NUMAchine Global Ring Hardware Design

Guy Lemieux, Steve Caranci, Robin Grindley, Kelvin Loveless

March 4, 2001

Abstract

This is a document about the global ring hardware, the top-level memory interconnection medium, designed for NUMAchine. Planning began in the summer of 1997 and board manufacturing ended in January 1998.

1 Introduction

The NUMAchine multiprocessor is a shared-memory parallel computer employing multiple, hardware-coherent caches (*i.e.*, it is a CC-NUMA system) that is designed to be scalable, modular and cost-effective. By being modular, a NUMAchine system can be constructed from only one processor, or it can easily grow, one by one, to 64 processors. Ideally, expanding such a system should not incur unnecessary expenses such as discarding old components or purchasing large components which contain excess capacity for future upgrades. As well, these components must still be designed to perform well when the system is operating at its largest size, and herein lies the challenge.

The NUMAchine design addresses this challenge by breaking up the system into smaller distributed-memory nodes, or *stations*, which are connected together in a hierarchical-ring network topology. Such a network is depicted in Figure 1. Memory transactions are be divided up into one or more *packets* for transmission across the rings.

This paper will discuss the design of the switching elements used within the ring-based network, beginning first with an understanding of the NUMAchine system environment.

The remainder of this paper is organized as follows. Sections 2 and 3 discuss the NUMAchine prototype specifications and the global ring design requirements.

Sections 4 and 5 discuss different choices that were considered during the design process, including provisions for monitoring. Section 6 describes the final global ring design. Sections 7, 8, and 9 describe the verification, manufacturing, and testing stages of the design. Section 10 provides a summary of lessons learned and suggestions for future possible designs. Lastly, a multi-part appendix describes details of the global ring implementation which are difficult to glean from the design files.

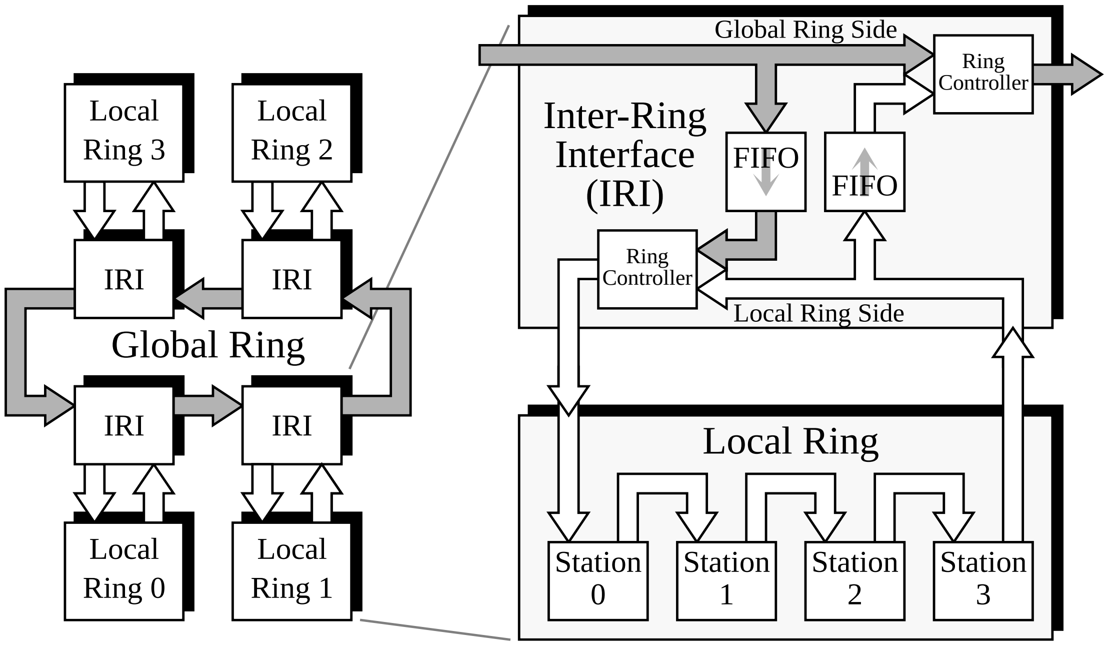

Figure 1: NUMAchine architecture.

