data_16	AE05

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xc_xw_data_17	Y09
xc_xw_data_18	AB08
xc_xw_data_19	W10
xc_xw_data_2	AC04
xc_xw_data_3	AB05
xc_xw_data_4	AD06
xc_xw_data_5	V09
xc_xw_data_6	AB06
xc_xw_data_7	AD04
xc_xw_data_8	AA06
xc_xw_data_9	W08
xc_xw_datardy_n	AA07
xd_wx_clk	M09
xd_wx_clk_n	L08
xd_wx_data_0	K09
xd_wx_data_1	K07
xd_wx_data_10	L01
xd_wx_data_11	L04
xd_wx_data_12	K03
xd_wx_data_13	J01
xd_wx_data_14	M04
xd_wx_data_15	M10
xd_wx_data_16	M05
xd_wx_data_17	M11

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xd_wx_data_18	M01
xd_wx_data_19	M02
xd_wx_data_2	K05
xd_wx_data_3	L10
xd_wx_data_4	H03
xd_wx_data_5	H02
xd_wx_data_6	L05
xd_wx_data_7	J04
xd_wx_data_8	L02
xd_wx_data_9	H01
xd_wx_datardy_n	J03
xd_xw_clk	T01
xd_xw_clk_n	R12
xd_xw_data_0	N11
xd_xw_data_1	P05
xd_xw_data_10	U01
xd_xw_data_11	V03
xd_xw_data_12	R02
xd_xw_data_13	V01
xd_xw_data_14	P11
xd_xw_data_15	R09
xd_xw_data_16	R10
xd_xw_data_17	R04
xd_xw_data_18	V04

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xd_xw_data_19	V02
xd_xw_data_2	P02
xd_xw_data_3	P01
xd_xw_data_4	R05
xd_xw_data_5	T02
xd_xw_data_6	P04
xd_xw_data_7	P03
xd_xw_data_8	T03
xd_xw_data_9	U03
xd_xw_datardy_n	R01
xe_wx_clk	H10
xe_wx_clk_n	G10
xe_wx_data_0	K11
xe_wx_data_1	A07
xe_wx_data_10	E08
xe_wx_data_11	A04
xe_wx_data_12	E07
xe_wx_data_13	D07
xe_wx_data_14	B06
xe_wx_data_15	E06
xe_wx_data_16	B04
xe_wx_data_17	D05
xe_wx_data_18	B03
xe_wx_data_19	C04

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xe_wx_data_2	A06
xe_wx_data_3	C05
xe_wx_data_4	D09
xe_wx_data_5	F09
xe_wx_data_6	A05
xe_wx_data_7	C06
xe_wx_data_8	J10
xe_wx_data_9	D06
xe_wx_datardy_n	D08
xe_xw_clk	H07
xe_xw_clk_n	G05
xe_xw_data_0	C02
xe_xw_data_1	D04
xe_xw_data_10	H05
xe_xw_data_11	H06
xe_xw_data_12	E01
xe_xw_data_13	F02
xe_xw_data_14	H04
xe_xw_data_15	J07
xe_xw_data_16	G04
xe_xw_data_17	F01
xe_xw_data_18	K10
xe_xw_data_19	J06
xe_xw_data_2	D03

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xe_xw_data_3	D02
xe_xw_data_4	E04
xe_xw_data_5	F05
xe_xw_data_6	F06
xe_xw_data_7	F04
xe_xw_data_8	E03
xe_xw_data_9	G06
xe_xw_datardy_n	F03
xf_wx_clk	J14
xf_wx_clk_n	G16
xf_wx_data_0	C17
xf_wx_data_1	K15
xf_wx_data_10	C16
xf_wx_data_11	E14
xf_wx_data_12	M13
xf_wx_data_13	A16
xf_wx_data_14	C14
xf_wx_data_15	A13
xf_wx_data_16	L12
xf_wx_data_17	A14
xf_wx_data_18	B14
xf_wx_data_19	K14
xf_wx_data_2	J15
xf_wx_data_3	F15

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xf_wx_data_4	L14
xf_wx_data_5	D15
xf_wx_data_6	A17
xf_wx_data_7	D17
xf_wx_data_8	E15
xf_wx_data_9	B15
xf_wx_datardy_n	A15
xf_xw_clk	C09
xf_xw_clk_n	D11
xf_xw_data_0	E13
xf_xw_data_1	L13
xf_xw_data_10	J11
xf_xw_data_11	A10
xf_xw_data_12	E11
xf_xw_data_13	H11
xf_xw_data_14	A09
xf_xw_data_15	C08
xf_xw_data_16	B07
xf_xw_data_17	F11
xf_xw_data_18	E10
xf_xw_data_19	A08
xf_xw_data_2	A12
xf_xw_data_3	B12
xf_xw_data_4	C10

Table 3 **CCGA PIN ASSIGNMENTS**

SIGNAL NAME	PIN ASSIGNMENT
xf_xw_data_5	C12
xf_xw_data_6	K12
xf_xw_data_7	D12
xf_xw_data_8	B11
xf_xw_data_9	H12
xf_xw_datardy_n	B08

Table 3 **CCGA PIN ASSIGNMENTS**

CHAPTER 3

Programmers Interface

3.1 Overview

The Crossbow ASIC, although not a physical widget, still has a register set accessed as logical widget 0. There are 2 kinds of registers found in the Crossbow: those which are general to the entire chip and those which are associated with a link. The per link register set consists of the following:

- Link(x) Input Buffer Flush
- Link(x) Control
- Link(x) Status
- Link(x) Arbitration Upper
- Link(x) Arbitration Lower
- Link(x) Status
- Link(x) Reset
- Link(x) Auxiliary Status

Registers common to the entire Crossbow:

- Crossbow Identification
- Crossbow Error Command Word
- Crossbow Error Upper Address
- Crossbow Error Lower Address

- Crossbow Interrupt Destination Upper Address
- Crossbow Interrupt Destination Lower Address
- Crossbow Arbitration Reload Register
- Crossbow Packet Time-out Register
- Crossbow Widget 0 Control Register
- Crossbow Widget 0 Status Register
- Crossbow LLP Control Register
- Crossbow Performance Counter
- Crossbow NIC register

There are 3 major functions associated with the register set found in the Crossbow: Error handling, Buffer Flushing, and Arbitration Control.

3.1.1 Error Handling

In general when an error occurs in a link controller, the nature of the error is indicated by flags which are asserted in the Link(x) Status register. If there is packet information associated with the error, the information is logged in the Crossbow Error Command Word register, Error Upper Address register, and Error Lower Address register. The Link's Control register contains interrupt enable bits associated with most error conditions on the link. If the interrupt enable bit is asserted an interrupt to the host processor will be generated. Only one outstanding interrupt will be generated to the host at anytime regardless of the number of errors that have occurred.

An interrupt to the host processor consists of a write to an interrupt register in the host. When issuing an interrupt, the crossbow will generate two double word write request packets. The first packet is sent when the interrupt condition occurs. The second packet is transmitted when the interrupt condition is cleared in the appropriate status register. The programmer should not disable an interrupt bit The packets are routed from widget 0, the crossbow, to the destination of the interrupt which is typically the host processor. This destination is indicated by the target id field found in the Crossbow Interrupt Destination Upper Address register. The packets are sent with the address of the destination's interrupt register which is taken from the values programmed into the Crossbow Interrupt Destination Upper Address and Lower Address registers. The data sent with the first packet consists of the interrupt vector value in bits [7:0] of the data field and bit [8] set to a one. When the packet is received by the interrupt destination, bit 8 in the data field indicates it should decode

the interrupt vector and set the appropriate bit in its interrupt register. The same data is sent with the second packet with the exception of bit [8] of the data field which is set to a zero indicating to the interrupt destination that it should clear the appropriate bit in its interrupt register. The interrupt vector value is programmed into the interrupt vector field of the Crossbow Interrupt Destination Upper Address register.

When an error occurs the Crossbow will update the global error registers if necessary and assert error flags in the appropriate Link(x) Status register. If an interrupt for this error condition is enabled, the first interrupt packet to the host will be dispatched. If subsequent errors requiring interrupts occur before the first interrupt condition is serviced, no further interrupts will be dispatched. However, **all** interrupt conditions must be cleared before the second interrupt packet will be dispatched. Interrupt conditions are cleared by a read of the Link(x) Status register at its read and clear address. This operation will return the value of the register and then clear all error condition and pending interrupts. Once all the Link(x) Status registers have been accessed, all interrupt requests should be cleared and the second interrupt packet should be dispatched. A read of the interrupt register in the host should indicate a cleared interrupt condition. If this is not the case it implies that the another interrupt has occurred before prior interrupt service routine could complete.

In a similar manner the global error registers, Crossbow Error Command Word register, Error Upper Address register, and Error Lower Address register, can only accept information from the first erring packet. A write to the Crossbow Error Command Word register will clear the global registers and re-arm them to accept information from future erring packets.

The types of actual errors that can occur be reported by a link controller are:

- Retry on Link
- Max Retry Limit exceeded
- Packet Time-out
- Overallocated input buffer
- Under allocated Bandwidth
- Invalid Micropacket

As previously described the LLP module uses a "go back n" retry policy to ensure reliable data transmission. This LLP module will indicate when a retry is occurring on the transmit side of the link. It will also indicate when data is received incorrectly on the receive side of the link. To mon-

itor the general integrity of the link, there are two counters in the Link(x) Status register which count the number of retries which have occurred on both the receiving and transmitting link. There are bits in the Link(x) Control which enable interrupt requests to be generated every time a retry occurs or every time a retry counter rolls over. No packet information is logged for this error.

A max retry error occurs when the transmitter has attempted a number of times to transmit the same micropacket and failed. The maximum retry number is based on a programmable value in the Crossbow LLP Control register. When the number is exceeded a flag will be set in the Link(x) Status register. If the max retry interrupt enable bit is set in the Link(x) Control register, an interrupt request will occur. Once a link fails due to a max retry its retry buffer is usually backed up. This causes stale packets destined for the port to remained backed up in source input packet buffers. For this reason the buffer stall signal is removed when a max retry error occurs, thus allowing the packets to be flushed out of their respective source input packet buffers. The packets are essentially dropped. The sources effected when this occurs are indicated by the time-out error location field in the Link(x) Auxiliary Status register.

In addition to max retry error handling each link controller is responsible for detecting packet time-out errors. There are two basic types of packet time-out errors. A max request time-out error and a source time-out

A Max request time-out occurs when due to a crosstalk protocol error or the inability of the widget to return request credits on its link, no request packets are allowed to be forwarded from a source to a given destination. When a destination reaches the its max request limit a watchdog timer is set. If the destination remains at its max request limit for more than one time-out interval, the watchdog timer will detect this. When this condition occurs the Max request time-out bit in the Link(x) Status register will be set and an interrupt request is generated provided the interrupt condition is enabled. In additions the port will then accept request packets to allow the packets to be flushed from the source input packet buffers and sent to the destination. As before, when this error occurs the sources effected are indicated by the time-out error location field in the Link(x) Auxiliary Status register.

The other type of packet time-out that can occur is a source time-out. A source time-out occurs when a packet is being streamed through the crossbow to it's destination and transmission of the packet from the source stalls for the same given maximum time interval.

If a source time-out occurs, the crossbow will complete transmission of the packet with "filler" data and set the appropriate error bits in the packet. The appropriate flag in the Link(x) Status register is asserted, and an interrupt request is generated provided interrupts are enabled.

The time-out value is set globally for all links by programming the interval into the Crossbow Packet Time-out register. Interrupts on time-out detection may be disabled on a link by link basis by setting the proper values in the Link(x) Control registers.

An overallocated input buffer error occurs when a source widget attempts to transfer a packet to the Crossbow and there is no space available to accept the packet in the input packet buffer. When this condition occurs the input over-allocation flag is asserted in the Link(x) Status register. An interrupt request will then be generated provided the proper interrupt enable bit is asserted in the Link(x) Control register.

Please refer to section 3.1.3 Arbitration Control for information on the under allocated bandwidth error.

There are also error conditions specific to widget 0. An error condition occurs when a widget attempts to read or write to an illegal address in widget 0. An error condition will also occur when widget 0 receives a request packet of any size other than a double word packet or a write request to a read only register. Widget 0 can also experience packet time-out errors when attempting to send response packets and issues interrupts. This error is logged in the Crossbow Widget 0 Status register, the packet header information is logged in the global error registers (Error Command Word, Error Upper Address, Error Lower Address), and an interrupt request is generated, provided that the Interrupt on Error bit is set in the Widget 0 Control register.

Please refer to Table 3 for a summary of link controller errors, which are errors which can occur during the routing of a Crosstalk packet through the Crossbar. Please refer to Table 4 for a summary of Widget 0 errors that can occur when the Crossbow processes Crosstalk packets during internal register accesses.

LLP asserts squash data	***LLP Receiver Error*** bit in the Link(x) Status register is set. If the ***Enable Interrupt on LLP Receiver Error*** bit is set in the Link(x) Control register an interrupt will be generated.

Receiver retry counter increments from FF -> 00	The ***LLP Receiver Retry Overflow*** bit in the Link (x) Status register is set. If the ***Enable Interrupt on LLP Receiver Retry counter overflow*** bit is set in the Link(x) Control register, an interrupt will be generated.
LLP asserts transmit retry	***LLP Transmitter Retry*** bit in the Link(x) Status register is set. If the ***Enable Interrupt on LLP Transmitter Error*** bit is set in the Link(x) Control register an interrupt will be generated.
Transmit retry counter increments from FF -> 00	The ***LLP Transmit Retry counter overflow*** bit in the interrupt status register is set. If the ***Enable Interrupt on LLP Transmit Retry counter overflow*** bit is set in the Link(x) Control register, an interrupt will be generated.
Transmitter max retry occurs	Fatal Error condition LLP shuts down, requires link reset from external widget or the port to be reset by a write to the Link(x) Reset register to restart. The ***LLP Max Transmitter Retry*** bit in the Link (x) Status register is set. If the ***Enable Interrupt on LLP Transmitter Max Retry*** is set in the Link(x) Control register, an interrupt is generated.The ***Timed out source location*** will be updated in the Link (x) Status register to indicate other ports have been effected by this error.
Received Crosstalk packet cannot be routed because the destination does not map to a legal crossbow port.	The ***Illegal Destination*** bit is set in the Link(x) Status register. An interrupt will be generated if the ***Enable Interrupt on Illegal Destination*** in the Link(x) Control register is set. The Crossbow Command Word Error, Error Upper, and Error Lower registers will be updated.
Crosstalk packet is received when input packet buffer is full.	The ***Input Overallocation Error*** bit is set in the Link(x) Status register. An interrupt will be generated if the ***Enable Interrupt on Input Overallocation Error*** is set.
While receiving a Crosstalk packet the link stalls during transmission of a Crosstalk packet for a time period > packet time-out value.	The ***Packet Time-out Error Source*** bit is set in the Link(x) Status register. An interrupt will be generated if the ***Source Time-out Interrupt Enable*** in the Link(x) Control register is set. The packet will be forwarded to the destination and the un-received portions of the packet will be filled with data which has the Crosstalk Sideband Error bit set.

A Crosstalk destination has reached it Max Request Limit and has remained in this condition for greater than the time-out interval.	The ***Max Request Time-out Error*** bit is set in the Link(x) Status register. An interrupt will be generated if the ***Max Request Time-out Interrupt Enable*** in the Link(x) Control register is set. The ***Timed out source location*** will be updated in the Link (x) Status register to indicates which ports have been effected by this error.
For diagnostic purposes, if a source with guaranteed bandwidth requirements has reached its bandwidth allocation, an error will be flagged.	The ***Bandwidth Allocation Error Port ID*** will indicate which sources have been under allocated. An interrupt will be generated if the ***Enable Interrupt on Bandwidth Allocation Error*** is set.

Table 4 **Link Controller Error Summary**

link_control_reg

Error	Action
Invalid Packet Type: **-Fetch and Op Packet** **-Store and Op Packet** **-Special Request Packet** **-Special Response Packet** **-Reserved Entries** **-Response packets** These are packet operations not supported by the Crossbow. Also, the Crossbow never issues requests which result in responses.	The ***Register Access Error*** bit in the Widget 0 Status register is set. In addition the 48-bits of the *Crosstalk* address is stored in the Crossbow Error Address Registers and the command word is stored in the Crossbow Error Command Word register. An interrupt will be generated if ***Interrupt on Register Access Error*** is set.

