NUMAchine

NUMAchine Hardware Reference and Maintenance Manual

**** THIS IS A PRELIMINARY DOCUMENT. ****

It is continually evolving and is subject to change at any time.

Steve Caranci Alex Grbic Robin Grindley
Mitch Gusat Orran Krieger Guy Lemieux
Kelvin Loveless Naraig Manjikian Zeljko Zilic

Department of Electrical and Computer Engineering
University of Toronto
Toronto, Canada
M5S 3G4

June 16, 1998

Contents

List of Figures

List of Tables

Part I

Overview

Chapter 1

The NUMAchine Multiprocessor

1.1 Overview of Hardware Components and Interconnection

NUMAchine is a scalable, cache-coherent, shared-memory multiprocessor designed to be modular, cost-effective and easy to program [V+95]. The architecture of the NUMAchine multiprocessor consists of a number of *stations* connected by a hierarchical ring interconnection network, as shown in Figure 1.1. This organization provides modularity for incremental growth in system size.

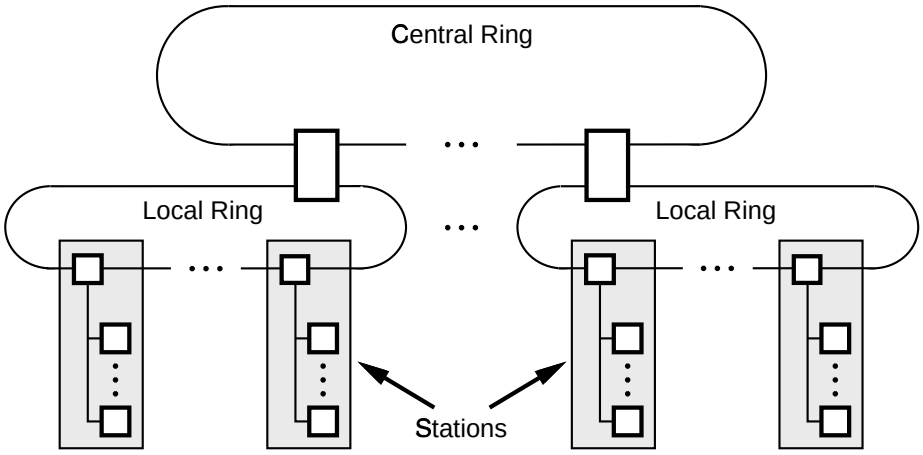

Figure 1.1: The NUMAchine hierarchy

The NUMAchine station is shown in Figure 1.2 on the next page. Each station contains a number of processor cards with caches, local memory, I/O, and a network interface. The network interface includes a network cache for caching remote data, and a ring interface connected to the local ring. The components of a station consist of commodity parts for cost-effectiveness, notably field-programmable devices. Not shown in Figure 1.2 on the following page is a centralized bus arbiter to control access to the station bus.

The local memory implements a portion of the global shared memory, and the network interface includes a network cache for copies of data from the memory in remote stations. A hardware-implemented cache coherence protocol [Grb96] is implemented in the local memory and network interface. This protocol automatically maintains valid copies of data in caches throughout the system in order to provide ease of programming.

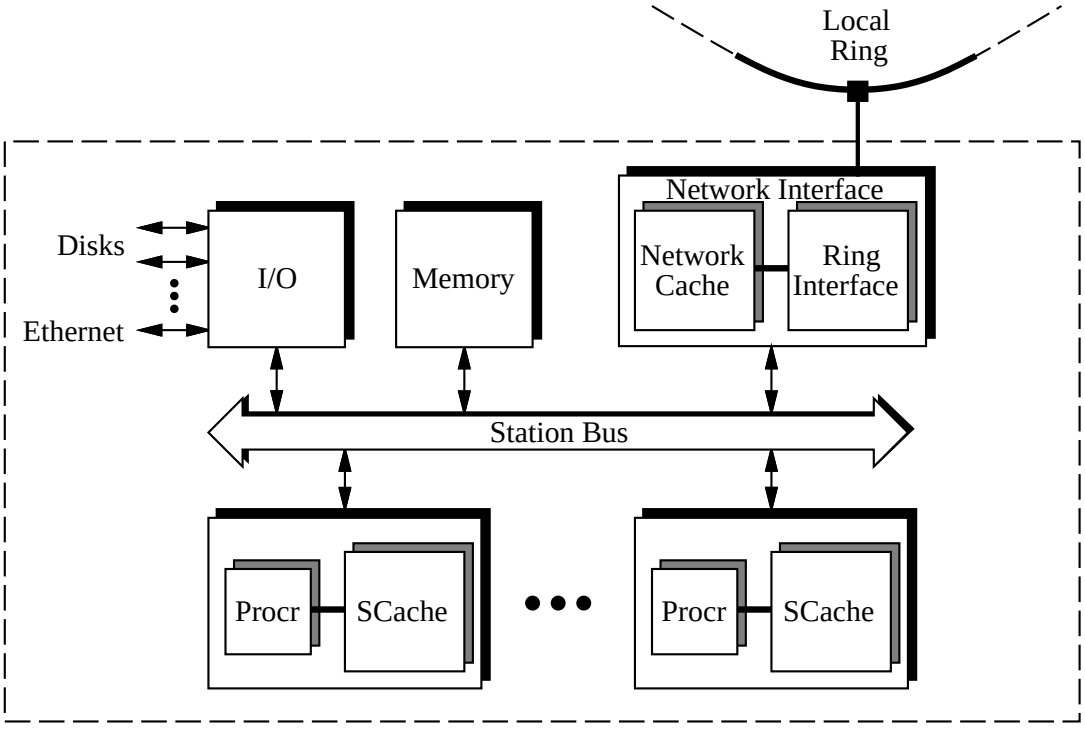

Figure 1.2: Components in a NUMAchine station

NUMAchine is built using the MIPS R4400 microprocessor [Hei94]. The R4400 is a single-issue processor with on-chip primary instruction and data caches, and support for a large external secondary cache. Only one outstanding memory request is supported by the R4400.

However, the design of the backplane and interconnection network can support a future implementation based on the MIPS R10000 microprocessor [MIP94]. The R10000 is a superscalar processor with support for up to four outstanding memory requests, including prefetch requests. The design of NUMAchine includes provisions for request identifiers to support multiple outstanding requests for the same processor.

1.2 Organization of This Document

The purpose of this document is to provide hardware details and maintenance information to support the operation of the NUMAchine multiprocessor.

Part II on page 9 of this document provides hardware details on the following topics:

- command encoding (Chapter 2 on page 9)
- system interconnection] (Chapter 3 on page 15)
- system clocking (Chapter 4 on page 21)
- arbiter card (Chapter 5 on page 25)

- processor card (Chapter 6 on page 27)
- memory card (Chapter 7 on page 45)
- I/O card (Chapter 8 on page 59)
- network interface card (Chapter 9 on page 67)
- inter-ring interface I/O card (Chapter 10 on page 75)

Part III on page 85 of this document provides information relevant for the operation and maintenance of NUMAchine, specifically:

- power distribution and powerup/power down procedures (Chapter 11 on page 85)
- replacement of cards (Chapter **??** on page ??)
- maintenance for programmable logic devices (Chapter **??** on page ??)
- guidelines and procedures for testing (Chapter 14 on page 97)

Further details regarding NUMAchine are found in a number of related documents:

- cache coherence protocol specification [Grb96]
- design of station bus and ring hierarchy [Lov96]
- implementation details for I/O card [Gus96]
- design of monitoring hardware [Lem96]
- principles of operation for system programmers [CGG+97]
- overview of the Tornado operating system [O/S96]

Part II

Hardware Details

Chapter 2

Packet Format and Command Encoding

2.1 Packet Format

The various hardware components of the NUMAchine multiprocessor communicate by exchanging information in units of *packets*. The format of the NUMAchine packet is given in Table 2.1. Each packet contains a set of command bits, and 64 address/data bits. Parity/ECC bits are also provided for error detection.

Table 2.1: NUMAchine packet format

Field	Size (num. of bits)
Command	16
Command parity	2
Address/data	64
Address/data parity	8

There are two types of packets:

- a *header* packet,
- and a *data* packet.

Requests and responses are sent throught the interconnect as a sequence of packets consisting of one header packet, followed by zero or more data packets. The principal packet sequences are given in Table 2.2 on the next page.

For header packets, the address/data field contains an address, specifically a 40-bit *NUMAchine* address augmented with routing information, as shown in Figure 2.1 on the following page. Further details on the interpretation of each field of the address are found in *NUMAchine Principles of Operation for System Programmers* [CGG+97, Chapter 2].

2.2 Command Encoding

The interpretation of the *Command* field of the basic NUMAchine packet depends on whether the packet is a header or contains data. *Bit 8 of the Command field is used to distinguish between header and data packets.*

- For a header packet, the Command field specifies the nature of the associated request or response.

Table 2.2: Principal NUMAchine packet sequences

Packet sequence	Header packets	Data packets
Read request, cached or uncached	1	0
Intervention request for cached data	1	0
Uncached read response	1	1
Cached read response	1	8 or 16†
Uncached write request	1	1
Cached write request (block write or writeback)	1	8 or 16†
Update request for cached data	1	1
Invalidation request for cached data	1	0*

†Note: 8 packets for 64-byte cache lines, 16 packets for 128-byte cache lines.
*The R4400 issues and expects a dummy data packet for invalidations. The dummy packet is dropped by the NUMAchine external agent hardware for outgoing requests, and is reintroduced for incoming requests. The R4400 details are described elsewhere [Hei94, pp. 335,348].

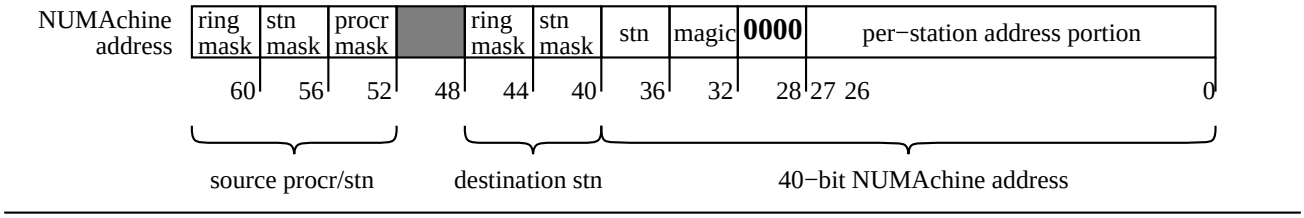

Figure 2.1: Format of address field in header packet

- For a data packet, the Command field contains a *data identifier* to indicate:
  - whether this packet is the last in a series,
  - whether the parity of the data should be checked,
  - or whether the data contains an error.

2.2.1 R4400 vs. R10000 Command Encoding

The command encoding used by the MIPS R4400 and MIPS R10000 differ in the positioning and interpretation of the relevant bits. To permit a future implementation based on the R10000, the NUMAchine command encoding is based on the R10000.

The external agent hardware that is connected to the R4400 system interface translates R4400-encoded commands into R10000-encoded commands, and vice-versa (see Section 6.4 on page 34).

2.2.2 NUMAchine Command Encoding

Encoding for Header Packets

The encoding of the command field for header packets is shown in Table 2.3 on the facing page.

- Bit 8 is 0.

Table 2.3: Interpretation of bits in *Command* field of NUMAchine header packets

Mon. 15..13	Req. Num. 12..10	Special 9	Header or Data 8	Cmd. Type 7..5	Nack 4	NS/S 3	Cmd. Modifier 2..0
	SysCMD[10..8] (*R10000 only*)		0	SysCMD[7..5] (*R10000-encoded*)			SysCMD[2..0]

- Bit 9 and bits 7..3 identify the command; the values of these bits are derived from the SysCMD bits issued by processors (see [Hei94, Section 12.9]).
- The modifier in bits 2..0 is also obtained from the SysCMD bits supplied by processors, and specifies additional information such as the number of bytes to write for a Byte_W operation (see [Hei94, Section 12.9]).
- For future use of the R10000 microprocessor, bits 12..10 contain a unique request number to identify one of several outstanding requests (see [MIP94]).
- Finally, bits 15..13 are reserved for monitoring.

Table 2.4 on the next page provides the NUMAchine encoding for the *Command* field in header packets. Table 2.5 on the following page provides the interpretation of the monitoring bits in the *Command* field in header packets.

Table 2.4: NUMAchine encodings for *Command* field of header packets

Command Name	Hex Encoding for bits 9..0*	Special bit 9	Procr. SysCMD bits 7..5	Procr. SysCMD bits 2..0	Nack bit 4	Req/Res bit 3	Alias Name
R_Req	008	0	000	from procr.	0	1	ITN_S
R_Res	000				0	0	ITN_S_Res
R_Res_N	010				1	0	ITN_S_N
MV_Req	208	1		X	0	1	
RE_Req	028	0	001	from procr.	0	1	ITN_E
RE_Res	020				0	0	ITN_E_Res
RE_Res_N	030				1	0	ITN_E_N
RE_Res_W	220	1			0	0	
SP_RE_Req	228	1			0	1	
IC_R_Req	048	0	010	from procr.	0	1	
IC_R_Res	040				0	0	
IC_R_Res_N	050				1	0	
Byte_R_Req	068	0	011	from procr.	0	1	
Byte_R_Res	060				0	0	
Byte_R_Res_N	070				1	0	
WB	080	0	100	from procr.	0	0	
SP_WB	280	1			0	0	
Byte_W	0A0	0	101	#bytes-1	0	0	
UPGD	0C8	0	110	X	0	1	
UPGD_N	0D0				1	0	
INV	0C0				0	0	
SP_INV (kill?)	2C0	1			0	???	
BC_Res	0E1	0	111	001	0	0	
BC_Inv_Req	2E9	1			0	1	
BC_Inv_Res	2E1	1			0	0	
UPD_Req	0EA	0		010	0	1	
UPD_Res	0E2	0			0	0	
UPD_Res_N	0F2	0			1	0	
Other	???	0/1		$>$010	0/1	0/1	

*If not specified, bits 2..0 are assumed to be zero. Bit 8 *must* be zero for header packets.

Table 2.5: Interpretation of monitoring bits in *Command* field of header packets

bit 15	bit 14	bit 13
Network Cache Hit/$\overline{\text{Miss}}$	Valid/$\overline{\text{Invalid}}$	Local/$\overline{\text{Global}}$

Table 2.6: Interpretation of bits in *Command* field of NUMAchine data packets

Mon. 15..9	Header or Data 8	Mon. 7	Ignore Parity 6	Error Data 5	End of Data 4	Req. /Resp. 3	Cache State 2..0
—	1	—		*R4400:* SysCMD[5]	*R4400:* SysCMD[7]	*R4400:* SysCMD[6]	SysCMD[2..0]
				R10000: —	*R10000:* SysCMD[4]	*R10000:* SysCMD[3]	

Encoding for Data Packets

The encoding of the command field for data packets is shown in Table 2.6.

- Bit 8 is 1.
- The Req./Resp. bit indicates whether a data packet forms part of a request (i.e., a write), or if it is part of a response (i.e., for a read). This bit must be set correctly, otherwise processors will hang when they receive incorrectly-marked packets.
- The end-of-data field indicates that this is the last data packet in a multi-packet sequence (see [Hei94, Section 12.9,]).
- If the Error Data bit is set, it indicates that an error has been detected in this data packet. It is derived from the corresponding R4400 SysCMD bit (see [Hei94, Section 12.9, p.375]).
- The ignore parity bit is set when parity checking is not required for response data in order to prevent processors from taking exceptions.
- The cache state bits are derived from the bits in the R4400 SysCMD (see [Hei94, Section 12.9]).
- The remaining bits are used for monitoring.

Table 2.7 provides the NUMAchine encoding for the *Command* field in data packets.

Table 2.7: NUMAchine encodings for *Command* field of data packets

Command Name	Hex Encoding*
EOD	10X
Data	11X
Error-data and EOD	12X
Error-data	13X
Ignore-parity and EOD	14X
Ignore-parity	15X

*Note: 'X' depends on Req./Resp. bit 3 and Cache State bits 2..0.

Chapter 3

System Interconnection

3.1 Bus Backplane

The bus backplane for NUMAchine is based on the Futurebus+ physical standard. [IEE90]

The majority of signals are simply bussed across all slots, although umber of the signals are reserved for special uses, as described below.

- Five wires are reserved for what Futurebus+ calls a *Geographical Address* (GA). These five bits form a physical address for each slot. The upper four bits number the slots, from left to right, 0 to 13. The least significant bit is a 1 if the slot is wired for a central arbiter, and 0 otherwise. The physical backplane used for NUMAchine only supports a central arbiter in *slot 0*.

- Three wires are connected in a point-to-point manner from all of the slots in the system to the central arbiter slot, i.e., each slot has a set of three pins that go directly to, and only to, the arbiter slot.

- One wire is reserved for a *Reset* signal.

- *Address/Data* and *Address/Data Parity* comprise a total of 144 wires. NUMAchine uses 72 of these bits for Address/Data and Address/Data Parity. A number of the remaining bits are assigned to other NUMAchine backplane protocol specific signals.

Select and Response Select This set of signals identifies the target device(s) for a request, and the target device(s) for the corresponding response. Table 3.2 on page 17 provides the bit assignment for each device.

Source processor/device ID (PID) This set of bus signals identifies the originator of a request. Of the bits 3..0, bit 3 is defined as "response requested." This bit is normally set, but under certain circumstances will be cleared to signify that no response is to be returned to the original requestor.

Bits 2..0 are set as shown in Table 3.2 on page 17.

Busy (sinkable + nonsinkable) Assertion of Busy (sinkable) indicates that a device can no longer accept *any* new incoming packets. Assertion of Busy (nonsinkable) indicates that a device means that the device can *only* accept sinkable packets, i.e., those that will not generate any new outgoing packets. The bit assignment for the Busy signals is the same as for the Select signals, as shown in Table 3.2 on page 17.

Table 3.1: NUMAchine station bus signals

Signal type	Logic	Signal names and sizes				
packet	positive	Sender ID				4
		Command + parity				18
		Response Select				9
		Address/Data	parity			8
			Data	64	address	40
					destination station mask	8
					monitoring phase ID	4
					source processor/device ID	4
					source station mask	8
arbitration	positive	Bus Request				1
		Bus Hold†				1
		Bus Grant				1
		Bus Busy				1
		Busy Sinkable				9
		Busy Nonsinkable				9
		Select				9
miscellaneous	positive	Station/Ring ID				4
	OC (neg)	Bus Reset				1
	OC (neg)	Bus Reset Request				1
	OC (neg)	System Reset Request				1
	positive	Cache Line Size				1
		Memory Present				2
		IO Present				2
	PECL	Bus Clock				2
	positive	Uart Data				1
		Uart Poll Request				1
		GA Test				4
monitoring	positive	Local Ring Busy				1
		Global Ring Busy				1

†In some documentation this signal is referred to as the *long* or *S_L* bit.

3.1.1 Operation of the Bus

At the station level, a split-transaction bus is used to connect the station components. Arbitration, flow control, and device selection are implemented using custom hardware protocols.

Device Select

Devices on the NUMAchine station bus explicitly select the destination(s) for each transaction. For this purpose, the bus provides a set of *Select* lines, one for each device, which must be asserted by the sending device. The appropriate destinations are decoded by the *sending* hardware based on the address, magic, and routing bits in header packets, and the corresponding Select lines are driven. Furthermore, each device continuously monitors its own Select line in order to recognize transactions that it must accept from the bus.

Table 3.2: Detailed bit assignment and encoding of NUMAchine station bus signals

Signal name	Bit(s)	Description	Encoding for source procr/device ID 2..0*
Select and Response Select	0	Memory	000
	1	Network cache	001
	2	Ring interface	001
	3	I/O	010
	4	Processor 0	100
	5	Processor 1	101
	6	Processor 2	110
	7	Processor 3	111
	8	Arbiter	011
Busy (sinkable)	0-8	see assignment for Select/RSelect	—
Busy (nonsinkable)	9-17	see assignment for Select/RSelect	—
Station/Ring ID	0-1	specify station number	—
	2-3	specify ring number	—

*See text for interpretation of source processor/device ID bit 3.

Bus packets with remote destinations are addressed to the Ring and/or Network Cache, and sent across the ring. The *remote* Ring Interface decodes the ultimate destination for the packet, and asserts the appropriate Select lines on the remote bus.

Arbitration and Busy Lines

Bus arbitration is performed automatically in hardware. The arbitration also enforces the NUMAchine flow control and deadlock avoidance protocols. Each device has associated Busy signals for two classes of requests. Sinkable requests may not have a response, whereas non-sinkable requests may have a response. Each device may selectively deny service to one or both of these classes based on its instantaneous buffer capacity.

Every card in the backplane has three dedicated bus arbitration lines: Bus Request, Bus Hold, and Bus Grant. On the backplane, there is one dedicated slot that must be used for the central arbiter. The other slots are available for any card. The bus arbitration lines are implemented as point-to-point connections between the arbiter slot and every slot in the station.

Each arbitration cycle begins with one or more devices having asserted their individual Bus Request signals. At the conclusion of the arbitration cycle, the Bus Grant signal is asserted for exactly one device; only that device may use the bus.

- If a requesting device wishes to issue a cache line to the bus (i.e., a Long transaction consisting of a header and 8 or 16 data packets), the device asserts the Bus Hold signal along with the Bus Request signal at the start of the arbitration cycle. The Bus Hold signal continues to be asserted for the duration of the Long transaction, but is deasserted 3 cycles before the device relinquishes the bus.

- For Medium length transactions the Bus Hold signal must be asserted when requesting the bus, and the de-asserted imediately once the bus is granted (Medium transactions have a header and one data packet). Unfortunately this wastes a few bus cycles, but we have observed slow turnoff on the BTL driveers causing corruption of the first and or last piece of data.

- For Short transactions a requesting device asserts only the Bus Request signal (Short transactions have only a header packet).
- Once the Bus Grant signal is asserted for a requesting device, the bus must be used for the requested transaction size.

It is important that there is one unused bus cycle between transactions to ensure that there is no interference between the BTL drivers turning on and those turning off.

For flow control, bus arbitration relies on two additional control signals from each card on the bus: Busy (Sinkable) and Busy (Nonsinkable). Although bus arbitration is centralized in the bus arbiter card, *the responsibility for checking whether the destination(s) is(are) busy rests with the bus requestor*.

- Each card on the bus always assumes that the destination(s) of a bus transaction is(are) *not* busy, and makes an initial bus request without checking the status of the destination(s).
- Once the bus is granted, the requestor checks if the destination(s) is(are) ready; if so, the request is placed on the bus and the appropriate Select lines are asserted.
- If at least one destination card has the busy signal asserted, no request is issued, and the requestor relinquishes the bus. The requestor then waits until all busy signals are deasserted and repeats the arbitration procedure.

Given sufficient buffer capacity in each card, unfulfilled bus requests should be relatively infrequent.

Finally, because only arbitration is centralized, it is possible for bus requestors to issue requests to nonexistent devices. Writes to nonexistent devices do not cause errors and are accepted because there is no acknowledgement. However, reads from nonexistent devices cause a bus error due to timeout (further details are in *NUMAchine Principles of Operation for System Programmers* [CGG+97].) Bus errors may be caused either by too many negative acknowledgements (likely a coherence error) or timeout (no response from destination).

3.2 Ring Hierarchy

The second level of NUMAchine is implemented using a hierarchy of rings (4 local rings and a global ring). Each ring is a unidirectional, bit-parallel, slotted ring. The segments of a ring between nodes being connected are referred to as slots. These slots can either be full, sygnifying that they contain valid data, or empty signifying that the slot is available for new data. Every ring clock cycle the data moves one position around the ring.

In order to transfer data across the hierarchy, every station and ring needs a distinct address. A station is uniquely identified by the combination of its ring and station number. These numbers are automaticly assigned during a system reset.

Table 3.3: The filtermask bit encoding

Ring bits				Station bits			
R3	R2	R1	R0	S3	S2	S1	S0

When data is sent across a ring, it is partioned into ring packets. A ring packet is a superset of its corresponding bus packet. A ring header is added that contaqins both routing information and tags required for

transaction reassembly at the destination(s). The destination address is usually specified using the *dest* field contained in the address portion of a bus transaction. For some transaction types the packet is always routed to the home station specified by the station bits in the address. When the *dest* field is used the address is specified using an 8 bit mask called a *filter mask*. The bits in the mask can be interpreted as shown in Table 3.3 on the facing page

For further details on the ring interconnect protocol please refer to [Lov96].

The signals used on the local ring are detailed in Figure 3.4. Similar signals are used on the global ring. For details on the implementation of the local ring please refer to Chapter 9 on page 67, and for details on the Global ring please refer to Chapter 10 on page 75.

Table 3.4: NUMAchine ring signals

Signal names and sizes			Description
Header	Free Slot	1	1=slot free for use
	Watchdog	1	Set as packet traverses watchdog/sequencing point
	Sequence Request	1	Forces packet to traverse sequencing point for multicasting
	Destination Mask	8	Identifies stations that have not yet received packet
	Monitoring Select	1	Indicates ring packet is intended for ring monitoring
	Global Free	1	Monitoring option sampling if the global ring is free
	SMA	11	SRAM mapping address for ring interface
	Stop Up	1	Stops packet from being put on the ring by NIC cards. Allows IRI to still send packe
	Stop Down	1	Stops all nodes from putting packets on the ring allowing the NIC cards to drain
	Config	1	Signal local ring configuration
	Error	1	Mark this packet as a ring error packet
	Wen	1	Write this packet to the down FIFO on the next station
Command + parity		18	
Address/data + parity		72	
Total		120	

3.3 Global Ring

3.4 Rack Mounting and Physical Layout

The interconnection hierarchy between NUMAchine stations consists of local rings and a global ring. These stations must be physically positioned in racks such that the required cable lengths are kept short. At the same time, easy access should be provided for maintenance, and cooling requirements should be satisfied.

Figure 3.1 on the following page illustrates a physical rack layout of several NUMAchine stations to form two local rings. To ensure that the cable lengths are kept as short as possible, two stations in each local ring are placed in an inverted orientation in the racks. With this arrangement, the connection between local rings and the global ring is centralized. The inter-ring interface board may physically reside in one of the stations in the center, or it may be separate. The layout shown in Figure 3.1 on the next page consists of 8 stations, each with 4 processors, for a total of 32 processors. A full complement of 64 processors is realized by placing two pairs of racks back-to-back. All connections to the inter-ring interface are made in the center.

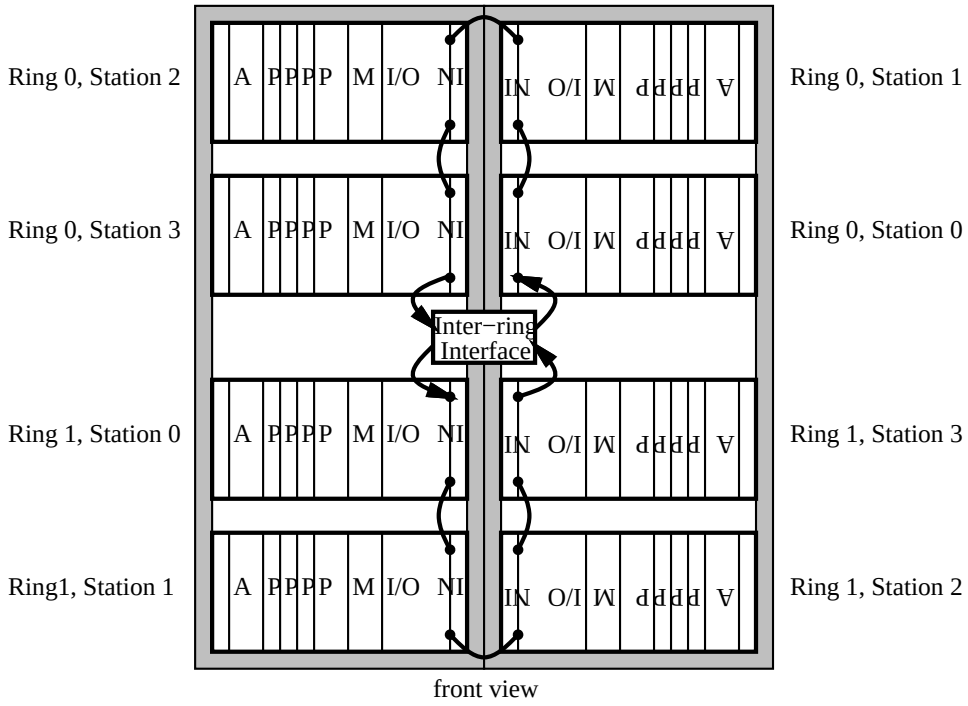

Figure 3.1: Side-by-side rack arrangement to form local rings

Chapter 4

System Clocking

NUMAchine requires clean and precise clock signals to support high-speed synchronous communication. Each card on a station bus requires clock synchronization with the other cards on that bus. Each ring interface connected to a local ring requires clock synchronization with the other ring interfaces on that local ring. However, different busses on the same ring do not require synchronized clocks because the FIFOs in the ring interfaces decouple the ring from each bus. The clock used for the local ring need not be the same clock used on any station bus connected to that ring.

4.1 Clock Generation and Distribution

4.1.1 Station Level

In each station, a clock generation and distribution card generates a Bus Clock whose frequency can be varied between 25 and 50 MHz in small increments using DIP switches. Low-skew copies of the Bus Clock are distributed to each card in the station. Figure 4.1 on the following page provides an overview of this scheme.

- PECL (Positive Emitter Coupled Logic) is used for clock generation and distribution because it provides clean and sharp edges using the same power supply levels as TTL.
- A clock fan-out tree is built on the clock generation board using the National Semiconductor 100310 and MC100EL11 to provide a low-skew distribution of the PECL Bus Clock.
- These PECL clock signals are distributed to each card in the station using differential, terminated, twisted-pair cables of equal length. Each twisted-pair cable is connected to 2 pins on the bus backline for each card slot.
- Each card in the station is then responsible for taking the copy of the Bus Clock that is delivered to it, replicating it in TTL, and distributing it locally on the card. On each card, a fan-out tree is built with MC100EL11 and MC100H641 chips to generate a maximum of 18 TTL copies of the Bus Clock.
- If more local TTL copies of the Bus Clock are needed, on-card PLL (Phase Lock Loop) chips must be used to generate low-skew copies.