2 NUMAchine Prototype Specifications

A prototype NUMAchine system was constructed using commodity parts and FPGAs assembled onto custom-designed printed circuit boards. This section describes the different components used in NUMAchine, the types of memory transactions, and the use of *filtermasks* as routing directives.

2.1 System Components

The prototype design supports a 64-processor system containing four local rings and one global ring. Each station contains up to four processors, two memory cards, one Input/Output (I/O) card, and a Network Interface Card (NIC). The entire system operates slightly below the target clock rate of 50 MHz.

The prototype is constructed using 150 MHz MIPS R4400 processors. Although MIPS R10000 processors were originally intended, they were too costly and unavailable early enough to be used. The memory hierarchy includes a 1MB external secondary cache, an 8MB network cache, and up to 512MB of main memory per station. The cache coherence protocol uses a write-back/invalidate scheme which supports a sequential consistency programming model.

The NUMAchine station bus is based on the Futurebus+ physical backplane, but a custom communication and arbitration scheme is used. This split-transaction bus supports a peak bandwidth of 400MB/s, or 64 bits per 20ns cycle.

The ring network supports the same 400MB/s bandwidth target. The local rings are formed by bit-parallel, unidirectional connections of high-density, impedance-controlled flexible ribbon cable. Each packet is sent one at a time according to a slotted-ring protocol and contains 64 bits of address/data payload plus 49 bits of parity/ECC, routing, and other control information.

2.2 Memory Transactions

Within a station, up to four processors communicate with local memory and (I/O) cards (or modules) over a shared, split-transaction bus. A typical bus arrangement is illustrated in Figure 2. Each memory card is mapped to a unique physical address and is accessible to any processor in the system. When a processor reads from a memory address belonging to a remote memory card, the read request must leave the bus through the (NIC), travel over the ring network according to the filtermask specification (described below), and arrive at the destination bus through the remote NIC. Eventually a response will travel back through the network to the original requester. Write transactions are similar, except that only the write itself needs to be transported — there is no acknowledgement (not even a negative acknowledgement), so correct delivery must be guaranteed.

Memory transactions involving coherence operations get more complicated. For example, suppose that the remote memory card has determined this data is already being widely shared among many processors, but the requesting processor wishes to have exclusive access (this is likely to satisfy a pending store instruc-

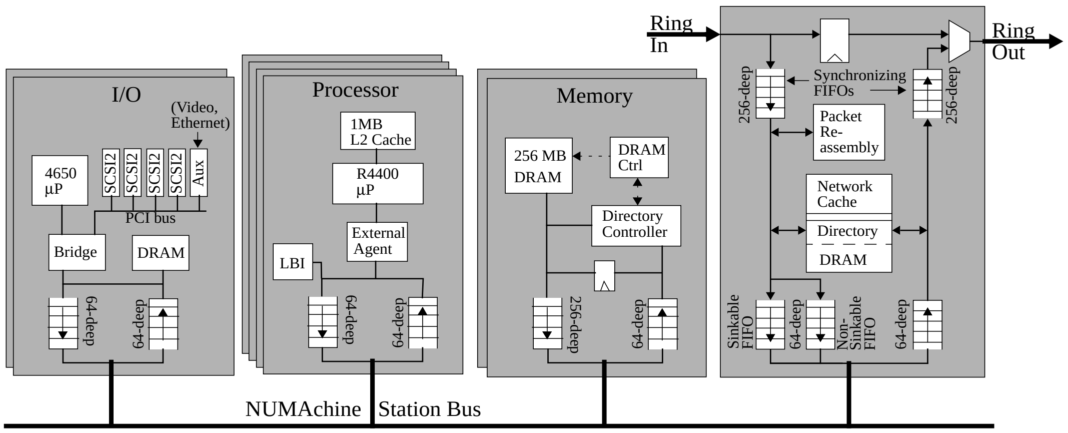

Figure 2: NUMAchine station.

tion). In this case, the memory must send out an invalidate message to each shared copy, including the original requester. A single invalidate message leaves the NIC at the remote memory station, but it is marked as a multicast transaction. The ring network must deliver multiple copies of this message, one to each cache. When a copy is delivered to the original requesting processor, it signifies completion and that processor is allowed to proceed with the pending instruction. Even more complicated examples are possible, including cases where the destination is too busy and must return a negative acknowledgement.