Table 5 **Widget 0 Error Conditions**

Error	Action
Request Packet Data size / Packet size mismatch or Request Packet Unsupported Data size. All Crossbow register accesses are double word any other size packets which are received will cause an error.	The ***Register Access Error*** bit in the Widget 0 Status register is set. In addition the 48-bits of the *Crosstalk* address is stored in the Crossbow Error Address Registers and the command word is stored in the Crossbow Error Command Word register. An interrupt will be generated if ***Interrupt on Register Access Error*** is set.
Request Packet Command Word Error bit set or Sideband Invalid bit set.	The ***Crosstalk Error*** bit in the Widget 0 Status register is set. In addition the 48-bits of the *Crosstalk* address is stored in the Crossbow Error Address Registers and the command word is stored in the Crossbow Error Command Word register. An interrupt will be generated if ***Interrupt on Crosstalk Error*** is set.
Request Packet Invalid Address. Indicates that the request packet contains an address not supported by the Crossbow.	The ***Register Access Error*** bit in the Widget 0 Status register is set. In addition the 48-bits of the *Crosstalk* address is stored in the Crossbow Error Address Registers and the command word is stored in the Crossbow Error Command Word register. An interrupt will be generated if ***Interrupt on Register Access Error*** is set.
Destination time-out. Widget 0 attempts to send a response packet back to a requestor or issue an interrupt packet, however the destination does not respond for a time period > packet time-out value.	The ***Connection Time-out Error*** bit in the Widget 0 Status register is set. In addition the 48-bits of the *Crosstalk* address is stored in the Crossbow Error Address Registers and the command word is stored in the Crossbow Error Command Word register. An interrupt will be generated if ***Interrupt on Connection Time-out Error*** is set.

Table 5 **Widget 0 Error Conditions**

3.1.2 Buffer Flush mechanism

The buffer flush mechanism in the Crossbow allows a process to guarantee all data in the Crossbow has been flushed from initiator to target.

Each link has associated with it a Link Input Buffer Flush register. A read of this register causes all entries in the link's input buffer to be marked. The read response is delayed until all packets in the input buffer have been dispatched to their targets. Once this occurs the read response returns a value of zero. Only one input buffer flush request may be active in the Crossbow at a time. If another Widget attempts to access any other Link Input Buffer Flush register while a flush is in progress the Crossbow will return a value of one indicating that the flush request was denied the "lock" and the widget should retry later to obtain it. For examples of system scenarios where this mechanism would be employed please refer to the Crosstalk specification, section 2.2, ordering.

3.1.3 Arbitration Control

As also detailed in the Crosstalk bus specification, there are two ring priorities of packets received by the Crossbow. These are the guaranteed bandwidth ring (GBR) and the remainder ring (RR). The guaranteed bandwidth ring is for real- time widgets which require a deterministic worst case response time and bandwidth. The remainder ring is for widgets which are non GBR widgets and GBR widgets that have exceeded their bandwidth allocation.

Associated with each link is the Link(x) Arbitration Control Upper register and the Link(x) Arbitration Control Lower register which contain Guaranteed Bandwidth Ring Counts and Remainder Ring Weight values. In the Arbitration Control Upper and Arbitration Control Lower registers there are seven count values which may be programmed. The seven values are associated with the seven other initiating widgets which may use the link controller's output link as the destination of a packet. The Guaranteed Bandwidth Ring Count represents the maximum number of packets that may be sent from an initiating link to this target link as guaranteed bandwidth packets within a given time period. The time period is fixed globally for all links via the Crossbow Arbitration Reload register. Each link reloads all values in its Arbitration Control register at the start of each arbitration reload interval.

Request packets sent to the Crossbow may have a bit set in the command field requesting arbitration on the guaranteed bandwidth ring. Once a GBR packet is received at the Crossbow's crossbar port arbiter, the link's bandwidth counter for the desired initiator/target path is checked. If the value of the counter is greater than zero, the request is treated as a guaranteed bandwidth request. The crossbar port arbiter upon receiving the request will service it in round-robin order with guaranteed bandwidth requests from other links. Whenever a guaranteed bandwidth request pack-

et is serviced, the guaranteed bandwidth count associated with that initiator/target path is decremented. If the counter value reaches zero before being reloaded, all further requests for the remainder of the reload time interval will assume the same priority with the crossbar port arbiter as a remainder ring priority packet. Response packets which have their guaranteed bandwidth bit set always maintain guaranteed bandwidth priority regardless of any other factors and do not effect guaranteed bandwidth counts.

Widgets which participate on the remainder ring share any remaining bandwidth not used by guaranteed bandwidth requestors during the arbitration reload time interval. When the crossbar port arbiter receives an arbitration request from a remainder ring widget it is serviced in a modified round robin fashion with remainder ring requests from other links. When a remainder ring widget's packet wins arbitration it will continue to win subsequent arbitration requests provided it is not preempted by a gbr arbitration request, it is continuing to make arbitration requests for subsequent packet transfers, and it has not exceeded it's remainder weight value. The remainder weight value is essentially a burst count which allows a widget to transfer n packets, where n is equal to the remainder weight value, before it has to give up the destination port to another remainder ring requestor.

For diagnostic purposes an interrupt may be generated any time a widget reaches its guaranteed bandwidth count during a reload interval. The condition is flagged when a request packet reaches its guaranteed bandwidth limit As with other error conditions the condition is marked in the source Link's status register. The packet's destination ID is also logged in the Link(x) Status register so it can be determined which path was under allocated. Unlike normal error packets, the packet is eventually forwarded to its destination widget. This feature may be enabled on a link by link basis by asserting the ***Bandwidth Allocation Error Enable*** bit in the Link (x) Control register.

3.2 Address Map

All register are 32 bits or less in size and are aligned to a 64 bit boundary with the data residing bits 31:0 (Crosstalk Data_Enable = 0x0f).

The registers can be accessed by Crosstalk double-word packet type only. An access to a reserved address in the Crossbow or an any access other than a double word packet access will result in an error condition. Write

accesses to read only and read and clear registers are also flagged as errors.

The Crossbow common registers start at crosstalk address space 0x00_0000_0000. The link 8 specific registers start at 0x00_0000_0100. Each subsequent group of link specific registers is offset 0x40 from the previous group.(ie. link 9 specific registers start at 0x00_0000_0140, etc.)

Registers	Hwreg name	Addresses	Remarks
Crossbow Identification	XB_ID	0x00_0000_0000	read-only
Crossbow Widget 0 Status	XB_STAT	0x00_0000_0008	read-only
Crossbow Error Upper Addr	XB_ERR_UPPER	0x00_0000_0010	read-only
Crossbow Error Lower Addr	XB_ERR_LOWER	0x00_0000_0018	read-only
Crossbow Widget 0 Control	XB_CTRL	0x00_0000_0020	read/write
Crossbow Packet Time-out	XB_PKT_TO	0x00_0000_0028	read/write
Crossbow Interrupt Destination Upper Address	XB_INT_UPPER	0x00_0000_0030	read/write
Crossbow Interrupt Destination Lower Address	XB_INT_LOWER	0x00_0000_0038	read/write
Crossbow Error Command	XB_ERR_CMDWORD	0x00_0000_0040	read/write
Crossbow LLP Control	XB_LLP_CTRL	0x00_0000_0048	read/write
Crossbow Widget 0 Status	XB_STAT_CLR	0x00_0000_0050	read&clear
Crossbow Arbitration Reload	XB_ARB_RELOAD	0x00_0000_0058	read/write
Crossbow Perf Counter A	XB_PERF_CTR_A	0x00_0000_0060	read/write
Crossbow Perf Counter B	XB_PERF_CTR_B	0x00_0000_0068	read/write
Crossbow NIC register	XB_NIC	0x00_0000_0070	read/write
Reserved		0x00_0000_0078 through 0x00_0000_00F8	

Registers	Hwreg name	Addresses	Remarks
Link 8 Input Buffer Flush	XB_LINK_IBUF_FLUSH_8	0x00_0000_0100	read-only
Link 8 Control	XB_LINK_CTRL_8	0x00_0000_0108	read/write
Link 8 Status	XB_LINK_STAT_8	0x00_0000_0110	read-only
Link 8 Arbitration Upper	XB_LINK_ARB_UPPER_8	0x00_0000_0118	read/write
Link 8 Arbitration Lower	XB_LINK_ARB_LOWER_8	0x00_0000_0120	read/write
Link 8 Status	XB_LINK_STAT_CLR_8	0x00_0000_0128	read&clear
Link 8 Reset	XB_LINK_RESET_8	0x00_0000_0130	write-only
Link 8 Auxiliary Status	XB_LINK_AUX_STAT_8	0x00_0000_0138	read-only
Link 9 Input Buffer Flush	XB_LINK_IBUF_FLUSH_9	0x00_0000_0140	read-only
Link 9 Control	XB_LINK_CTRL_9	0x00_0000_0148	read/write
Link 9 Status	XB_LINK_STAT_9	0x00_0000_0150	read-only
Link 9 Arbitration Upper	XB_LINK_ARB_UPPER_9	0x00_0000_0158	read/write
Link 9 Arbitration Lower	XB_LINK_ARB_LOWER_9	0x00_0000_0160	read/write
Link 9 Status	XB_LINK_STAT_CLR_9	0x00_0000_0168	read&clear
Link 9 Reset	XB_LINK_RESET_9	0x00_0000_0170	write_only
Link 9 Auxiliary Status	XB_LINK_AUX_STAT_9	0x00_0000_0178	read-only
Link A Input Buffer Flush	XB_LINK_IBUF_FLUSH_A	0x00_0000_0180	read-only
Link A Control	XB_LINK_CTRL_A	0x00_0000_0188	read/write
Link A Status	XB_LINK_STAT_A	0x00_0000_0190	read-only
Link A Arbitration Upper	XB_LINK_ARB_UPPER_A	0x00_0000_0198	read/write
Link A Arbitration Lower	XB_LINK_ARB_LOWER_A	0x00_0000_01A0	read/write
Link A Status	XB_LINK_STAT_CLR_A	0x00_0000_01A8	read&clear
Link A Reset	XB_LINK_RESET_A	0x00_0000_01B0	write-only
Link A Auxiliary Status	XB_LINK_AUX_STAT_A	0x00_0000_01B8	read-only

Registers	Hwreg name	Addresses	Remarks
Link B Input Buffer Flush	XB_LINK_IBUF_FLUSH_B	0x00_0000_01C0	read-only
Link B Control	XB_LINK_CTRL_B	0x00_0000_01C8	read/write
Link B Status	XB_LINK_STAT_B	0x00_0000_01D0	read-only
Link B Arbitration Upper	XB_LINK_ARB_UPPER_B	0x00_0000_01D8	read/write
Link B Arbitration Lower	XB_LINK_ARB_LOWER_B	0x00_0000_01E0	read/write
Link B Status	XB_LINK_STAT_CLR_B	0x00_0000_01E8	read&clear
Link B Reset	XB_LINK_RESET_B	0x00_0000_01F0	write_only
Link B Auxiliary Status	XB_LINK_AUX_STAT_B	0x00_0000_01F8	read-only
Link C Input Buffer Flush	XB_LINK_IBUF_FLUSH_C	0x00_0000_0200	read-only
Link C Control	XB_LINK_CTRL_C	0x00_0000_0208	read/write
Link C Status	XB_LINK_STAT_C	0x00_0000_0210	read-only
Link C Arbitration Upper	XB_LINK_ARB_UPPER_C	0x00_0000_0218	read/write
Link C Arbitration Lower	XB_LINK_ARB_LOWER_C	0x00_0000_0220	read/write
Link C Status	XB_LINK_STAT_CLR_C	0x00_0000_0228	read&clear
Link C Reset	XB_LINK_RESET_C	0x00_0000_0230	write_only
Link C Auxiliary Status	XB_LINK_AUX_STAT_C	0x00_0000_0238	read-only
Link D Input Buffer Flush	XB_LINK_IBUF_FLUSH_D	0x00_0000_0240	read-only
Link D Control	XB_LINK_CTRL_D	0x00_0000_0248	read/write
Link D Status	XB_LINK_STAT_D	0x00_0000_0250	read-only
Link D Arbitration Upper	XB_LINK_ARB_UPPER_D	0x00_0000_0258	read/write
Link D Arbitration Lower	XB_LINK_ARB_LOWER_D	0x00_0000_0260	read/write
Link D Status	XB_LINK_STAT_CLR_D	0x00_0000_0268	read&clear
Link D Reset	XB_LINK_RESET_D	0x00_0000_0270	write_only

Registers	Hwreg name	Addresses	Remarks
Link D Auxiliary Status	XB_LINK_AUX_STAT_D	0x00_0000_0278	read-only
Link E Input Buffer Flush	XB_LINK_IBUF_FLUSH_E	0x00_0000_0280	read-only
Link E Control	XB_LINK_CTRL_E	0x00_0000_0288	read/write
Link E Status	XB_LINK_STAT_E	0x00_0000_0290	read-only
Link E Arbitration Upper	XB_LINK_ARB_UPPER_E	0x00_0000_0298	read/write
Link E Arbitration Lower	XB_LINK_ARB_LOWER_E	0x00_0000_02A0	read/write
Link E Status	XB_LINK_STAT_CLR_E	0x00_0000_02A8	read&clear
Link E Reset	XB_LINK_RESET_E	0x00_0000_02B0	write-only
Link E Auxiliary Status	XB_LINK_AUX_STAT_E	0x00_0000_02B8	read-only
Link F Input Buffer Flush	XB_LINK_IBUF_FLUSH_F	0x00_0000_02C0	read-only
Link F Control	XB_LINK_CTRL_F	0x00_0000_02C8	read/write
Link F Status	XB_LINK_STAT_F	0x00_0000_02D0	read-only
Link F Arbitration Upper	XB_LINK_ARB_UPPER_F	0x00_0000_02D8	read/write
Link F Arbitration Lower	XB_LINK_ARB_LOWER_F	0x00_0000_02E0	read/write
Link F Status	XB_LINK_STAT_CLR_F	0x00_0000_02E8	read&clear
Link F Reset	XB_LINK_RESET_F	0x00_0000_02F0	write-only
Link F Auxiliary Status	XB_LINK_AUX_STAT_F	0x00_0000_02F8	read-only
Reserved		0x00_0000_0300 through 0x00_00FF_FFF F	

3.3 Register Definitions

Conventions: All register fields unless otherwise noted will default to zeros after power-up. Fields that are marked as reserved will be ignored when written to and return zeros when read.

3.3.1 Link (x) Input Buffer Flush

This read only register returns a value of 0 after the input buffer for the addressed port has been flushed. If a flush operation is already in progress a value of one is returned and the flush is not performed.

3.3.2 Link (x) Control Register

The Link Control register is a read/write register which contains the bits used to enable the Crossbow feature set for each link..

Bits	Definition
31	***Enable interrupt on Link_Alive:*** 0 = disable, 1 = enable. default = 0. Allow an interrupt to be generated to alert host if a device comes on-line after initial start-up.
30	***Reserved***
29:28	***Performance Monitor Mode Select:*** these bits select the performance monitor mode. 00 = no monitoring. 01 = monitor source link. Every time the source link receives a micropacket an increment is sent to the widget 0 performance counter. 10 = monitor destination link. Every time a micropacket (excluding admin packets) is sent by the destination link, an increment is sent to the widget 0 performance counter. 11 = monitor input packet buffer level. Every 100 Mhz clock cycle that the number of locations in the input packet buffer equals the value in the input packet buffer level value, an increment is sent to the widget 0 performance counter. Default = 00.
27:25	***Input Packet Buffer Level:*** for performance monitoring, a way of determining that over a period of time the amount of time there were "n"entries in the input packet buffer. When the performance monitor mode bits are set to "11" each clock cycle the number of crosstalk packets in the input packet buffer is equal to the input packet buffer level. 000 = no entries, 001 = 1 entry, 010 = 2 entries, 011 = 3 entries, 100 = 4 entries, as there are 4 input packet buffer location in the crossbow, values greater than 4 will have be ignored. Default = 000.