4.1.2 Ring Level

The clock distribution for the local rings is similar to that of the stations, except that the ring clock also goes across a catagory five twisted pair cable which is used also for reset and powerup control. Figure 4.2 on page 23 illistrates this.

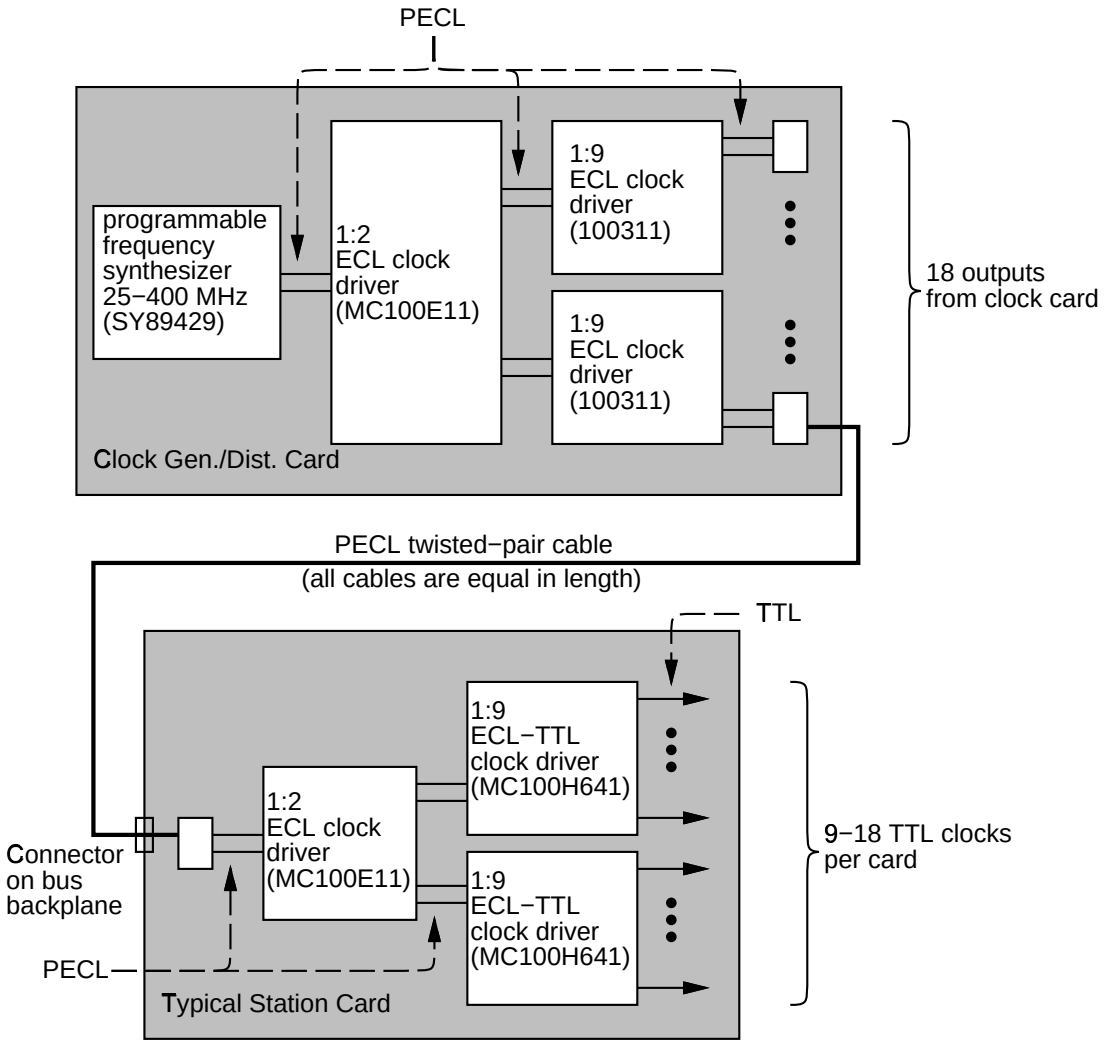

Figure 4.1: Clock generation and distribution scheme within a NUMAchine station

4.2 Board-level Clocking

The clocking scheme used on the system cards differs depending on the requirements of each individual card.

4.2.1 Processor

See Section 6.3 on page 32.

4.2.2 Memory

See Section 7.7 on page 55.

4.2.3 IO

See Section 8.6 on page 63.

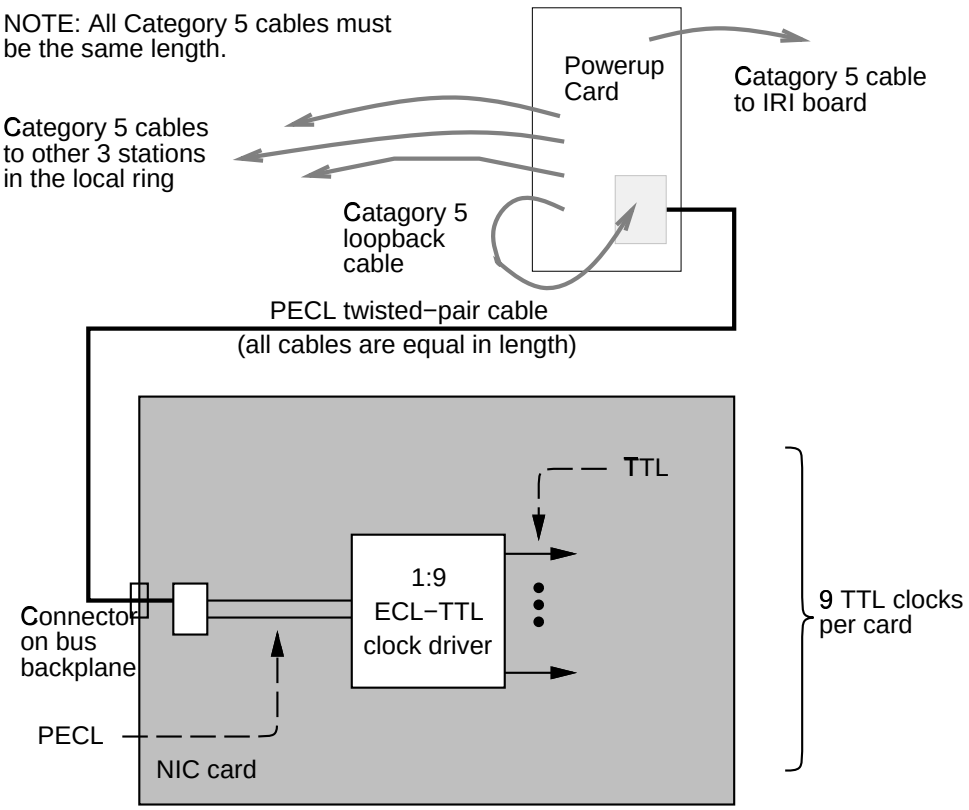

Figure 4.2: Ring clock distribution scheme for a local ring

4.2.4 Network Interface

See Section 9.8 on page 74.

4.2.5 Global Ring

See Section 10.5 on page 81.

Chapter 5

Arbiter Card

5.1 Overview

The arbiter card contains functionality for central arbitration, station level reset, as well as a small test circuit.

5.2 Arbitration

See Section 3.1.1 on page 16.

5.3 Test Circuit

A small number of test circuits are available for testing the function of a NUMAchine bus backplane.

5.4 NUMAchine Bus Interface

The current arbiter card does not interface to the NUMAchine bus so there is no bus control circuitry. This does not, however, preclude future versions of the arbiter card from having functions that are available to the NUMAchine bus — the arbiter slot is completely functional as a NUMAchine slot.

5.5 Bus Debugging Headers

The arbiter card provides a number of headers to be used with the HP logic analyzer for debugging purposes. Figure vreffig:arb.headers shows the location of these pods (viewed from the back of the pcb) and their primary use. Table vreftab:arb.bus.headers details the signals which are connected.

When using these headers the voltage level needs to be adjusted since these headers all provide BTL level signals. We have found that setting the threshold to 1.55V works well. Most commonly the Select, CMD and AD headers are used for debugging.

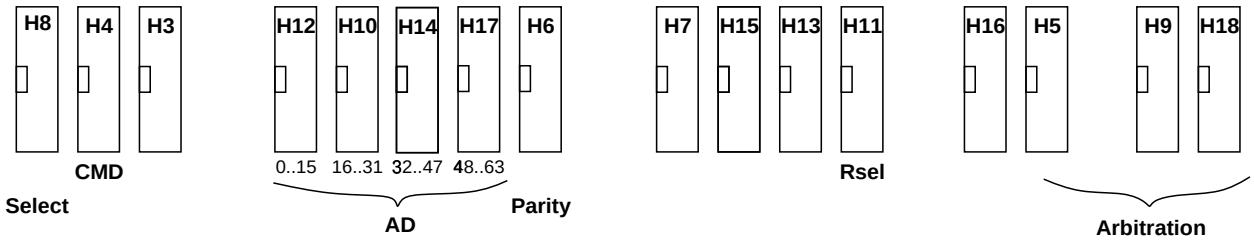

Figure 5.1: Locations of the Bus headers on the Arbiter card

Table 5.1: Bit assignments on the Bus Headers

Header	Name	Bit	Description
H8	Select	3..0	Undefined
		4	Bus Busy
		15..5	Select[8..0]
H4	CMD	13..0	Command[13..0]
		15..14	Command parity[1..0]
H3		1..0	Command[15..14]
		15..2	undefined
H12		15..0	Address/Data[15..0]
H10	AD	16..31	Address/Data[16..31]
H14		32..47	Address/Data[32..47]
H17		48..63	Address/Data[48..63]
H6	Parity	7..0	AD parity[7..0]
		8	Busy Sinkable[8]
		9	Busy Non-Sinkable[1]
		15..10	Obsolete
H7		7..0	Busy Sinkable[7..0]
		14..8	Obsolete
		15	Busy Non-Sinkable[0]
H15		5..0	Busy Non-Sinkable[8..2]
		15..6	Obsolete
H13			Obsolete
H11	Rsel	8..0	Rselect[8..0]
		15..9	Obsolete
H16			Obsolete
H5		0	Bus request test
		1	Short/Long test
		2	Bus Grant test
		15..3	Bus Request[12..0]
H9	Arbitration	12..0	Short/Long [12..0]
		15..13	Bus Grant[2..0]
H18		9..0	Bus Grant[12..3]
		15..10	undefined

Chapter 6

Processor Card

6.1 Overview

The NUMAchine processor card provides one processing node per card. The card includes the R4400 processor, a secondary cache, FIFO buffering, a bus interface and local resources including monitoring and basic I/O. The processor card can be booted from an on-board EPROM, or through the gizmo connector on each card. In the finished system all processors should be able to boot from EPROM. Further details on the processor card, particularly with respect to the address space, are found in *NUMAchine Principles of Operation for System Programmers* [CGG+97].

An overview of the components of the NUMAchine processor card is shown in Figure 6.1 on the following page. The names of the devices reflect the names chosen for the design description files. The key components of the processor card are:

- the R4400 processor and its secondary cache
- the External Agent datapath and controller
- FIFO buffers
- local resources (Local Bus Interface and Monitoring)
- the NUMAbus interface (bus transceivers and controllers)
- the Reset controller

The central component of the processor card is the R4400 microprocessor; a brief summary of its features and behavior is provided in this chapter. Other commodity components on the processor card include the FIFO buffer chips and the BTL bus transceivers; these components are described in Appendix A on page 101.

6.2 R4400 Microprocessor

The R4400 microprocessor is the most complex component on the processor card; considerable detail regarding its behavior is discussed elsewhere [Hei94]. The intent of the summary in this section is to highlight the important aspects which are required to understand the design of the *external agent*, which is discussed in Section 6.4 on page 34. As such, details related to instruction execution are not covered here. Only the signals and protocol for transferring commands and data in and out of the processor and the secondary cache are discussed in this section.

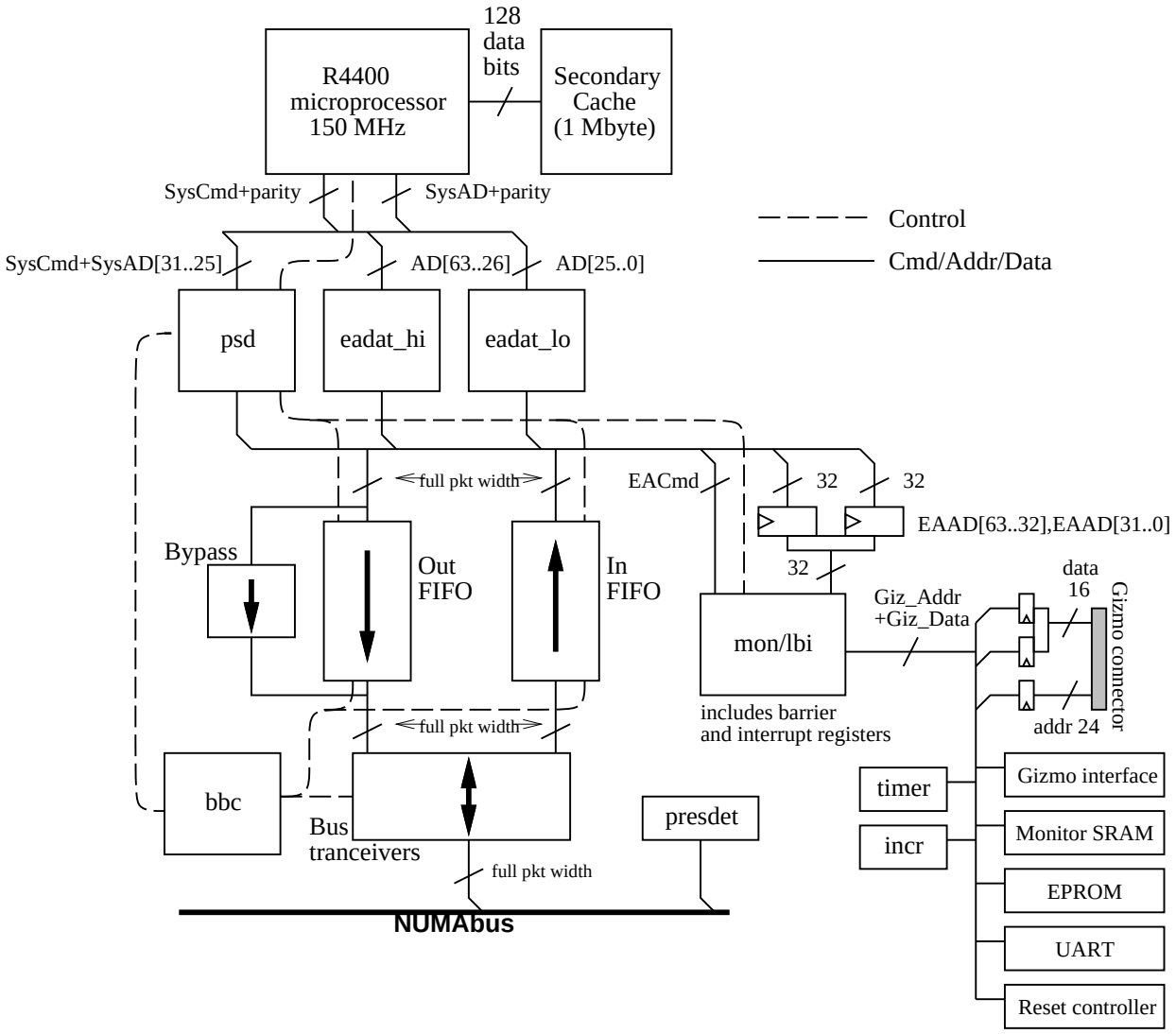

Figure 6.1: Overview of components on the NUMAchine processor card

6.2.1 System Interface

Figure 6.2 on the next page depicts the external system interface of the R4400 processor. These signals connect to the external environment, typically to a collection of logic referred to as the external agent. The system interface has two associated states for the bidirectional signals. In *master* state, the R4400 drives the bidirectional signals, while in *slave* state, the external agent drives them. The system interface protocol describes the conditions under which transitions between these two states occur. In switching between these two states, there is always one idle cycle to permit bus drivers to turn on and off safely.

The description of the system interface signals is given below.

SysAD[63..0] 64-bit bidirectional address/data bus

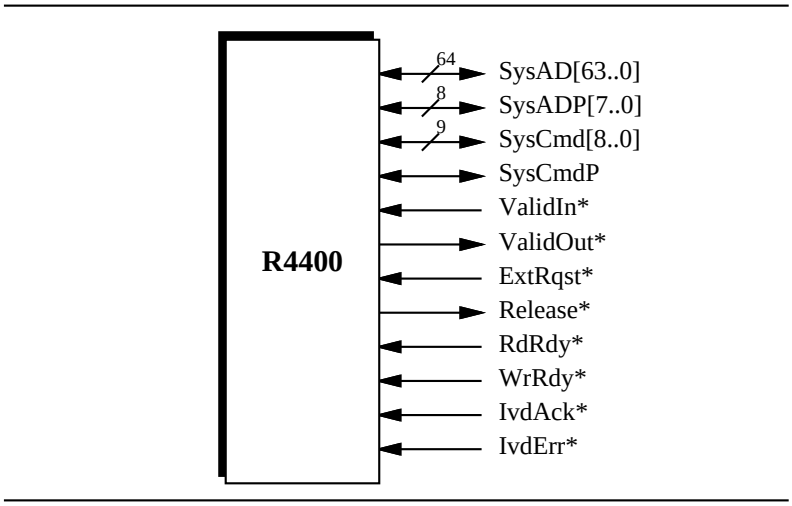

Figure 6.2: System interface of R4400 processor

SysADC[7..0] 8-bit bidirectional check bit bus for address/data; when programmed to operate in "byte parity" mode, these check bits implement even parity checking

SysCmd[8..0] 9-bit bidirectional command/data-identifier bus

SysCmdP 1-bit bidirectional even parity bit for command/data-identifier bus

ValidIn* This signal must be asserted by the external agent on each cycle in which valid address/data along with command/data-identifier is sent to the processor.

ValidOut* This signal is asserted by the processor on each cycle in which valid address/data along with command/data-identifier is sent to the external agent.

ExtRqst* This signal is asserted by the external agent to request a transition to slave state.

Release* This signal is asserted by the processor as an acknowledgment to ExtRqst*. One cycle after it is asserted, the processor stops driving the bidirectional signals and the system interface enters slave state.

RdRdy* This signal serves as an indication from the external agent to the processor that the external agent can accept processor read, invalidate, or update requests. If it is asserted at least two cycles before the processor issues such a request, the processor considers the request to accepted. If the processor asserts a request when RdRdy* is high, it must wait until RdRdy* goes low, then it must wait for two cycles before the request is considered to be accepted by the external agent.

WrRdy* This signal serves as an indication from the external agent to the processor that the external agent can accept a processor write request. If it is asserted at least two cycles before the processor issues such a request, the processor considers the request to accepted. If the processor asserts a request when WrRdy* is high, it must wait until WrRdy* goes low, then it must wait for two cycles before the request is considered to be accepted by the external agent.

IvdAck* This signal serves as an indication from the external agent to the processor that a pending invalidate request has been completed successfully, permitting the processor to continue executing instructions.

IvdErr* This signal serves as an indication from the external agent to the processor that a pending invalidate request has *not* been completed successfully, resulting in a bus error exception.

Specific Behavioral Details

The R4400 system interface does not check parity on incoming command bits. Because address cycles only use 36 bits out of the 64 bits on the SysAD bus, there is also no error detection on incoming addresses. However, proper check bits are generated on the SysADC bus for outgoing addresses [Hei94, pp. 412–413].

External requests from the external agent to the R4400 processor may assert a *cancellation bit* in the command. When the R4400 has an pending invalidate, asserting the cancellation bit forces re-execution of the instruction which caused the processor invalidate request. This re-execution forces a re-evaluation of the cache line state for the data address in question, which may then cause the R4400 to reissue the processor invalidate request, or to issue a read-exclusive request. The read-exclusive request will be issued if the external request resulted in the invalidation of the cache line which the processor was attempting to write [Hei94, p. 336].

There is an undocumented feature of the R4400 regarding the manner in which it responds to external intervention requests. The R4400 always responds to interventions in *sub-block* order, even if the R4400 is configured with the sequential ordering. To ensure that the data in the response is in sequential order, the external agent must reset the least-significant address bits for intervention requests before they are forwarded to the R4400.

The following are important aspects of the system interface protocol regarding transitions between master state and slave state:

- After issuing a read request in master state, the R4400 performs an *uncompelled change to slave state* to await the response [Hei94, p. 301].
- When a uncompelled change to slave state occurs, the external agent must note this transition so that it does not perform arbitration for the system interface in order to send an external request to the R4400.
- *After each external request is issued by the external agent, the R4400 reverts to master state* [Hei94, p. 300].
- When the R4400 is awaiting a read response, and an external request is issued by the external agent, and then completed by the processor, *the R4400 makes another uncompelled change to slave state* to further await the response [Hei94, pp. 301,332].

Some additional aspects which were discovered when testing the first version of the processor card are given below:

- The R4400 seems to use a *counter* when receiving cache lines, i.e., it is not possible to simply send a single 64-bit packet with the end-of-data identifier when the processor is expecting a multi-packet cache line.
- The R4400 expects the cache line state in the data identifier for incoming cache lines to be a *valid* state; if the state is “invalid” or “reserved,” it simply ignores the cache line.

6.2.2 Secondary Cache

The MIPS R4400 processor contains separate on-chip primary caches for instructions and data. However, the capacity of each on-chip cache is only 16 Kbytes. Hence, an external secondary cache of much larger capacity

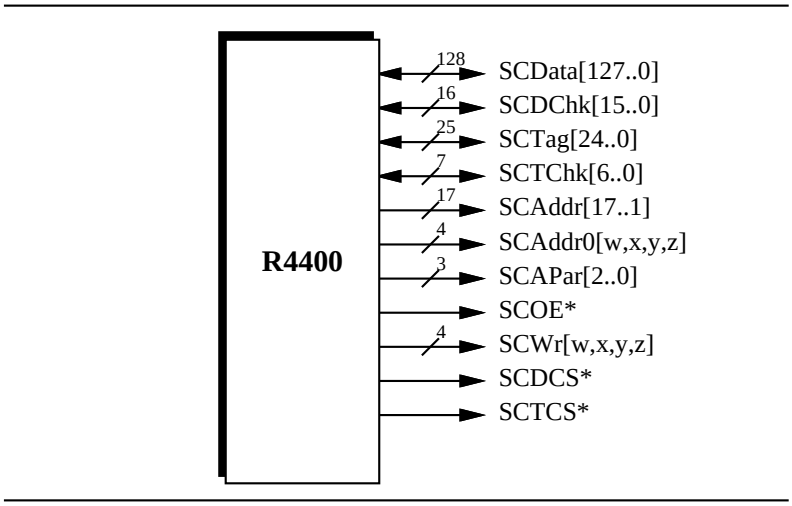

Figure 6.3: Secondary cache interface of R4400 processor

is required to ensure that the R4400 performs well. For the NUMAchine processor card, the secondary cache capacity is 1 Mbyte. The secondary cache is a *unified* cache for both data and instructions (however, it may be configured as a split data/instruction cache with appropriate programming). More details on the secondary cache interface of the R4400 are described elsewhere [Hei94].

The R4400 secondary cache interface is shown in Figure 6.3. The full description of signals is given below:

SCDAta,SCDChk 128-bit data and 16-bit ECC busses; both are bidirectional

SCAddr,SCAddr0 18-bit address bus; bit 0 is replicated to provide additional drive capability

SCWr data/tag write enable signal; replicated to provide additional drive capability

SCTag,SCTChk 25-bit tag and 7-bit ECC busses; both are bidirectional

SCOE* output enable for data/tag chips

SCDCS* data chip select

SCTCS* tag chip select

SCAPar parity bus for SCAddr, SCDCS*, and SCTCS*

The SC control signals generated by the processor should not *be terminated.* The data lines are not terminated. The address lines are buffered.

The 1-Mbyte secondary cache capacity for the NUMAchine processor card is provided by a number of SRAM chips, namely TC551664-15 64kbit$\times$16 chips from Toshiba with 15-nsec access times [Tos94]. The capacity of each chip is 1 Mbit or 128 Kbytes, so 8 chips are required for cache data, and one for the associated parity.

The 25-bit tag and its 7-bit ECC are stored in the tag RAM; the total number of bits per tag is 32, which is 4 bytes. At least 2 SRAM chips are required for the 32-bit tag. The secondary cache line size is programmable at boot time. For NUMAchine, there are two choices: 64 bytes and 128 bytes. The maximum number of tags

Table 6.1: Timing parameters for SRAMs on NUMAchine processor card

Parameter	Number of PCycles
TWrSUp	3
TWr2Dly	1
TWr1Dly	1
TWrRC	0
TDis	2
TRd2Cyc	3
TRd1Cyc	4

results with 64-byte cache line, requiring a capacity of (1M/64)$\cdot 4 = 65,536$ bytes, which easily fits into 2 SRAM chips. Hence a total of 11 chips are required for both data and tags.

A number of boot-time programmable parameters govern SRAM access timing (see [Hei94]). For the SRAMs used in the NUMAchine processor card, the parameter settings are provided in Table 6.1. The unit ⟸ ⋆ of time is PCycles of the PClock, whose frequency is twice the external system clock of the R4400.

[NOTE: Parameters may not be correct. Currently more conservative numbers are used.]

6.3 Clocking

The processor card utilizes three independent clock signals:

- a local clock for the R4400 processor,
- a global clock from the NUMAbus,
- and a local clock for the reset circuitry.

There are two additional clock signals:

- the output *System Clock* or *SClock* derived from the R4400, which drives most of the external agent circuitry,
- and the external *Gizmoclock* that drives the Gizmo side of the local bus interface.

Figure 6.4 on the facing page illustrates the clock generation and distribution on the processor card.

The R4400 processor accepts a single clock input (MasterClock) from which it derives its internal *PClock* for processor logic operation, and several output clock signals. The most important output is the *SClock*. The processor can also generate separate read and write clock signals for the external interface logic, with programmable skew between these signals. However, this feature is not used in the current implementation.

6.3.1 Clock Synchronization

[a very important issue for the interface circuits which needs to be discussed??]

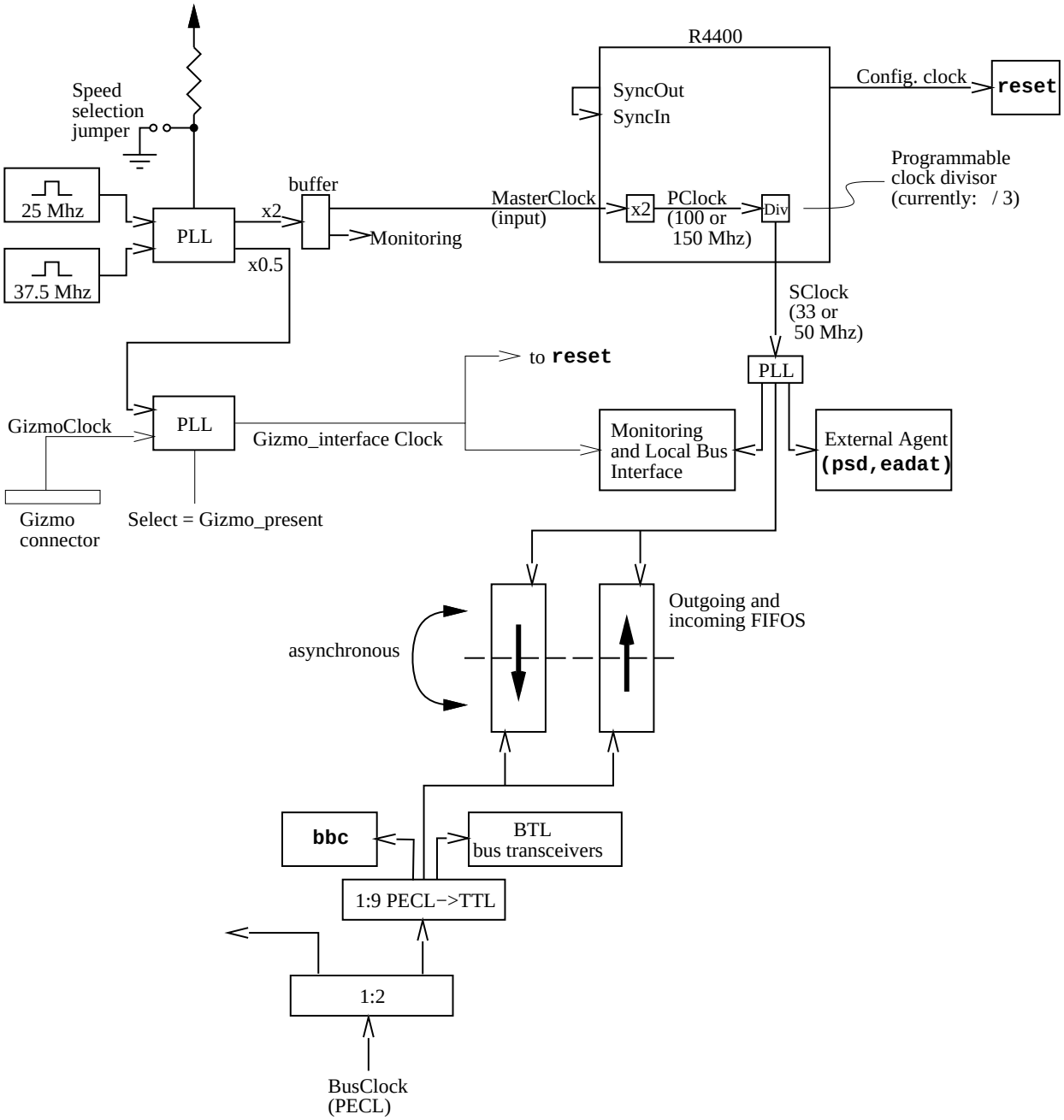

Figure 6.4: Clock generation and distribution on processor card

Processor clock synchronization

Processor clock synchronization requires a low clock skew. For that purpose, there are several clock distribution chips available, and the processor internally has a phase-locked loop that synchronizes its clock inputs and outputs.