The delivery of coherence operations imposes a number of restrictions on the interconnection network such as guaranteed delivery, preservation of ordering, and efficient support for multicasts. These requirements and other technical details will be specified more thoroughly and precisely in the next sections.

2.3 Filtermasks

All memory transactions which leave a NUMAchine station are routed over the ring network according to the destination encoded in a set of *filtermask* bits. These bits can precisely specify the location of a single station in the ring hierarchy, or they can identify the location of a number of stations in an imprecise manner described below. The filtermasks are used for both routing and storing cache coherence directory state.

The filtermask is divided into two groups, one per level in the hierarchy. Each group is a bitmask that indicates which ring (or which station on that ring) is of interest in that level. When multiple stations are specified, bitwise ORing of the filtermasks for the individual stations is done. This may result in multiple set bits in each filtermask group, and it can lead to imprecision by specifying more stations than required. This over-specification is acceptable and accounted for in the coherence protocol. An example of two filtermasks, one which is precise and one which is over-specified, is given in Figure 3.

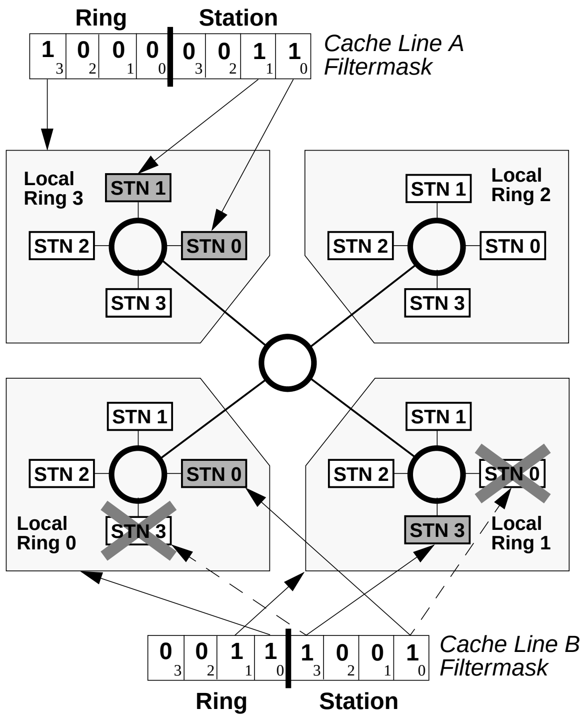

Figure 3: NUMAchine filtermask examples.

3 Global Ring Design Requirements

The global ring acts as a switch or router: packets from local rings are received, switched, and delivered to the correct destination or destinations. To create a ring hierarchy, one module called the Inter-Ring Interface (IRI) is connected into each local ring. These IRI modules are, in turn, connected to each other to form the global ring, as shown in Figure 1. A total of four connections are required for incoming and outgoing traffic from/to the local and global rings. Each IRI acts as a simple switch between the local and global rings, receiving data from the incoming side and sending it to the outgoing side of each ring.

3.1 Switching Data

Each IRI must handle four different types of packet operations, depending on the state of the filtermask bits. These four operations are described as follows:

1. **Pass-through.** In the simplest operation, data on one ring passes through the switch and continues on the same ring.

2. **Upstream and downstream insertion.** An *upstream packet* from the local ring is inserted into the global ring, or a *downstream packet* from the global ring is inserted into the local ring. Any packet which is to be inserted must first have been extracted by one of the following two operations.

3. **Upstream extraction.** Data on the local ring may be destined for a remote location, so a packet may be switched to travel in the upstream direction. This case always results in a free slot on the local ring, which the IRI may utilize with a downstream packet as a protocol option (see below).

4. **Downstream extraction.** Data on the global ring may be switched to travel in the downstream direction. This generates a free slot on the global ring *only* if this is the final destination for the packet. Otherwise, the packet is being multicast, so it remains on the global ring and is sent to the next ring segment. Like the previous operation, a free slot generated on the global ring may be utilized immediately with an upstream packet if a protocol option is enabled (see below).