Table 6 **Link(x) Control Register**

Bits	Definition
24	***Send Bit Mode 8:*** When this bit is set the link will attempt to come up in 8 bit mode only. 0 = disable, 1 = enable. default = 0.Note: this bit is not reset by a link reset. This allows the programmer to change the sense of ***Send Bit Mode 8*** and then reset the link. The link will then try to come up in the selected mode.
23	***Force Bad LLP Micro-Packet Enable*** this bit enables generation of bad LLP micro packets on first transmission of the packet. The retry will generate a valid packet.1= Force Bad, 0 = normal, default = 0. Note: THis bit acts as a "one shot" after it is set to one it only remains a one for a single cycle and then resets to zero, hence it will always return zero when read from.
22:18	***LLP Widget Credit*** This five bit value is used to determine the maximum number of outstanding request packets that can be sent to a widget. default value = 2.
17	***Enable Interrupt on Illegal Destination:*** 0 = disable, 1 = enable. default = 0.
16	***Enable Interrupt on Overallocated Input Buffer:*** 0 = disable, 1 = enable. default = 0.
15:9	***Reserved***
8	***Enable Interrupt on Bandwidth Allocation Error***: Enables an interrupt when a Bandwidth Allocation Error occurs. 0 = disable 1 = enable. default = 0.
7	***Enable Interrupt on LLP Receiver Error counter overflow:*** Enables an interrupt to be generated when the LLP receives more than 255 erroneous micropacket. 0 = disable, 1 = enable. default = 0.
6	***Enable Interrupt on LLP Transmit Retry counter overflow:*** Enable Interrupt to be generated when the LLP retries more than 255 erroneous micropacket. 0 = disable 1 = enable. default = 0.
5	***Enable Interrupt on LLP Transmitter Max Retry:*** Enables an interrupt when the LLP indicates when the maximum number of retries to transmit a single micropacket has been reached. 0 = disable 1 = enable. default = 0.
4	***Enable Interrupt on LLP Receiver Error:*** Enables an interrupt to be generated when the LLP receives an erroneous micropacket. 0 = disable 1 = enable. default = 0.
3	***Enable Interrupt on LLP Transmitter Retry:*** Enables an interrupt when the LLP indicates a micropacket retry has occurred. 0 = disable 1 = enable. default = 0.
2	***Unused.*** This bit is writable and readable but has no functional effect.

Table 6 **Link(x) Control Register**

Bits	Definition
1	***Max Request Time-out Interrupt Enable*** this bit enables an interrupt when a Max Request Time-out error occurs. 0 = disable 1 = enable, default = 0
0	***Source Time-out Interrupt Enable***: this bit enables an interrupt when a packet has not been received from a source within the time out interval. 0 = disable 1 = enable, default = 0.

Table 6 **Link(x) Control Register**

3.3.3 Link (x) Status Register

The Link Status register which contains status information for link (x) in the Crossbow ASIC. When the Link (x) status register is read from its read and clear address, the value of the register is returned and all error bits are negated.

A note on the link_alive, max_retry, widget_present, and link_failure bits. There are four possible states for a Crossbow port: (1) it is attempting to come out of reset, (2) it has failed to come out of reset, (3) it is out of reset and functional, (4) it is non-functional due to a failure after it successfully came out of reset. If widget_present is inactive, the port will go from (1) to (2) and stay at (2). When the port is in (1), link_alive, max_retry, and link_failure_mode are all inactive. When the port is in (2) max_retry and link_failure_mode are active. When the port is in (3) link_alive is active. When the port is in (4) max_retry is active.

Bits	Definition
31	***Link_alive:*** indicates that the link came out of reset properly and is operational. 1 = link_alive. 0 = dead_puppy.
30-19	***Reserved***

Table 7 **Link(x) Status Register**

Bits	Definition
18	***Multiple_error:*** indicates that while a particular error flag was set another error of the same kind was detected. Valid for ***Illegal Destination, Input Overallocation Error Packet Time-out Error Source, Connection Time-out Error, and Packet Time-out Destination*** flags. Multiple_errors detected = 1, no Multiple_errors detected = 0, default 0.
17	***Illegal Destination:*** indicates that the link received a packet with an illegal value in the destination field. Error = 1, no Error = 0, default 0
16	***Input Overallocation Error:*** indicates that the widget attempted to transfer a packet to the Crossbow and there was no space in the input packet buffer. Error = 1, no Error = 0, default 0
15:8	***Bandwidth Allocation Error Port ID:*** when a bandwidth allocation error occurs this field indicates which source was under allocated.Each bit is a sticky if multiple allocation errors occur, multiple bits will be set. bit 15 = link F,..., 8 = link 8.
7	***LLP receive error counter overflow***
6	***LLP transmit retry counter overflow***
5	***LLP Max Transmitter Retry*** indicates that the max retry count was reached on the transmitter side of the LLP. This error is based on LLP micropackets not *Crosstalk* packets. Error = 1, no Error = 0, default 0
4	***LLP Receiver error*** indicates that an error was received by the receiver section of the LLP. This error is based on LLP micropackets not *Crosstalk* packets.Error = 1, no Error = 0, default 0
3	***LLP Transmitter Retry*** indicates that a retry was required on the transmitter side of the LLP. This error is based on LLP micropackets not *Crosstalk* packets. Error = 1, no Error = 0, default 0
2	***Reserved***
1	***Max Request Timeout:*** indicates that due to a crosstalk protocol error or the inability of the widget to return request credits on its link, this port has allowed no new outstanding requests for more than one time-out interval. Once this error occurs, request packets are allowed to pass to this destination and the source of all packets sent are logged in the time-out error location field in the auxiliary link status register. Error = 1, no Error = 0, default 0.

Table 7 **Link(x) Status Register**

Bits	Definition
0	***Packet Time-out Error Source*** indicates that as a packet was being streamed from a source widget to a destinations widget, transmission from the source ceased and the packet timed out.Error = 1, no Error = 0, default 0

Table 7 **Link(x) Status Register**

3.3.4 Link (x) Auxiliary Status

This register is used for additional status information. When the Link(x) Status Register is cleared, this register is also cleared.

[

Bits	Description
31-24	***LLP receive retry counter*** Default 0x00
23:16	***LLP transmit retry counter*** Default 0x00
15:8	***Timed out source location:*** When a Max Request time-out error or Link Max Retry error occurs, packets are allowed to pass to the destination to allow them to be cleared from the source input packet buffers. They may be considered to be lost packets. This field indicates which sources are effected. Each bit is a sticky if multiple allocation errors occur, multiple bits will be set. bit 15 = link F, 8 = link 8
15:7	***Reserved***
6	***Link_failure_mode:*** This bit has meaning only if ***Link_alive*** is deasserted and ***Widget Present*** is asserted. 1 = link never came out of link reset. 0 = if max_retry is set it implies link came out of reset and failed while transferring data.
5	***Widget Present:*** this bit indicates there is a widget physically attached to this port. widget present = 1, widget not present = 0.
4	***Bit mode 8*** this bit indicates whether the link is a 16 bit link or an 8 bit link. 16 bit link = 0, 8 bit link = 1
3:0	***Reserved***

Table 8 **Link(x) Auxiliary Status**

3.3.5 Link (x) Arbitration Upper

The Link Arbitration Upper register is a read/write register which contains GBR and RR arbitration information for a portion of the links in the Crossbow ASIC. The GBR count value indicates the amount of the destination link bandwidth for a link(x) which is allocated to the other source links. Similarly the remainder weight value is the remainder weight allocated to each of the other source links.

Please note that since a link cannot send packets to itself, (loopback) the value for link A in the Link (A) Arbitration register has no meaning. As an artifact of the Crossbow implementation, this location will be used for widget 0 and should be programmed to 0x1F.

RWV 0x0 => 1 crosstalk packet transfer is allowed from this source before next remainder ring source is serviced. RWV 0x7 => 8 crosstalk packet transfers are allowed to occur before next remainder ring source is serviced.

GBR 0x00 => source is allowed no GBR priority all crosstalk request packets sent to this destination from this source have remainder ring priority. GBR 0x01 => source is allowed to transfer one GBR request packet to this destination during the current GBR time interval. etc.

Bits	Definition
31-29	***Remainder Weight Value Count Link B*** Default 0x00
28-24	***Guaranteed Bandwidth Ring Count Link B*** Default 0x00
23-21	***Remainder Weight Value Count Link A*** Default 0x00
20-16	***Guaranteed Bandwidth Ring Count Link A*** Default 0x00
15-13	***Remainder Weight Value Count Link 9*** Default 0x00
12-8	***Guaranteed Bandwidth Ring Count Link 9*** Default 0x00
7-5	***Remainder Weight Value Count Link 8*** Default 0x00
4-0	***Guaranteed Bandwidth Ring Count Link 8*** Default 0x00

Table 9 **Link (x) Arbitration Register**

3.3.6 Link (x) Arbitration Lower

The Link Arbitration register is a read/write register which contains GBR arbitration information for a portion of the links in the Crossbow ASIC.

Bits	Definition
31-29	***Remainder Weight Value Count Link F*** Default 0x00
28-24	***Guaranteed Bandwidth Ring Count Link F*** Default 0x00
23-21	***Remainder Weight Value Count Link E*** Default 0x00
20-16	***Guaranteed Bandwidth Ring Count Link E*** Default 0x00
15-13	***Remainder Weight Value Count Link D*** Default 0x00
12-8	***Guaranteed Bandwidth Ring Count Link D*** Default 0x00
7-5	***Remainder Weight Value Count Link C*** Default 0x00
4-0	***Guaranteed Bandwidth Ring Count Link C*** Default 0x00

Table 10 **Link (x) Arbitration Register**

3.3.7 Link (x) Reset

The Link(x) Reset register provides a means for software to individually reset a widget after the power on reset sequence (link reset). This signal when asserted will reset all local and remote LLP circuitry. It forces the LLP to assert the remote link reset signal and causes mode arbitration. A write to this address will act as a "one-shot" forcing link reset to be asserted for 2 core clock cycles. Please refer to the Crosstalk Specification concerning Widget/LLP reset and mode arbitration for greater details.

3.3.8 Crossbow Identification

The Crossbow Identification register is a read only register used by the host cpu during configuration to determine the type of the crossbow. The

format is the same as defined in IEEE 1149.1 JTAG Device Identification Register.

Bits	Description
31-28	***Revision Number***: current revision of the crossbow starting at 1.
27-12	***Part Number*** "Crossbow Type" = 0x00
11-1	***Manufacturer Identity*** = 0x00
0	Always read as 1

Table 11 **Crossbow Identification Register**

3.3.9 Crossbow Error Command Word

The Error Command Word register is a read/write register that holds the command word of the packet when errors occur. Subsequent errors are not logged until this register is written. A write to this register will clear this register and the Crossbow Error Upper and Lower Address registers.Data is ignored on a write. For a complete definition of packet command fields please refer to the Crosstalk Bus specification. The command word register defaults to 0x0..

Bits	Definition
31-28	Destination ID Number (DIDN)
27-24	Source ID Number (SIDN)
23-20	Packet type(PACTYP)
19-15	Transaction NUMber (TNUM)
14	Coherent Transaction (CT)
13-12	Data size(DS)
11	Guaranteed Bandwidth Ring enable (GBR)
10	VBP Message (VBPM)
9	Error Occurred (ERROR)

Table 12 **Crossbow Error Command Word Register**

Bits	Definition
8	Barrier
7-0	Reserved

Table 12 **Crossbow Error Command Word Register**

3.3.10 Crossbow Error Upper Address

The Crossbow Error Upper Address register is a read only register which contains the upper 16 bits of the address when any error occurs. Subsequent errors are not logged until the Error Command Word register is written.

Bits	Description
31-16	Reserved
15-0	Address Bits 47-32, Default 0x0

Table 13 **Crossbow Error Upper Address Register**

3.3.11 Crossbow Error Lower Address

The Crossbow Error Lower Address register is a read only register which contains the lower 32 bits of the address when any error occurs. Subsequent errors are not logged until the Error Command Word register is written.

Bits	Description
31-0	Address Bits 31-0, Default 0x0

Table 14 **Crossbow Error Lower Address Register**

3.3.12 Crossbow Interrupt Destination Upper Address

The crossbow interrupt destination upper address register is a read/write register containing the upper 16 bits of address of the register in the host to which the interrupt write request packet is targeted. The target id contains the destination of the interrupt packet and the interrupt vector contains a portion of the data to be sent in the data field of the interrupt packet.

Bits	Description
31-24	***Int_Vector***: Interrupt vector value, Default 0x00
23:20	***Reserved***
19:16	***Target ID Number*** for interrupt destination, Default 0x8 Note: bit 19 is always 1, because 0-7 are not valid interrupt target IDs.
15-0	Address Bits 47-32, Default 0x0000

Table 15 **Crossbow Interrupt Destination Upper Address Register**

3.3.13 Crossbow Interrupt Destination Lower Address

The crossbow interrupt destination lower address register is a read/write register which contains the lower 32 bits of the address of the interrupt register in the host to which the interrupt is targeted.

Bits	Description
31-0	Address Bits 31-0, Default 0x00000000

Table 16 **Crossbow Interrupt Destination Lower Address Register**

3.3.14 Crossbow Arbitration Reload

The Crossbow arbitration reload register determines the reload interval for the GBR counters Range is 0 to 80 usec in 1.28 usec steps. Value

should be > 0 for proper operation. New GBR reload interval values take effect after the current GBR reload interval has completed.

Bits	Description
31-6	Reserved
5-0	***GBR Reload Interval***(1.28 usec/tic) Default = 0x002

Table 17 **Crossbow Arbitration Reload Register**

3.3.15 Crossbow Packet Time-out register

The value in this register determines the time-out interval for packets in the Crossbow. If a packet has been received in a link's input packet buffer and has not progressed through the crossbow in a minimum of one and a maximum of two timer intervals a time-out error will occur. The time interval is computed as: (interval value x 1.28 usec).Value should be > 0 for proper operation. Note: the first time-out interval after reset is 2.56 usec. The next time-out interval will be 0xFFFFFF x 1.28 usec unless a new value has been written first. New time-out interval values take effect after the current time-out interval has completed.

Bits	Description
31-20	Reserved
19-0	Packet time-out interval value(1.28usec/tic) Default = 0x0FFFFF

Table 18 **Crossbow Packet Time-out register**

3.3.16 Crossbow Widget 0 Status

This register contains the error conditions which may result when writing Crossbow internal registers. All status bits default to 0. A read of this register from its read&clear address will clear the Register Access, Crosstalk, and Connection Time-out error bits and, as a side effect, the Widget 0 Interrupt bit. The Link Interrupt Request bits will not be cleared

until their respective status registers are cleared. An active interrupt request bit indicates only that the link or widget0 attempted to send an interrupt; it may have been failed due another outstanding interrupt. See the section on Error Handling for more information.

Bits	Description
31	***Link F Interrupt Request***: Link F has observed an error whose interrupt bit is enabled.
30	***Link E Interrupt Request***: Link E has observed an error whose interrupt bit is enabled.
29	***Link D Interrupt Request***: Link D has observed an error whose interrupt bit is enabled.
28	***Link C Interrupt Request***: Link C has observed an error whose interrupt bit is enabled.
27	***Link B Interrupt Request***: Link B has observed an error whose interrupt bit is enabled.
26	***Link A Interrupt Request***: Link A has observed an error whose interrupt bit is enabled.
25	***Link 9 Interrupt Request***: Link 9 has observed an error whose interrupt bit is enabled.
24	***Link 8 Interrupt Request***: Link 8 has observed an error whose interrupt bit is enabled.
23	***Widget 0 Interrupt Request***: Widget 0 has observed an error whose interrupt bit is enabled.
22-6	Reserved
5	***Register Access Error***: Indicates a register access violation such as attempt to access an address mapped to a reserved portion of widget 0 address space, an attempt to write a read only register, an attempt to read a write only register, an attempt to send a request packet size to widget 0 which is not double word size. Error = 1, No error = 0. Default = 0.
4-3	Reserved
2	***Crosstalk Error***: A crosstalk packet was received with a sideband error bit set or the error bit set in the command word portion of he packet. Error = 1, no Error = 0, default 0
1	Reserved
0	***Multiple_error:*** indicates that while a particular widget0 error flag was set another error of the same kind was detected. Multiple_errors detected = 1, no Multiple_errors detected = 0, default 0.

Table 19 **Crossbow Widget 0 Status Register**

3.3.17 Crossbow Widget 0 Control

This register controls error reporting for widget 0 accesses.

Bits	Description
31-6	Reserved
5	***Interrupt on access error:*** Enable interrupt on register access error. Enable = 1,Disable = 0. Default = 0.
4-3	Reserved
2	***Interrupt on Crosstalk Error:*** Enable interrupts for widget 0 when crosstalk error occurs. Enable = 1, Disable = 0. Default = 0.
1	***Interrupt on Connection Time-out Error:*** Enable interrupts for widget 0 when destination time-out occurs. Enable = 1, Disable = 0. Default = 0.
0	Reserved

Table 20 **Crossbow Widget 0 Control Register**

3.3.18 Crossbow LLP Control

This register controls the parameters for the LLP modules in the crossbow.

Bits	Description
31-26	Reserved
25-16	***Max Micropacket Retry***: Represents the number of time a micropacket transmission will be retried before the link will Retry timeout.Default =0x3FF

Table 21 **Crossbow LLP Control Register**

Bits	Description
15-10	***Null time -out:*** Idle time (in clocks) required before a LLP null micro-packet is transmitted. Minimum value is 3. Default = 0x06
9-0	***Max Micropacket Burst:*** Maximum number of micropackets that may be continuously transmitted before a null cycle is inserted Default =0x010.