[more details about the actual clock distribution circuitry??]

Synchronization Among Independent Clocks

Since there are several independent clock sources, there is a possibility for metastability. To remove (or at least decrease) this possibility, the following techniques are used:

1. Data interchange through FIFOs use double-latched status signals.
2. Asynchronous handshaking is used for circuits that interface between the processor-derived clock and the bus clock.
3. Multiple latching is used for signals that interface Gizmo bus to the processor card local bus.

Because independent clock sources can be tuned independently, the circuits contending with multiple clock sources must be possess some degree of flexibility. Currently the BBC and PSD controllers take special precautions to ensure that they function correctly when one side is working at 33 MHz and the other at 50 MHz. Caution must be taken because one side may try to sink data faster than it is being sourced if false assumptions are made.

6.4 External Agent

The external agent (henceforth abbreviated as EA) is the key component of the NUMAchine processor card, as it is the interface between the R4400 processor and the NUMAbus that is the connection to the rest of the system. The EA provides the following functions:

- From the processor side, the EA accepts R4400 memory read/write requests and forwards them to the appropriate destinations.
- From the NUMAbus side, the EA accepts responses to R4400 read requests and forwards them to the waiting processor.
- From the NUMAbus side, the EA accepts external intervention, invalidation, or write requests, and forwards them to the processor.
- From the processor side, the EA accepts R4400 responses to external intervention requests and forwards these responses to the NUMAbus.
- For both the processor side and the NUMAbus side, the EA provides access to local resources (e.g., monitoring, Gizmo interface) on the processor card.
- In all cases involving communication between the processor and bus sides, the EA performs the necessary translation of the address space and command encoding (R4400$\leftrightarrow$NUMAchine).

The EA consists of: (1) a bidirectional datapath between the processor and the external environment (the FIFOs from/to the NUMAbus), and (2) a controller to coordinate data transfers through the datapath.

The controller is implemented in a single Altera chip called /emph psd. The *psd* controller conforms to the processor system interface handshaking protocol for accepting processor requests/responses and sending external requests/responses to the processor (see Section 6.2 on page 27). On the external side of the datapath, the controller must sample the status of the FIFOs and control read/write operations from/to the FIFOs. Processor requests can only be accepted if there is space available in the outgoing FIFO. Conversely, if the

incoming FIFO is non-empty, then the controller must arbitrate for the processor system interface in order to forward external requests and responses to the processor. The controller implementation consists of two state machines, one for the processor side, and one for the FIFO side.

The data path is implemented in 2 Altera chips called `eadat_lo` and, `eadat_hi`. The `eadat_hi` chip performs address bit manipulation in order to translate between R4400 and NUMAchine address spaces.

6.4.1 Data-path

A simplified view of the EA datapath is illustrated in Figure 6.5. On the processor side, the datapath connects to the R4400 64-bit SysAD bus. On the external side, the datapath connects to the 64-bit EAAD bus. The figure is a simplified illustration of the buffering and multiplexing employed for data transfer between the SysAD and EAAD busses; the address bit manipulation that is performed in `eadat_hi`) is not shown.

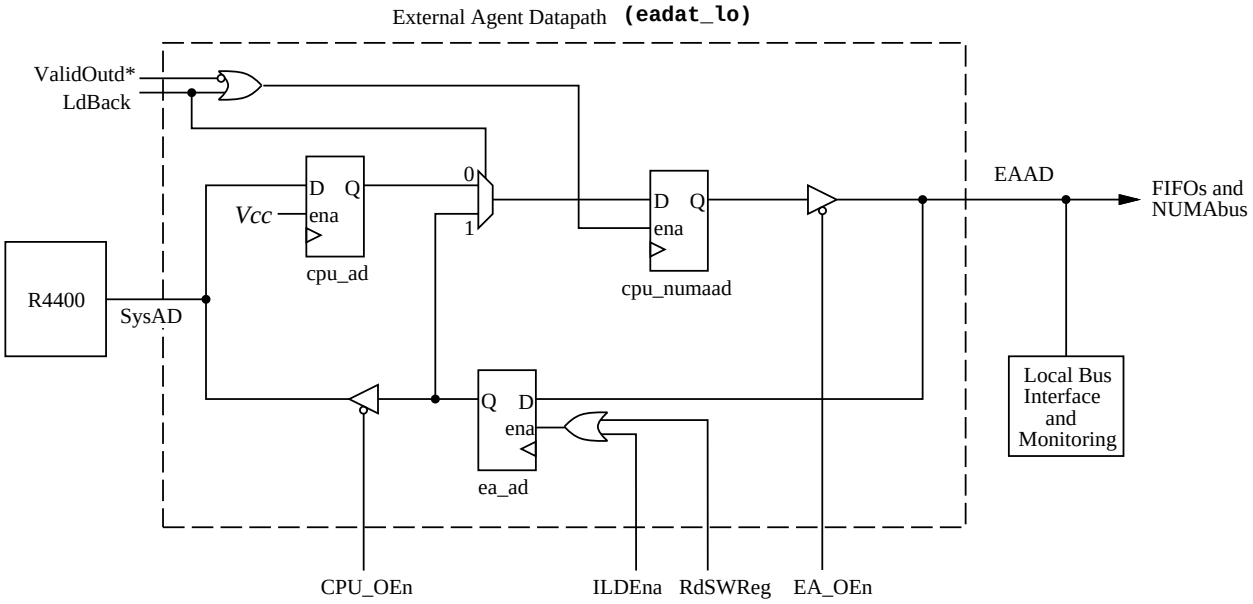

Figure 6.5: External agent datapath

In the implementation, the 64 data bits from the processor SysAD bus and the associated 8 check bits from the SysADC bus are divided among two Altera programmable logic devices. Table 6.2 on the next page provides the assignments of data and check bits to particular devices in the EA implementation. All of the check bits are assigned to one device. Although there is parity generation and checking for data packets, there is none for address packets, and the external agent datapath (specifically `eadat_hi`) manipulates the address bits. For data packets, there is no manipulation. In either case, the parity bits are passed through unchanged.

Outgoing Path

Two levels of latches are required for outgoing address/data (R4400→system) to permit sufficient time to decode and manipulate commands and addresses as they pass through the datapath. Outgoing read, write, and upgrade requests proceed from the SysAD bus through the *cpu_ad* and *cpu_numaad* latches to the outgoing

Table 6.2: Assignment of SysAD bits to Altera devices

Device Name	Data Bits	Check Bits
eadat_hi	SysAD[63..26]	—
eadat_lo	SysAD[25..0]	SysADC[7..0]

FIFO chips. Read and upgrade requests are placed in the outgoing FIFO bypass mailbox register, while write requests are placed in the outgoing FIFO itself. Outgoing processor data responses also proceed through these latches. Requests to Local Bus Interface and Monitoring resources do not enter the FIFOs, and are instead routed to the appropriate local resource.

The first level latch for outgoing traffic is *always* enabled. The second level latch connected to the EAAD bus is normally enabled with a one cycle delay on *ValidOut** from the R4400. This one cycle delay is provided by the signal *ValidOutd**, which is generated by the psd component (the datapath controller). The second level latch for EAAD is also enabled when there is a loopback from the incoming latch *ea_ad*.

Incoming Path

For the incoming path (system→R4400), the single input latch is enabled by the ILDEna signal. For eadat_hi, which performs address bit manipulation, the D-inputs to the input latch *ea_ad* and the output latch *cpu_numaad* have more complex combinational logic to perform the required manipulation discussed in Section 6.4.2.

Incoming external requests/responses from the incoming FIFO are latched into *ea_ad*. Responses from Local Bus Interface and Monitoring also pass through the same latch.

Loopback Path

A special feature to note in Figure 6.5 on the page before is the presence of a loopback path for the EAAD bus, through the *ea_ad* latch, and into the *cpu_numaad* latch. This loopback path is required for external intervention requests. An incoming intervention request is forwarded to the R4400 from the incoming FIFO, and also copied through the loopback path to be placed in the outgoing FIFO as the header packet for the intervention response.

Path for Commands

The EAAD bus also has an associated EACMD bus that corresponds to the SysCmd bus of the R4400. The datapath for commands is similar to the datapath for the address/data portion (i.e., two latch levels for outgoing commands, one latch for incoming commands), *except that there is no loopback path for commands*. The command bits do not go through the same Altera devices as the address/data bits. Instead, the command datapath is part of the psd controller device due to the close relationship between command decoding for control purposes, and the need to manipulate command bits to interface between the R4400 and the NUMAchine system.

6.4.2 Address Bit Manipulation in Datapath

Normally, incoming and outgoing data is passed unchanged through the EA datapath. However, when addresses are passed through the datapath, some bit manipulation is required to translate between physical addresses generated by the R4400 processor and the system-wide NUMAchine physical address space, as well as to introduce the routing information maintained in the high-order bits of an address (see Figure 2.1 in

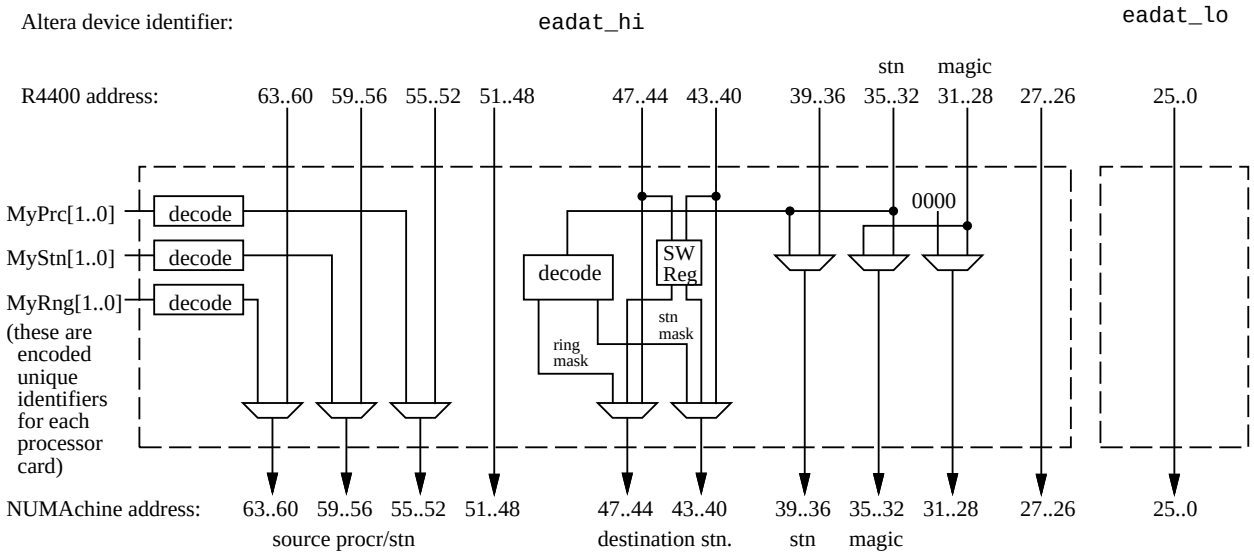

Figure 6.6: Manipulation of outgoing address bits in external agent datapath

Chapter 2). The logic implemented in the Altera devices for the datapath performs this function. Figure 6.6 illustrates the manipulations performed in the datapath for *outgoing* addresses, identifying the devices and address bit ranges. The multiplexers permit passing data through unchanged for normal data cycles, or performing the required manipulation for address cycles.

Note that in `eadat_hi`, an additional source for the destination ring/station masks is a special software-programmable register. For *incoming* addresses, the reverse manipulation is performed to transform a NUMAchine physical address into an R4400 physical address. The logic for the reverse manipulation is straightforward and not shown here, involving a simple shift from the 40-bit NUMAchine physical address to the 36-bit R4400 physical address. The high-order bits are left unaltered, as the R4400 does not use or check them for incoming addresses (see Section 6.2 on page 27).

6.4.3 State Machines in Controller

The `psd` controller for the EA datapath consists of two state machines:

- `cpu_sm`: the state machine for controlling processor-side data transfers,
- `fifo_sm`: the state machine for controlling FIFO-side data transfers.

Processor-side State Machine

The process-side state machine (`cpu_sm`) accepts requests and responses from the R4400 processor and sends them to the outgoing FIFO or to local resources on the processor card. The state machine remains in an idle state until it detects that the processor is attempting to issue a memory request, as indicated by the assertion of the ValidOut* signal on the system interface. The machine then moves into a decode state to determine the nature of the request and to take the appropriate action. In each state, the state machine samples or controls signals for both the R4400 and the outgoing FIFO. It must be reiterated that the outgoing

EA datapath consists of two levels of latches. As a result, the sampled signals and control signals are also latched for proper sequencing with the address/data flowing from the R4400 to the outgoing FIFO.

Responses from the R4400 for external requests are handled by the FIFO-side state machine.

FIFO-side State Machine

The FIFO-side controller accepts external requests and responses from the incoming FIFO or local resources on the processor card and forwards them to the R4400 processor.

For external interventions, the FIFO-side state machine arbitrates for the system interface, forwards the ⟸ ⋆ intervention to the processor, then awaits the response from the processor.

[add more details as appropriate]

6.5 NUMAchine Bus Interface

The NUMAchine Bus Interface connects the processor board FIFOs to the bus transceivers on the NUMAchine station bus. The specifications for the NUMAchine Bus Interface are as follows:

- provide a bidirectional datapath between FIFOs and the NUMAchine bus,
- perform additional encoding and bit manipulation that cannot be performed by External Agent (e.g., command bits in intervention responses),
- permit reconfiguration to handle different cache line sizes (in the same manner as the `psd` controller in External Agent),
- generate Select bits for devices connected to the NUMAchine bus,
- handle bus arbitration and handshaking, and provide Busy signals to indicate the status of the processor board for incoming requests,
- and provide a retry mechanism with backoff for negatively-acknowledged requests.

The implementation consists of a controller called `bbc` that incorporates the following components:

- a *Send* state machine for outgoing requests and responses,
- a *Backoff* module that implements a binary exponential backoff retry mechanism,
- a *Timeout* module for unacknowledged requests,
- and a *Receive* module that accepts incoming requests and responses.

6.5.1 Datapath

The datapath in the NUMAchine Bus Interface manipulates bits in the command field of outgoing data packets, as shown in Figure 6.7 on the facing page. The datapath in the `bbc` controller completely replaces the EACMD bits with the BusCMD bits. One reason for this is the Nak/Resp bits which are generated by the `psd` controller and passed on to the `bbc` controller because they cannot be put in the command field by the External Agent datapath. The `bbc` controller also uses address bits to generate Select lines for the devices on the NUMAbus. Finally, some bits from *incoming* commands are read in order to control the operation of the Receive state machine (see Section 6.5.3 on the next page).

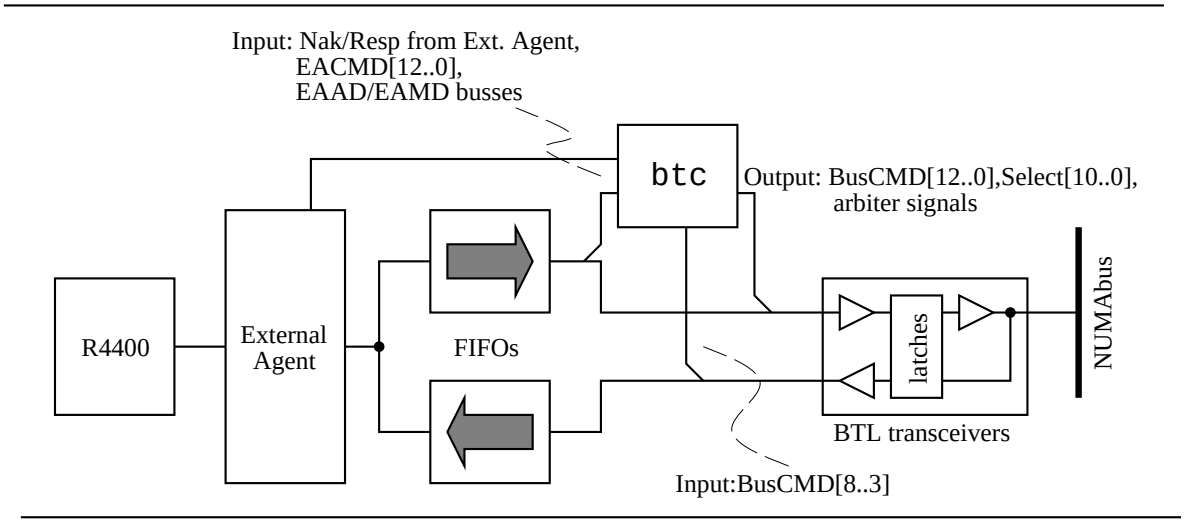

Figure 6.7: NUMAchine Bus Interface datapath

6.5.2 Send Machine

The Send machine in the `bbc` controller monitors signals from the `psd` controller of the External Agent and the flags of the outgoing FIFOs to determine when there is data to send to the NUMAbus. Priority alternates between read/upgrade (short) requests in the outgoing mailbox register, and write requests or processor responses (long) in the outgoing FIFO buffer. A bus request is made when the Send machine detects a outgoing data in the mailbox or FIFO. After the bus grant is received, the data is sent to the bus if the destinations are free. If the destinations are not free when the bus is granted, and a read/upgrade request is pending, the request priority is allowed to alternate in order to permit any write requests or processor responses in the FIFO to be sent.

The Send machine calculates the device Select signals for outgoing data in order to: (a) determine if the destinations of the data are free, and (b) route the data to the appropriate destinations when the data is sent on the bus. This calculation is based on bits in the command and address fields.

The Send machine is more complex. First, outgoing data must be examined to determine if it will involve a *short* or *long* transaction on the NUMAbus, and if all the data for a *long* transaction is in the FIFO, ready to be sent. In so doing, the Send machine must support different cache line sizes for long transactions. Furthermore, the Send machine must determine the destinations for the data. Next, an appropriate bus arbitration request is made, and the data is sent onto the bus as soon as a bus grant is received, provided that the destinations are able to accept the data.

The Send machine must also interface with the `bak` controller which provides the support for a retry mechanism with backoff for outgoing requests that require a response to the processor. If a timeout occurs or if the maximum retry count is exceeded, a bus error is sent to the processor.

6.5.3 Receive Machine

The Receive machine of the `btc` controller is trivial. There are only two states: *idle* and *rx*. When the processor board is selected during a bus transmission by another device on the NUMAbus, the machine switches from the idle state to the rx state. The machine remains in the rx state so long as the board is selected or until an end-of-data indicator is detected in the incoming command bits.

The control over the incoming FIFO in order to accept data from the bus is actually located in the `bak` controller. This controller monitors the Select lines for each packet on the bus, comparing it to the bit pattern that corresponds to the processor board. When the patterns match, the FIFO is enabled, and the data from the bus is transferred to the FIFO buffer. The `bak` controller also monitors the almost-full flag for the incoming FIFO; when there is insufficient capacity, it asserts a busy signal indicating that the processor board is unable to accept incoming data.

6.5.4 Backoff/Retry Mechanism

Read and upgrade requests from the processor require a response from the system. It is possible for the destinations of such requests to reply with negative acknowledgments, in which case it is necessary to support a retry mechanism. It is also possible that the destinations of such requests may not respond at all, in which case a timeout mechanism is required. Both mechanisms are implemented in the `bak` controller using a number of counters.

If the timeout counter should expire before a response is received to a read/upgrade request, a bus error is sent to the processor. If a negative acknowledgment is received prior to a timeout, the request is reissued. The interval between issuing retries increases using a binary exponential backoff function. If the maximum number of retries is exceeded, a bus error is sent to the processor.

6.6 Local Bus Interface and Monitoring

The Local Bus Interface (LBI) permits access to local resources on the processor card, including:

- local EPROM memory,
- the Reset Mechanism,
- the interface to the *Gizmo* [PVL92], a 68000-based single-board computer,
- local monitoring resources.

The LBI is responsible for decoding addresses it receives from the External Agent to select the appropriate local resources. An overview of the LBI connections is illustrated in Figure 6.8 on the facing page.

Between the External Agent and the LBI, transceiver chips are used to buffer the data on the EAAD bus, as shown in Figure 6.9 on page 42.

6.6.1 Datapath

The LBI connects the EAAD bus on the External Agent side to the Local Bus connected to the Gizmo interface. Multiplexing is required due to the differences in data widths on either side of the LBI and because the Local Bus side has separate address and data lines. There are two levels of multiplexing.

- The 64-bit EAAD bus first is reduced to a 32-bit bus using transceiver chips, as illustrated in Figure 6.9 on page 42. Pullups are required on the transceiver output-enable lines to ensure that data lines are not driven by multiple chips at powerup when the LBI chip is being programmed.
- On the other side of the LBI chip, the 32-bit *data* bus is then reduced to the 16-bit width needed for the Gizmo interface.

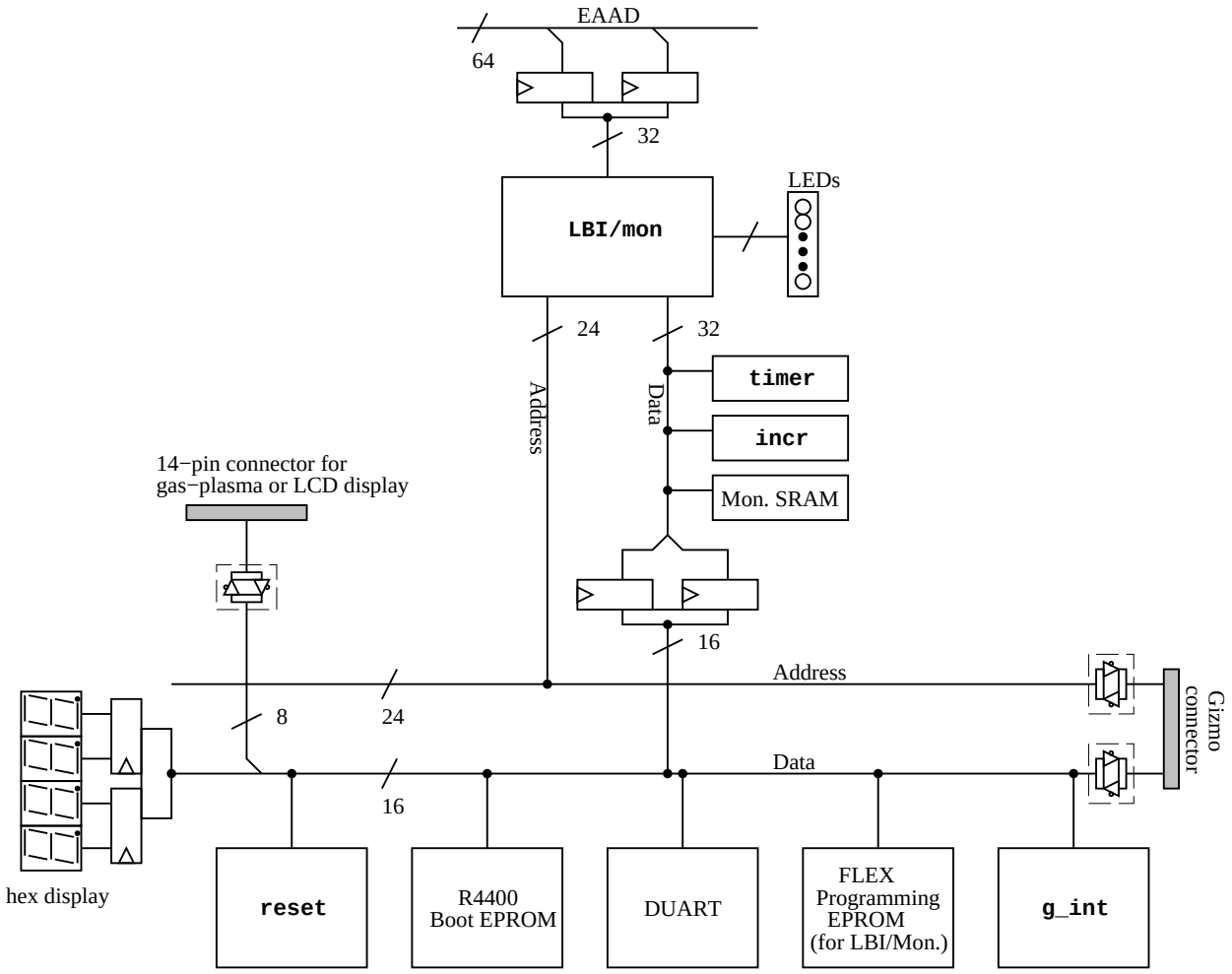

Figure 6.8: Overview of Local Bus Interface connections

6.6.2 Reset Controller

The Reset controller provides the following functions:

- controls the reset sequence for the R4400,
- provides control over R4400 boot-time configuration,
- inhibits accesses to the Gizmo interface on request from the R4400,
- controls loading of values into the hex display registers,
- provides control and status for the DUART on the processor card.

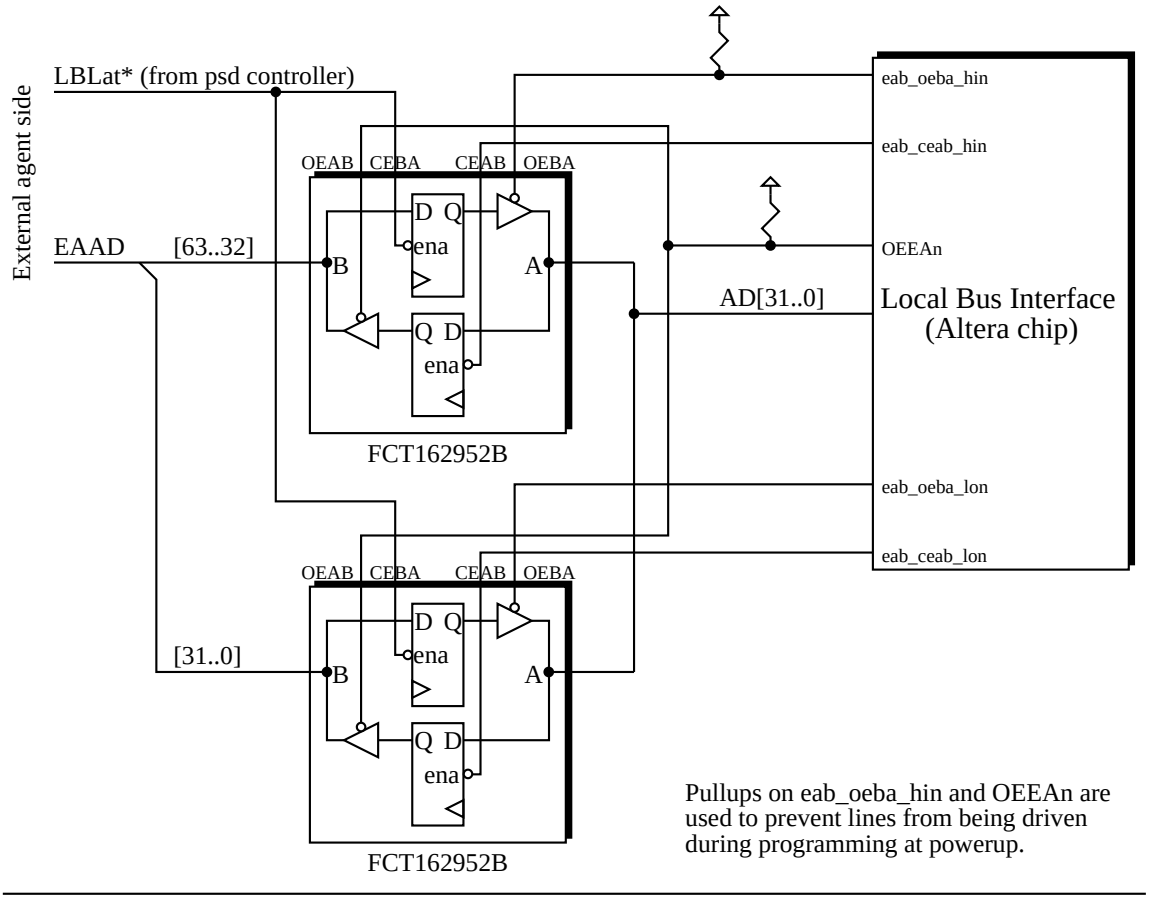

Figure 6.9: Transceivers connected to the Local Bus Interface

6.6.3 Gizmo Interface

The Gizmo interface supports both 32- and 64-bit transfers. The 64-bit external agent data path is multiplexed down to the 16-bit data width on the Gizmo bus (with the 24-bit address multiplexed separately).

There is a special signal *LBI_inhibit* that controls R4400 access to the Gizmo bus. This signal is asserted when the Gizmo is being reset to prevent the Gizmo bus from hanging. It may also be asserted/deasserted explicitly by the R4400 through a special write to the Reset controller.

6.6.4 Monitoring

6.6.5 DUART

The DUART on the processor card provides two ports intended for debugging data. Port A is connected to a multidrop serial line via an on-board connector, and Port B is connected to a common serial bus on the backplane. The intent is to use port A for initial debugging, and then once the system is quite stable to use port B, through the I/O card to collect debugging information. If port A is to be connected to an RS232 device the small adaptor board must be used. In addition to providing multiple ports there is some status information available on the general purpose input ports on the DUART. These alternate uses are detailed in *NUMAchine Principles of Operation for System Programmers* [CGG+97]

6.6.6 Hardware Displays

7-Segment LED Display

A 4 character 7-segment LED display is available on the local bus for use by software. The addressing is detailed in *NUMAchine Principles of Operation for System Programmers* [CGG+97]

Status LEDs

Table 6.3 describes the status LEDs on the processor card.

If the FLEX programming light should fail to flash on reset, it is most likely because the Gizmo bus is hung.**[is this correct?]** ⇐ ⋆

Table 6.3: LEDs on processor card

Number	Color	Description
1	red	FLEX programming in progress*
1	green	`presdet` is ready†
1	green	proc_id[1]
1	green	proc_id[0]
1	green	Gizmo access(programmable)
1	green	interrupt(programmable)
1	green	EA activity(programmable)
5	red/yellow/green	programmable‡

*This should flash on momentarily after reset.