Since data may be simultaneously received for both pass-through and insertion to the same ring, some type of buffering is required. The pros and cons of various buffering options are addressed later in Section 4.1.

The extraction operations generate free slots when the destination is reached. The policy for using these free slots is described next.

3.2 Free Slot Policy Option

Whether the IRI can immediately utilize the free slot that is generated by the upstream or downstream extraction is left as a protocol option called *use_free_slots*.

The global ring design must be able to separately enable the use_free_slots protocol for the global ring and each of the local rings. In this way, the best policy can be determined experimentally by running real programs.

If the use_free_slots option is not used, free packet slots must be passed on the ring segment to the next node. By doing this, the IRI is being more fair and is supposed to prevent this segment from saturating the network.[1] Unfortunately, this policy also guarantees that maximum ring utilization will never be attained because an empty slot must always be transmitted for one unit of time on at least one segment. This results in a total bandwidth reduction of between 50% (worst case) and $1/N$ (best case), where N is the number of ring segments.[2] In contrast, by immediately filling the free slot with a new packet, the risk of starvation may increase but maximum ring bandwidth can be reached as needed.

3.3 Delivery Requirements

The coherence operations described in the previous section require a number of delivery requirements which must be satisfied for correct operation. These requirements are as follows:

1. **Guaranteed delivery.** All packets must be delivered to the destination. This is particularly important since write operations are unacknowledged. For other transactions, a negative acknowledgement may be allowed, but the rings should not generate them. Instead, buffering must be sufficiently large to accept all in-flight packets, and the flow-control scheme described below can be used to prevent overflow.

2. **Delivery order.** Data being sent on a ring must be delivered in-order. It is acceptable to interleave packets from different sources, but the ordering from each source must be preserved.

3. **Descending multicast order.** Multicast packets normally continue along the ring as a duplicate copy is sent down the hierarchy. The duplicate copy must not cause a response to be generated which can race past the continuing multicast. A simple but overly strict way to ensure this delivers the

[1]Note, however, that starvation is still possible if all injected data from one ring segment travels to the previous node on the ring.

[2]In a full NUMAchine system, peak utilization would be limited to 80% (since $N = 5$).

continuing multicast to the next ring segment at the same time as the duplicate descends. Some IRI buffering organizations are unacceptable because they cannot guarantee this ordering.

4. **Sequencing.** Some packets, notably invalidation messages, must be seen to occur in the same global order by all processors. The NUMAchine solution is to require a sequencing point on each ring. Packets requiring sequencing are specially marked by setting a *sequence_request* bit. Prior to delivery, these packets must ascend the ring hierarchy as high as required and pass the designated sequence point on that ring to clear the sequence_request bit. Only then can the packet proceed down the ring hierarchy for delivery.

5. **Flow control.** Finite-sized buffers make flow control a practical requirement: for example, a sufficient number of cache writebacks can easily fill any buffer. The NUMAchine flow control policy is designed to avoid deadlock and requires that an individual ring stop delivering all packets if any of its destination buffers becomes full.

3.4 Flow Control and Deadlock Avoidance

For deadlock avoidance, NUMAchine flow control policy requires an unblocked downstream path to exist, even if flow control is invoked and the rings are stopped. This allows the network buffers to be emptied so that ring traffic can resume. The NIC design creates separate logical non-sinkable and sinkable networks (sometimes called request and response networks) so that separate physical networks are not required.

3.4.1 NUMAchine Deadlock Problem

During the global ring IRI design, we considered the deadlock problem to have been solved. However, there may be a deadlock situation in NUMAchine. Suppose the following events occur at exactly the same time:

1. The upstream buffer from Local Ring 0 (LR0) fills. LR0 stops, waiting for it to flush.

2. Since LR0 is stopped, the downstream buffer to Local Ring 0 fills. The global ring (GR) stops, waiting for it to flush.

In this case, the GR will not restart because its downstream buffer is full, assuming that it will be emptied eventually. Likewise, LR0 will not resume as it is assuming the GR will eventually drain the upstream buffers.

This situation has never been encountered during simulation or during the limited use of the global ring. Hopefully, the global ring buffers are sized sufficiently large to avoid this problem, but it cannot be completely eliminated with the current design.