Table 21 **Crossbow LLP Control Register**

3.3.19 Crossbow Performance Counter A

This register contains the performance counter and the select signals which indicate which link's performance will be monitored. Only bits 22:20 will be affected by a write operation.

Bits	Description
31-23	Reserved
22:20	***Link Performance Monitor Select:*** determines which link will control the incrementing of the performance counter. 000 = Link 8,.....,111 = link F. Default = link 8
19:0	***Performance Counter Value***

Table 22 **Crossbow Performance Counter A**

3.3.20 Crossbow Performance Counter B

Same a performance counter A. Use of A and B allows monitoring of a source and a destination simultaneously.

Bits	Description
31-23	Reserved
22:20	***Link Performance Monitor Select:*** determines which link will control the incrementing of the performance counter. 000 = Link 8,.....,111 = link F. Default = link 8
19:0	***Performance Counter Value***

Table 23 **Crossbow Performance Counter B**

3.3.21 Crossbow NIC register

This register contains the NIC/microlan support

Bits	Description
31-20	Reserved
19:10	***NIC_BMP:*** determines Bus Master pulse duration in uS. Default = 0
9:2	***NIC_OFSSET:*** Sampling offset relative to the deassertion of the Bus Master strobe. Default = 0
1	***NIC_DATA_VLD:*** Sample Data Valid. Default = 1.
0	***NIC_DATA:*** Sample Data. Default = 0.

Table 24 **Crossbow NIC register**

3.4 Function Errata

3.4.1 Crossbow NIC register

Is readable and writable but the microlan controller is not functional.

3.4.2 Link (x) Status Register

Read and Clear operations are not atomic. There is a two cycle window where an error can occur and be cleared without being detected. Hence read and clear operations should be reserved for clearing error conditions in normal operation.

3.4.3 Link (x) Reset

When this register is written using a crosstalk write with response packet type, no response packet is returned. The workaround is to use only normal write request packets when writing this register.

3.4.4 Double Overflow

Double overflow can hang source, destination ports.An overflow occurs when all entries in the input packet buffer are filled and another packet is received. This condition may only occur if the widget is improperly programmed to believe the Xbow has greater than 4 input packet buffers.

3.4.5 Supported Crosstalk Packet Types

(This is meant as a note for future designer that may want to use reserved crosstalk packet types)

When it receives a crosstalk packet, Xbow is hard-wired to route a fixed number of upkts for that packet, based on a decode of the packet type and data size fields (i.e. it ignores the tail bit). The following size table is currently used:

pac_typ	data_sz	upkts
0001	00	1
0001	01	3
0001	10	9
0001	11	9
0010	00	2
0010	01	3
0010	10	9
0010	11	9
0011	XX	1
0100	00	2
0101	00	1
0110	XX	1
0111	00	1
1000	00	2
1001	00	1
1010	00	1
1011	00	1
1100	00	1
1101	00	1
1110	XX	9
1111	XX	9

Table 25 **Supported Crosstalk Packet types**

CHAPTER 4

Architectural Description

4.1 Overview

The Crossbow chip consists of the following major blocks:

- Crossbar switch
- link controller
- Widget 0 (internal registers and error processing)

The core of the Crossbow chip consists of a 9 port, 68 bit crossbar switch. Each port on the crossbar consists of a 68 bit output bus and a 68 bit input bus. The busses consists of 64 bits of packet data, 3 bits of sideband data(packet_start,packet_stop,invalid packet), and 1 bit of control information(xbar_data_valid). Each port, with the exception of widget 0, is connected to a link controller.

A link controller consists of all the necessary circuitry to transmit and receive data on the links via the crosstalk protocol. In addition it contains all necessary buffers and arbitration logic to control data flow to and from the links.

As shown in Figure 13, each link controller occupies one port on the crossbar switch. The Crossbow supports four 16 bit links and four 8 bit links. Data is transferred through the crossbar at a data rate equivalent to that required for a 400 Mbaud 16 bit link. The 8 bit ports contain rate

matching buffers which allow them to transmit and receive from the crossbar at the 16 bit data rate and transmit and receive data on their physical links at half the data rate of the crossbar. 16 bit ports may be configured to run as 8 bit ports and therefore also contain rate matching buffers.

Widget 0 occupies one port on the crossbar. Widget 0 handles the accessing of Crossbow internal registers and performs error handling. When a link controller detects an error in a Crosstalk packet, it forwards the packet to the widget 0 port and the widget 0 error handling logic updates the necessary registers and issues interrupts if so enabled. All read and write requests of internal registers in the Crossbow are in the form of request packets sent to widget 0.

A more detailed description of these blocks follow.

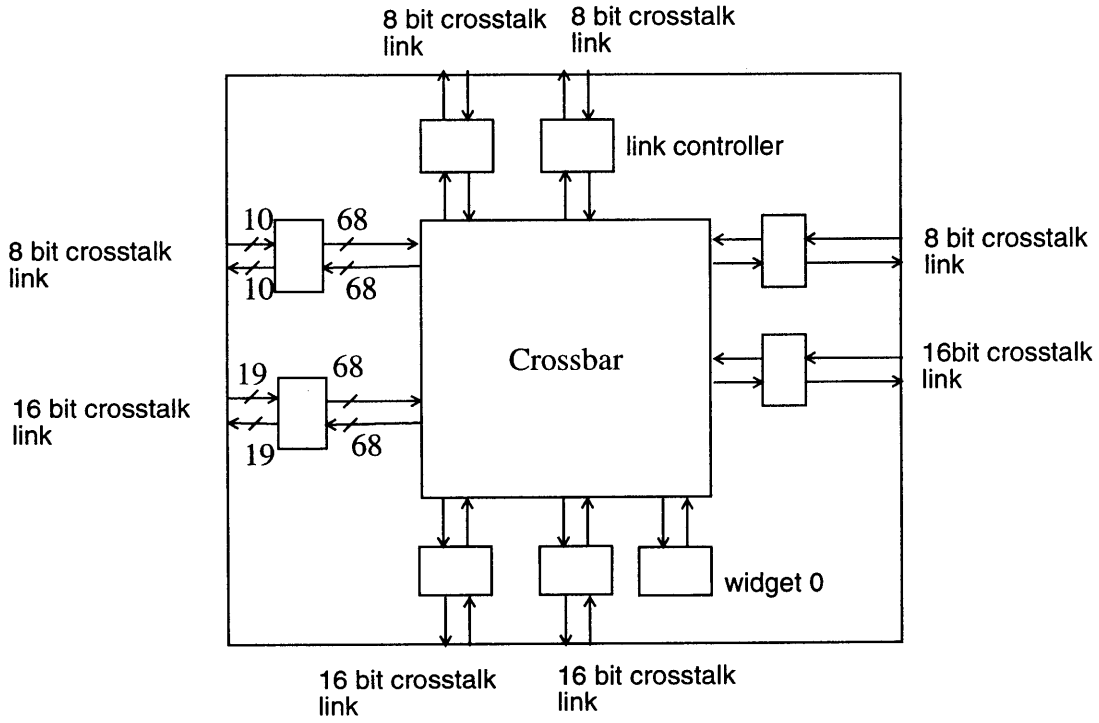


Figure 13 Crossbow Block Diagram

4.1.1 Clocking regimes

The majority of the Crossbow operates off single 100 Mhz clock known as the core_clock. Unless otherwise noted it should be assumed that any block discussed are in the core_clock clock regime.

The two notable exception are the Crosstalk bus receivers and Crosstalk bus transmitters which are known as respectively as the Source Synchronous Receiver (SSR) and Source Synchronous Driver(SSD).

Each SSD receives a 400 Mhz clock, this in turn is used to generate a local 100 Mhz clock, local_100, which is used internal to the SSD. Since the SSD is a fixed standard cell layout, the delay from the 400 Mhz clock to the internal local_100 clock is known. If the on the board the skew from the 400 Mhz source to each pin is closely controlled, the skew between each local_100 clock will also be minimal.

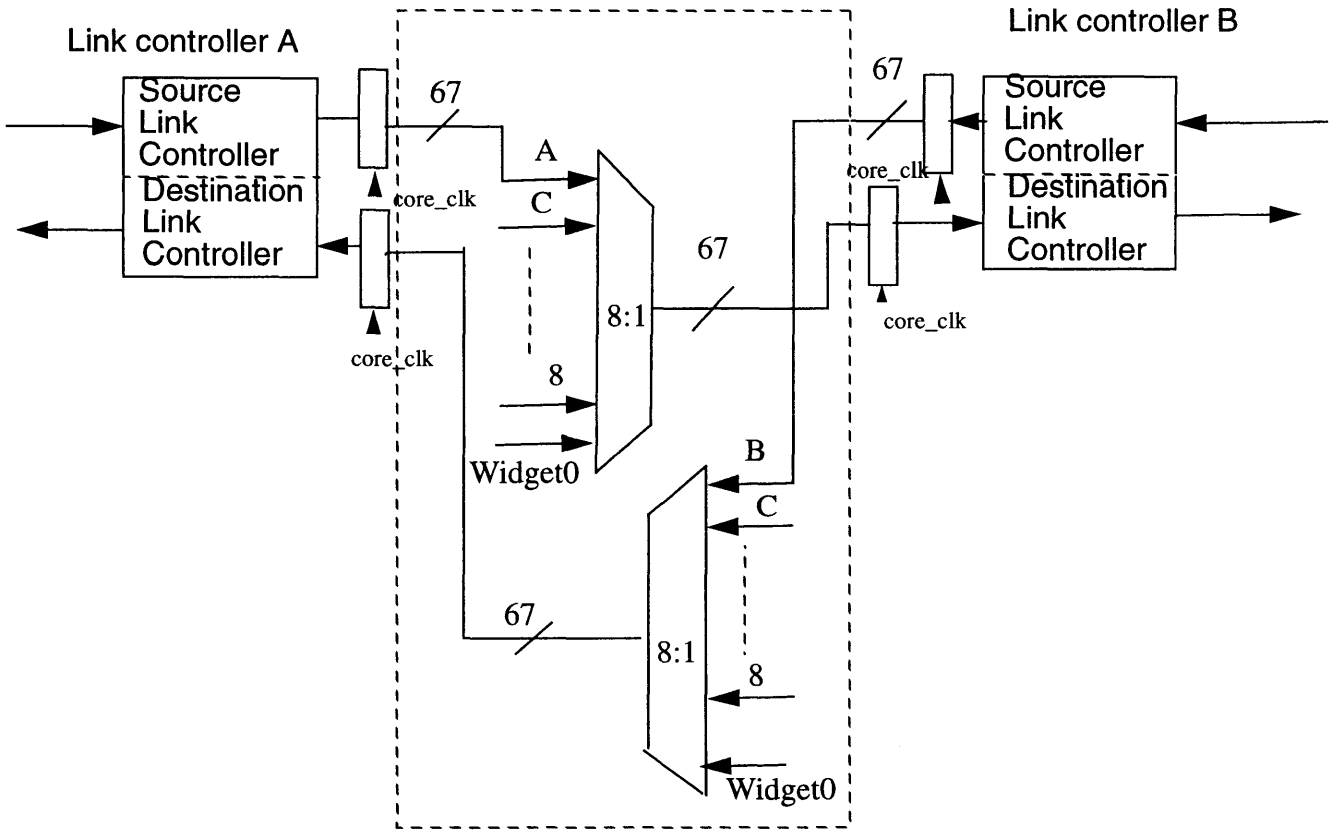
One of the received 400 Mhz clocks is used to generate the 100 Mhz core_clock. The SSD assumes that there is a fixed relationship between the local_100 clock and the core_clock. This relationship is achieved by using a PLL that will cancel the skew between one of the locally generated 100 Mhz clocks and the core_clock. The core_clock will then be delayed to have a fixed relationship between core_clock and the local 100 Mhz clocks.

Each SSR uses a clock which is transmitted with the data and there is a assumed relationship between the frequency of the transmitted clock and frequency of the core_clock, however, there is no phase relationships assumed and no skew requirements. The boundary between the two regimes may be treated as asynchronous for layout purposes

4.2 Crossbar Switch

The crossbar switch is made up of nine 68 bit wide 8:1 muxes. Any of the source ports can be connected concurrently to any of the destination ports with one significant exception. The structure of the crossbar does not allow the connection of a link controller's source crossbar port to its own destination crossbar port (no loopback). Also only single point to point connection are supported (i.e. no broadcast mode). At present it is assumed that the Crossbar switch interconnect will be traversed by data in one core clock cycle. Hence, it is necessary for source link controllers to

drive the crossbar with registered outputs and it is necessary for destination link controllers to register the data in.



TOTAL: NINE 68 bit 8:1 Muxes

Figure 14 Crossbar Switch

4.3 Link Controller

There are essentially two kinds of link controllers. Link controllers differ based on the size of the link port which they are connected. An 8 bit port link controller, as the name implies, controls traffic on 8 bit links and operates only in 8 bit mode. A 16/8 port link controller may operate in either 8 bit mode or 16 bit mode, hence they may connect to 8 bit or 16 bit widgets. In truth the two types

of link controllers differ only in their LLP modules and the controllers which interface to the LLP's.

The Link Controller actually consists of two fairly distinct modules. The Source Link Controller handles all packet traffic between the source link and the crossbar. Conversely, the Destination Link Controller handles all traffic between the crossbar and the destination link. The two modules are more or less completely independent with the exception of status information that is passed from the Source Link Controller to the Destination Link Controller and transmitted back to the initiator widget via the destination link.

4.3.1 Source Link Controller

The source link controller consist of the following submodules:

- Source Synchronous Receiver (SSR)
- Link Level Protocol Receiver (LLPr)
- Input Packet Buffer
- Packet Receive Control
- Free Buffer Stack
- Crossbar Request Manager
- Packet Dispatch Control
- Exception module

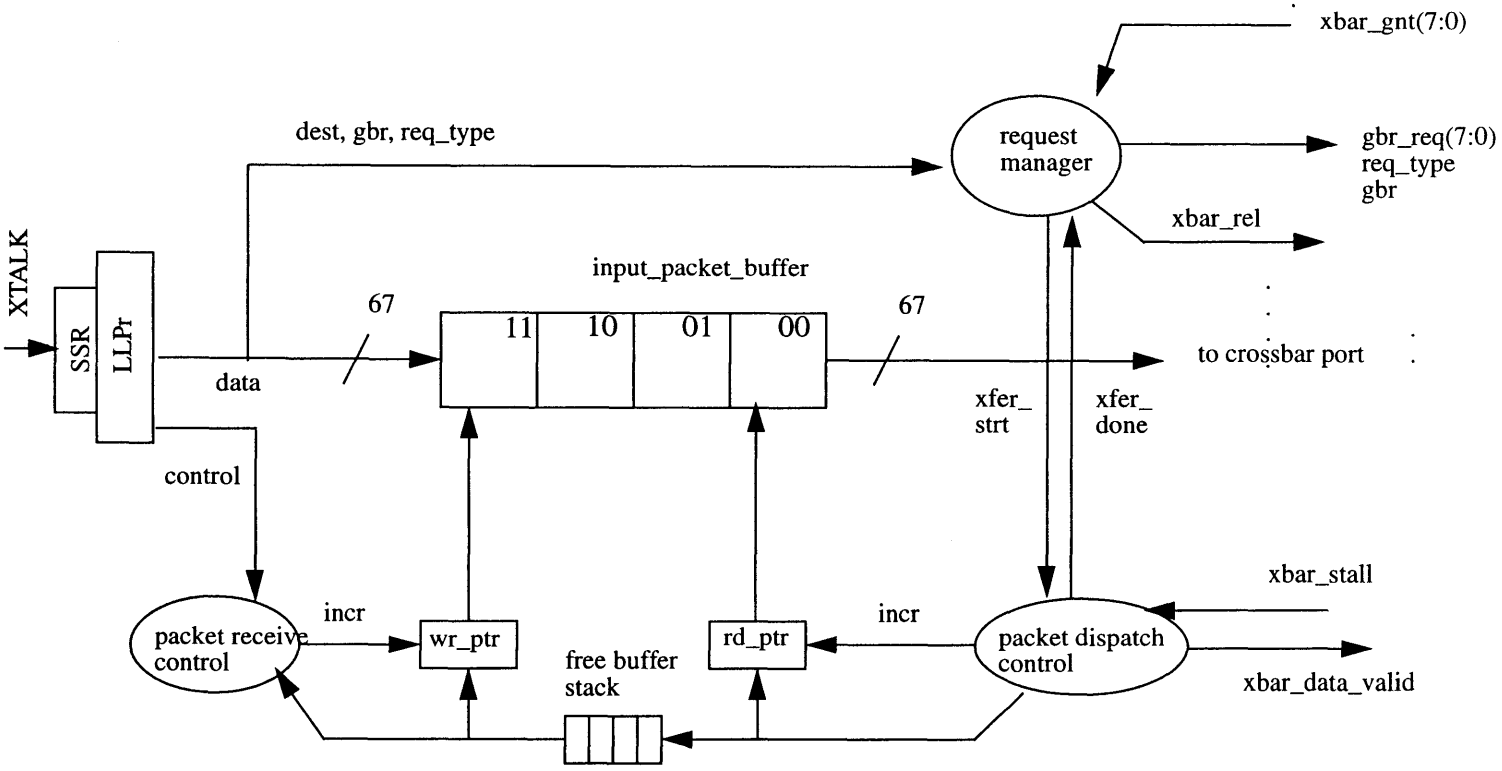


Figure 15 Source Link Controller

4.3.2 Source Synchronous Receiver and Link Level Protocol Receiver

For the purpose of this document the LLPr and SSR will essentially be treated as a black box to which the rest of the Source Link Controller interfaces. The architecture of the LLPr, LLPt, SSR and SSD are covered in great detail in the **Crosstalk Bus Specification** and the **Source Synchronous Driver/Receiver Specification**.