†This should flash off momentarily after reset.

‡This is currently a “chaser” whose speed increases with increased External Agent activity.

Multi-character Display

A connector conforming to a standard multi-character display interface is available on the local bus for use by software.

Chapter 7

Memory Card

7.1 Overview

The NUMAchine memory card stores data and provides the primary hardware support for maintaining data coherence. The memory card maintains a directory to record the current state and location(s) of data in units of cache lines, and serves as a sequencing point for coherent data requests from processors. For each request, the memory card uses the state information to select an appropriate course of action: respond with data from memory, issue an intervention request to retrieve valid data from another processor, or issue invalidation requests to other processors in response to an exclusive data request from a processor. The memory card also accepts writebacks from processors when cache lines are ejected from caches due to replacement or explicit flushing.

In addition to supporting operations on cached, coherent data, the memory card also supports noncoherent and uncached operations. Noncoherent requests still involve data transfers in units of cache lines, but coherence is not enforced by the hardware, and the directory information need not be valid. Uncached operations involve data transfers in units less than the cache line size, i.e., bytes, words, and double-words. Coherence not enforced for uncached operations.

Finally, the memory card also supports a number of special functions. External manipulation of the directory information is supported to enable system software to initialize the directory, or alter its contents in an application-specific manner. This feature is useful for I/O and noncoherent requests. Provisions are also made to apply operations on ranges of cache lines in the directory to reduce software overhead. Support is included for generating interrupts to signal completion of range operations or error conditions to system software. Finally, explicit broadcasts or multicasts of cache lines may be initiated to send new data to other stations throughout the system.

An overview of the components of the NUMAchine memory card is shown in Figure 7.1 on the following page. The names of the devices reflect the names chosen for the design description files. The key components of the processor card are:

- the DRAM data memory, interleaved to increase data bandwidth
- the SRAM directory memory for the coherence protocol
- FIFO buffers
- the NUMAbus interface (bus transceivers and controllers)
- the Master controller

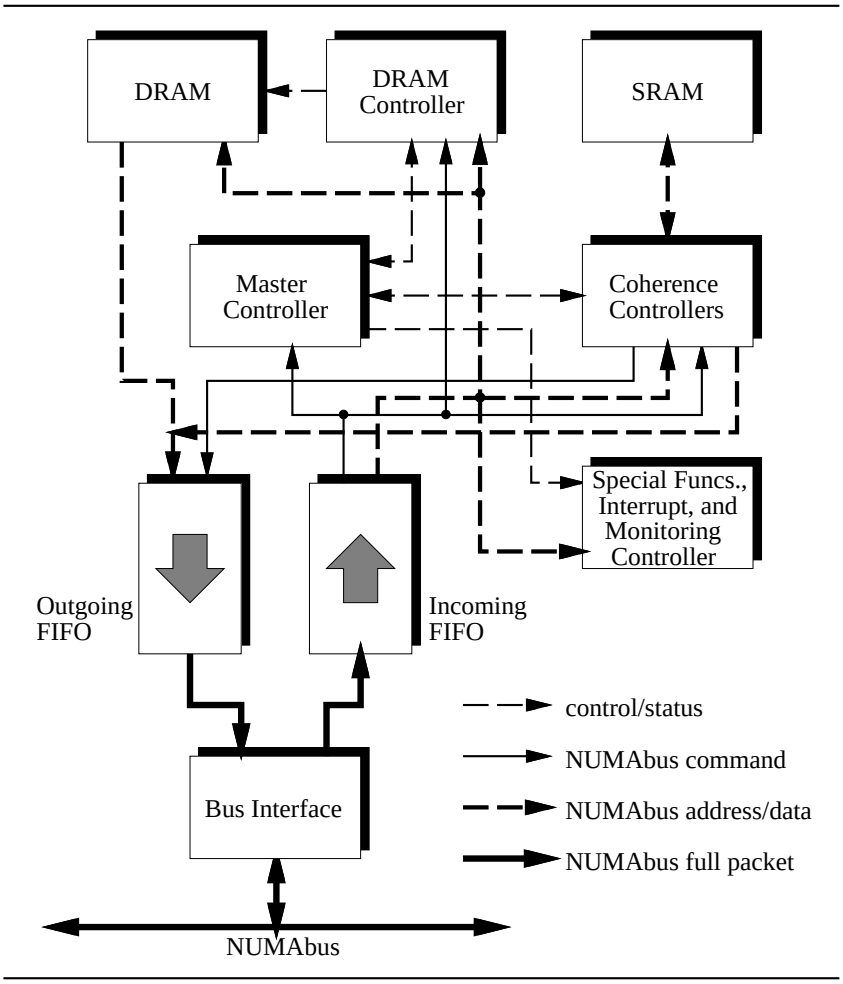

Figure 7.1: Overview of components on the NUMAchine memory card

- the Cache Coherence controllers
- the DRAM controller
- the Special Functions, Interrupt, and Monitoring controller

The design of the control units is such that requests are pipelined as much as possible to improve performance. For example, the Master controller may begin processing a new request after handing off the preceding request to the Cache Coherence controllers or DRAM controller. Of course, the new request may have to wait if its resource requirements conflict with the preceding request.

The commodity components found on the memory card include the FIFO buffer chips and the BTL bus transceivers; these components are described in Appendix A on page 101.

7.2 DRAM Controller

The DRAM controller reads or writes memory under the control of the Master controller. The DRAM controller is optimized for cache line transfers with 2-way interleaving. Each leaf is 128-bits wide, but data from

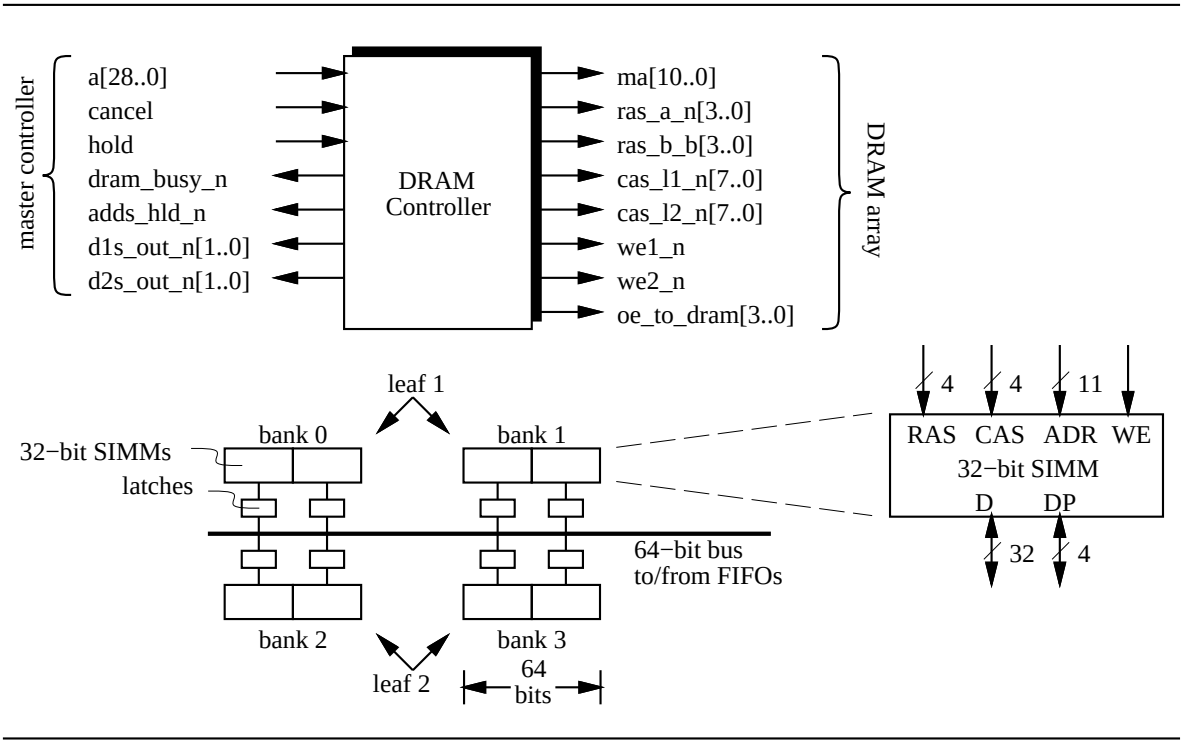

Figure 7.2: Controller and DRAM array interconnection

each leaf is transferred in units of 64 bits (a double-word). By switching between leaves, data may be read or written in consecutive 50 MHz clock cycles in 64-bit units. This provides effectively a 4-way interleaved, 64-bit wide memory, as seen by the master controller.

In addition to cache line transfers, the DRAM controller supports byte, word, and double-word accesses to memory. In particular, the DRAM controller supports unaligned memory accesses arising from the use of the LDL/R and SDL/R instructions of the MIPS R4400 [Hei94, chap. 2].

The DRAM controller initiates data transfers at the request of the Master controller. The two controllers employ a handshake to ensure that data transfers begin only one or the other is ready to accept the data. The handshake is required to ensure that the DRAM timing is satisfied.

7.2.1 Interconnection

Figure 7.2 illustrates the signals between the controllers and the DRAM array organization. The two leaves of the DRAM array are connected to the data bus through a set of latches to match the 128-bit data width of the DRAM leaves with the 64-bit datapath connected to the FIFOs.

Figure 7.3 on the next page illustrates the physical layout of the SIMMs on the memory card, relating subsets of bits with the leaves and banks for interleaving.

7.2.2 Handshake with Master Controller

Figure 7.4 on the following page illustrates the handshake protocol with the master controller for read requests. Figure 7.4 on the next page(a) shows the handshake for normal DRAM accesses, and Figure 7.4(b)

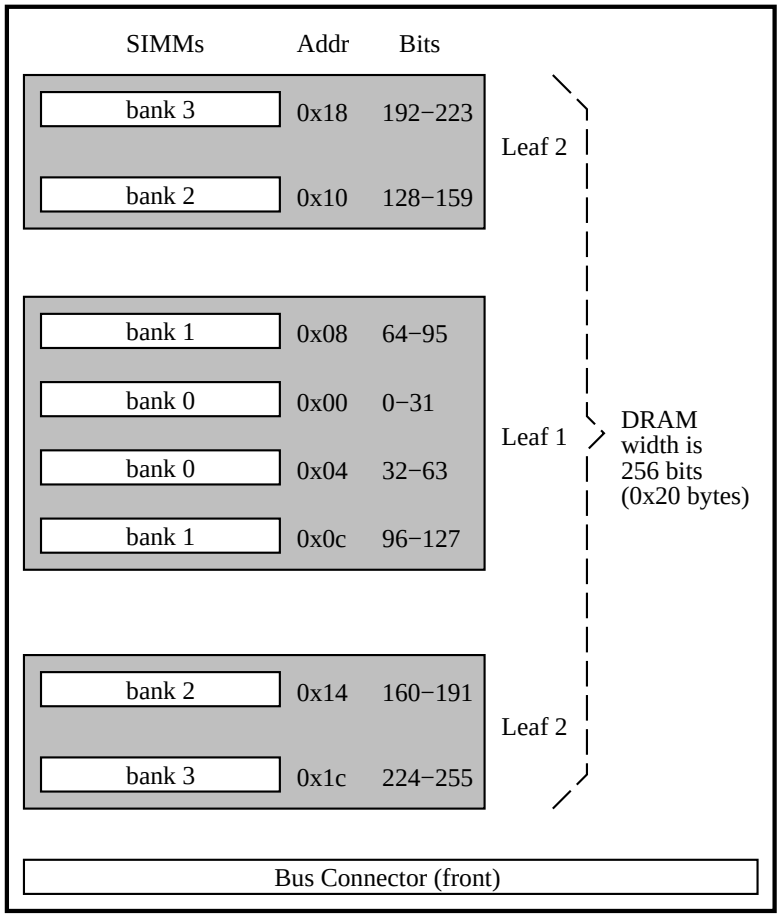

Figure 7.3: SIMM layout on memory card

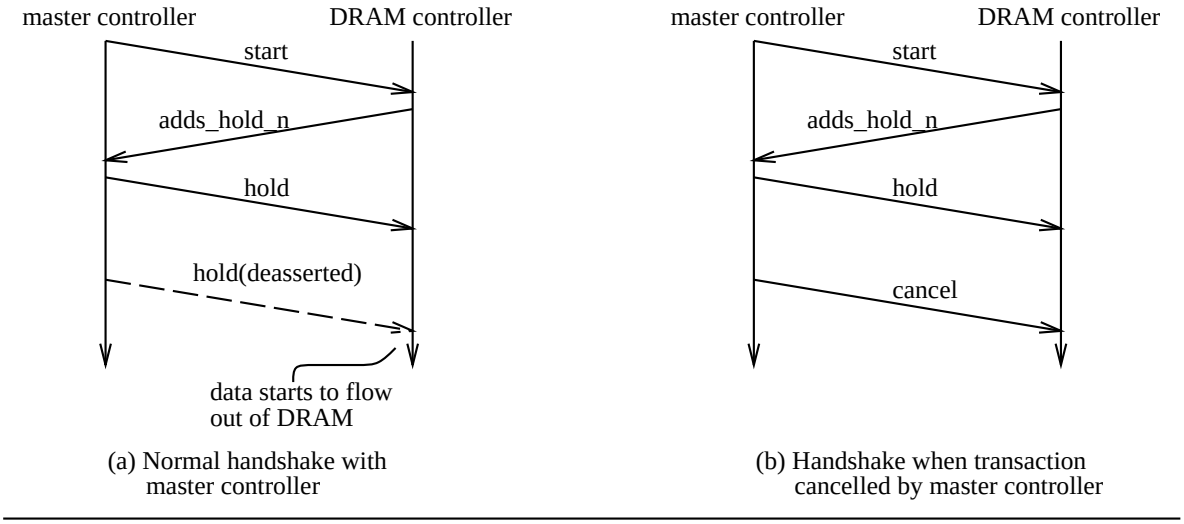

Figure 7.4: Handshake between DRAM and master controllers

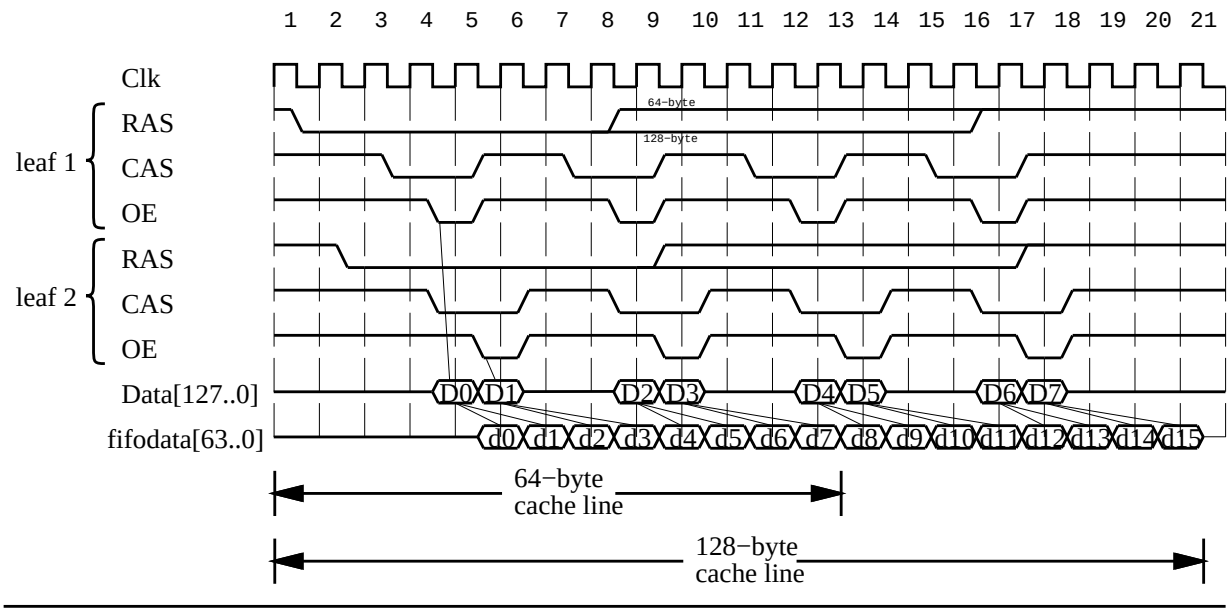

Figure 7.5: DRAM access timing for cache line reads

shows the handshake when the Master controller cancels the current memory transaction (i.e., when the directory lookup indicates that the valid copy of the data is located in a processor cache). The *hold* signal is asserted by the Master controller when the required resources are available, namely the datapath to the FIFOs.

On memory write requests, the Master controller must wait until the *dram_busy_n* signal is De-asserted by the DRAM controller.

The *d1s_out* and *d2s_out* signals (for leaf 1 and leaf 2 respectively) indicate that data from the DRAM array is valid on read. There is one bit for each bank in a leaf. These signals notify the Master controller to write the data from the DRAM into the outgoing FIFO (hence, these signals should *only* be asserted when data from the data is valid).

For writes, the Master controller must assert *data_in* to indicate that the stream of write data to the DRAM array is ready.

7.2.3 DRAM Access Timing

Figure 7.5 illustrates the basic timing for interleaved DRAM accesses when reading cache lines. A total of 128 bits is read from each leaf into a set of latches. Then, the data is moved 64 bits at a time into the FIFOs. The data flow is reversed on cache line writes, but the timing for accessing the DRAM leaves is similar.

7.3 Cache Coherence Controller

The coherence controllers of the memory board are responsible for enforcing the NUMAchine cache coherence protocol. Collectively, they perform two important tasks:

- cache line directory lookup and maintenance,
- response packet generation.

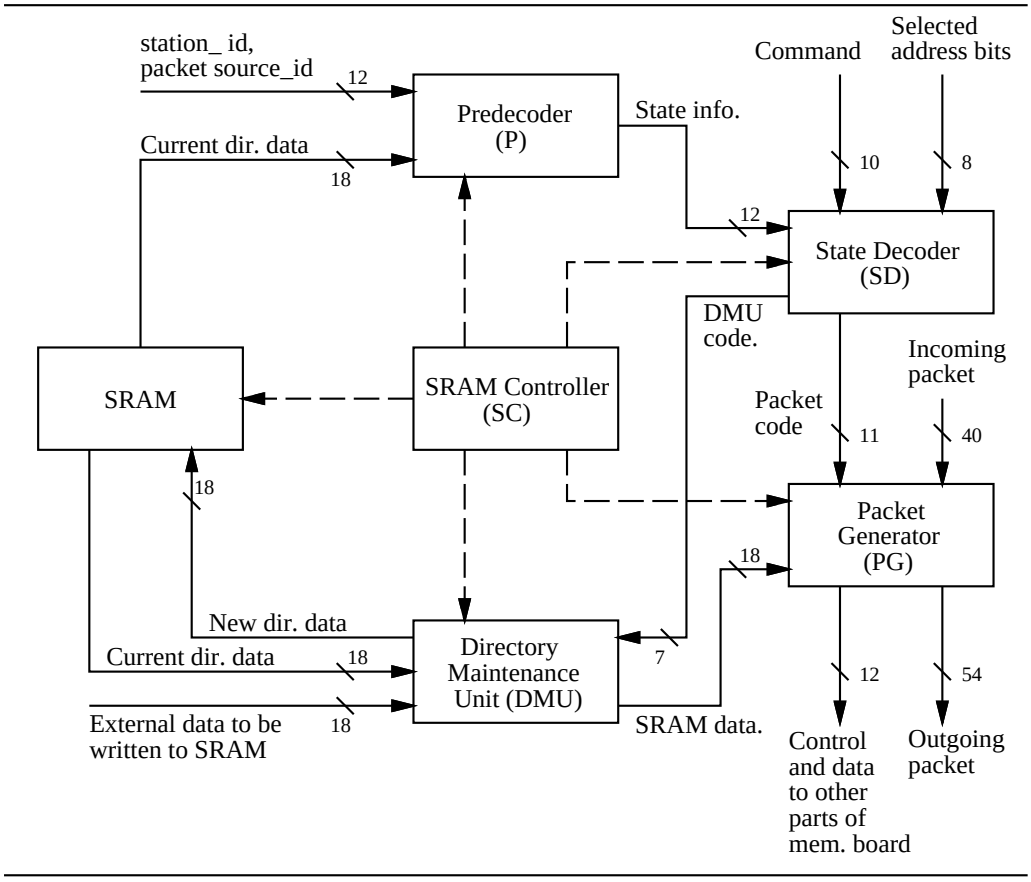

Figure 7.6: Interconnection of Coherence Controllers

Figure 7.6 illustrates the relationships between the coherence controllers and the input/output signals.

7.4 Master Controller

The Master controller is implemented in two logical parts. Mast_in which controls the incoming FIFO, and the routing of its request, and Mast_out which controls the outgoing FIFO and interfaces to the DRAM controller.

7.4.1 Incoming Controller (Mast_in)

The incomming controller, or *Mast_in*, routes the packets from the incomming FIFO to their appropriate destination based on their address. Figure 7.7 on the facing page shows the datapath controlled by the *Mast_in* controller.

Some important details regarding state transitions:

- The normal path IDLE→RD_FIFO→FIFO_HOLD. . . extracts commands/data from the incoming FIFO and sends them to the coherence controllers and DRAM.
- The alternative path IDLE→*SFGNT*→RD_FIFO→FIFO_HOLD. . . is taken when the request originates from the special functions controller. In this case, the command is *not* read from the FIFO, but

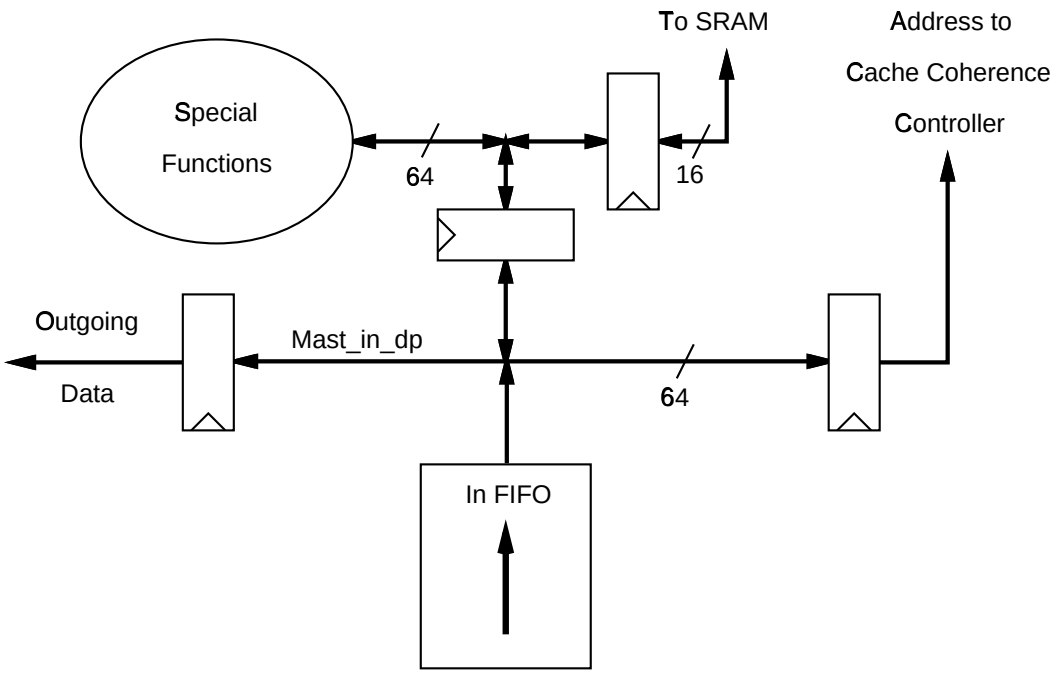

Figure 7.7: Data Path controlled by Mast_in

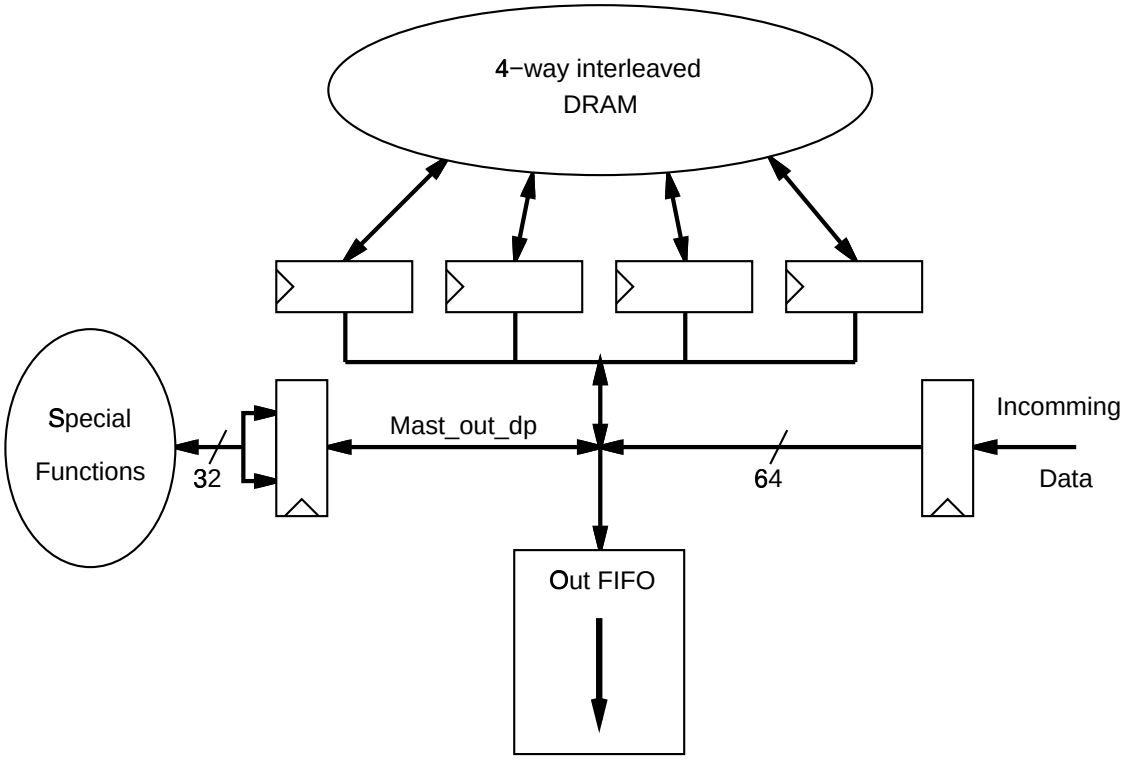

Figure 7.8: Data Path controlled by Mast_out

from the special functions unit. However, the same states are traversed in either case (except for the path through SFGNT). A special flag (InFifo_OE_n) is used to disable the incoming FIFOs when the source of the request is the special functions controller.

7.4.2 Outgoing Controller (Mast_out)

The Outgoing Controller transfers data to the DRAM and the Special Functions unit, as well as ready data from them. Its primary responsibility is to control the many buffers connected to the outgoing FIFO. Figure 7.8 on the page before shows this portion of the data path.

An important note about the control of the outgoing FIFO writes, is that they are inferred in most cases from the output enables of the buffers in the datapath.

7.5 Special Functions and Interrupt Controller

Each memory card in the system contains a *special function unit* to perform block operations. This unit performs one or two special functions on a contiguous block of up to 256 cache lines. Special functions are initiated by writing the special function registers on the Memory board. Completion of special functions is signaled with an interrupt. The operations supported by the special function unit are:

- block write of SRAM tags
- block kill-copy (removes copies of cache lines throughout the system before physical memory pages are replaced by I/O initiated by the operating system)
- block obtain-valid-copy-in-memory (brings any remote dirty copies of cache lines to memory)
- block memory move (must do obtain copy first)
- block zero region

Writing the special function register that contains the command initiates the special function process. All special function processes are interruptible at cache-line granularity by the master controller on the memory card; the master controller processes special functions only when there are no pending requests.

The special functions unit contains a number of registers as outlined in Table 7.1 . The address for Memory card special functions and interrupts is currently AD[31..24] = 0x4c.

The Special Functions and monitoring unit (*SF_mon*) is connected through registers to both the mast_in and mast_out data paths as illistrated in figure 7.9 on the facing page. The connection to the mast_in data-path facilitiates monitoring of the transactions being processed, as well as depositing special function transactions on to the mast_in data path for processing.

The connection to the mast_out data path allows for reading and writing of *SF_mon*, and the monitoring SRAM, as well as sending out interrupt packets. Since *SF_mon* is an FPGA it must be programmed after powerup. The current design reprogramms *SF_mon* once on reset. The *sf_ctl* CPLD controls the bus between *SF_mon* and the mast_out data path. It is also responsible for programming *SF_mon*. The option also exists to reprogram *SF_mon* from the Monitoring SRAM when different monitoring options are desired.