3.4.2 Avoiding the NUMAchine Deadlock Problem

Paul McHardy suggests a way to avoid encountering this situation using 2 flow control signals on the LR: *upstream_blocked* and *downstream_blocked*. This scheme hasn't been completely thought out, but would work as follows.

The first signal, downstream_blocked, indicates that a downstream path to a station is full. This would stop all stations and the GR from injecting packets into the LR. NUMAchine policy guarantees that the downstream path will be eventually drained, so eventually this signal will be cleared and normal ring operation can resume.

The second signal, upstream_blocked, would indicate the GR upstream path is full. All stations must stop injecting packets to the LR, but the GR would always be allowed to inject packets since they always travel downstream. A downstream path will become available to empty the GR queues, so eventually this signal will be cleared and normal ring operation can resume.

Even if the global ring stalls and both upstream_blocked and downstream_blocked are asserted simultaneously, stalling the local ring, the rings should not deadlock. The downstream path on each NIC should drain for all stations, eventually clearing downstream_blocked. This would allow the GR to inject LR packets, eventually draining its queues and clearing upstream_blocked.

As an optimization, it may be possible to allow packets from stations to continue if they travel upstream yet the downstream_blocked signal is asserted, for example. This is probably fine until the first packet to be blocked is encountered. At this point, it is easiest to stall all packets leaving the station due to possible memory coherence problems. It isn't clear whether the memory coherence policy will tolerate upstream-destined packets racing past downstream-destined (i.e., blocked) ones (or vice-versa). Hence, the safe thing to do is stop sending any packets as soon as the first blocked one is encountered.

This scheme needs to be thought out more carefully to ensure it does solve the deadlock problem.

3.5 Global Ring Wish List

There are a number of desirable features to include in the ring design. These desirables are listed as follows:

1. The global ring must impose low latency delays on any packet that travels through it, in both a loaded and unloaded system.
2. The global ring should support a variety of clock rates to explore the effects of varying bandwidth.
3. The global ring should follow the NUMAchine ideals of a modular system.
4. The global ring should have a higher bandwidth than the local rings, particularly since rings are criticized for having poor bisection bandwidth.
5. The global ring should be modular: easy to test and debug, easy to repair or re-spin boards, and easy to upgrade.
6. The global ring should include network monitoring. (Time pressures forced us to exclude the monitoring design.)
7. It should be easy to add/remove local rings to/from the global ring. To minimize mechanical fatigue caused by rearranging cables, a programmable connection is considered superior.
8. The design should be flexible to permit exploration of different network topologies. For example, in this design, the global ring PLD logic can be reprogrammed to operate as a crossbar switch instead with no changes to the wiring.
9. Connecting four NICs directly to a global ring (instead of in a local ring) would be an interesting way to create a higher-bandwidth local ring (but with additional queuing latency). If the global ring could be reprogrammed to operate as a crossbar switch (see above), this would significantly improve available bandwidth. A hierarchy of global ring switches could even be formed.

4 Design Alternatives

A number of practical design alternatives were considered in the light of different buffering options, ring speeds, topology flexibility, physical construction and packaging constraints.

4.1 Buffering Options

IRI switching consists of taking packets from two input ports and switching them to two output ports. Any switching topology which satisfies this could have been used, provided two main constraints are satisfied: packets from each input are kept in-order, and a multicast from the global input is delivered to the global ring output no later than when it is delivered to the local ring output. Both of these constraints are requirements from the NUMAchine coherence protocol.

There are four primary ways to do switching, illustrated in Figure 4 as options a) through d). The first option, input buffering, is not reasonable because the local and global ring clocks cannot be decoupled. The second option, full output buffering, solves this problem but uses two additional FIFOs. The third option, a partial output buffering which gives switched data priority, has the same drawback as the first. The fourth option, which is the one chosen for use in NUMAchine, effectively decouples the ring clocks and uses only two FIFOs.