The interface between the SSR and the outside world consists of the following pins: **16 bit link:**

Data[19:0] (i)Data, check bit, and SN information

DataReady_N (i)Valid data is being transmitted, active low

Clk[1:0] (i) Differential clock

8 bit link:

Data[9:0] (i)Data, check bit, and SN information

DataReady_N (i)Valid data is being transmitted, active low

Clk[1:0] (i) Differential clock

The SSR operates in the clock regime of the source synchronous differential clock which it receives(clk[1:0]). Between the LLPr and the SSR is an asynchronous boundary crossing between the received clock and the 100 Mhz core_clock. All SSR outputs interface directly with the LLPr.

The interface between the rest of the Source Link Controller and the LLPr consists of the following signals:

RcvData[63:0] (o) Data from remote transmitter.

RcvSideband[7:0] (o) Sideband data from remote transmitter.

RcvDataValid (o) Data lines contain valid data.

SquashData (o) Last micro-packet must be discarded.

RcvCBError (o) Last micro-packet encountered check bit error.

RcvSNError (o) Last micro-packet encountered sequence number error (only if no RcvCBError detected.)

RcvLinkReset (o) Asserted when local receiver is in link reset state.

RcvWarmReset(o) Asserted when local receiver is in reset state due to a remote or local WarmReset.

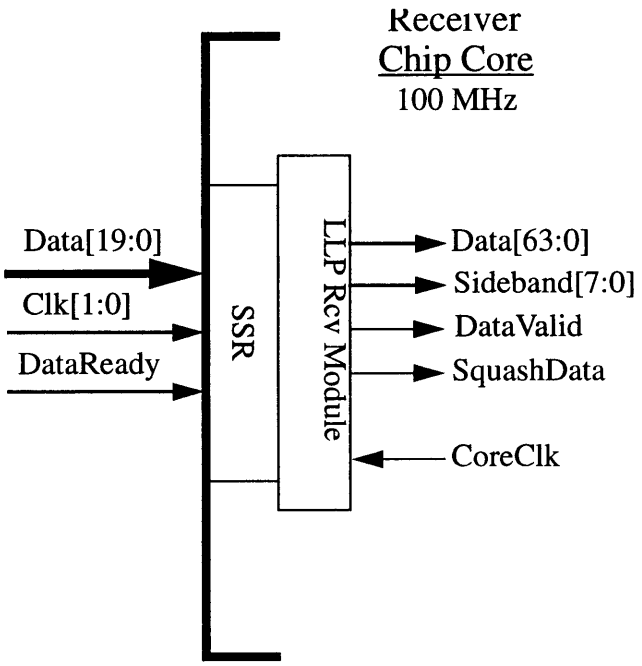


Figure 16 SSR, LLPr Pair

The reset signals from the block namely, RcvLinkReset and RcvWarm-Reset are interpreted by the Link Controller's exception block. The RcvCBError and RcvSNError are also sent to the exception block.

The RcvData from the LLP is received and stored in the input packet buffer until it can be forwarded through the crossbar. Command word portions of the Crosstalk packet data are forwarded to the crossbar request queue. The forwarded information is used to properly route the packet. Portions of the sideband data are also placed the input packet buffer, whereas other portion of the sideband data are forwarded to the destination link controller.

The packet receive control block is responsible for interpreting the control signals from the LLP and writing the input packet buffer. It also handles updating the crossbar request queue. The signals which packet receive control block receives include RcvDataValid, SquashData, BitMode8, MicropacketHead and portions of the Sideband data.

Please refer to Figure 16 for examples of the data transfer cycle timings. In the diagrams D0 refers to the first portion of a micropacket and D1 refers to the second portion. The CRC check code is computed for the entire micropacket. I f the micropacket is found to be in error, SquashData

is asserted. At this point the entire micropacket(D0 and D1) must be discarded.

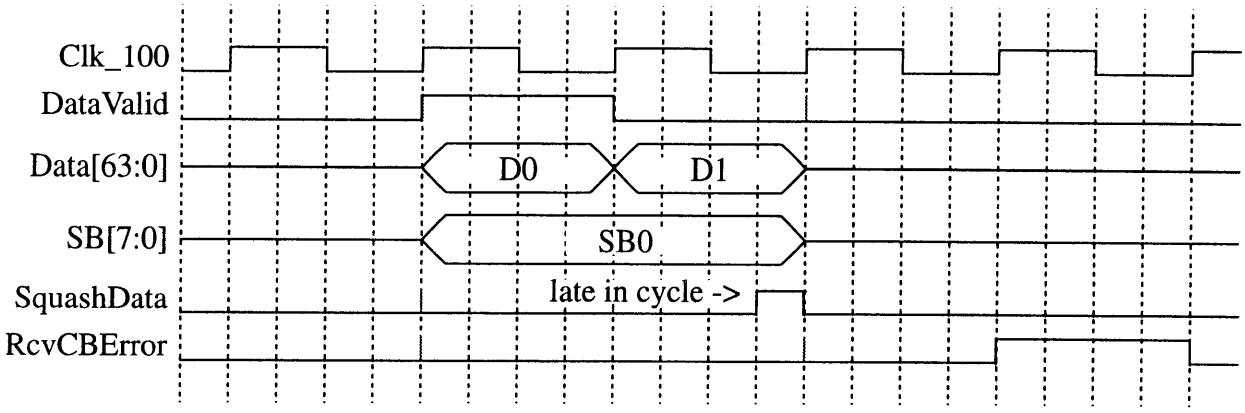


Figure 17 LLPr interface 16 bit mode, with packet error

4.3.3 Input Packet Buffer

The input packet buffer consists of an embedded SRAM 67 bits wide by 72 locations deep. The buffer is subdivided into 4 separate sub-buffers each capable of storing a single crosstalk packet up to a full cache line in length. To simplify the design of the buffer management, each sub-buffer can store only one crosstalk packet.

The input packet buffer stores 64 bits of data from the LLP and 3 bits of sideband data (start, stop and valid). Data is written into the input packet buffer by the packet receive control, and the sub-buffers are managed by the free buffer stack. The packet dispatch module reads data out of the input packet buffer.

The exact implementation of this array is dependent on the speed and size of the selected vendor's array. The array must be capable of a 10 ns read and write cycle times.

4.3.4 Packet Receive Control

The packet receive control module monitors sideband and control signals from the LLP and writes data to the input packet buffer.

A packet is delimited by the sideband signals start and stop. The packet receive control uses these signals, along with a data_valid signal from the LLP, to determine when to write to the input packet buffer. Data is written into the input packet buffer directly as soon as it is received from the LLP.

When the LLP detects a CRC error, it generates a signal called squash, which appears during the last half of a micropacket. By the time the packet receive control module detects a squash, it will have already written the micropacket. The packet receive control must maintain a shadow write pointer so that when a squash is encountered, the write pointer can be backed up to the beginning of the bad micropacket. This micropacket will be overwritten by the next micropacket.

The packet receive control also controls packet length counters. As full crosstalk packets are correctly received (no squash is detected during the stop bit), the packet receive control generates an increment signal to a counter associated with this packet buffer entry.

There is one significant exception when an administrative packet is received. An administrative packet is a Crosstalk packet which has only valid sideband data. The packet receive control logic should detect such packets and inhibit any writes to the input packet buffer.

After a crosstalk packet is correctly received, the packet receive control updates the write pointer to point to the address of the next available subbuffer in the input packet buffer.

Other miscellaneous functions provided by the packet receive control module include alerting error handling logic when errors occur. The errors detected include, micropacket error detected, and over allocated packet buffer. Additionally, it passes valid sideband information such as the freebuf signal onto the crossbar connection arbiter. The meaning of freebuf will be covered in the section related to the packet dispatch control.

4.3.5 Free Buffer Stack

The free buffer stack maintains the locations of the available cache line buffers in the input packet buffer.

As soon as the chip comes out of reset, the free buffer stack pops off the first available location in the buffer and loads the write pointer to point to that location. As complete crosstalk packets are received, the free buffer stack pops off the next location and the packet receive control loads the write pointer. As packets are read out by the packet dispatch block, free locations are pushed back onto the stack.

The free buffer stack also generates a full signal when all its input buffers are in use.

4.3.6 Crossbar Request Manager

The request manager determines priorities of requests and issues them. Requests can be generated in accelerated, scheduled or normal mode depending on the fullness of the input packet buffer and the speed of the source and destination.

The first portion of a packet contains the packet routing and priority information. As the destination is decoded, so is the priority. The decoded destination is then routed to the correct priority request generator (RR or GBR).

In accelerated mode, the decoded destination is passed directly to the request manager as soon as the first half of the first micropacket is received from the LLP. A request is generated the next clock cycle, at which time the second half of the micropacket and the squash signal is available. If there is no error, the request is passed through a mux and latched. If squash is asserted, a null request is muxed in and the request in the request manager is purged. Accelerated mode saves 2 clock cycles over normal mode by not waiting for the squash signal to determine if a request will pushed into the request manager. Accelerated mode will be used in the cases summarized in Figure 16.

In normal mode, the request is held until the first micropacket has been received correctly. If a squash signal is not detected during reception of the second half of the first micropacket, the request is pushed into the request manager.

A third mode is used for transfers from an 8 bit port to a 16 bit port when a full cache line is being transferred. In scheduled mode, the request is held until half the cache-line is received. This allows the source link to build up enough data so as to not slow down the destination link to 8 bit

speed. (The destination link would be waiting for the 8 bit source link to receive data.)

SRC	DEST	INPUT BUFFER EMPTY	Scheduler action
16	16	Y	Select accelerated xbar request
16	16	N	Select normal xbar request
16	8	Y	Select accelerated xbar request
16	8	N	Select normal xbar request
8	8	Y	Select accelerated xbar request
8	8	N	Select normal xbar request
8	16	N	Packet size in micropackets: 1,2,3 Select normal xbar request 9 Select scheduled request
8	16	Y	Packet size in micropackets: 1 Select accelerated xbar request 2 Select normal xbar request 3 Select normal xbar request 9 Select scheduled request

Figure 18 XBAR Request Dispatch

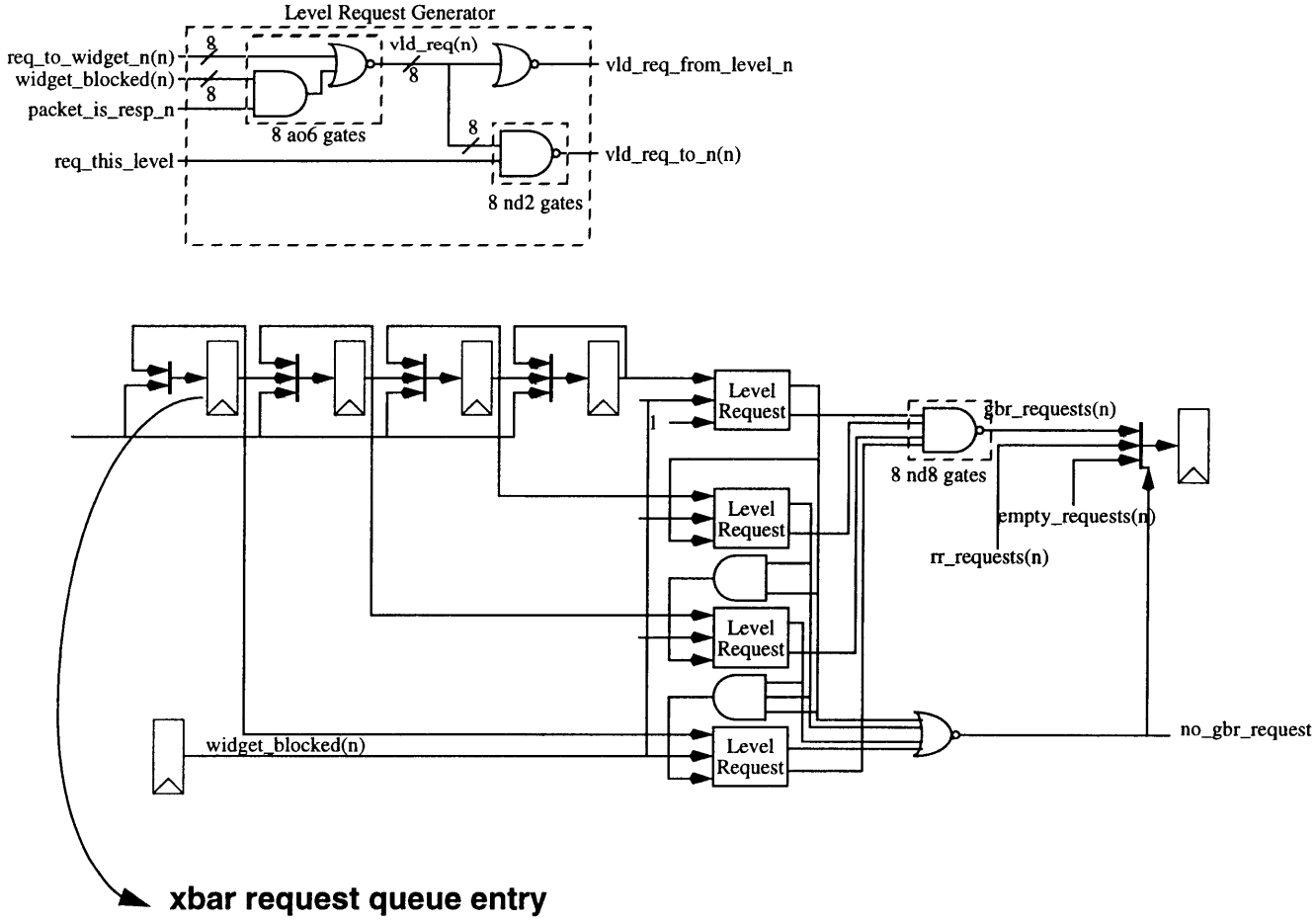


Figure 19 Request Manager Detail

Each priority request generator consists of a fifo-like array of registers. The top of the fifo is the newest request and the bottom is the oldest. A request is generated based on the following criteria:

1. If a packet's destination is blocked, meaning it can accept no more request packets, only response packets destined for that destination may be sent. A request packet cannot be selected until its destination can accept more request packets.

2. The oldest packet in a request generator is promoted ahead of younger packets.

A destination is blocked if it has received the maximum number of requests or if the source has depleted its bandwidth to the target. Counters tracking maximum request and bandwidth available are maintained in the destination port arbiter. The arbiter sends a block signal to all source ports indicating if that source is blocked from sending to this destination.

Each request generator (RR and GBR) independently determines its next request. By taking into account whether the destination is blocked and if the packet is a response or request, the request generators determine the winning request in one clock cycle. A mux at a higher level selects between GBR and RR requests. If there are GBR requests pending, then GBR requests will get priority of RR requests. Please reference Figure 16.

The request manager continues to assert the request until status conditions change causing another packet in the input packet buffer to assume higher priority or the crossbar connection request is granted.

Once the packet has been selected for crossbar connection arbitration it will alert the packet dispatch control which will take appropriate action to prepare for transferring the packet out of the input packet buffer and into the crossbar.

When the packet receives a grant from the crossbar connection arbiter it will clear the packets valid bit in the crossbar request queue, thus removing it from arbitration consideration. When the packet dispatch control is nearing completion of the current transfer it will signal the request manager and a new crossbar connection request will be dispatched. In this manner crossbar connection arbitration for the following packet may overlap the transfer of the current packet.