7.6 NUMAchine Bus Interface

The NUMAchine bus interface acts as an intermediary between the memory card FIFOs and the NUMAchine bus itself. It provides the following functions:

Table 7.1: Registers for special functions and monitoring in the memory card

Addr 11..0†	Registers			
	63..48	47..32	31..17	15..0
000	SF status register [R]		status, HW config. [R]	monitoring config. [R/W]
100	CMD2 [W]	CMD1 [W]	SF Rsel [W]	phase1 cnt-1 [W] / phase0 cnt-1 [W]
200	SF reg1 (Add1) [W]			
300	SF reg2 (Add2/Data) [W]			
400	Error Address (AD1) [R]			
500	——			Intr_vector0 (Error) [R]
600	Monitoring init. data [R/W]		SF done intr. [R/W] station / device(s)	Intr_vector1 [R/W] (special functions)
700	——		Mon. interrupt [R/W] station / device(s)	Intr_vector2 [R/W] (monitoring)
800	HW Special Function lock register, returns the same data a s Addr 000 [R]			

†With more than one memory card, the LSB of a cache line address selects the card.
[R]=read-only, [W]=write-only, [R/W]=read/write

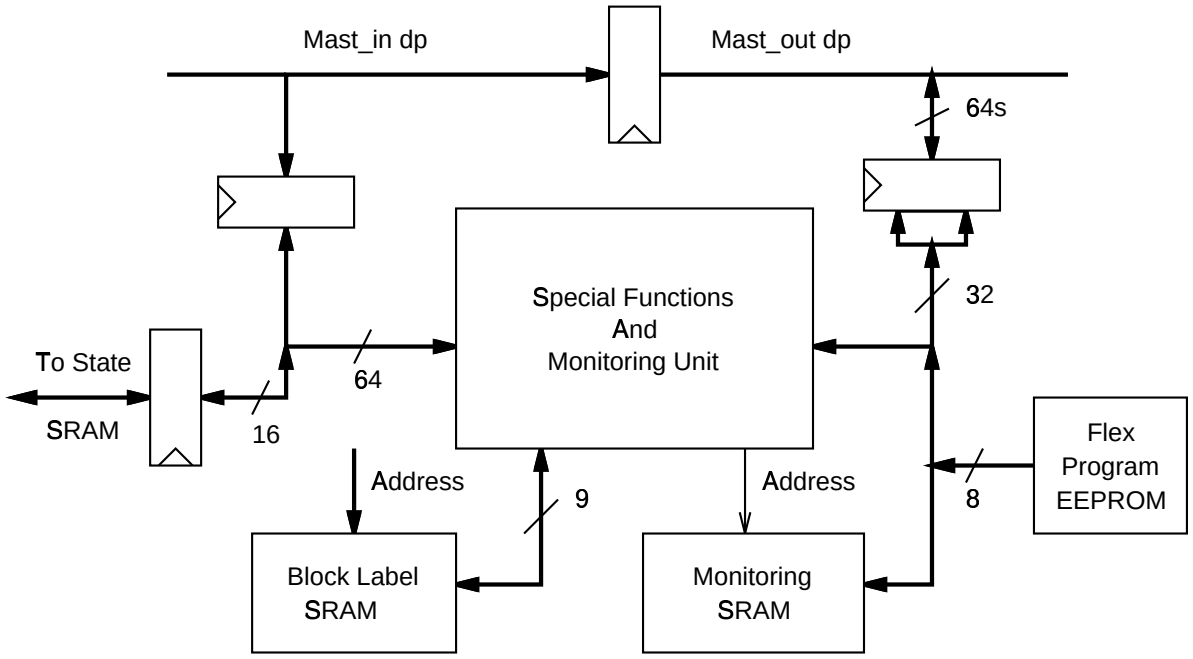

Figure 7.9: Data Path Connected to the Special Functions Unit

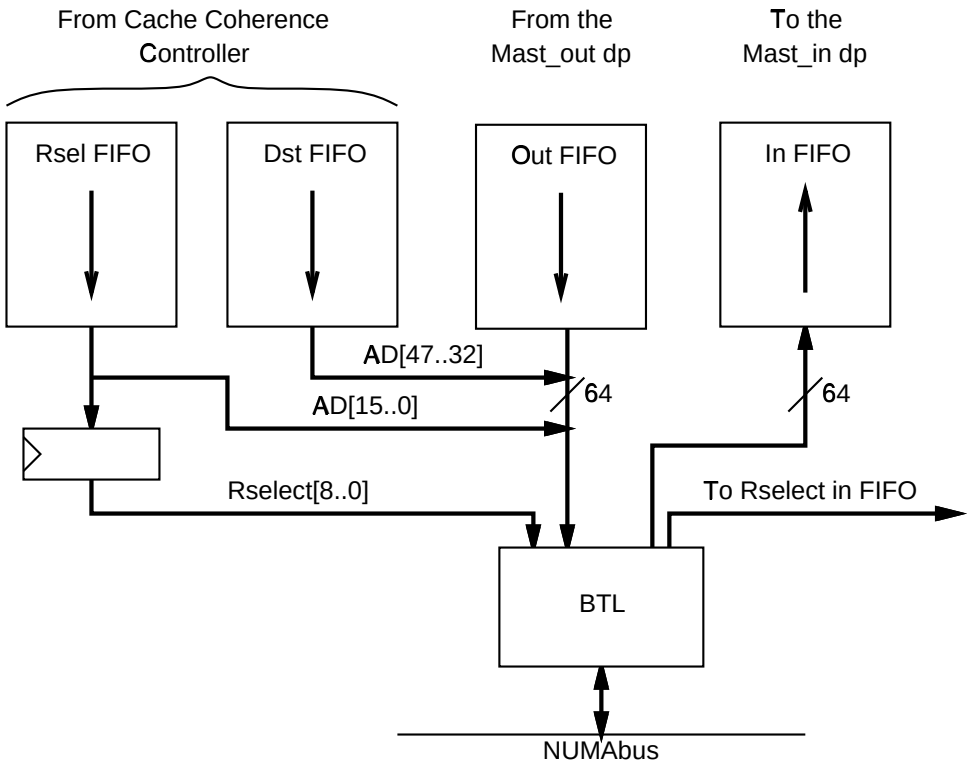

Figure 7.10: Data Path Connected to the NUMAbus

- Accepts NUMAbus packets and places them in the incoming FIFOs.
- Watches when packets are placed in the outgoing FIFOs and sends them appropriately to the proper destinations.
- Determines number of memory cards on a local station.

7.6.1 FIFO to NUMAbus

The FIFO to NUMAbus control is implemented in *Buscon*. This controller has the followiing primary functions:

- Transfer packets from FIFOs to bus transceivers.
- Perform proper arbitration procedures. (See Section 3.1.1 on page 16.)

There are multiple sources for the outgoing data as illistrated in Figure 7.10. The data comming from the cache coherence controller allows reading of the state SRAM, and also it makes it possible to modify the *dst* bits in the address when the remote routing needs to be changed. The CMD datapath is similar, but their is only one source for the command. It is important to note that for transactions longer than one packet, only two entries are written into the command FIFO. The first one is the command itself, and the next is a sample data identifier, without the *EOD* bit set. The *buscon* is responsible for duplicating it, and setting the *EOD* bit when nescessary.

7.6.2 NUMAbus to FIFO

The NUMAbus to FIFO control is implemented in *incon*. This controller has the following functions:

- Transfer packets from bus transceivers to FIFOs that select this memory card.
- Proper generation of memory busy bus signals.

7.6.3 Memory Card Presence Detection

This functionaity is implemente in the *pres_det* controller. Following reset it is responsible to:

- Determine the number of memory cards on the local station.
- Determine the serial number of the current card (0 or 1).

7.7 Clocking

The memory card uses two clocks. The bus clock is used for all of the onboard controllers, and the refersh clock, 2.4576Mhz is used by the DRAM controller to ensure that enough refreshes occur. The refresh clock is fed by an oscillator, and the bus clock is distributed from the backplane via an PECL 1:2 replicator followed by a pair of 1:9 PECL to TTL converters. The skew between bus clocks on the memory card should be within 1 ns.

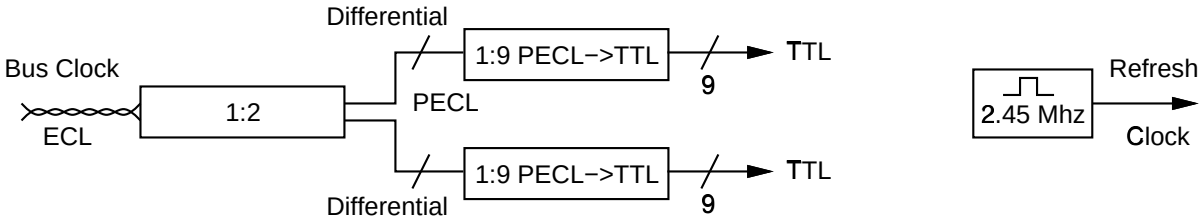

Figure 7.11: Clocking on the Memory Card

7.8 Debugging Output

7.8.1 LEDs

A number of LEDs are provided on the memory card for high level debugging. In general red lights are used to indicate errors, green lights are used for configuration, and yellow lights for utilization. Though this was the original intention, this process has not been followed for all cards.

There are two yellow utilization lights on the memory card, *bus busy* and *DRAM busy*. The intensity of these lights gives a first order indication of how busy the bus, and/or the DRAM is. During normal operation the DRAM busy light should be dimly lit, representing refresh, and the bus light should be off. If either light is on solid there is likely an error. Even for very high intensity tests the lights tend to flicker.

Table 7.2: Memory Card Parity Error Codes

Character	Fatal	Description
6	Yes	Address parity error on incoming FIFO
5	Yes	Parity error on latch between the incoming FIFO and the outgoing FIFO
0	Selectable	Parity error on write to DRAM bank 0
1	Selectable	Parity error on write to DRAM bank 1
2	Selectable	Parity error on write to DRAM bank 2
3	Selectable	Parity error on write to DRAM bank 3
4	Selectable	Parity error on write to the Special Functions unit
F	No	Parity error on reads of multiple DRAM banks
A	No	Parity error on read from DRAM bank 0
B	No	Parity error on read from DRAM bank 1
C	No	Parity error on read from DRAM bank 2
D	No	Parity error on read from DRAM bank 3

The two green configuration lights indicate whether memory card configuration has completed, *presdet ready*, and the memory card number, *card number*. The *presdet ready* light should flash during reset, but remain on otherwise. The *card number* LED should only be on the second of two memory cards installed in a station.

There are three red error lights on the memory card. The *Flex program* LED should flash during reset. This indicates that the Special Function unit is being programmed. If it stays on than the programming of the Special Function unit has failed and the card will not provide valid data if the special functions unit is accessed (i.e.. memory size).

Another error light is referred to as *Fatal Error*. This light is asserted when a fatal error is detected by the master controller. Fatal errors are primarily caused by data corruption on incoming data, or addresses, but can also be caused by garbage commands being received. If a fatal error occurs the master controller requests a system reset unless this feature is disabled using the *sys_rst_en* jumper (refer to section **??** on page ?? for details).

The final error light is *underflow error*. This light is asserted if the bus controller detects that more non-sinkable requests have been processed than were sent into the memory card. There is a feature in the special functions unit which allows it to send out processed non-sinkable requests without interfering with this checking process.

In addition to the seven LEDs a seven segment display is included which helps to detail the cause of a fatal error. If the decimal place is set it indicates that multiple types of fatal error have occurred. The number displayed always indicates the highest priority error, that is the error which occurred closest to the incoming FIFO. Table 7.2 details the error codes in priority order.

7.8.2 Debugging Headers

There are five debugging headers on the rev.2 memory card. Two for debugging the cache coherence controller, one for special functions, and two for the master controllers and the bus side controllers. Table 7.3 details their bit assignments.

There are two jumpers on the memory card. One entitled *sys_rst_en*, and the other *data_per_en*. Neither jumper is required for normal operation.

Table 7.3: Memory Card Debugging Headers

Header	bit(s)	Description
H1		Cache coherence controller
H2		Cache coherence controller
H3		Incon Buscon Mast_out
H5		SF_mon SF_ctl
H6		Mast_in Mem_dram

If the *sys_rst_en* jumper is installed the memory card will freeze when a fatal error occurs rather then causing an automatic system reset. This jumper should always be installed when debugging the memory card or it may get stuck in an infinite reset loop.

The *data_per_en* jumper disables a fatal error on on writes to DRAM to allow for more effective debugging. Normally if a parity error is detected while writing the DRAM a fatal error is asserted and the card will freeze.

Chapter 8

I/O Card

8.1 Overview

Support for I/O in NUMAchine is distributed across stations. Each NUMAchine may contain up to two I/O cards. I/O hardware is implemented on separate cards to offload as much I/O overhead as possible from the R4400 processors in each station. Each I/O card includes a separate processor and local memory to perform low-level operations such as communicating with I/O devices and formatting data.

A block diagram of the I/O card is shown in Figure 8.1 on the following page. The I/O card uses an R4650 microprocessor, a simplified derivative of the R4400 microprocessor. The R4650 executes local I/O software from the on board DRAM memory to control the transfer of data between the FIFO buffers connected to the NUMAchine bus and the I/O devices on the PCI bus. *All data transfers in either direction are staged through the DRAM memory.*

8.2 Embedded Processor and Sub System

At the heart of the I/O board is the R4650 processor and the GT bridge chip. Under the control of software executed by the R4650, the DMA Engine in the GT transfers data between DRAM and either the PCI bus or the NUMAchine bus. Data is transferred in units of cache lines between the I/O card and the NUMAchine memory. Appropriate Address and Command packets need to be also written to the out FIFO to ensure that the data is processed correctly. It is essential that all outgoing packets from the I/O card comply with the full hardware protocol, or the system may crash or otherwise show strange behavior. *Great care should be taken when interacting with the station.*

A block diagram of the Embedded processor and sub system is shown in Figure 8.2 on page 61. The *GT_glue* chip is responsible for manipulating control signals when those comming from the GT are not adaquit.

8.2.1 The GT Bridge Chip

The External Agent/PCI bridge is implemented using the GT-64010A (*GT*) made by Galileo Technologies. For detailed information on the controller please refer to their web site at www.galileoT.com. The GT includes a full interface to the R4650, and directly controls the on board memory, DUART and EPROM. The access speed of these devices is configurable in the GT. Table 8.1 on page 62 describes how the GT selects the devices that it is connected to.

It is important to note that the MAD bus is a multipurpose bus. During the begining of a transaction the address and chip selects are driven on it, and later it is used for transfering data.

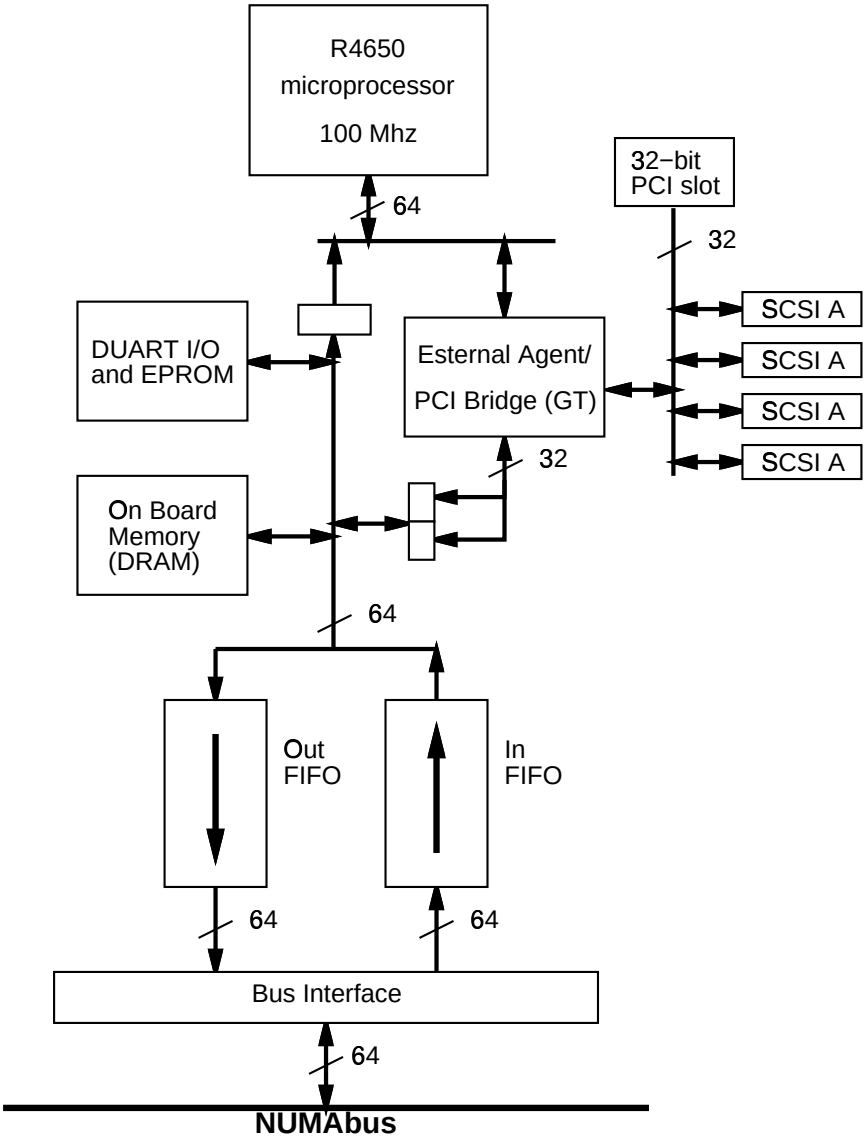

Figure 8.1: NUMAchine I/O card (Rev. 2)

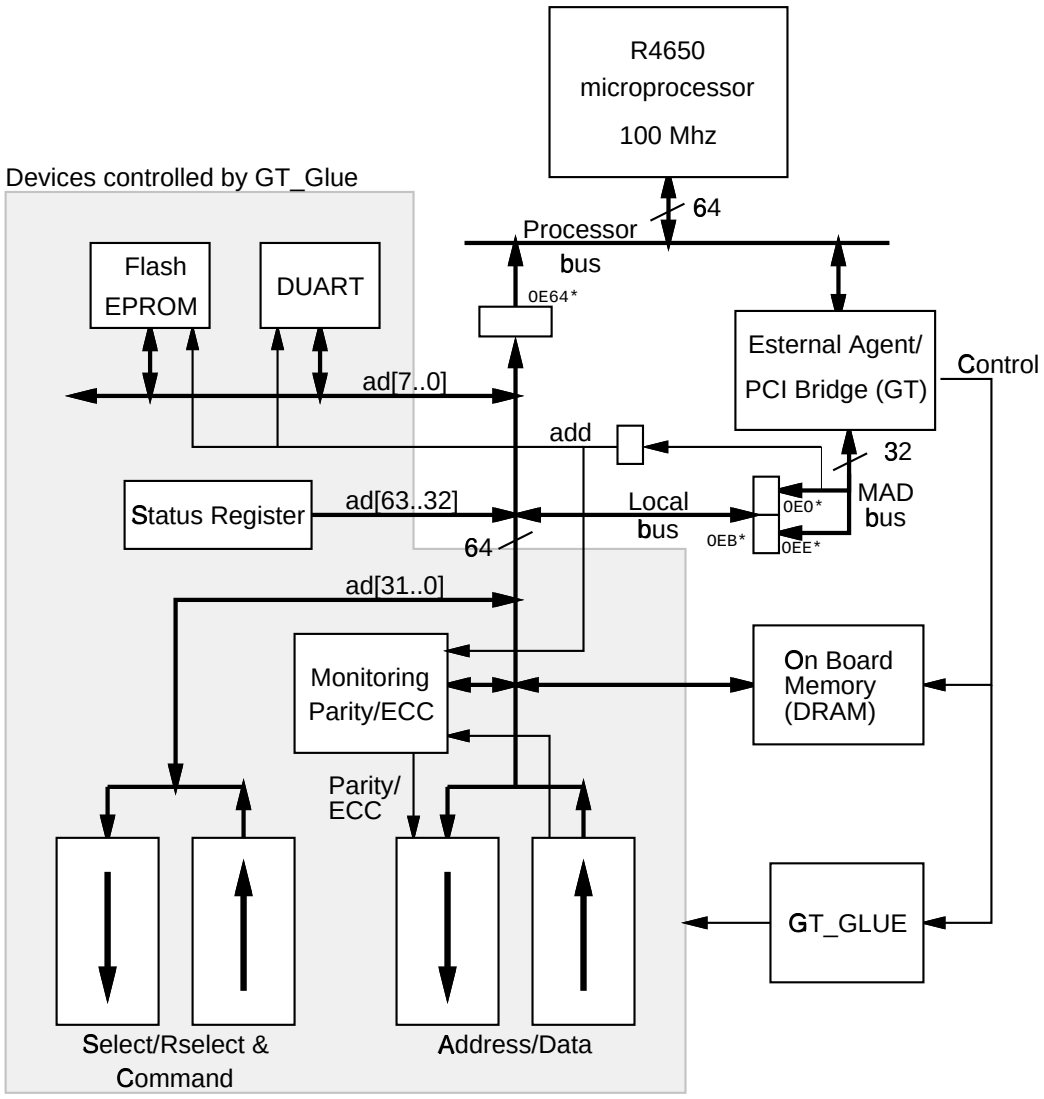

Figure 8.2: Connections to the GT Local Bus and the Processor bus on the I/O card

Table 8.1: Devices connected to the GT

Select	Parameters 50MHz	33MHz	Device	Notes
BootCS	(6,6)		Flash EPROM	8 bits wide, connected to the least signifigant byte
CS3			DUART	Connected to the least signifigant byte
CS2			Monitoring	connected to all 64 bits
CS1	(3,3,3)		Data FIFO	Address bit 3 forces parity generation mode
CS0			CMD FIFO Status Register	Lower Word Upper Word, read only
			Local DRAM	Dedicated control lines

The parameters are as follows: (AccToFirst,AccToNext[,ADSToWr])

8.2.2 The Status Register

The status register on the I/O card is designed so that the key flags of all of the devices on the card can be polled, rather than constructing a complex interrupt structure. Since the processor on the I/O card is primarily responsible for moving and manipulating the I/O data, its core control loop should poll the status register on a very frequent basis. If for example the in FIFO on the I/O card fills up, and the memory card is trying to send a transaction to the I/O card the memory card will be blocked until there is sufficient space on the I/O card to send the transaction. If this happens for too long the processors may time out and give fatal bus errors. If the I/O card does nothing else it must ensure that the in FIFO is emptied as soon as possible.

The *in FIFO command* and the status register are read at the same time. The upper 32 bits are the status register, and the lower 32 are the *in FIFO command* and associated Rselect. The contents of the status register are detailed in *principles of operation for system programmers* [CGG+97].

The Status register is implemented as a 32 bit wide register which is always clocked. Consequently the register always reflects the current status of the system. All signals that feed the status register must be cleared at their source.

8.2.3 DUART details

The DUART on the I/O card has a few dedicated I/O pins which are used for controlling the flow control on the RS232 port, as well as other functions we have defined. For details on the pin definitions please refer to *principles of operation for system programmers* [CGG+97]. Figure 8.3 on the next page shows how the ports of the DUART are connected to the system. Port B is intended for gethering debugging data from the processors, and port A is intended for sending that data to a remote console for debugging purposes.

There are additional connections to the bus which are used for flow control on the bus lines used by the DUART. These control lines are Poll_request_in, Poll_request_out, UART_CTS and UART_RTS. The two Poll_request lines connect to one bus line refered to as Poll_request. One is used as the output, and the other as the input. Since the BTLs provide efectively an open-collector type circuit multiple cards in the bus can assert Poll_request if desired. UART_CTS is also refered to as the Guy_bit_in, and UART_RTS is refered to as the Guy_bit_out. These two bits map to one bus signal called the Guy_bit which has multiple functions. Initially it is used to establish if ECC or Parity mode is being used on the station. Following reset it is available to be used as flow control for DUART port B. The original use has been designed out so the name Guy_bit has stuck for historical reasons only.

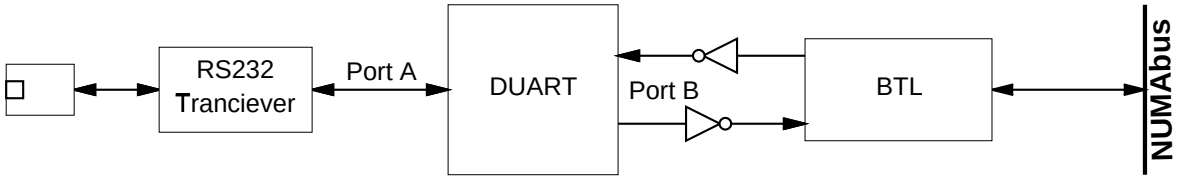

Figure 8.3: Duart port connections on the I/O card

8.3 PCI sub-system

The PCI subsytem comprises the GT bridge, 1-4 SCSI controller a PCI arbiter and a PCI slot. Currently only two of the possible four SCSI controllers are installed for cost reasons.

8.4 NUMAchine Bus Interface

The NUMAchine bus interface acts as an intermediary between the I/O card FIFOs and the NUMAchine bus as illistrated in Figure 8.4 on the following page. The NUMAbus interface control is implemented in *FIFOcon*. This controller has the followiing primary functions:

- Transfer packets from FIFOs to bus transceivers.
- Perform proper arbitration procedures.
- Transfer packets from bus transceivers to FIFOs that select this I/O card.
- Proper generation of I/O busy bus signals. (See Section 3.1.1 on page 16.)

The presence detect functionaity is implemented in the *presdet_reset* controller. Following reset it is responsible to:

- Determine the number of I/O cards on the local station.
- Determine the serial number of the current card (0 or 1).

8.5 Arbiter

The IO card contains arbitration and reset circuitry for the NUMAchine bus. See Section 3.1.1 on page 16.This functionality is almost identical to that which is found on an arbiter card, but the reset switch has been removed. The arbitration and reset circuitry has no effect if the I/O card is not instelled in the arbiter slot.

8.6 Clocking

There are a number of clock sources on the I/O card. Figure 8.5 on page 65 shows these clock sources on the I/O card, and where they are distributed. The Bus side clock is delivered the same way as the memory card. The clock for the local bus can be either connected synchronously to the PCI clock, or to the NUMAbus

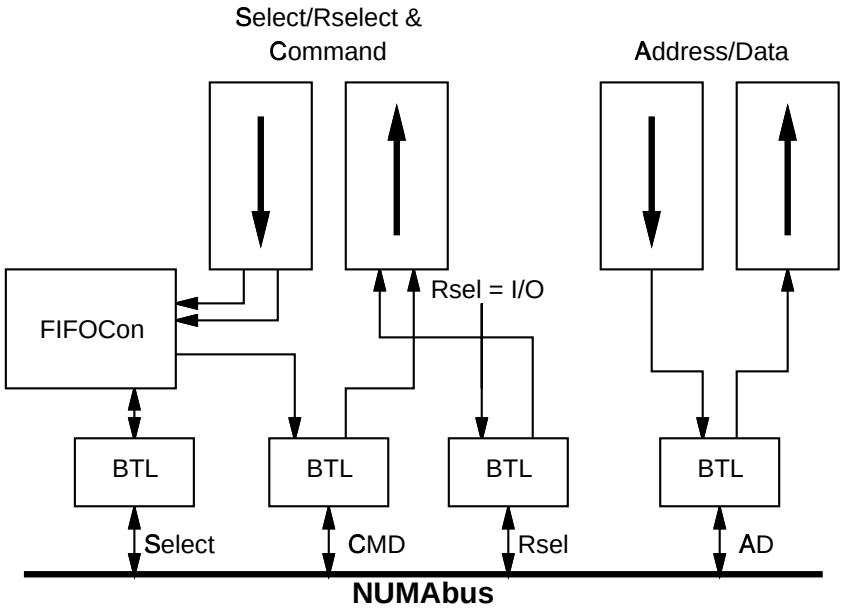

Figure 8.4: Bus side data path on the I/O card

clock. It has been discovered that the PCI bridge chip exibits metestability problems if the clocks are not the same frequency on both sides, but are not synchronous. As a result running with the PCI clock feeding the local clock is the safest. Using a 50Mhz NUMAbus clock also works well, but reliability problems are expected. The PCI clock and the SCSI clock are both provided by onboard oscilators.

8.7 Reset operations

8.7.1 CPU configuration

When the I/O card detects a bus reset it performs a cold reset on the embedded R4650. This includes sending a 256 bit configuration stream. The details of the cpu reset are detailed in the R4650 manual [[Int95]].

8.7.2 FLEX programming

On the falling edge of the local reset the flex programming circuit programs the ECC/parity chip. It is set up for a serial bit stream from bit zero of the EPROM.

8.7.3 Optional reset control from the I/O card

There are optional conections to both the system reset request line and the PCI reset request line. In order to activate these features jumpers JH1 and/or JH2 respectively need to be installed. This feature is untested and may require some minor code changes in the reset controller depending on what the powerup state of the duart output pins is. It should be noted that the PCI reset will also reset the GT chip.

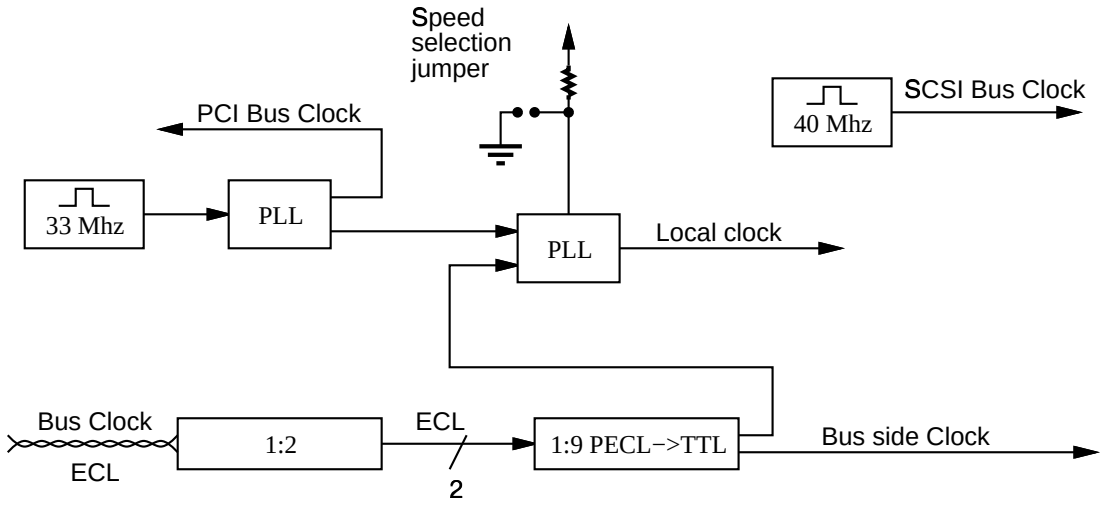

Figure 8.5: clocking on the I/O card (Rev. 2)

Chapter 9

Network Interface Card

9.1 Overview

The Network interface provides connectivity between a given station and the rest of NUMAchine. It also provides a tertiary cache for caching remote accesses. This card was originally designed as two cards, the ring interface which connects to the network, and the Network cache which is the tertiary cache. Figure 9.1 on the following page is a high level block diagram of the nic.

9.2 Local Ring Interface

To be filled in by Steve

9.3 NUMAchine Bus Interface

To be filled in by Steve

9.4 Network Cache Controller

To be filled in by Alex

9.5 SDRAM Controller

To be filled in by Robin

9.6 Ring Packet Assembler

The *packet assembler* is the final stage (stage 3) of a ring transfer. It re-assembles all transactions that are longer than one packet in length. To avoid deadlock, this stage must absorb all non-sinkable transactions that can be in the system. The reasoning behind this is discussed in [Lov96, Chapter 4]. Figure 9.2 on page 69 shows the logical design of stage 3.

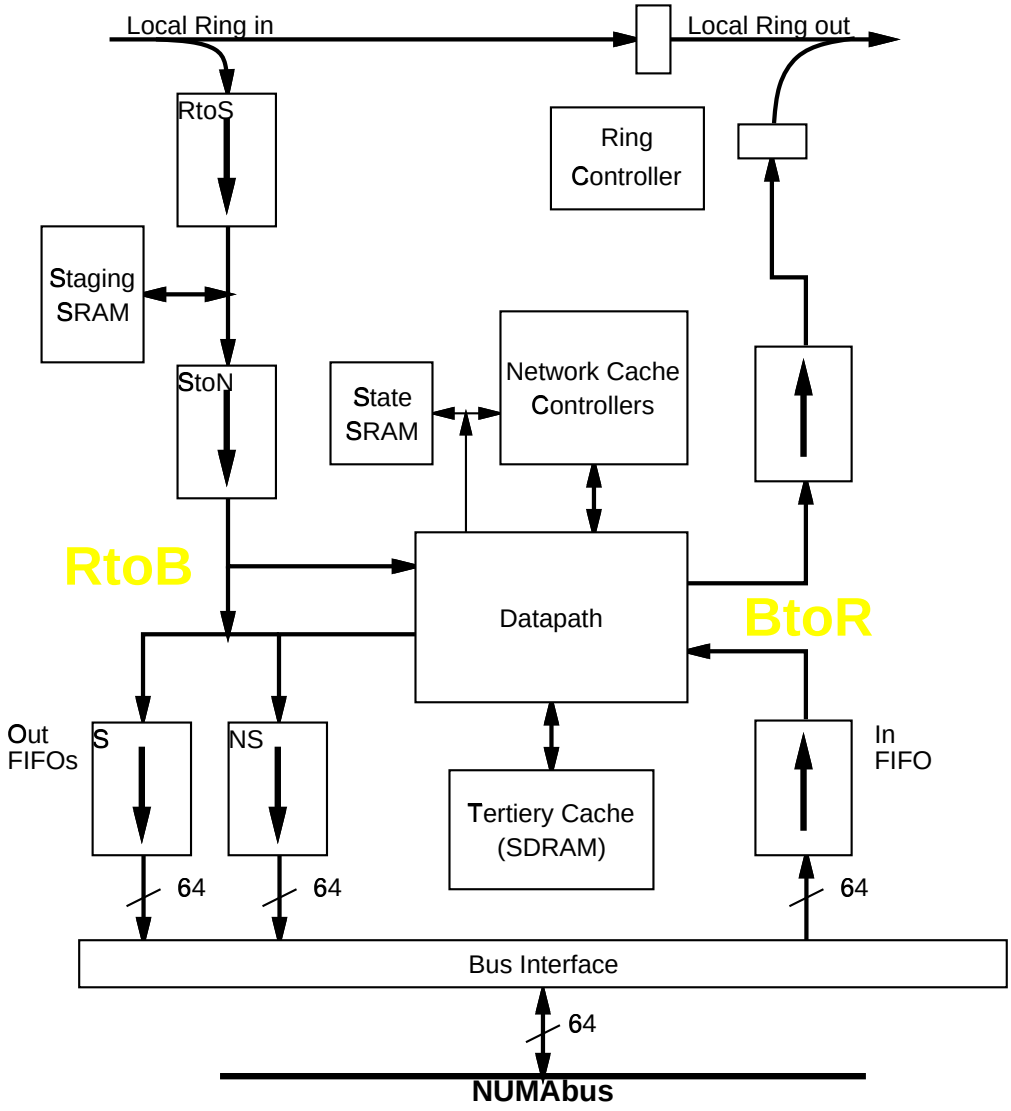

Figure 9.1: NUMAchine NIC card (Rev. 2)

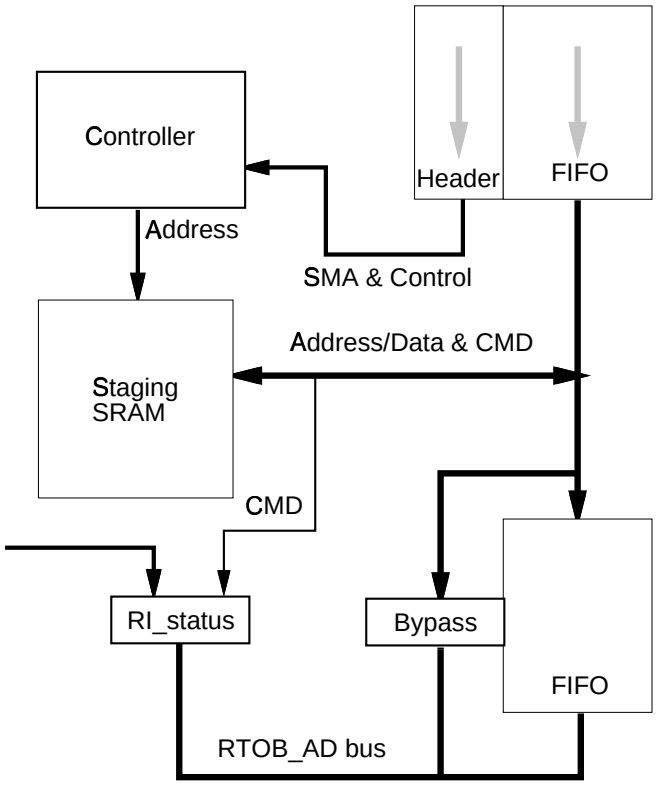

Figure 9.2: Block diagram of the packet assembler stage (Stage 3)

9.6.1 Staging SRAM Memory

Since the interconnect is implemented using a slotted ring, multiple packets in one transaction may not arrive in consecutive slots. Non consecutive packets are reassembled in a dedicated staging area implemented using SRAM memory. The staging SRAM is partitioning into a number of staging locations based on the transaction type along with some virtual Queues. Table 9.1 shows how the SRAM is logically partitioned.

Table 9.1: SRAM Partitioning

Starting Address	Size in entries	Description
0x0000	0x3FF	Solicited data staging area (mapped in using the SMA)
0x0400	0x3FF	Unsolicited data staging area (mapped in using the SMA)
0x1000	0x7FF	Virtual Non-Sinkable Queue (NS_FIFO)
0x1800	0x1F	Virtual Sinkable Queue (S_FIFO)
0x18FF	0x1	Ring error register (This must be explicitly read)
0x19FF	0x1	All subsequent ring errors until the ring error register is read. (this is not readable)

The solicited data staging are of the SRAM is reserved for use by each of the logical cards on the sta-

tion that is capable of requesting data from a remote station. Logical cards which fit into this category are Processor, Memory, I/O, and Network Cache. For each of these cards, the amount of space reserved in the Staging Memory is sufficient to hold a total of eight cache lines of data per requester. Eight spaces are provided so that each logical card is permitted to have up to eight outstanding requests. This allows for flexibility in processor selection (e.g., MIPS R10000), and allows other logical cards, such as the I/O card, to pipeline multiple requests to memory.

The unsolicited data staging area of the SRAM is used to absorb sinkable transactions sent to this station which were not explicitly requested. This section is divided up so that there is space for four cache lines per station.

9.6.2 Classification of Ring Packets

Ring packets are classified as either header packets or data packets.

- *Header* packets contain one address packet from the bus. These packets are classified as either sinkable (S) or non-sinkable (NS). Header packets are used to signify the end of a transaction for data transfers.
- *Data* packets contain one data packet from the bus. To aid in re-assembly, data packets are classified as either solicited or unsolicited.
  - *Solicited* data packets are data response packets, which correspond to requests that have been issued by this station (that is, the station the RI is connected to). Data associated with most R_Resp transactions and Byte_R_Resp transactions fall into this category.
  - *Unsolicited* data packets refer to all other data packets received which are not the direct response to a request issued from the station. This includes write-backs and write operations.

When a packet is read from the Down_FIFO, it is characterized as either a header or a data packet. Header packets are always placed in either the NS_FIFO or the S_FIFO, based on their class. The header packet always signifies that a complete transaction has been received. This is true for two reasons. First, the header packet for a transaction is always sent last. Second, since the ring hierarchy guarantees ordering, packets will be received from a given station in the same relative order that they were sent. Data packets are written into the Staging Memory at the address identified by the SMA, as described below.

9.6.3 SRAM Mapping Address (SMA) Field

The *SRAM Mapping Address* (SMA) field in ring packets is used for re-assembly of transactions that contain one or more data packets (short transactions with only one address packet do not use the SMA field). Table 9.2 provides the SMA encoding. Two different encodings are used: one for solicited data packets, and one for unsolicited data packets.

Table 9.2: SMA Definition

Type	Bits							
	10	9	8	7	6	5	4	3..0
Solicited Data	0	Req_num (CMD[12..10])			PID (AD[54..52])			Seq_num
Unsolicited Data	1	senderSR_ID					W_cnt	Seq_num

Encoding for Solicited Data Packets

- The most significant bit of the SMA is set to 0 to mark a packet as a solicited data packet.
- The PID field, which is extracted from an upper portion of the address bits in the corresponding bus packet, identifies which logical card on the station is the requester of the data. The PID is set initially when a request leaves a logical card, and is maintained as part of the address until the transaction is completed. PID is encoded using three bits: the most significant bit indicates if the requester is a Processor, as opposed to another type of logical card, and the other two bits specify which card in that class is the requester. Refer to Table 3.2 on page 17 in Section 3.1 on page 15 for the PID encoding.
- The requester has a three-bit number, referred to as the Req_num, which indicates which outstanding request this transaction corresponds to. It is the responsibility of the requester to ensure that if it wishes to have multiple outstanding requests, then all outstanding requests have distinct values of Req_num. All cards in the system that process the request and the response maintain the Req_num.
- The lower four bits of the SMA, Seq_num, are used to identify the ordering of packets within a transaction. The ring packets are numbered consecutively starting with zero for the first packet, and ending with n-1 for the nth packet. This numbering is essential because the packets are likely to arrive interspersed between packets from other stations. Numbering of the packets means that the Ring Packet Assembler does not need to maintain a pointer for each cache line that is being re-assembled.

Encoding for Unsolicited Data Packets

- The most significant bit of the SMA is set to 1.
- The sender SR_ID is the station and ring ID of the station that is sending the unsolicited data.
- W_cnt is the value of a two-bit counter located on each station. This counter is incremented after an unsolicited transaction, usually a write transaction, is sent onto the ring. The intention of the W_cnt field is to create four logical groups for the purpose of flow control in stage 3. The Staging Memory has space for four unsolicited cache lines from each station in the system. Since this data is unsolicited, and there is only space for four cache lines per source, it is possible to overwrite the data in the Staging Memory before it is sent to the station. To prevent this, the Staging Memory is partitioned into four logical groups based on the value of W_cnt. Each logical group has space for one cache line from each source. When a logical group is full, an associated flag is set. Processing of the Down_FIFO is stopped if an unsolicited data packet arrives which is destined for a group which is flagged as containing a full cache line. This flag is then cleared when the cache line is removed from the staging area. It would be preferable to have a flag for each unsolicited data cache line location, but this is not feasible for our implementation in FPDs.
- The lower four bits of the SMA, Seq_num, are used to identify the ordering of packets within a transaction.

9.7 Datapath and Associated Controllers

9.7.1 Datapath

The datapath on the nic2 card is implemented using FLEX6000 series FPGA‘s, rather than a sea of buffers as was the case in the previous revision. The datapath is partitioned into five devices. One contains the CMD

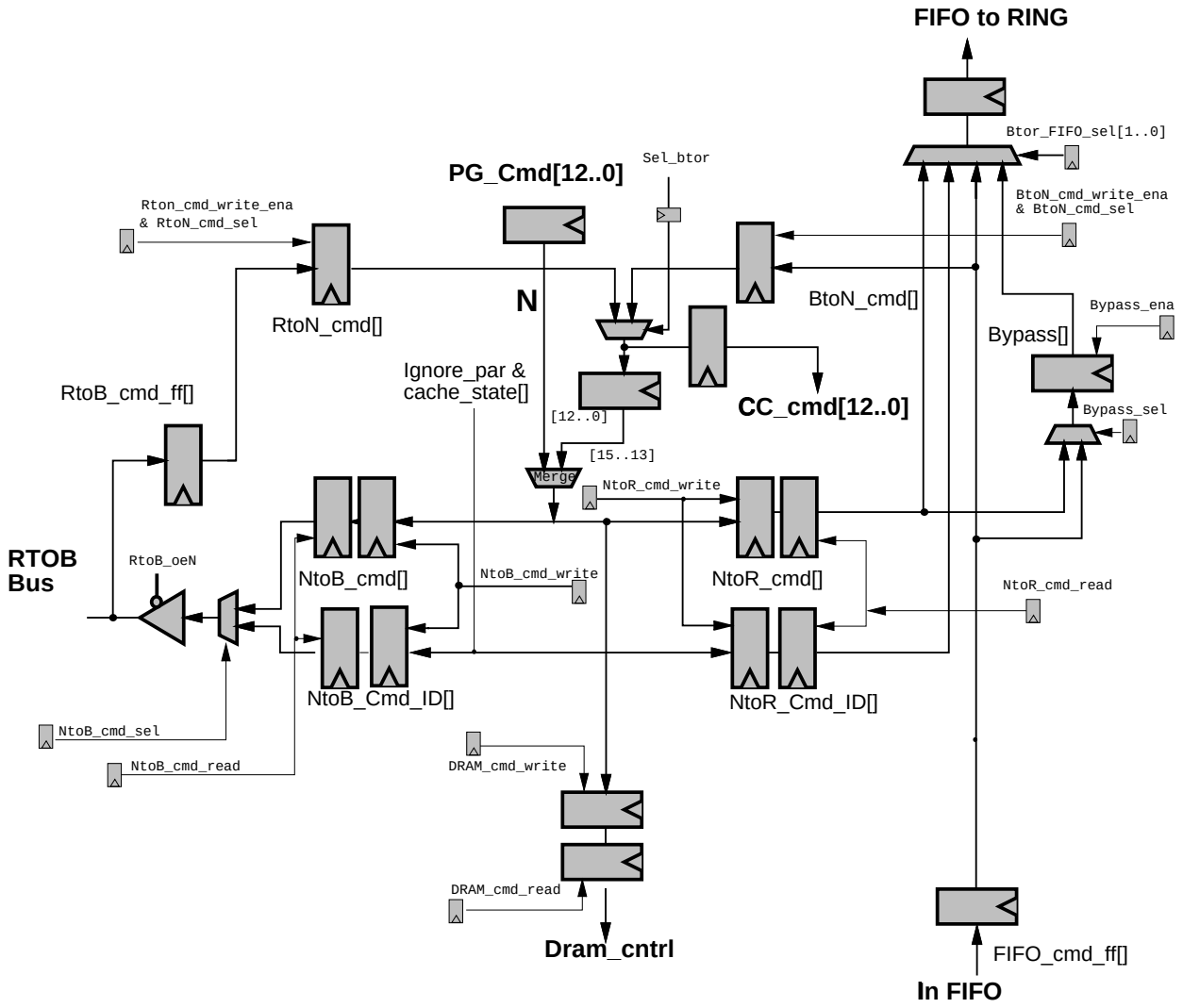

Figure 9.3: Core components of the CMD data path

datapath (nidp_cmd) and the other four each include an 18 bit slice of the Address/data datapath (nidp0 - nidp3). There are a few additional datapath signals such as Rselect and select which are distributed wherever they fit best.

Figure 9.3 illustrates the core of the CMD datapath. These diagrams are included to give a high level overview of the datapath. In order to understand the detailed implementation the actual design must be examined. Some of the details are not clearly shown in this diagram.

Figure 9.4 on the next page illustrates the core of the Address/data datapath. In the 18 bit slices (16 bits of data & 2 parity bits), only portions of this datapath are implemented. The signals to and from the cache coherence controllers are only part of the 64 bit datapath (i.e. SRAM_ADD is only in the first two datapath segments).

In all of the datapath designs all external control signals are registered in order to provide reasonable

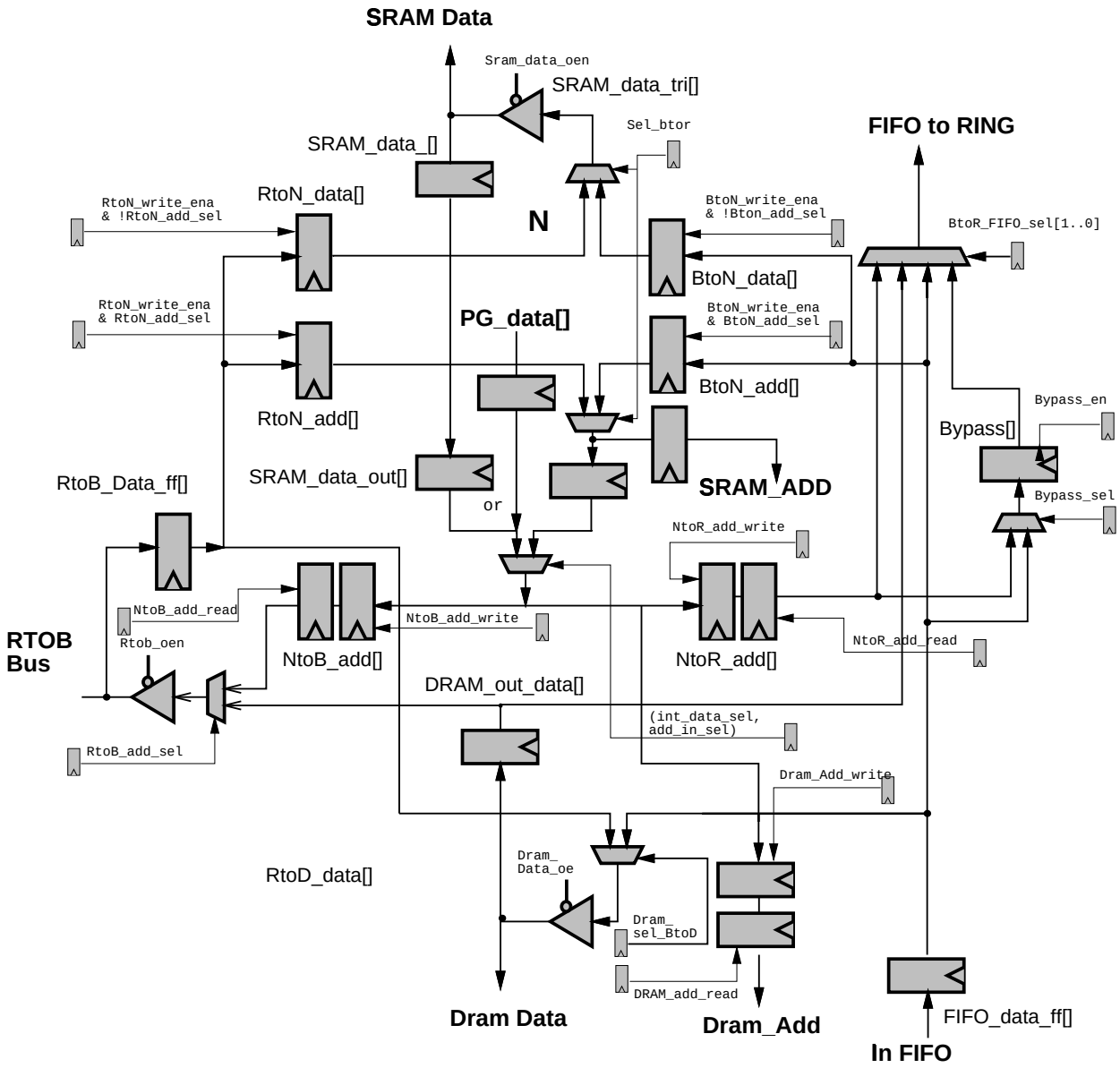

Figure 9.4: Core components of the Address/data data path

setup times. The output slew rate was adjusted on the FIFO to Ring signals because of transmission line problems casing unreliable operation at 50Mhz. The datapath FPGAs are all programmed following reset from an on-board flash EPROM. The data is stored in a bit slice manner starting with nidp0 - nidp3, followed by nidp_cmd.

9.7.2 BtoR Controller

To be filled in by Robin.

9.7.3 RtoB Controller

The RtoB Controller is responsible for taking data that has been processed by the packet re-assembler and sending to the network cache and/or the bus. It is also responsible to send out any packets that the network cache requests should be sent out. The RtoB controller is partitioned into two state machines. The first state machine, referred to as *rtob_sm*, is responsible to listen to the FIFO between the packet re-assembler and rtob (StoN_FIFO), and process the data as required. The second controller, referred to as *ntob_sm*, is responsible to listen to the datapath to see what currently needs to be sent to processed as per the network caches instructions. Only one of the two state machines can be active at a time, except when *ntob_sm* starts *rtob_sm*.

The *rtob_sm* can process both sinkable and nonsinkable data from the StoN_FIFO. Any transactions which may effect the network cache are sent into the network cache for processing. Only the address packet is sent in for processing, and the data is left in the FIFO until the datapath signals what is to be done with it. The datapath signals the target operation via the src_dst bits. These are detailed further in section 9.4 on page 67

9.8 Clocking

Clocking on the NIC is similar to that on the memory card, with the addition of a ring clock. Figure 9.5 illustrates the clock distribution on the NIC.

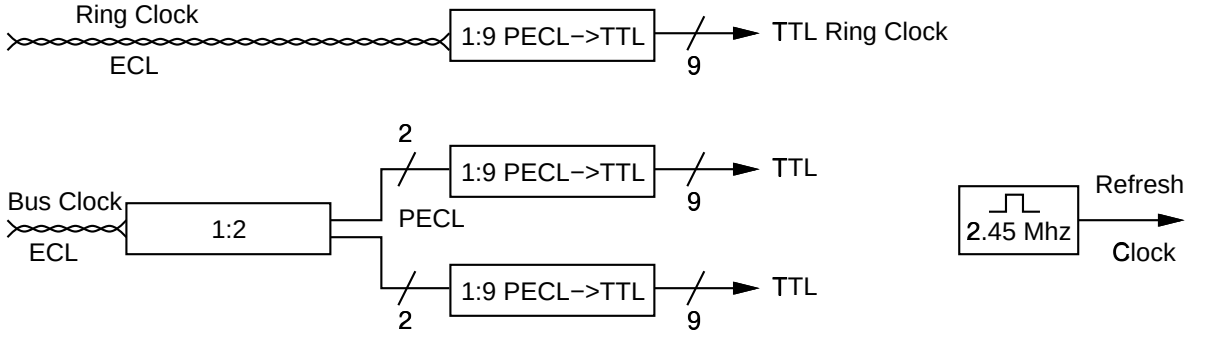

Figure 9.5: Clocking on the Network Interface Card Rev 2A

Chapter 10

Inter-ring Interface Card

10.1 Introduction

10.2 Mainboard

Notes: document cct board mods, status of revs to 2nd mainboard, 2ns early clock on mainboard, no skew between daughterboards, dip switch clock settings.

10.2.1 Component Locations

See Figure 10.1 to view the location of components on the mainboard.

The datapath card bitslices are allocated as follows.

Table 10.1: Datapath bitslice assignments.

Datapath Slot	Description
Slot A	EXTRA<2..1> GLOBAL_MON_GTOL SMA<10..0> FMASK_STN<3..0>
Slot B	CMD<17..0>
Slot C	AD<17..0>
Slot D	AD<35..18>
Slot E	AD<53..36>
Slot F	AD<71..54>

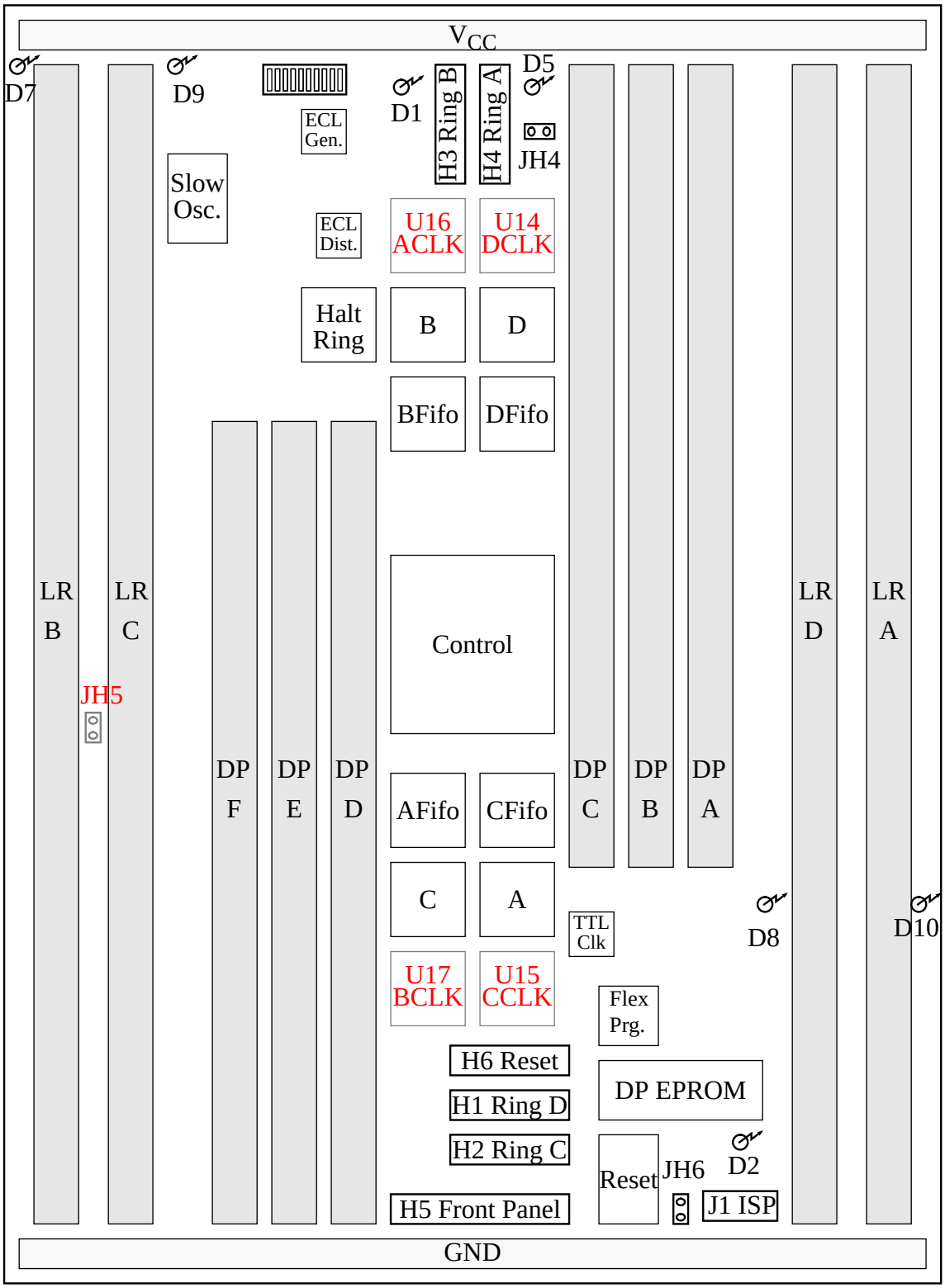

Figure 10.1: This is the mainboard component floorplan.

Table 10.2: Mainboard jumpers.

Jumper	Description
JH4	use_fresh_slots (see also D5, amber LED)
JH5	V_{CC} sense ***DO NOT SHORT!!!!***
JH6	disable clocks (for JTAG programming reliability)

Table 10.3: Mainboard LEDs.

LED	Colour	Description
D1	red	global ring controller error
D2	red	datapath FPGA programming active
D5	amber	use_fresh_slots (see also jumper JH4)
D7	green	local ring B presence detect
D8	green	local ring D presence detect
D9	green	local ring C presence detect
D10	green	local ring A presence detect

Table 10.4: Mainboard JTAG programming connector J1.

Pin	Description
1	ISP TCK
2	GND
3	ISP TD7 (TDO?)
4	VCC
5	ISP TMS
6	no connect
7	no connect
8	$\overline{\text{ISP SENSE}}$
9	ISP TD0 (TDI?)
10	GND

Table 10.5: Mainboard debug connector H1, *ring D*.

Signal Position	Description
15	gr_participants<0>
14	hs_halt_ring_l
13	hsmon_sending_lrtogr_d
12..11	hs_d_muxsel<1..0>
10	hsmon_freeslot_d
9	hs_dn_we_d*
8..0	hs_debug_d<8..0>
clk	unconnected

Table 10.6: Mainboard debug connector H2, *ring C*.

Signal Position	Description
15	gr_participants<1>
14	hs_halt_ring_l
13	hsmon_sending_lrtogr_c
12..11	hs_c_muxsel<1..0>
10	hsmon_freeslot_c
9	hs_dn_we_c*
8..0	hs_debug_c<8..0>
clk	unconnected

Table 10.7: Mainboard debug connector H3, *ring B*.

Signal Position	Description
15	watchdog_error
14	hs_halt_ring_l
13	hsmon_sending_lrtogr_b
12..11	hs_b_muxsel<1..0>
10	hsmon_freeslot_b
9	hs_dn_we_b*
8..0	hs_debug_b<8..0>
clk	unconnected

Table 10.8: Mainboard debug connector H4, *ring A*.