As mentioned before the age of the packet plays a part in the request managers algorithm. The reason for this is that there is a requirement that request packet issue order is maintained. If this were not the case if two request packets of the same priority were sent to the same destination and they were dispatched in an order different than the one received, a Write After Read hazard may occur.

This problem is circumvented by always selecting the oldest packet. Using a fifo like structure to generate requests, the highest priority is given to requests further in the fifo (and older).

In the event that a change in status occurs among the entries in the input packet buffer and a crosstalk packet assumes a higher priority than the

one currently being selected for crossbar arbitration, the request manager will perform a request context switch. One example of when this would occur would be when the input packet buffer contains only remainder ring priority packets and then a guaranteed bandwidth packet is received. When a context switch occurs the request manager will withdraw all crossbar connection requests. It will wait for a period of time equal to the request to grant latency. If a grant is received during the waiting period the lower priority packet which is associated with the previous request will be sent. If no grant is received a new request will be issued to the destination associated with the higher priority packet.

Size	Bit name	Signal Definitions
1	**Pt**	***Packet timer flag***: when a packet timer interrupt occurs and this entry is valid this bit will be set. When the xbar request associated with this entry is acknowledged this bit will be cleared.
1	**Dto**	***Destination time-out flag***: when a packet timer interrupt occurs and the packet timer flag is set, this bit will be set. This bit is cleared when the xbar request associated with this entry is acknowledged or by the exception processing logic.
1	**Valid**	***Valid:*** indicates a valid location. When this bit is active a crossbar connection arbitration cycle will begin.
1	**Flush**	***Input Buffer Flush*** indicates that this entry was present when a flush command was issued. When an input buffer flush command is issued, all current entries are marked. When a packet is forwarded through the crossbar, the entry is retired and this bit is reset. When all marked entries have been retired, the flush is completed.
1	**ReqType**	***Request type*** indicates whether the packet is a request or a response.
8	**Dest**	***Destination field*** contains the packet's target id field, used to indicate the proper link controller to send the crossbar connection request to.
1	**GBR**	***Guaranteed Bandwidth Ring*** indicates this is a high priority packet for purposes of crossbar arbitration. High priority =1, Normal priority = 0.

Table 26 **Xbar Request Queue Entry**

Size	Bit name	Signal Definitions
2	**BufLoc**	***Buffer location:***indicates which input packet buffer location is associated with this entry.

Table 26 **Xbar Request Queue Entry**

4.3.7 Packet Dispatch Control

The packet dispatch control module is responsible for transferring crosstalk packets out of the input packet buffer and into the crossbar router. Once a crossbar connection request is issued the request manager will forward a pointer which correspond to which input packet buffer the data should be transferred from. This pointer is used to determine the starting address of the data in the input packet buffer and indicates which input packet buffer status register should be accessed.

The packet dispatch control module will upon receiving this data will pull the first value out of the input packet buffer and load it into the crossbar input port. This reduces the start up latency needed when a crossbar grant is received. The packet dispatch control then waits until a crossbar grant is received. At this time it will also assert the xbar_data_valid signal and decrement the packet count value in the input packet buffer status registers.

When the crossbar grant is received the module will begin reading from the input packet buffer and transfer the packet into the crossbar. As each piece is transferred, the packet count is decremented. The presence of a complete packet in the input packet buffer is indicated by the presence of the tail flag in the buffers associated input packet buffer status register. When the tail flag is present and the count has reached zero, the complete packet has been transferred.

When the end of the transfer is near, the module will issue a crossbar release signal. This will allow the request manager to schedule the next crossbar arbitration request. It will also allow the crossbar connection arbiter to give the port away before the current transfer is complete.

In conjunction with releasing the crossbar connection arbiter, the packet dispatch control will make the location in the input packet buffer available. It will do this by pushing the packet buffer location onto the free buffer stack. Additionally, it will send a signal to the transmit side of the

link controller (Destination Link Controller) to issue a freebuf signal on the outgoing sideband data.

The widget has prior knowledge of the number of crosstalk packet buffers in the crossbow's input packet buffer. When a packet is dispatched from the widget to the crossbow the number corresponding to the number of available slots is decremented. When the widget receives the freebuf signal in the sideband portion of its destination link data, this indicates a new slot has been made available and the number is incremented. In this manner packet flow control is achieved between the widget and the crossbow input packet buffer.

The four separate counters in the input packet buffer status registers allow the Crossbow to simultaneously stream data into and out of a packet buffer location. If a source time-out occurs the tail flag will never get set and instead the source time-out error flag will be set. If in the course of a transfer the dispatch controller detects this error flag is set, it should continue to transfer data until it has transferred all existing data and then "pad" the packet with the last piece of data received until the required number of transfers has occurred. The invalid sideband bit should be set for all padded data transferred. The correct number of transfers to perform is indicated by the expected packet size value which is determined from the packet header information and stored in the input packet buffer status register.

If the dispatch controller receives an abort signal it indicates an error has occurred.When this occurs the controller should simply retire the input packet buffer location on to the freebuf stack.

There is a possibility that the destination may not have enough available buffers to store the packet in progress. In this situation, the destination will assert a stall signal. The dispatch controller must then wait for a micropacket boundary before it stops sending the packet. When the stall signal is de-asserted, the dispatch controller resumes transmission of the packet.

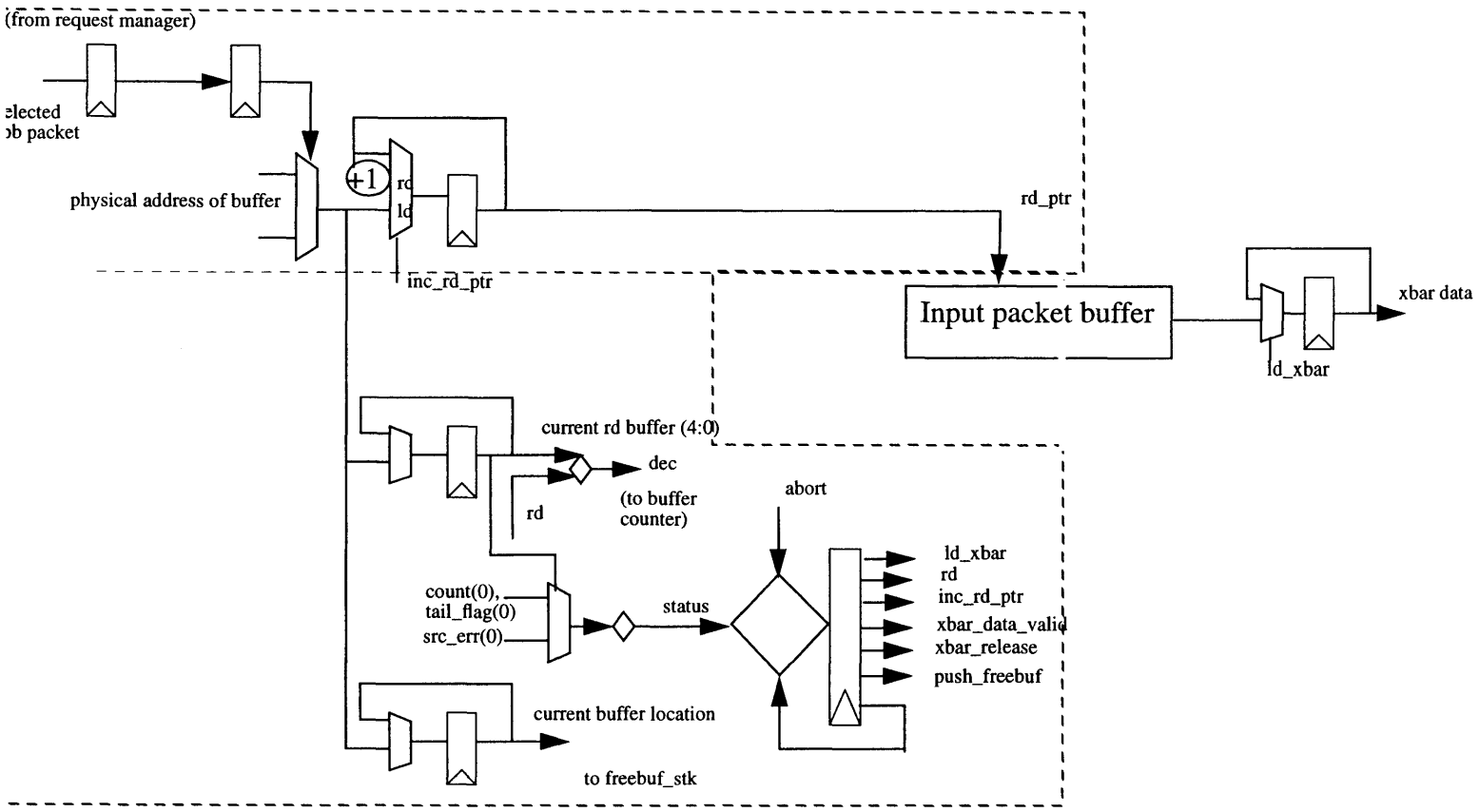


Figure 20 Packet Dispatch Control

4.3.8 Exception module/Error handling

The exception module is responsible for overseeing all error conditions which may occur on in the link. The detection processing of error conditions is somewhat distributed throughout the module in the link controller and their interaction is discussed in this section also. These error conditions include:

- Data Error occurred, receiver
- Data Error occurred, transmitter
- Packet time-out destination
- Packet time-out source

- Illegal destination
- Retry time-out
- Over-allocated input packet buffer
- Under-allocated bandwidth
- Link Reset

A receiver data error occurs when the packet receive control logic detects a corrupted micropacket is being received by the LLP (SquashData active). In this case the packet receive control module will signal the exception module. The exception will update the retry counter in the Link Status register. If the Interrupt on LLP Receiver Error bit is set in the Link(x) Control register, the exception module will send a generate interrupt request signal to the widget 0 each time a receiver error occurs. If the Interrupt on LLP Receiver Error Counter Overflow bit is set an interrupt request will be generated whenever the counter overflows. When widget 0 receives this signal it will issue an interrupt packet to the host if there none currently outstanding.

Transmitter data errors which are indicated by LLPt MicroPktRetry signal are also similarly reported and logged,

Destination time-out errors occur when a packet has been received in the input packet buffer, but has not been dispatched to its destination in a reasonable period of time. The way this is determined is by means of time-out clock. The time-out clock is a central timer in widget 0 which sends a timer interrupt to all the link controllers at a programmable interval.When the time-out clock "interrupt" occurs all valid entry in the request manager's xbar request queue are marked by having their packet timer flag set. As the xbar requests are honored the entries are cleared out of the request queue. If time-out interrupt occurs and the are crossbar request queue entries with their packet time-out flags set, these packets are considered timed-out. At this point the exception module will detect this condition and pass a halt request to the request manager.

The request manager will finish its current transaction, halt, and send a acknowledgment back to the exception module. The exception module will then send a signal to the request manager which will cause all timed out packets in the queue to have their error bit set and their destination redirected to widget 0. The exception module will then release the request manager.

The request manager will then go about normal processing. When it arrives at the entry for the a timed-out packet and make the xbar port re-

quest to widget 0, the error request bit will be set. If the global error registers are available in widget 0, the xbar port request is sent to widget 0. Widget 0 will acknowledge the request. The packet dispatch controller will transmit the packet through the crossbar, retire the buffer location onto the freebuf stack, and signal the exception module. The exception module will update the Link(x) Status register. Widget 0 upon receiving the packet it will update the global error registers. Once the registers have been written the global_err_reg_avail flag is de-asserted.

As subsequent time-out packets are processed if the global_err_reg_avail flag is de-asserted the exception module will intervene and suppress the crossbar port request from being sent to widget 0. It will also "fake" a xbar port grant to the request manager such that the entry is removed from the queue. The packet dispatch controller when it detects that the error registers are not available will simply retire the packet and alert the exception module.

A source time-out occurs when a packet has been partially received and is being forwarded to the destination. Partial transmission will occur when the LLP goes into retry. When the head of a packet is received the packet receive control module will note this. If the packet receive control logic detects two time-out clock interrupts before the tail of the packet will be considered timed-out. The packet receive control will set the packet time-out bit in the packet count registers and prepare to accept the start of the next crosstalk packet. When the dispatch control logic detects that the time-out bit is set, once it has dispatched all available packet data it will pad the rest of the packet and set the invalid sideband bit in each piece of data until the correct number of transfers has occurred. It will then set the **Packet Time-out Source** bit in the Link(x) Status register which will cause an interrupt request to be generated if the interrupt is unmasked.Please reference for a flow diagram hardware handling of source and destination time-outs.

A destination time-out occurs when a connection between a source and destination has been established but, the packet transfer does not complete before two time-out clock interrupts have occurred. This situation can occur when a destination link stalls indefinitely due to multiple link errors or a MaxRetry situation occurring. When this occurs the **Packet Time-out Destination** bit in the Link(x) Status register will be set and an interrupt request to be generated if the interrupt is unmasked. Additionally, the location of the destination which timed-out will be stored in the **Timed_Out Destination** field of the Link(x) Auxiliary Status register. The packet's location in the input packet buffer will then be retired.

An illegal destination error occurs when a packet is received and its destination field does not correspond to any crossbow port destinations. This condition is detected by the destination mapping logic. When an illegal destination is detected the destination mapping logic will map the destination to widget 0 and set the destination error bit. This information will then be entered into the request manager's xbar request queue. From this point the packet is handled the same as a destination time-out packet with the notable exception that the **Illegal Destination** bit is set in the Link(x) Status register.

A retry time-out occurs when the LLPt has tried to send a packet for the programmed maximum number of retry times. When this occurs the **RetryTimeout** signal on the LLPt will become active. This will set the ***LLP Max Transmitter Retry*** bit in the Link(x) Status register and create a interrupt request that is sent to widget 0.

An over-allocated input buffer occurs when a packet arrives at the input and no entries are available on the freebuf stack. When this condition arises the packet receive control will let the packet fall on the floor and set the ***Input Overrallocation Error*** bit in the Link(x) Status register. This will send an interrupt request to widget provided the ***Enable Interrupt on Overallocated Input Buffer*** bit is sent in the Link(x) Control register.

An under-allocated bandwidth error occurs when a GBR request comes in and it's associated bandwidth counter indicates it has used all of it's allocated transfers for the current guaranteed bandwidth interval. The crossbar connection arbiter will detect that this condition has occurred and set the ***Bandwidth Allocation Error*** bit in the Link(x) Status register. In addition it will indicate which source the request(s) came from. The contents of this location will not change, regardless of further under-allocation occurring until the error condition is cleared by a read of the Link(x) Status register at it's read and clear address.

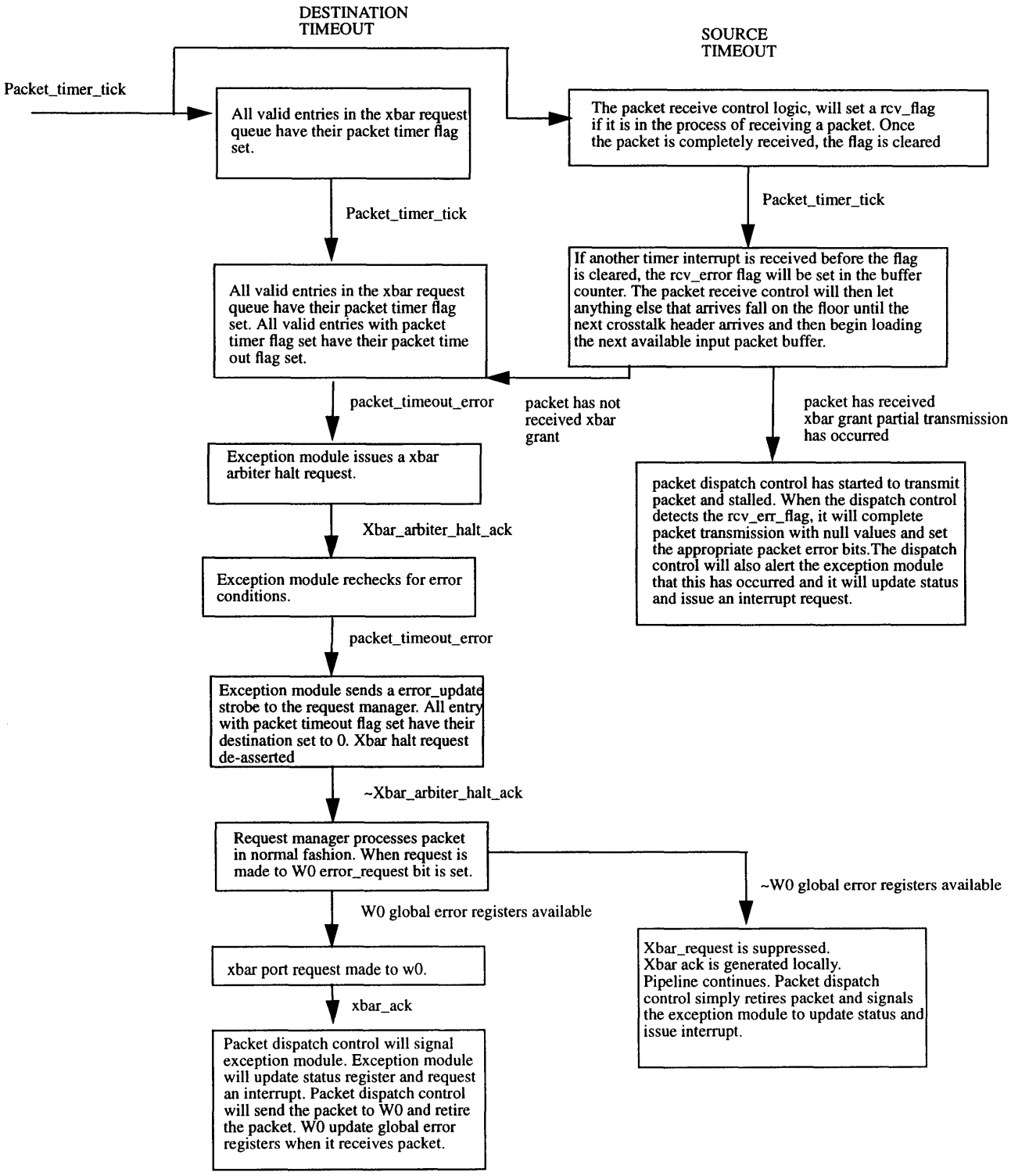


Figure 21 Flow diagram packet time-out processing

4.3.9 Destination Link Controller

The destination link controller consist of the following submodules:

- Crossbar Connection Arbiter
- Freebuf Issue Logic and Crossbar Flow Control
- Sideband Datapath
- Link Level Protocol Transmitter. LLPt
- Source Synchronous Driver, (SSD)

4.3.10 Crossbar Connection Arbitration

Each destination link controller contains a crossbar connection arbiter. The crossbar connection arbiter controls access to the destination crossbar port associated with its link controller. It receives requests from all other source link controller request managers wishing to establish a path through the crossbar to transfer a packet to the destination link. The crossbar connection arbiter will arbitrate from among the outstanding requests, grant the connection, and establish the connection through the crossbar.

A crossbar arbitration cycle begins when one or more Crossbar connection requests are received from a link controller's request manager. The crossbar connection arbiter will send a grant acknowledge signal back to the winning link controller and switch the mux control signals to the crossbar to enable the path from the source link controller to the its destination crossbar port. The path is maintained until the source link controller relinquishes the connection by sending a crossbar release signal to the crossbar arbiter. At this time the arbiter will service other requests if there are any outstanding. If there are no requests outstanding the crossbar arbiter will put the datapath in idle mode.

The crossbar arbitration scheme consists of two round robin ring arbiters. From each request manager the crossbar arbiter receives a request signals which indicate the priority of the crossbar connection request. The highest priority requests are guaranteed bandwidth requests. All other requests are remainder ring requests. Guaranteed bandwidth requests are serviced first in round robin order starting at one greater than the last link

controller which won arbitration. If there are no guaranteed bandwidth requests, remainder ring requests are then serviced. Please refer to Figure 22 and Figure 24.

The crossbar connection arbiter maintains status signals which are used by a source link controller's request manager to determine in which order to send the packets present in its input packet buffer. One bit that is maintained by the crossbar arbiter is the MAX_REQ bit. As previously mentioned in the request management section, this bit indicates that the target widget has received its maximum number of outstanding crosstalk request packets. This maximum number is equal to the number of request packet buffers available in the target widget. When a crossbar connection request is made the link controller sends a signal indicating whether the connection request is for a response or request packet transfer. If the transfer is to be a request packet transfer, when the request is granted a counter is incremented. When the count value is equal to the programmed maximum value the MAX_REQ bit is asserted. When this flag is set link controllers will only send response packets to this target. The target widget will indicate when more request packet buffer space is available by sending the freebuf signal in the sideband data bits, which will eventually decrement the request packet counter and change the MAX_REQ bit.

The crossbar connection arbiter also maintains 8 status bits which indicate the status of each source link controller's guaranteed bandwidth counter. In each crossbar connection arbiter there is a counter associated with each source link controller. This counter contains the number of packet transfers from an initiator widget to this target widget that will be treated as guaranteed bandwidth requests during a programmable time period. All guaranteed bandwidth counters associated with all crossbar connection arbiters are reloaded with their initial values at a fixed interval which is determined by the value in the arbitration reload register. Each time a guaranteed bandwidth ring request packet is granted a connection, the value of the guaranteed bandwidth counter associated with the packet source is decremented. Response packet transfers have no effect on the counters. If the value of a guaranteed bandwidth counter reaches zero before the reload interval, the BW_aval bit associated with the counter is negated. This bit remains negated until the counter is reloaded. As guaranteed bandwidth requests are received for a source link controller, if the BW_aval bit for the source- destination pair is negated, the request will passed over to the remainder ring.

Widgets which participate on the remainder ring share any remaining bandwidth not used by guaranteed bandwidth requestors during the arbi-

tration reload time interval. When the crossbar connection arbiter receives an arbitration request from a remainder ring widget it is serviced in a modified round robin fashion with remainder ring requests from other links. When a remainder ring widget's packet wins arbitration it will continue to win subsequent arbitration requests provided it is not preempted by a gbr arbitration request, it is continuing to make arbitration requests for subsequent packet transfers, and it has not exceeded it's remainder weight value. The remainder weight value is essentially a burst count which allows a widget to transfer n packets where n is equal to the remainder weight value, before it has to give up the destination port to another remainder ring requestor.

The implementation of this ring is similar to the guaranteed bandwidth ring. When a member of the ring wins arbitration a counter is incremented. The value of counter is compared to the remainder weight value associated with it's position on the ring. The ring priority will remain at this position until one of two conditions occur. If the counter value reaches the remainder weight value, the counter is cleared and the "priority token" is passed to the next widget on the ring. Also, if another remainder ring widget wins arbitration, the priority token is passed to that widget and the counter is set to one. The second condition would occur when the remainder ring which previously won does not re-request for a connection.

Please note that while a gbr packet may interrupt a remainder ring burst, the remainder ring device does not give up its priority on the remainder ring. After the gbr requests have been satisfied the remainder ring burst will complete.

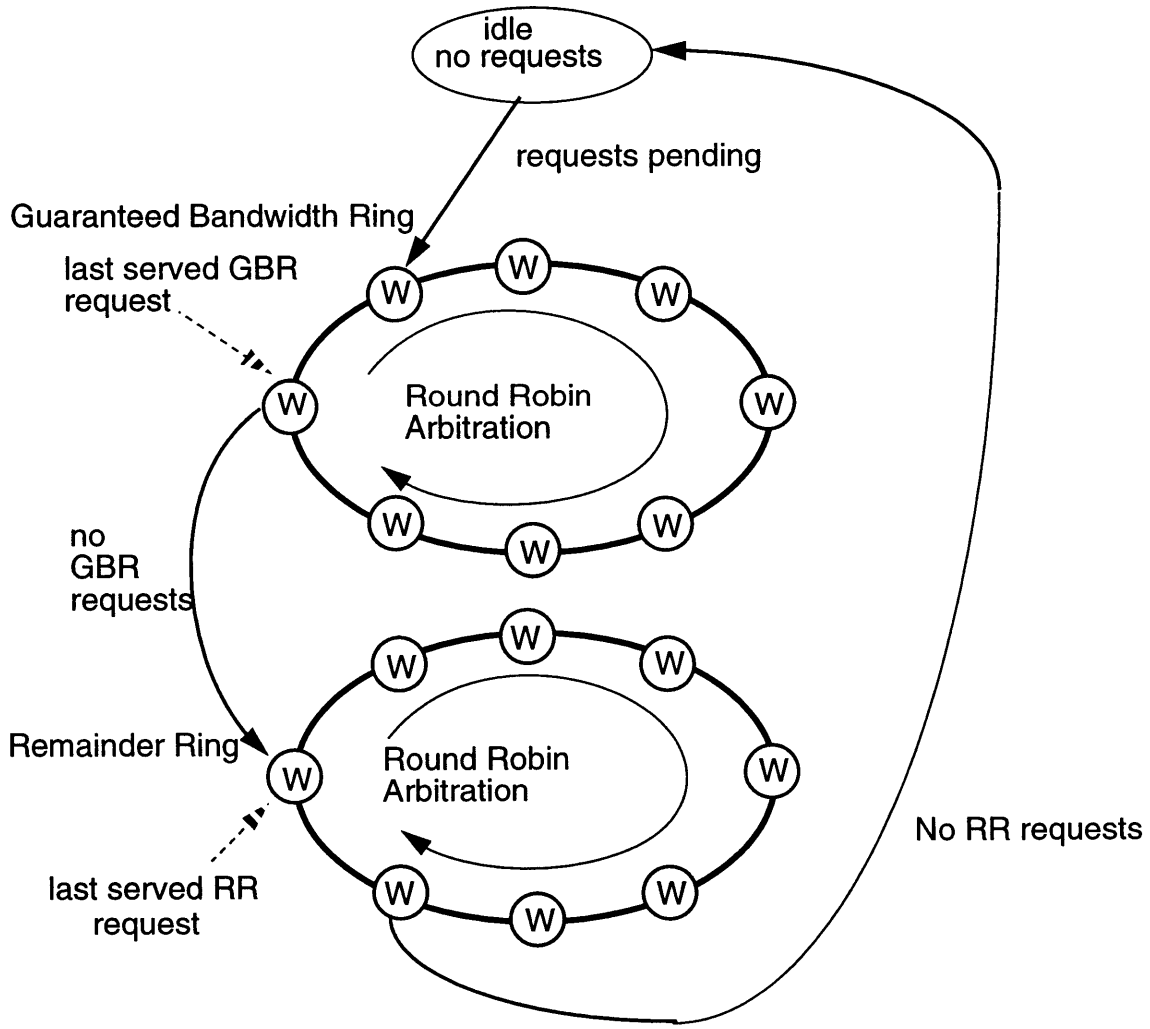


Figure 22 Crossbar Connection Arbitration Diagram

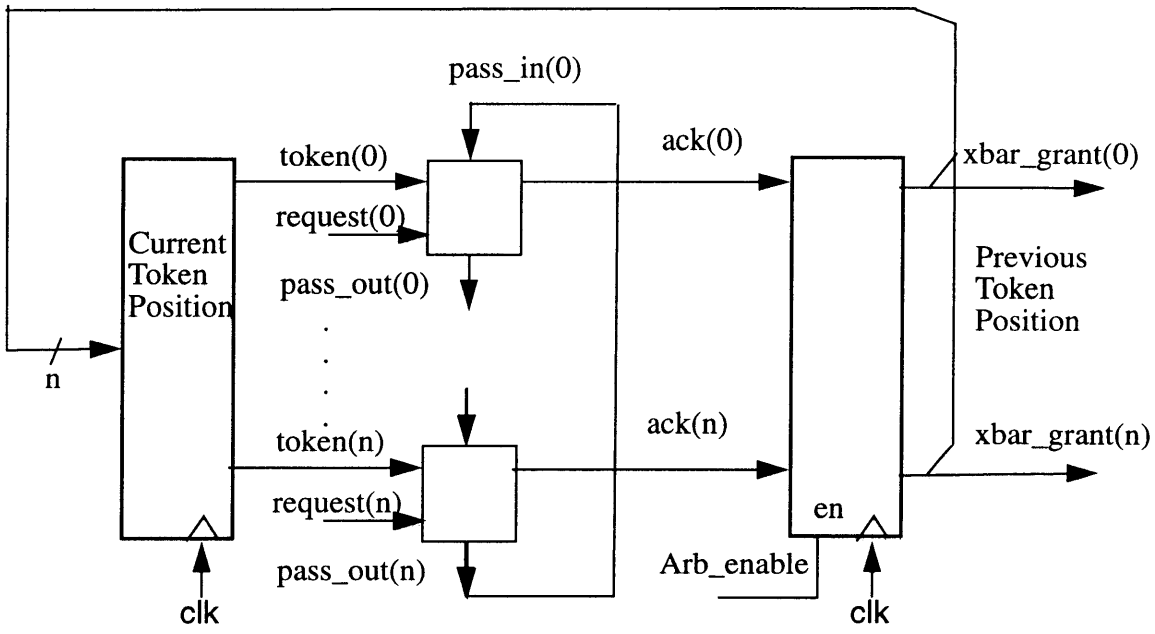


Figure 23 General Ring Arbitration Implementation

4.3.11 Free buffer issue logic and Crossbar Flow Control

As crosstalk packet are dispatched throughout the crossbow locations in the input packet buffer become available. The crossbow will signal the widget which sent the packet that sent the packet that there is space available for another packet to be transferred into the Crossbow. This accomplished by setting a bit in the micropacket sideband data flowing back to the initiating widget. If there is crosstalk packet information flowing back to the initiating widget the credit is simply sent along with the normal stream of data. If there is no traffic on the port the free buffer issue logic will simply generate a micropacket with a credit and a sideband administrative bit set. When the widget receives the micropacket it will simply discard the micropacket data and use the credit information in the sideband data.

As packets are dispatched across the crossbar they are received and passed directly to the LLPt. In the LLPt there is a retry buffer which serves a dual purpose. For normal LLPt functionality it functions as a retry buffer. As the LLPt transmits micropackets it saves them in the buffer

until the receiver acknowledges their receipt. If the acknowledgment does not come in a given time period, the packets are retransmitted.

During chip to chip communication normal turn-around to transmit a packet and receive an acknowledgment uses only a portion of the retry buffer. The remaining buffer location are used as a rating matching buffer to match transfer rates between the 16 bit sources and 8 bit destinations.

During a normal 8 bit transfer there is sufficient space left over in the retry buffer to contain a full cache line packet. This means a 16 bit source transferring to various destinations is not limited by the transfer rate of the destination. It simply dumps the cache line at full 16 bit rate to the 8 bit port and then goes to service another destinations.

In the event the acknowledgment is not received and the buffer fills up, a stall signal is sent out which is monitored by the transmitting source. The current skid distance through the crossbar is 2 micropackets. The LLPt provides a count indicating the amount of free space remaining. When this number reaches 2 micropackets the stall signal is asserted.

4.3.12 Sideband Datapath

Essentially, as described in the previous section, this simply means of muxing in values in the path between the crossbar and the LLPt.

4.3.13 Link Level Protocol Transmitter and Source Synchronous Driver

For the purpose of this document the LLPt and SSD will essentially be treated as a black box to which the rest of the Destination Link Controller interfaces. The architecture of the LLPr, LLPt, SSR and SSD are covered in great detail in (reiterate the specific pertinent documents).

The interface between the SSD and the outside world consists of the following pins: (16 bit link)

Data[19:0] (i)Data, check bit, and SN information

DataReady_N (i)Valid data is being transmitted, active low

Clk[1:0] (i) Differential clock

The interface to the LLPt consists of the following signals:

SendData[63:0] (i) Data to be transmitted

SendSideband[7:0] (i) Sideband data which accompanies each micro-packet

SendDataValid (i) Data lines contain valid data

BufFull (o) DataValid must be de-asserted on the next clock, as the LLP transmit buffer is now full.

CancelDataValid (i) Stomp current micro-packet and do not write it to xmit buffer. Available only in 16 bit mode, this signal can be asserted during the second clock of a micro-packet.

SendDataError (i) Force bad checkbits on first transmit attempt. Re-transmissions will not force errors. Asserted on first clock of micro-packet.

MicroPktRetry (o) Pulses each time the retransmit timer expires, indicating that data is being retransmitted due to lack of an acknowledge from the receiver. This signal, along with the receiver error signals, can be used to gather statistics on link integrity.

RetryTimeout (o) Asserted when a micro-packet has been retried MaxRetry times. Under this condition, the link will shut down all data transmission, de-assert BufFull, and wait until a LinkReset is received.

MaxRetry[9:0] (i) Total retry attempts a micro-packet can fail before RetryTimeout is asserted. This also controls the number of times a link will send reset micro-packet bursts without acknowledge before failing to reset.

MaxBurst[9:0] (i) Longest continuous data burst (in clocks) the LLP will send. Minimum value is 2. This value can be adjusted to account for slightly mismatched clocks.

BuffersEmpty[3:0] (o) Number of micropacket locations remaining in the Micropacket retry buffer. At time 0, this signal equals the usable LLP xmit buffer size. This signal is a registered output of the LLP, and is updated in response to new data from the core on the cycle following DataValid.