Signal Position	Description
15	use_fresh_slots
14	hs_halt_ring_l
13	hsmon_sending_lrtogr_a
12..11	hs_a_muxsel<1..0>
10	hsmon_freeslot_a
9	hs_dn_we_a*
8..0	hs_debug_a<8..0>
clk	global ring clock

Table 10.9: Mainboard front panel connector H5.

Pin	Description	Pin	Description
1	no connect	2	no connect
3	slow clock (4 MHz)	4	V_{CC}
5	data 0	6	V_{CC}
7	data 1	8	GND
9	data 2	10	GND
11	data 3	12	GND
13	data 4	14	GND
15	data 5	16	GND
17	data 6	18	GND
19	data 7	20	GND
21	$\overline{BREQ}$	22	GND
23	$\overline{BGNT}$	24	GND
25	read/$\overline{\text{write}}$	26	GND
27	$\overline{\text{address strobe}}$	28	next_start_pwr0
29	next_start_pwr1	30	next_start_pwr2
31	next_start_pwr3	32	next_start_pwr4
33	no connect	34	no connect

Table 10.10: Mainboard debug connector H6, *reset controller*.

Signal Position	Description
15..11	unused
10..8	flex_prg_st<2..0>
7..0	rst_dbg<7..0>
clk	unconnected

Table 10.11: Mainboard local ring clock connectors.

Connector	Description
U14	local ring D clock
U15	local ring C clock
U16	local ring A clock
U17	local ring B clock

10.2.2 Jumpers

10.2.3 LEDs

10.2.4 Connector Pinouts

10.3 Datapath Daughterboard

10.3.1 Debug Connector Pinouts

Table 10.12: Datapath daughterboard debug connector H1.

Pin	Description
15	Almost empty flag D
14	Almost empty flag C
13	Almost empty flag B
12	Almost empty flag A
11	Empty flag D
10	Empty flag C
9	Empty flag B
8	Empty flag A
7..0	debug<7..0>
clk	global ring clock

10.4 Local Ring Daughterboard

10.4.1 Debug Connector Pinouts

Table 10.13: Local ring daughterboard debug connector H1.

Pin	Description
15..8	unused (reserved for monitoring debug pins)
7..0	ringcon_debug<7..0>
clk	local ring clock

10.5 Clocking

The IRI main board provides the high speed clock used by the global ring. Figure 10.2 shows how the clock is generated and then distributed on the main board. Each of the datapath cards also have a 1:9 ECL - TTL converter.

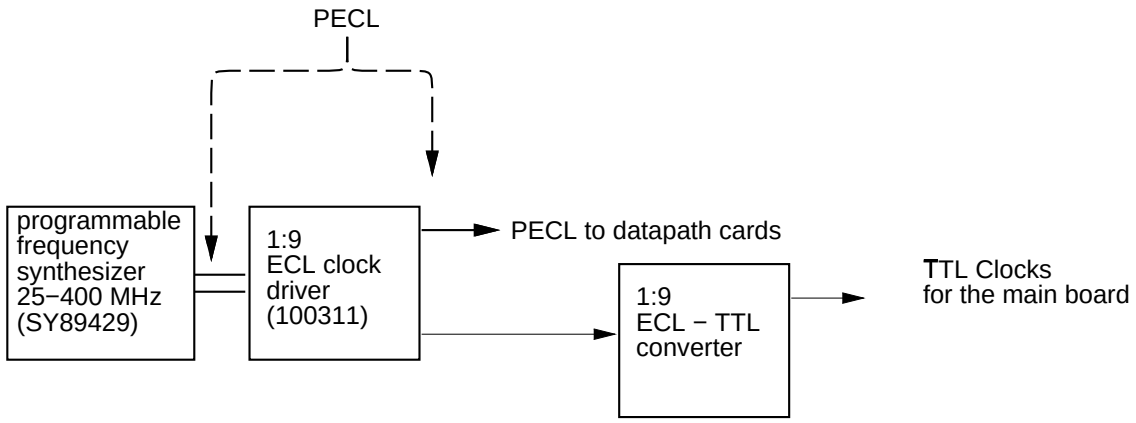

Figure 10.2: Clocking on the IRI main board

Part III

Maintenance Procedures

Chapter 11

Power

11.1 Station-level Power Requirements and Distribution

11.2 Procedures for Powering Up and Powering Down

11.2.1 Basic Issues and Concerns

All powered components in the system should be brought on-line in a controlled fashion in order to reduce the instantaneous power requirements to a reasonable level. To facilitate this, each station is powered up in sequence. The system will provice a system reset after the last station is powered up.

A means to power down the system in a controlled way — in case of a cooling outage, for example — should be provided, although although there is no such functionality at the present.

Powerup Each station powers up sequentially; they are daisy chained, and when one station has stable power, it sends a signal that allows the next station to powerup. This limits the instantaneous power required on start-up.

There will also need to be a way of powering up the disks to limit their instantaneous power requirement - apparently the SCSI protocol allows for sequential power-up of disks.

System-level Reset A system-wide reset line exists that causes each arbiter card to perform a station reset sequence. This system-wide reset is bussed to all arbiter cards making each station identical, and allowing any arbiter to initiate the system wide reset.

The system-wide reset is an active low OC line, with pull-ups. When a reset occurs, the line is pulled low, and then released. When an arbiter card senses the line is asserted, it starts a station reset. This allows any device attached to the system-wide reset line to initiate a reset. The OS can initiate a system reset by writing to register in any processor cards reset controller.

Chapter 12

Station Assembly

12.1 Overview

A station consists of a Futurebus+ backplane card, mounted in a cage. Several cards are inserted in the backplane slots. In the back of the backplane, there are two power supplies, a clock card and two disk drives.

To assemble the station, some small modifications have to be made to the backplane, the cage must be assembled, and the backplane mounted on the cage. The whole is then installed on a rack and the remaining parts are installed from the back of the rack. Power supply cables need to be connected to the backplane and clock lines (twisted pair wires) need to be soldered to the back of every slot.

12.2 Assembling the Cage and the Backplane

12.2.1 Backplane Modifications

Each station uses a Futurebus+, 14 slot, 128 bits wide backplane. However, small modifications need to be made to it, that is some resistors need to be removed: 5 on the front of the plane, the side with the slots, and 2 on the back. These resistors are surface mounted and are best removed with a hot air soldering iron which will simply blow them away when hot. The position of the resistors to remove on the front of the plane is given in Figure 12.1 on the next page.

In this figure the board is held so that the copyright notice is in the top left corner. The five resistors are:

- On the left side of the leftmost slot, the 3rd resistor from the bottom;
- On the right side of the leftmost slot, the last resistor on the bottom;
- On the left side of the rightmost slot, the 3rd resistor on the bottom;
- On the right side of the rightmost slot the last resistor on the bottom;
- In the top centre of the board, from the set of four resistors, the 2nd from the top.

The resistors to be taken out from the back of the backplane are shown in Figure 12.2 on the following page.

This board is held so that the short slots are on the bottom. The resistors are always close to a row of capacitors:

- In the first column of resistors from the left edge, the 3rd one from the top;

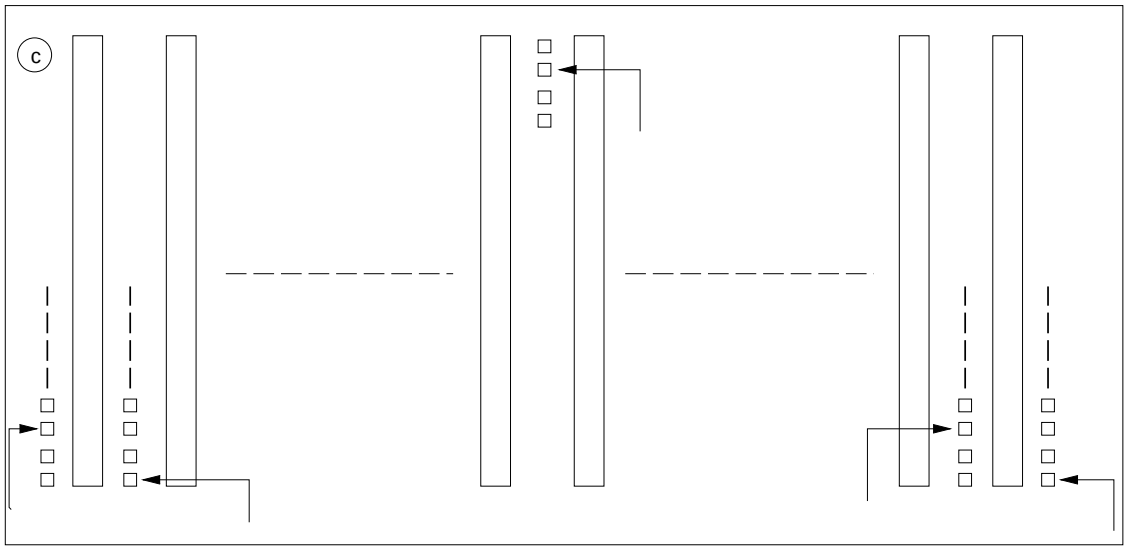

Figure 12.1: Front view of backplane

Figure 12.2: Rear view of backplane

- In the second column of resistors from the right edge, the third one from the top;
- In the first column of resistors from the left edge, the 3rd one from the bottom.

These eight resistors need to be removed from small, 4 slot backplanes too if they are used.

12.2.2 Jumper Settings

Several jumpers need to be set on the Backplane for proper operation. They are all on the back, the side without slots, and on the lower side when the copyright notice is on the top. The position of the jumpers and

their state is given in Figure 12.3. An empty square represents an open jumper, a checked square represents a set jumper and the long rectangles are the positions of the slots on the other side of the Backplane.

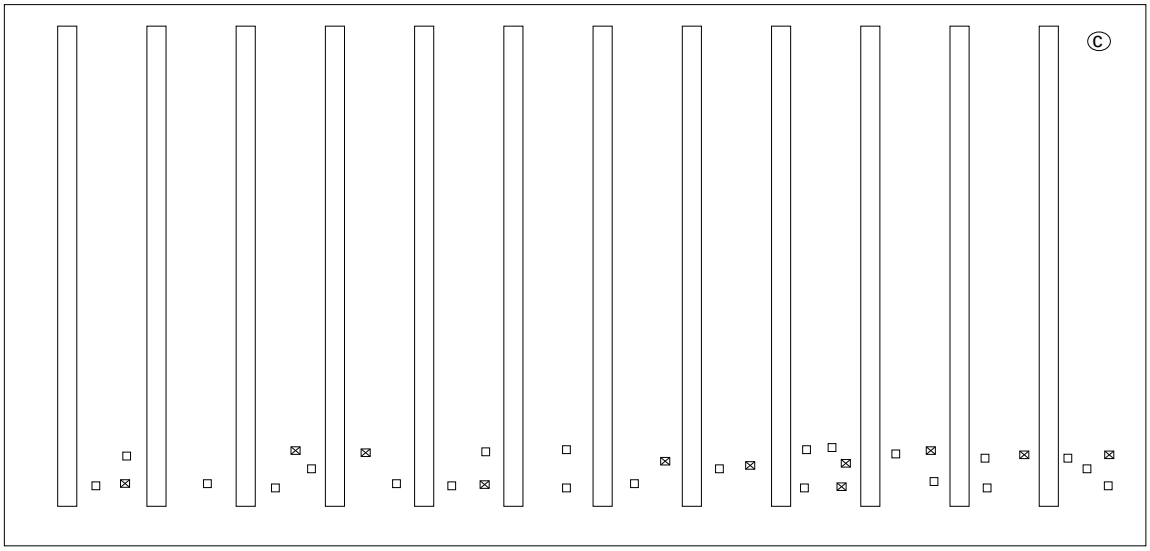

Figure 12.3: Jumper positions to be populated

12.2.3 Assembling the Cage

The cage used is a KM 25 Metric subrack. Its main structure is shown in Figure 12.4 on the following page.

Its main parts are:

- 2 side plates;
- 2 L rails (not symmetric);
- 2 front rails;
- 2 back rails;
- 2 middle rails.

The front rails are the widest and are installed with the numbers at the front. Before they are mounted, a guide location strip must be inserted through the groove in the rail with the smooth side out. They must be mounted together with the L rails. These rails stand vertically along the front edges of the side plates. These rails are not symmetric, the L rail with the smaller inner flap must be put on the right side of the cage.

The back rails are somewhat narrower. A guide location strip must be inserted in each of them as well as a tapped strip to which the backplane will be attached. (See Figure 12.5 on the next page.) The rail must be mounted with the tapped strip to the back.

The middle rails are the smallest. Another tapped strip must be inserted through them before they are mounted. They are mounted so that the card guides can be screwed to them later.

To attach the rails to the side plates, the supplied Allen head screws are used.

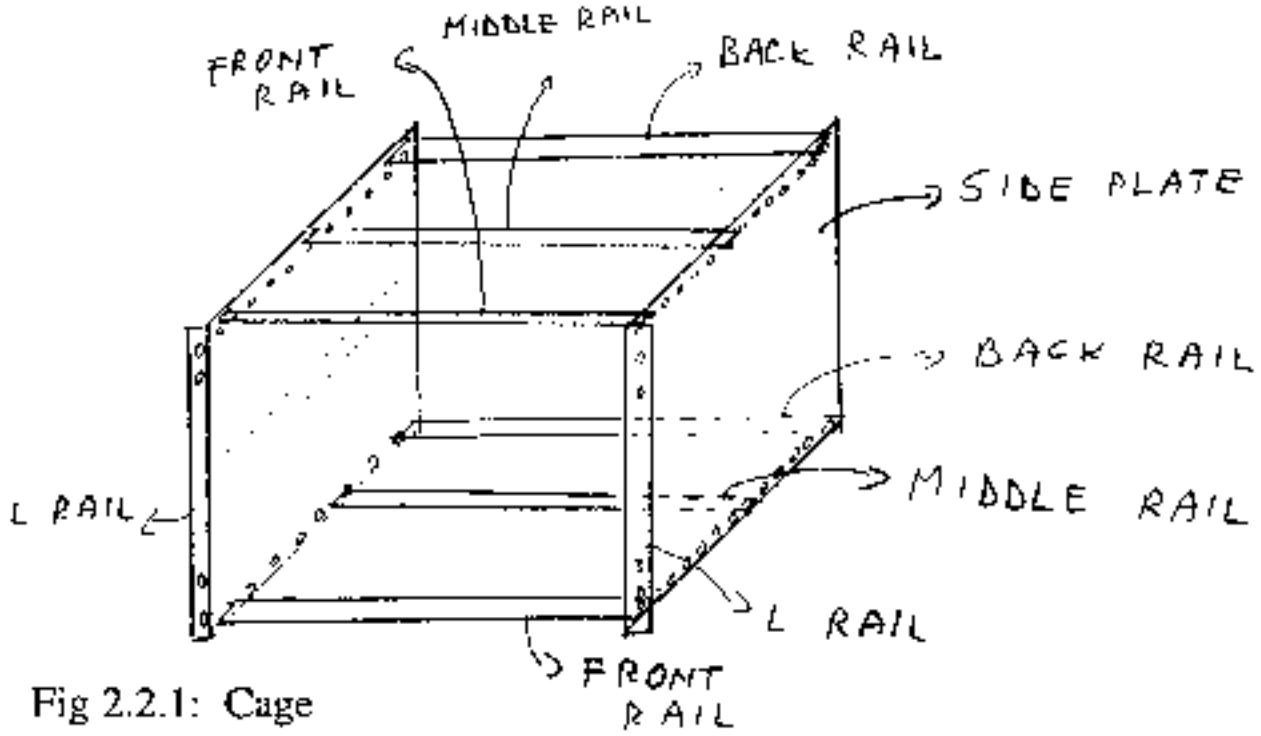

Figure 12.4: Station cage assembly

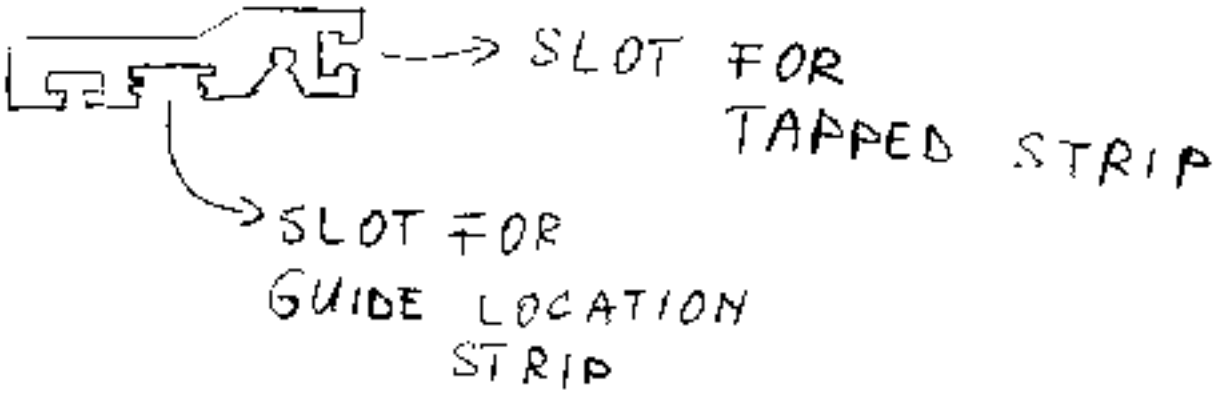

Figure 12.5: Back rail, cross section

12.2.4 Attaching the Backplane to the Cage

The backplane must be installed at the back of the cage, with the slots towards the inside of the cage, and the copyright notice in the top left corner. The insulating strips that came with the cage must be inserted between the backplane and the back rails. 2.5 x 10 mm screws are used to screw the backplane to the tapped strips of the back rails. There should be a screw in each corner and a screw in every third hole of the plane.

Finally the card guides must be inserted in the cage. Their groove must align with the right wall of each card slot. They snap into the guide location strip of the front and back rails, and they are screwed into the tapped strip of the middle rail. For this 2.5 x 8 mm screws are used.

The cage with the backplane can now be installed on a rack. The holes in the L rails are used for this.

12.3 Auxilliary Cage Hardware

12.3.1 Overview

Plastic and metal pieces, blah, blah, blah...

12.3.2 Assembly of Components

Tab A into slot B, blah, blah, blah...

12.4 The 2.1 V Supply

The backplane requires a 2.1 V, 10 A power supply. It is built using a LT1038CK voltage regulator. The schematic of the circuit is given in Section B.3 on page 103 and the parts used are included in Table B.1 on page 104. To assemble the supply, all the parts except the voltage regulator must be soldered to the card, the regulator is then bolted to the heat sink and only afterwards is soldered to the board.

12.5 Wiring of the Cages

12.5.1 Ground and 5 V Connections

The Backplanes are supplied with 5V from the Lambda RWS450A- 5 power supply. The connections are made using Belden 8916 cable, red for 5V and black for ground. 12 wires run from the ground terminal of the supply to the lower part of the Backplane and 10 wires run from the positive terminal of the supply to the upper part of the Backplane. See Figure 12.6 on the following page for the positions of the connections on the Backplane. These cables are connected to the plane using the screws on the rails provided. The connection is made using Panduit PN-14- 10R-C crimp terminals. At the power supply all the cables for each terminal are connected together in a Panduit CB175-38-Q connector. Three more ground and three 5V cables are attached to these connectors and led to the female part of a Molex 6 circuit connector. This is used to supply power to the clock card, the 2.1V power supply and the ring control card. Additionally 5 cm of heat shrink tubing are slid on the cables at the power supply end to provide insulation.

The 2.1 V supply is connected with blue cable to the two 2.1 V bolts on the backplane. The cables are soldered to the supply and are connected with crimp terminals to the backplane.

The use of the Molex connector is given in Figure 12.7 on the next page This shows the mating side of the female part.

12.5.2 Clock Connections

The clock card has 18 clock outputs. From these, 13 clock lines are connected to 13 of the slots of the backplane. Twisted pair wires taken from category 5 Ethernet cable must be used for this. It is best to leave three twisted pairs in each cable, connecting them to three adjacent clock outputs on the clock card and to 3 slots. For the last cable 4 pairs must be used. The wires are soldered both to the clock card and to the backplane. On

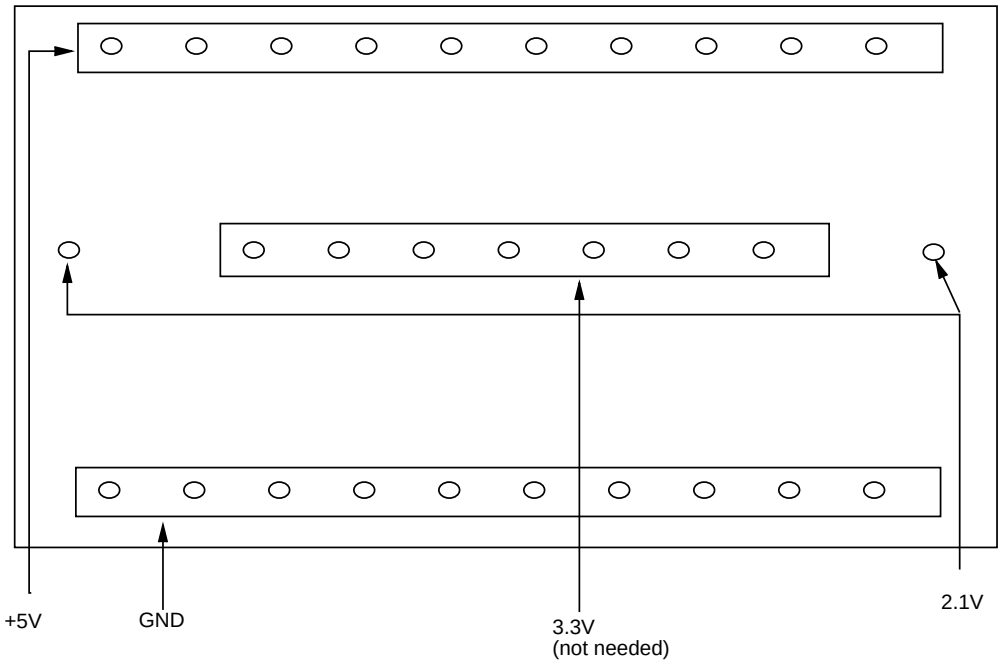

Figure 12.6: Backplane power connections

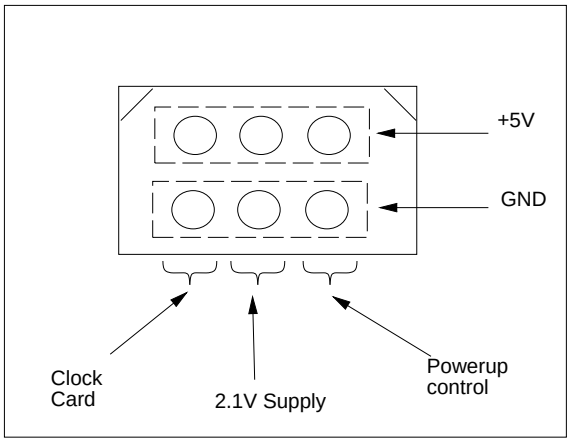

Figure 12.7: Connector to clock card, 2.1V power supply, and ring control card

the clock card they are connected to two adjacent pads. On the backplane they are soldered to two adjacent vias, on the back of connector X. Their position is shown in Figure 12.8 on the facing page. Since there are 14 slots on the backplane and only 13 clock lines, one slot must be left unconnected and thus it is unusable. The guiding rails should be taken out from the cage for this slot.

When connecting the clock lines, it is very important to keep the same clock phase for each slot. It is thus recommended to connect each white wire of a pair to the pad marked pin 1 of the output and to pin 2 at the slot, and the coloured wire to pin 2 on the card and pin 3 on the backplane.

The two power wires of the clock card must be connected to the male part of the Molex connector shown in Figure 12.7 and thus to the power supply.

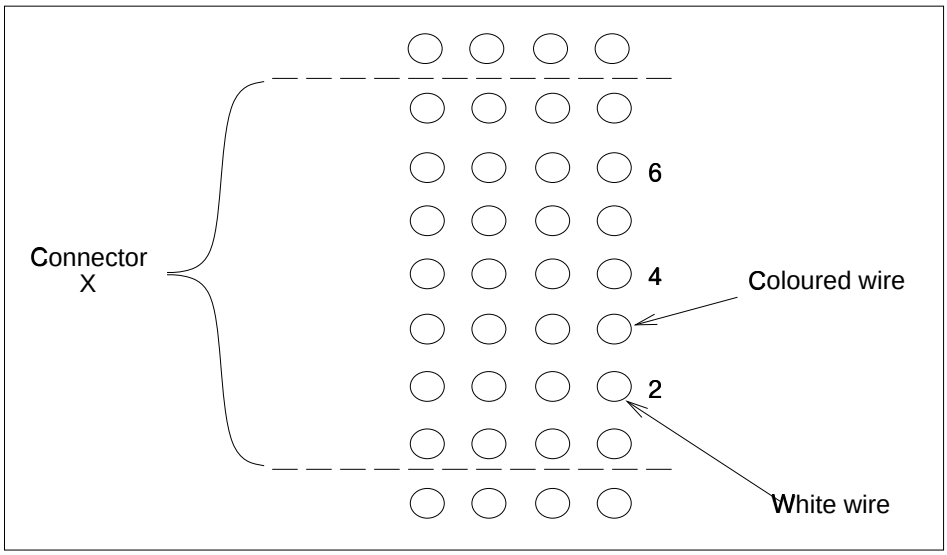

Figure 12.8: Clock connection to backplane

For proper operation of the clock card, the serial load pin on the set of jumpers beside the frequency selecting switches must be jumpered to ground. This is best done by wire wrapping a short wire to this pin and to one of the ground pins.

Chapter 13

Maintenance Procedures

13.1 Hardware Mainteneance

13.1.1 Card Replacement

Power should always be turned off when removing or inserting cards. It is possible to demage some of the Altera devices if boards are hot-swapped.

13.2 Programmable Logic Device Maintenenance

Compiling Source Code

Troubleshooting

Downloading Object Code to Chips

In-system Programmable Devices

Socketed Devices

Problems During Programming

13.3 Test Software Maintenance

13.4 Cadence Design Files

Chapter 14

Testing and Troubleshooting

14.1 Overview

This section describes the approximate testing procedure used to verify that cards are functioning correctly. It also includes a list of common symptoms, likely causes, and possible solutions.

14.2 Testing Processor cards (Rev. 3 & 3A)

14.2.1 Standard Test Sequence

When testing the processor card there are four levels of functionality that are tested.

- The processor reset properly, the FLEX chip is programmed and the hex count routine runs from the EPROM.
- All on-board features are tested, predominantly those on the local bus.
- The bus interface is tested by doing a sequence of memory tests
- The Multi-processor features are tested using various single station MP tests

14.2.2 Common problems

- Faulty bits on bus interface. Likely caused by FIFO pins being shorted, or having cold solder joints.
- Memory gets fatal error when running a memory test. Likely cause is parity error in the address packet, or a corrupt command packet. Assuming the memory is known to be good these errors are likely caused by a faulty bit in the bus interface.
- Processor is flaky when booting, or FLEX programming fails. Likely cause is one or more bad connections on the G_INT chip. The sockets are not the highest quality.

14.2.3 Notes on the Processor card

Some processor cards have processors installed that do not meet thermal requirements. The processors that do not meet this specification are marked as such on the ceramic package. When replacing a processor, ensure that if one of these processors is used a fan is also used on the heat sink. These processors need to have a fan-mounted heat sink installed.

14.3 Testing Memory Cards (Rev. 2 & 2A)

14.3.1 Standard Test Sequence

When testing the memory card there are four levels of functionality that are tested.

- The card resets, and both presdet and flex program complete.
- The state SRAM can be read and written successfully
- Uncached DRAM operations work, at least on linear and on pattern or rotating bit test should be performed.
- Cache coherent operations work successfully including cache coherent shared, cache coherent exclusive and non-coherent.
- Multiprocessor test works both shared and exclusive. This should include at least one remote station.

14.3.2 Common Problems

14.4 Testing Network Interface Cards (Rev. 2A)

14.4.1 Standard Test Sequence

When testing the NIC Card there are five levels of functionality that are tested.

- The card resets, and flex programming works correctly
- The local SRAM and SDRAM can be read and written correctly
- With the ring looped back to itself, the NIC can send packets across the ring correctly.
- Bursty uncached MP tests work with two stations.
- MP tests work with two rings both shared and exclusive.

14.4.2 Common Problems

There are some common problems with the NIC.

- Flaky ring transfers: One or more of the ring buffers may be broken. Unfortunately this often only manifests itself when the buffers oe are turned on and off. More complex MP tests are usually required to catch this.
- Packets get written to the staging SRAM but are not read out: There are two common causes for this. One is that the writeback buckets are full, likely because of corrupt packets. Two error packets were received. Writes of error packets superficially look the same as normal writes to the staging SRAM, but are sent to an error location, and no further processing is initiated.

14.5 Testing I/O Card (Rev. 2A)

14.5.1 Standard Test Sequence

When testing the I/O Card there are five levels of functionality that are tested.

- The card resets properly, and flex program works correctly
- The R4650 processor boots and displays the appropriate boot message through the DUART.
- The I/O board can down-load code and run the dma test.
- The PCI device inventory works correctly.
- The arbiter embedded on the card works at 50 MHz.

14.5.2 Common Problems

- The parity is wrong coming out of the IØcard. Likely case is that the GT_Glue chip is programmed for Rev. 1 rather than Rev. 2 for rev.2 cards, or vice-versa for rev.1 cards.
- A PCI command access returns all 'F'. The device is not present, or not responding.

14.6 Testing a Station Backplane

14.6.1 Standard Test Sequence

When testing the station backplane there are four levels of functionality that are tested.

- Test that the power is being supplied correctly to the backplane. 5V +/- 5
- Test that the all wired slots have a functioning clock.
- Test that a 4 processor, memory and arbiter configuration can run a local MP test.
- Add a NIC and test that the local ring clock works by running a multi-station MP test

14.6.2 Common Problems

There are a few common problems with the stations. They are as follows:

- Clocks are flaky: One or two of the clock lines on the back are not well connected.
- No ring clock: Either the ring clock isn't plugged into the right slot on the back, or the distribution of the ring clock through the power-up card is not working.