RequestBitMode8 (i) Indicates that the local physical port is 8 bits wide. Should be set to 0 for 16 bit ports.

BitMode8 (o) Port size currently being used by LLP. 1 indicates 8 bit, 0 indicates 16 bit.

Reset (i) Chip power on reset: resets all local LLP logic, begins LinkReset process.

SendLinkReset (i) Resets all local and remote LLP logic and asserts remote LinkReset signal. Forces mode arbitration.

SendWarmReset(i) Same as LinkReset, except causes remote WarmReset line to pulse instead of LinkReset line.

WarmResetAfterReset(i) This signal tells the LLP which type of reset-packet to send after the chip Reset signal de-asserts. If this signal is a '1' a WarmReset will negotiation packet will be issued. If the signal is a '0',a LinkReset negotiation packet will be issued. The size request by the negotiation packet will reflect the existing RequestBitMode8 LLP input signal. In Crossbow it is tied to a '0'.

NullTimeout[5:0] (i) Idle time (in clocks) required before a LLP null micro-packet is transmitted. Minimum value is 3.

CoreClk (i) 100 Mhz clock used by core logic.

The SendData signal comes from the rate matching buffer. SendSideband consists of data from the rate matching buffer and the sideband datapath. SendDataValid is provided by the rate matching buffer control. Please refer to Figure 24 for examples of the relationship of these signals

CancelDataValid is not used.

SendDataError is used for diagnostic purposes. It causes a micropacket when initially sent to be sent with the check bits corrupted. On retry transmission the packet is sent correctly.

MicropktRetry and the RetryTimeout signal are sent to the exception module which handles these condition.

MaxRetry, MaxBurst, and Nulltimeout are all provided by values in the Crossbow LLP Control register.

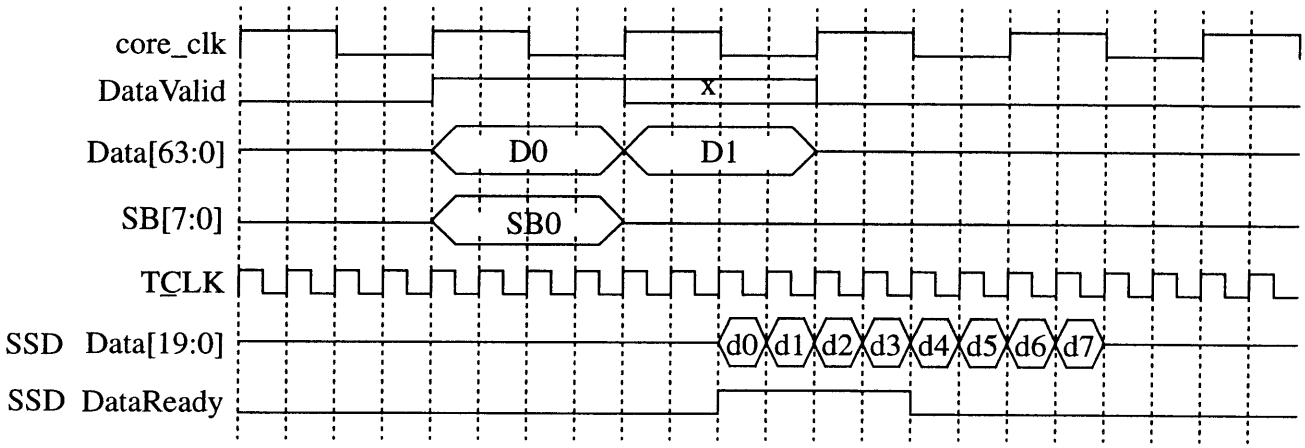


Figure 24 LLPt transmitting

4.4 Widget 0

Widget 0 occupies one crossbar port. All Crossbow internal register accesses are directed to this port. Widget 0 handles the writing of all internal registers and generation of read responses with register information. Additionally packet transfers which have resulted in errors are directed to widget 0. Widget 0 then handles the updating of the global error registers and the issuing of interrupts, if necessary. Link controllers wishing to send a packet to widget 0 send arbitration requests as they would to obtain access to a link. Arbitration for the port is done in a simple round-robin fashion.

4.4.1 Internal Register Access

All Crossbow internal registers are 32 bit. Therefore, all Crossbow registers accesses are assumed to be double word packets.

When a link controller wishes to transfer a packet to Widget 0 it arbitrates for the resource as in the case of a packet destined for any other

link controller. A request is made to the crossbar connection controller, the controller acknowledges the requests in round robin order disregarding any priority flags and sends signals to the crossbar to establish the link between initiator and target.

When the packet arrives at Widget 0 the command portion of the packet is stripped and placed in registers and the address and request type are decoded. If the request is a write request the data is stored in a temporary register and then written to the appropriate register when the address decode has completed. The access to remote link register is accomplished via a separate 32 bit datapath which connects each link controller to widget 0 and widget 0 to the link controllers. Please reference Figure 24.if a write response is requested, widget 0 will then arbitrate for the connection back to the initiating link. Once the connection is granted, the response packet will be formed using the information in the temporary registers and transferred. In the case of a read request, the pertinent request information is stored. Once address decode and register accesses are complete, the read data is placed in a temporary register. As before, arbitration for a connection to the initiating link is requested and the response packet is transferred.

Widget 0 typically processes only one transaction at a time, the one exception to this is a buffer flush read request which is done as a split transaction. When a flush read request is received, the necessary information to generate the read response is stored in a set of registers dedicated to this purpose. A flush request is sent to the link controller associated with the input packet buffer that is to be flushed. While the flush is occurring other requests to widget 0 may be serviced. Once the flush has completed the link controller will signal widget 0. Widget 0 will complete any transaction in process and then arbitrate and dispatch the response. Since the Crossbow supports only one outstanding flush request at a time from all requesting widgets, it is assumed one set of temporary registers to store information for the read response is sufficient.

4.4.2 Error Processing

When an error occurs at a link controller, the erring packet is forwarded to widget 0. This typically occurs for packet time-out errors. When the W0 crossbar connection arbiter receives the port request an additional signal will be sent the error request bit which indicates that the information being transferred will be for an update of the global error registers namely the Crossbow Error Command Word, Crossbow Error Upper Address, and Crossbow Error Lower Address registers.

When the packet is received, the command and address portions of the packet are extracted and stored in the global error registers. At the global_err_reg_avail bit will be de-asserted. The link controllers will detect the de-assertion of the signal and respond accordingly, meaning no more error information will be forwarded to w0. The w0 crossbar connection will then be released to service other requests. The global_err_avail flag is reset by writing the Crossbow Error Command Word register. Writing the Crossbow Error Command Word register also has the side effect of clearing the global error registers.

There are also errors that are specific to widget 0 these include illegal register accesses and destination packet time-outs. These errors are handled in much the same manner as an error on a link controller. The packet information is logged if the global error registers are available and an interrupt request is generated if the interrupt for the error condition is enabled.

4.4.3 Interrupt processing

Each link controller and widget 0 itself generate interrupts by means of an interrupt request signal which is sent to the widget 0 interrupt processing logic. The interrupt requests are essentially all "or" ed by widget 0. Interrupt request lines will remain active until there error condition bit in the Link(x) Status register is cleared or, its associated interrupt mask in the Link(x) Control register is enabled. When widget 0 detects the first interrupt it will send an interrupt halt to the widget 0 xbar connection arbiter. After any current transaction are completed, the arbiter will grant no more crossbar connection requests until released by the interrupt control logic. Widget 0 will then form an "interrupt on" write request packet, arbitrate for the host destination port, and transfer the packet. At this point it will release the widget 0 crossbar connection arbiter.

When all interrupt request have been de-asserted the same process will occur with the exception of the fact that the "interrupts off" interrupt packet will contain different data.

4.4.4 Buffer Flush Handling

As discussed in the programmer's interface section of this document, it sometimes becomes necessary to determine when all data in an input packet buffer has been flushed to a particular destination. As a simplification the flush mechanism actually verifies that all valid entries existing in the input packet buffer when the flush request was made will be flushed regardless of their destinations.

A widget wishing to flush an input packet buffer will send a read request to the link input buffer flush register associated with the packet buffer it wishes to flush. This request is sent to the widget 0 destination. The widget 0 controller will determine whether a buffer flush is currently in progress by checking internal flags. If no flushes are in progress the widget 0 controller will assert the flag which indicates a flush is in progress. The widget 0 controller will then save the necessary information to form a read response packet. It then issues a flush request to the appropriate link controller. The flush is performed as a split read request transaction. While the flush is in progress widget 0 continues to process future requests. The flush management logic in the link controller will then mark all valid buffer entries by setting the flush flag in their request manager's crossbar request queue entry. As the entries are retired this flag is cleared. When all flags have been cleared, the flush management logic signals widget 0. Widget 0 then sends a read response to the requesting widget indicating the flush has completed (a data value of 0) and de-asserts the flush in progress flag.

If a flush is in progress the widget 0 controller will form a read response and route it back to the requesting widget with a value of one. This indicates that the flush request was denied and the widget should retry the request later. Allowing only one outstanding flush in the Crossbow at a time eliminates the need for keeping context for multiple split transactions.

4.4.5 Timers

Widget 0 also contains the timer for the arbitration reload interval and the timer which sets the input packet buffer time-out interval. The timer outputs are sent to and monitored by all link controllers. The timer clock consists of a pre-scaled version of core_clock which provides 1.28 usecs per "tic". The arbitration reload timer is a 6 bit counter and the input packet buffer time-out timer consists of a 20 bit counter.

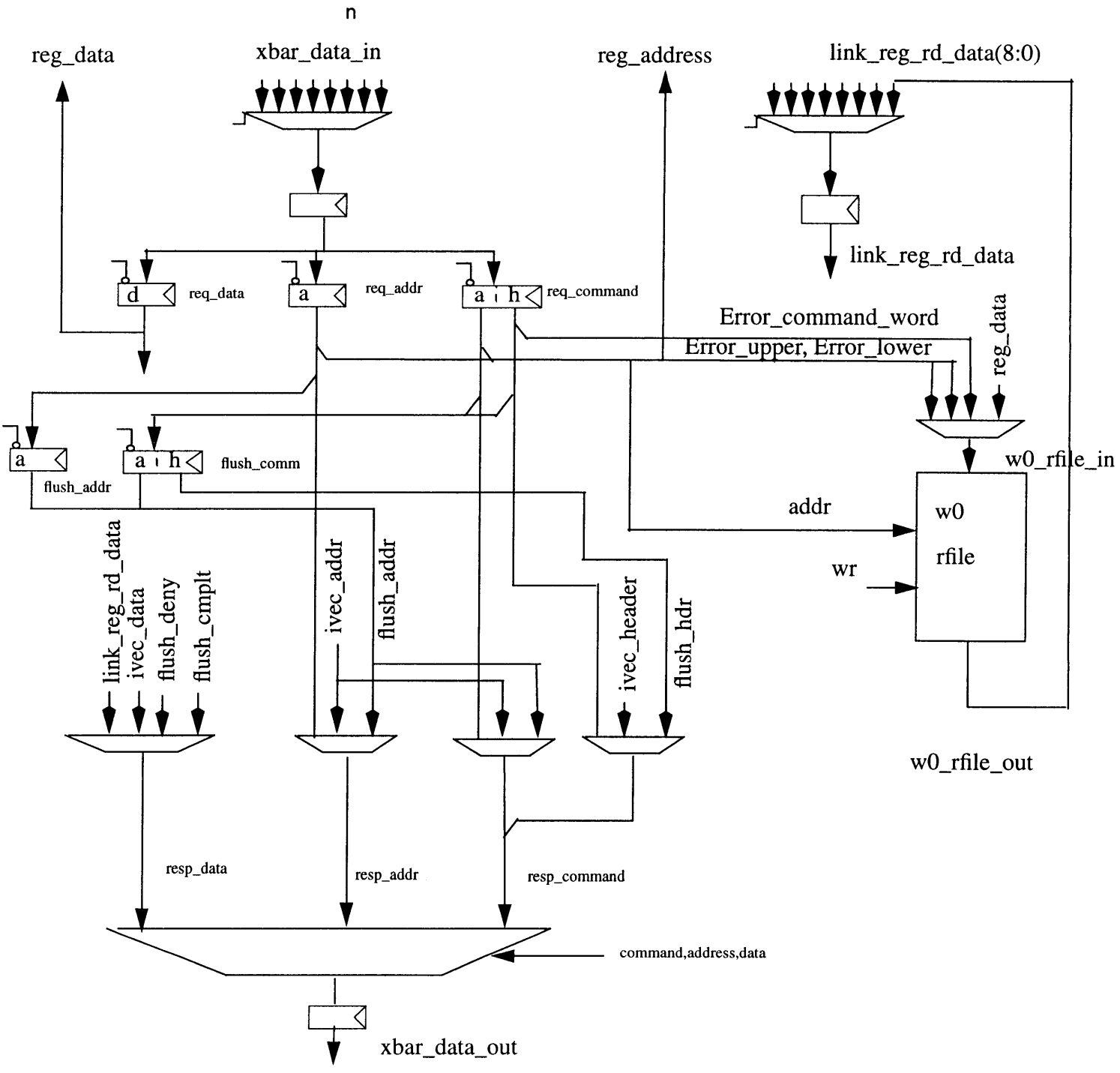


Figure 25 Widget 0 Datapath

4.5 Reset of Device

There a three reset conditions for the crossbow:

- power on Reset
- Link Reset
- Warn Reset

When power on reset is asserted all pertinent registers in the Crossbow will be initialized. Power on reset will also be sent to the LLP blocks in all link controllers. Power on Reset is then de-asserted and the LLP will begin their reset sequence.

The reset sequence for the LPP is described as follows:

The LLP send module can be reset with either Reset (power on), LinkReset, or WarmReset. Each of these resets will clear the sender's transmit buffer and all other LLP internal state. During the reset state, the LLP also transmits special reset packets to the remote LLP, forcing that LLP module into reset as well. LinkReset and WarmReset both exist so that a higher level protocol can decide if just the local LLP link should be reset, or if a reset pulse should be propagated throughout the entire system. Note that the power on Reset will look like a LinkReset to remote LLP receivers.

Before the LLP can exit the reset state, it must negotiate the link bit mode with the remote LLP. LinkReset will cause the LLP sender to transmit a continuous stream of reset micro-packets, with a small time gap between back-to-back packets. These packets contain the data width of the transmitter (8 or 16 bits), and continue to transmit until a bit mode has been negotiated. Since the physical data link width of the remote receiver is not know, reset packets are sent alternately in 8 and 16 bit modes. If a bit mode is not negotiated within MaxRetry reset micro-packet bursts, the link is assumed dead, and RetryTimeout is asserted.

Once a reset packet is received, the LLP knows it is connected on a live link. The LLP will then remain in a reset state for a number of clocks sufficient to cover the transmission of several reset micro-packets. This allows the LLP to negotiate for data link transmission size. Each side of the link transmits its actual physical size, and at the completion of the reset

After power on reset the LLP will eventually all come out of reset i.e Buffull de-asserted, RcvLinkReset,RcvWarmReset, de-asserted, and begin normal operation. Those that are not connected or failed to come out of reset will have their RetryTimeout signals active. Active link connec-

tions are indicated by the value of the ***LLP Max Transmitter Retry*** bit in the Link(x) Status Register.

An individual Crossbow port may be reset by an external widget when the widget sends a Link Rest pattern out onto the Crosstalk bus. The LLPr in the crossbow will receive the pattern and generate a RcvLinkReset signal. The exception control block will cause the crossbar release signal to become active for a number of cycles to release any crossbar connections. The Link Controller associated with the port is then reset. No other links or Link Controllers are effected.

The Crossbow itself may reset an individual port. This is done by setting the Link Reset bit is set in the Link(x) Control Register. Writing the bit causes a "one-shot effect" whereby SendWarmReset is asserted for a few cycles and then the Link Reset bit is cleared. This cause the LLP to begin transmitting warm reset patterns on the crosstalk bus. In a similar manner the exception control block will force a disconnect from the crossbar and reset the Link Controller.

The Crossbow may also receive a WarmReset pattern from an LLPr. A WarmReset differs from a Link reset in that it is propagated throughout the Crossbow. When the RcvWarmReset signal becomes active from any LLPr, for all intents the Crossbow will consider this a power on reset and reset all core logic. It will also propagate the reset and assert a SendWarmReset to all LLPt's in the Crossbow.

As the WarmReset is propagated out, WarmReset are received back. These reset loops are disabled by the LLP which after detect the first reset pattern will ignore all subsequent reset request for 1024 cycles.