14.7 Testing Inter-Ring Interface

Appendix A

Commidity Components

A.1 Integrated Circuits

A.1.1 FIFOs

TI ABT3611 Uni-directional 36-bit FIFO, with bi-directional mailbox registers. Has parity check and generate functions. Mailbox and FIFO share IO ports for both I and O functions. Used everywhere. Sprinkled like pepper.

Datasheet reference:
TI SN74ABT3611, 64x36 clocked first-in first-out memory.
TI publication number SCBS127C.

Original supplier: FAI

A.1.2 Buffers

TI FB1650 Bi-dirctional 18-bit buffer—translates between TTL and BTL. Has one IO port for BTL side, bus separate I and O ports for TTL side. Two functionally separate 9-bit buffers reside in package. Package is a 100 pin PQFP. Used to interface to Futurebus+ backplane. Every card has about seven of these on board. Sprinkled like salt.

Datasheet reference:
TI SN74FB1650, 18-bit TTL/BTL universal storage transceivers.
TI publication number SCBS178C.

Original supplier: FAI

IDT FCT162511 Bi-dirctional buffer with parity generation and detection features.

Original supplier: FAI

IDT FCT162827 Uni-dirctional address buffer.

Original supplier: FAI

IDT FCT162374 Uni-dirctional register.

Original supplier: FAI

IDT FCT162244 Bi-dirctional buffer.

Original supplier: FAI

A.1.3 Programmable Logic Devices

Altera CPLDs Assorted 7000 and 7000S series CPLDs

Original supplier: Semad

Altera FPGAs Assorted 10K, 8K and 6K series FPGAs

Original supplier: Semad

A.2 Support Hardware

A.2.1 Power Supplies

450W LAMBDA 5V 450W powersupply with remote regulation via sense lines.

1000W LAMBDA 5V 1000W powersupply with remote regulation via sense lines.

A.2.2 Fans

A 120-38R 11-S 120V 60Hz, 105 CFM Nidec fan.

Original supplier: FAI

Appendix B

Other Information to Assemble a Station

B.1 Parts Needed for One Station

Table B.1 on the following page lists the parts needed to assemble one station. Additionally some serial conversion boards, power up controllers and Byte Blaster programmers can be built. The serial conversion boards convert the duart output of the processors to RS232 which can be then connected to a terminal. The power up controllers, delay the time when each station is powered up so that the initial load on the wall sockets is not too large. Moreover, it distributes a synchronous clock to all the network cards on a local ring. The Byte Blaster programmer is used to program Altera PLDs on board by connecting it to the parallel port of a computer. The parts needed to assemble each of these boards is given in table B.2 on page 105.

B.2 Modifications To Processor Cards

For all the processor cards some modifications need to be done to the duart circuitry. On the top of the card, pin 3 of chip U72 needs to be lifted from its pad and bridged to pin 2. On the back of the card, two traces must be cut and two jumper wires soldered instead. The position of the traces to be cut is given in fig B.1 on page 106 and the position of the jumper wires is given in fig B.2 on page 106.

For the processor boards marked 44-96, the first manufacturing batch of the Proc R3 cards, three resistors must be added. These are 4.7 kohm, surface mount resistors. Their positions are shown in fig B.3 on page 107, fig B.4 on page 107 and fig B.5 on page 108. The third resistor is soldered between the lower pin of capacitor CB333 and the via on its left.

B.3 2.1V Power Supply

The schematic of the 2.1 V power supply used by each station is given in fig B.6 on page 108. The parts used to assemble one are given in table B.1 on the following page.

Table B.1: Parts List

Category	Parts	Man/Supplier	Part Number	Amount
Power supply				1
Cables	Power Cord	Belden/Simmonds	17236	1
	Red(5V), AWG 14	Belden/Simmonds	8916, colr 2	6 m
	Black(GND), AWG 14	Belden/Simmonds	8916, colr 10	5 m
	Blue(2.1V)	Belden/Simmonds	8916, colr 6	0.75 m
	Twisted Pair			3 m
Connectors	High Current	Panduit	CB175-38-Q	2
	Crimp	Panduit	PN-14-10R-C	24
Power Connector	Plug	Molex	6 circuit	1
	Receptacle	Molex	6 circuit	1
	Crimp Terminal	Molex	male/female	12
Clock/Reset Conn.	Female Housing	Molex	50-57-9202	3
	Male Housing	Molex	70107-0036	3
	Female Crimp	Molex	16-02-0086	6
	Male Crimp	Molex	16-02-0107	6
2.1 V Supply Parts	Heat Sink	Thermalloy	6401B-2	1
	Card			1
	Voltage regulator	LT/Electrosonic	LT1038CK	1
	100uF electrolytic cap.			1
	10uF srf. mnt. cap			1
	42.2 ohm, 1% res	Philips/Electros.	MRS25F42R2	1
	61.9 ohm, 1% res	Philips/Electros.	MRS25F61R9	1
	1N4002 diode			2

Table B.2: Parts for Serial conversion, Power up control and Byte Blaster boards

Board	Parts	Part Number	Amount
Serial	6 pin jack		2
conversion	SMD .1u cap.	0853C104K	8
	SMD 10u cap.	TAJB106M016	2
	SMD 49.9 ohm res.		2
	SMD 100 ohm res.		4
	SMD 1K ohm res.		4
	SMD 4.7K ohm res.		2
	RS232 controller	DS14C232	1
	One Shot	LS123	1
	Differential driver	SN75176B	2
	LEDs		2
Power up	Ethernet connector	Molex 95003-2881	6
controller	SMD .1u cap.	0853C104K	4
	SMD 10u cap.	TAJB106M016	1
	SMD 100u cap.	KME16VB101M6X11LL	4
	SMD 33 ohm res.	NRC10J330TR	1
	SMD 49.9 ohm res.	080549R9-F	4
	SMD 110 ohm res.	0805-110R-F	4
	22 K ohm, 5 % res.		3
	200 K ohm, 5 % res.		1
	One Shot	LS221	2
	NAND	LS00	2
	LEDs		4
Byte	10 pin connector		1
Blaster	10 pin ribbon cable		10 cm
	Parallel connector		1
	SMD 0.01u cap.		2
	SMD 33 ohm res.	NRC10J330TR	7
	SMD 49.9 ohm res.	080549R9-F	2
	SMD 100 ohm res.		2
	Tri state driver	LS244	1
	LEDs		1

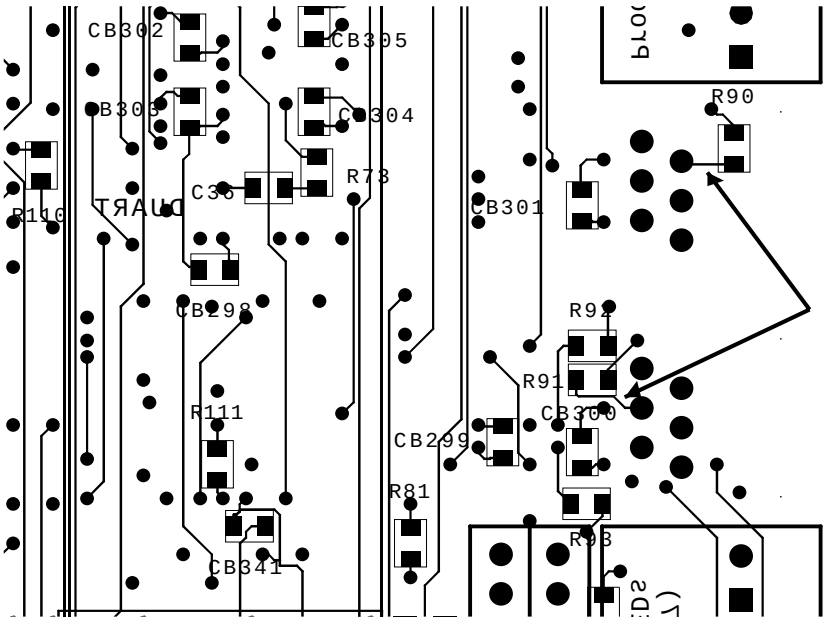

Figure B.1: Traces cut on the back of processor cards

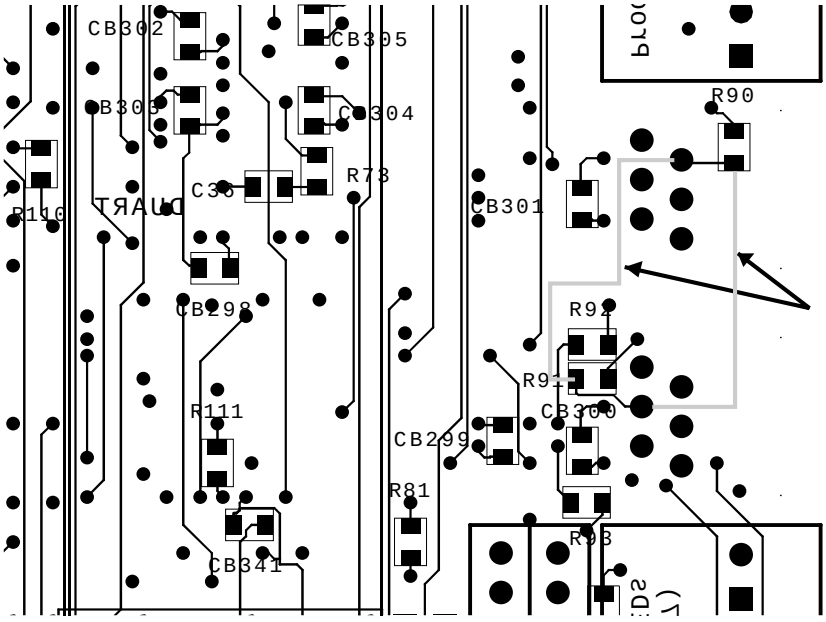

Figure B.2: Patches added to back of the processor cards

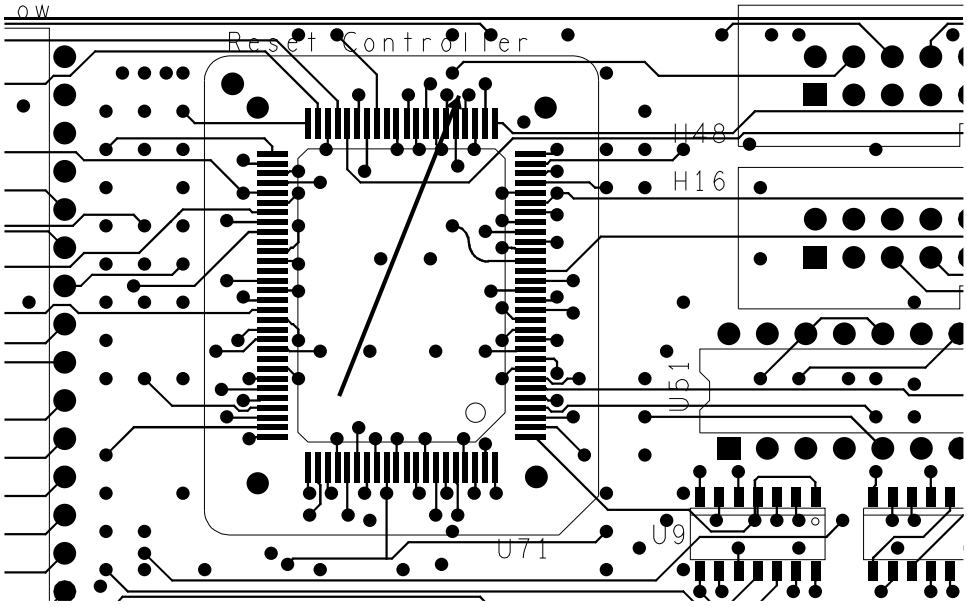

Figure B.3: 1st resistor to be added to the first batch of the Proc R3 cards

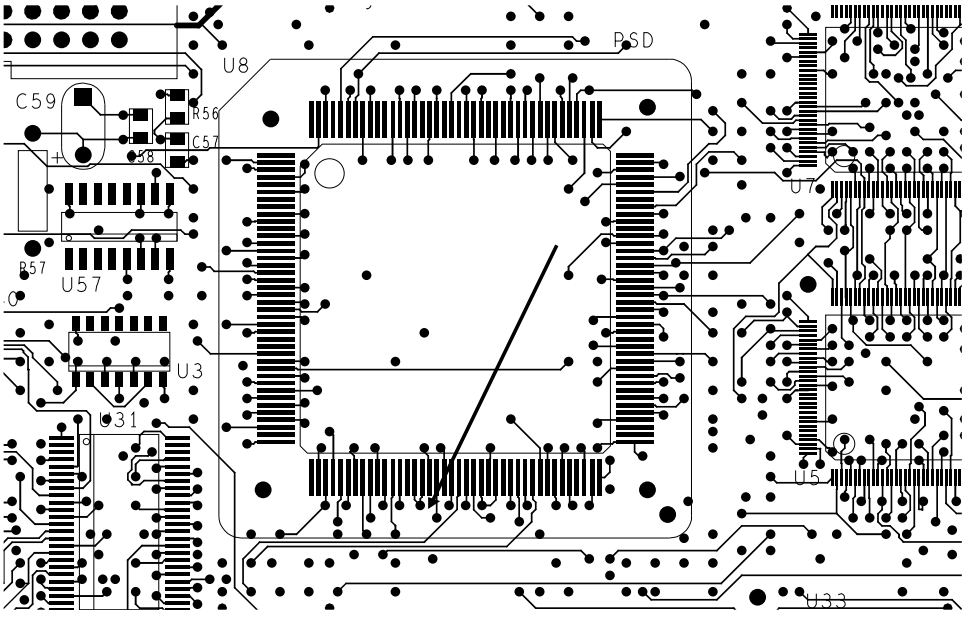

Figure B.4: 2nd resistor to be added to the first batch of the Proc R3 cards

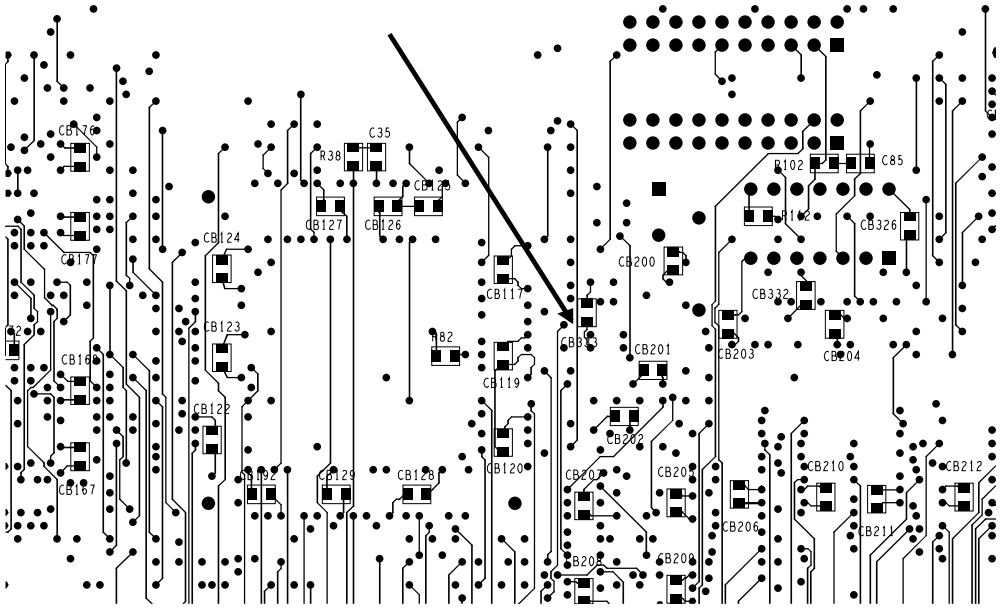

Figure B.5: 3rd resistor to be added to the first batch of the Proc R3 cards

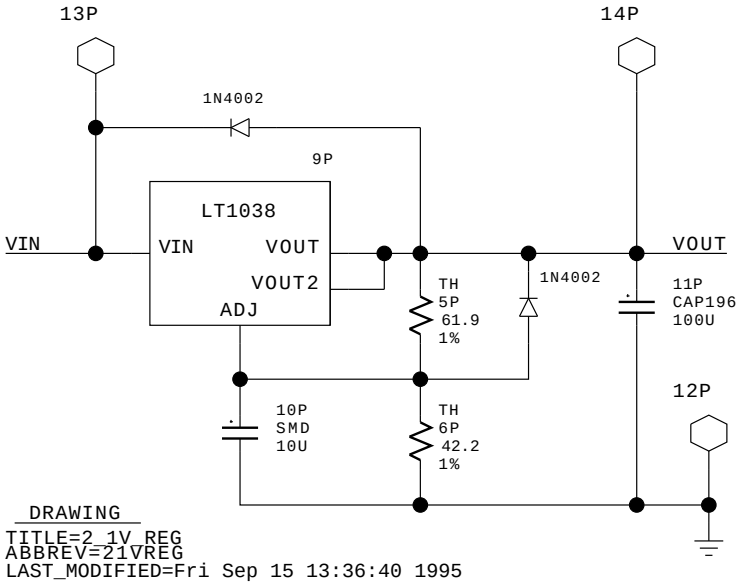

Figure B.6: Schematic of 2.1 V supply

Appendix C

Using The Powerup Controller

C.1 Overview

This document describes the use of the powerup controller designed for Numachine. Its purpose is to power up all the stations in sequence so that the wall sockets are not overloaded by all the stations coming up at the same time. The controller also distributes a synchronous clock and reset signal to all the cards on a local ring.

C.2 General Description

C.2.1 Structure

The powerup controller has 6 ethernet connectors as shown in fig. C.1 on the next page. From these, four connectors connect to each of the four stations on a local ring; the connector on the right is used to control the local station and the connector on the left is used to connect this board to the inter ring interface and through it to the other powerup boards on different rings. This board can be powered through the connector to the inter ring interface. This board needs to be supplied with +5 V and ground any time when the station power supplies are plugged in.

C.2.2 Using the controller

Each station controlled needs a powerup board. However, only one board on each local ring acts as a controller. The rest are used only as connectors to the stations. For each station used a category 5 ethernet twisted pair cable needs to be inserted in a connector, starting with connector 1. All the cables need to be inserted in adjacent slots. Each cable is then inserted in the local station connector of the boards. That is there is a cable returning to the initial board.

If more than one local ring is used, the controlling board must be connected to the inter ring interface, again using an ethernet cable. Section C.3 on page 111 describes the connections needed in the inter ring interface.

In the case when the board is connected to the inter ring interface, it can be powered from there. Only the controlling board needs to be supplied with power since the other three act only as connectors.

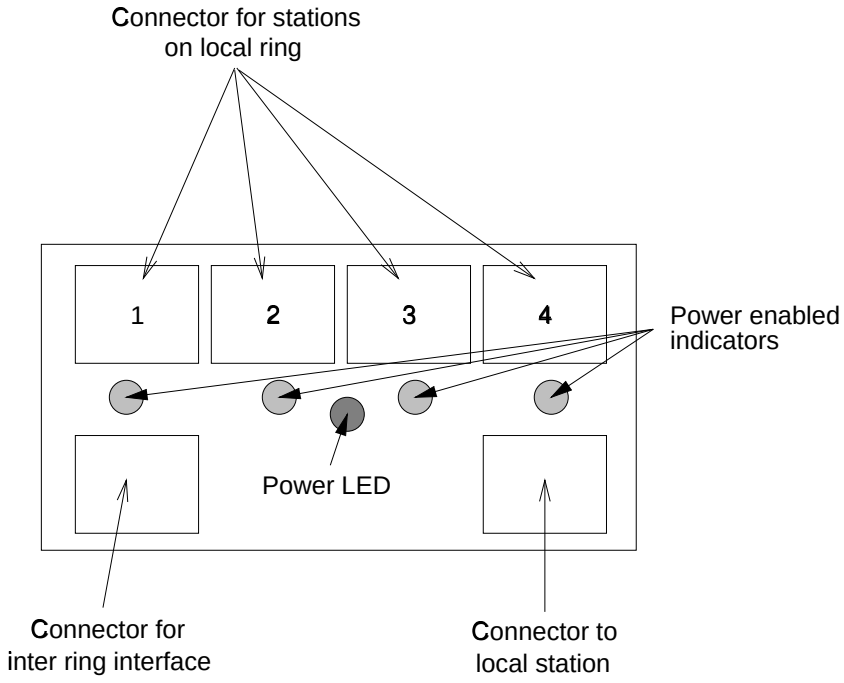

Figure C.1: Structure of the board

C.2.3 Connection to Local Station

There should be one powerup board for each station used. The board is connected to the stations using the jumper holes in the top right corner. The assignment of these holes is shown in fig C.2. The clock lines must be of equal length on each station.

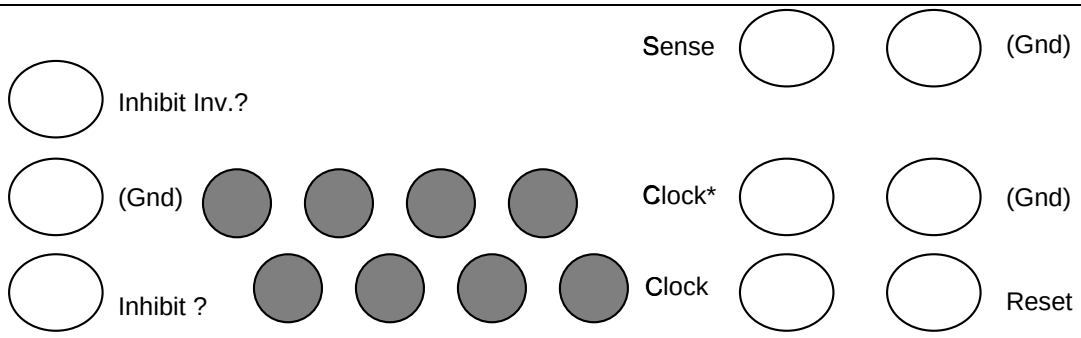

Figure C.2: Jumper holes to connect board to local station (back view)

See section C.4 on the facing page for a description of the labels in the diagram. The ground line is not connected on the board. It has not yet been determined if it is needed.

On all the boards, 5 clock lines must be connected from the clock card of the station to the clock jumper holes shown in fig C.1. Care must be taken to keep the phase of the clocks the same on all the connections.

C.3 External Circuitry Needed

The inter ring interface connector is used to connect the powerup controller to other controllers controlling different rings. The reset signal is also obtained through this connector. The powerup board supplies a clock signal to the inter ring interface as well.

If only one local ring is used there are two options to take. If the powerup controller is left unpowered, it will let all the power supplies come on. Otherwise, a switch can be mounted at the end of an ethernet cable which plugs into the inter ring interface jack. The position of the switch is shown in fig C.3. In this case it is easiest to remove the power LED and connect power there.

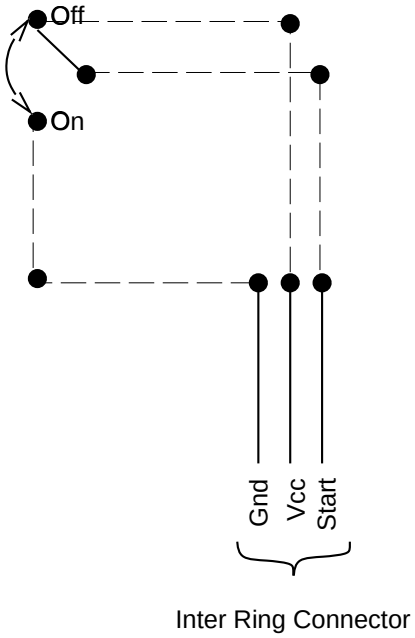

Figure C.3: Switch to be used if only one local ring is used

C.4 Pinouts

C.4.1 Connectors to Stations on the Ring and to Local Station

Figure C.4 on the next page shows the ethernet plug that needs to be connected to the stations on the local ring and to the local station connector. It is shown so that the release tab is on the bottom.

The inhibit pin turns off the power supply when asserted. Inhibit Inv. is the inverted version of Inhibit and it is used for older power supplies. Seq is high only for the first connector and low for the rest. Reset distributes a reset signal. Sense is used to sense when the power supply has come up. Clock and Clock* distribute a synchronous clock.

C.4.2 Connector to Inter Ring Interface

Figure C.5 shows the plug that is connected to the inter ring interface. Again, the release tab is on the bottom.

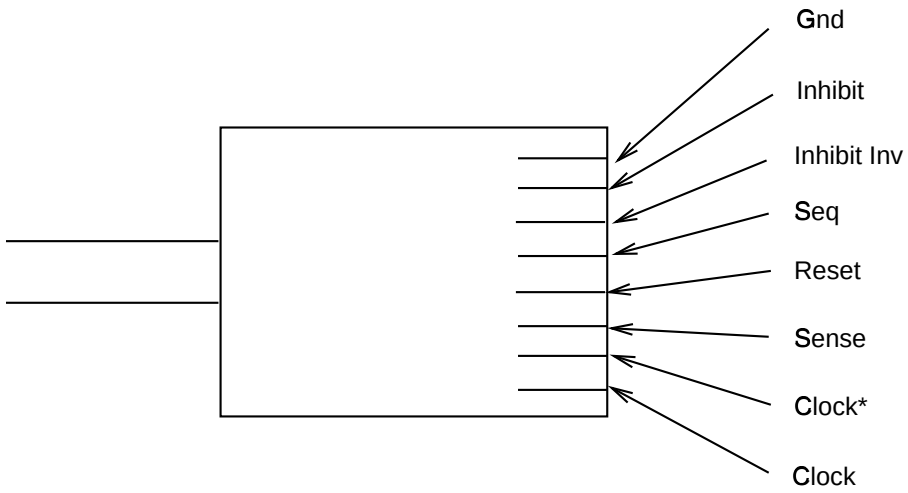

Figure C.4: Pinout of connector to local ring and local station

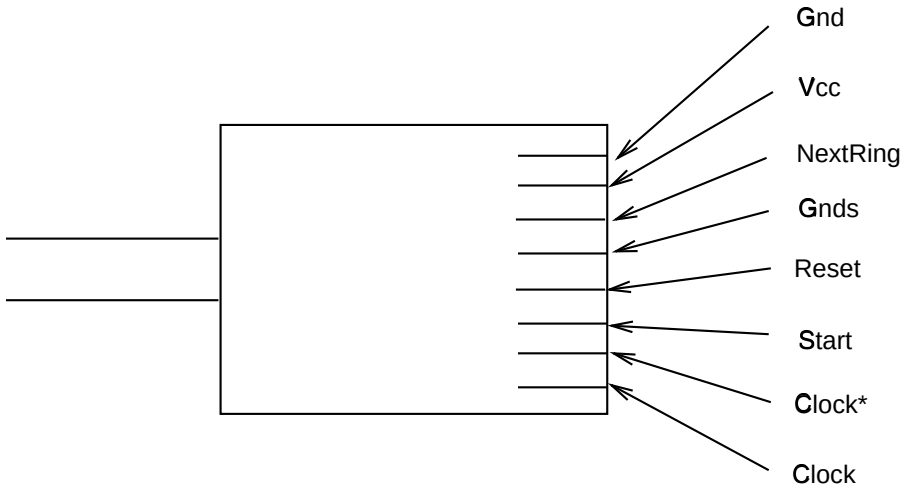

Figure C.5: Pinout of connector to inter ring interface

NextRing goes low when the next local ring should power up. Start signals when this ring should power up.

Appendix D

How to Modify this Document

D.1 Location of this document in CVS

This hardware maintenance manual is writen in LaTeX2ε. The files are all stored ion CVS and anyone wishing to modify this document can check out their own copy, make any cahnges, and then check it back in again. The repository is currently located at:

/stumm/d0/tornado/torncvs/tornado/doc/hw/hw_maintenance_manual

Bibliography

[CGG+97] Steve Caranci, Alex Grbic, Robin Grindley, Mitch Gusat, Orran Krieger, Guy Lemieux, Kelvin Loveless, Naraig Manjikian, and Zeljko Zilic. *NUMAchine Principles of Operation for System Programmers*. Dept. of Electrical and Computer Engineering, University of Toronto, Ontario, Canada, 1997.

[Grb96] Alexander Grbic. Hierarchical directory controllers in the NUMAchine multiprocessor. Master's thesis, University of Toronto, Dept. of Electrical and Computer Engineering, 1996.

[Gus96] Mitch Gusat. *Implementation of the I/O Card for the NUMAchine Multiprocessor*. Dept. of Electrical and Computer Engineering, University of Toronto, Ontario, Canada, 1996.

[Hei94] J. Heinrich. *MIPS R4000 Microprocessor User's Manual*. MIPS Technologies, Inc., Mountain View, CA., second edition, 1994.

[IEE90] IEEE. Futurebus+ specification ieee ???, 1990.

[Int95] Integrated Device Technologies. *IDT79R4640 and IDT79R4650 Risc Processor Hardware User's Manual*. Santa Clara, CA, 1995.

[Lem96] Guy Lemieux. Hardware performance monitoring in multiprocessors. Master's thesis, University of Toronto, Dept. of Electrical and Computer Engineering, 1996.

[Lov96] Kelvin Loveless. The implementation of flexible interconnect in the NUMAchine multiprocessor. Master's thesis, University of Toronto, Dept. of Electrical and Computer Engineering, 1996.

[MIP94] MIPS Technologies, Inc. *MIPS R10000 Manual*. Mountain View, CA., 1994.

[O/S96] O/S Group. *Implementation of the Tornado Operating System for the NUMAchine Multiprocessor*. Dept. of Computer Science, University of Toronto, Ontario, Canada, 1996.

[PVL92] Peter Pereira, Zvonko Vranesic, and David Lewis. *MC68000 Microprocessor Laboratory—Notes and Experiments (Version 1.3)*. Computer Group, Dept. of Electrical and Computer Engineering, University of Toronto, 1992.

[Tos94] Toshiba. *Static RAM Data Book*. USA, 1994.

[V+95] Zvonko G. Vranesic et al. The NUMAchine Multiprocessor. Tech. Rep. CSRI-324, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, April 1995.