Buffering options a) through c) all share one advantage and a corresponding disadvantage. Since a FIFO is used along the pass-through path, it is possible to keep multiple packets together in consecutive ring slots. This would group together all of the data for an entire cache line, for example. Since transactions would no longer be fragmented, the NIC design could be simplified as follows: the latency bottleneck introduced by the staging SRAM and packet re-assembly logic would be eliminated. The staging SRAM control signals (SMA numbers) which are sent over the ring would also be eliminated.[3]

The corresponding disadvantage, however, is that data which stays on the same ring must always pass through a FIFO at every stage along the ring. Most FIFOs require at least two cycles of latency to store the data and change the empty flag

[3]Fragmenting transactions into packets as done in NUMAchine also imposes a worst-case source of latency on delivery of large transactions. Interleaving the packets from different transactions makes all of them have high latency because each must wait for the last packet before continuing. If the packets were kept together, the first one would be delivered with a low latency and the last one would have the same high latency as the fragmented and interleaved case. Of course, this latency is more significant with the larger cache line size.

status, so the latency of an unloaded system increases. More importantly, ring latency is higher in a loaded system since it must wait for multiple cycles at every ring segment. Without proper detailed simulation, it is not clear whether the advantage outweighs the disadvantage.

Options b) and c) also have one critical drawback. Using either of these organizations introduces a race condition in the coherence protocol. This condition is listed as the **descending multicast order** requirement in Section 3.3. It is very difficult to work around this problem, so it is better to avoid these organizations.

Lastly, it is worth noting that the buffering options a) through c) might impact the NUMAchine deadlock avoidance scheme. Presently, the global ring is allowed to flush its data when a downstream FIFO becomes almost-full, but no new ring packets are injected upwards. With FIFOs on the ring path, this method is not appropriate because too much data is present on the ring and this would overflow the downstream buffers. Hence, the data flow on the global ring must actually be stopped rather than flushed if these buffering options are employed.

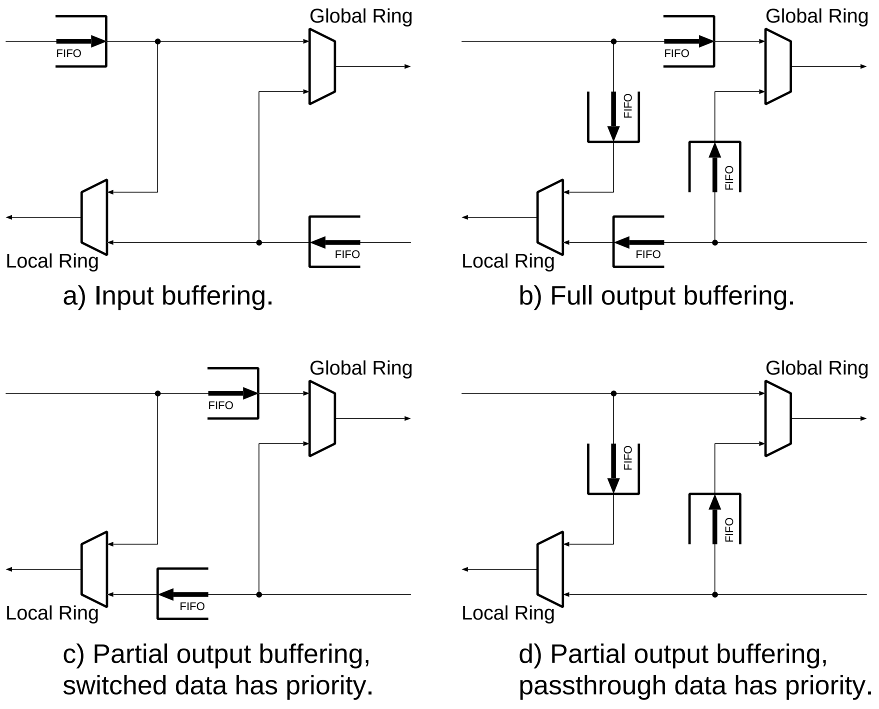

Figure 4: Of the four buffering options, the last one was chosen for the IRI.

4.2 Physical Packaging Options and Critical Path Reduction

The most apparent way to construct the global ring is to design a single IRI card which can be inserted into every local ring. Forming the global ring would simply require cables to connect the IRI cards in a ring. The primary drawback for this method is a limited global ring speed: the global ring would have to use the same commodity parts chosen for the local ring and this would have similarly limited the global ring speed to 50 MHZ.

4.2.1 High-Speed Signalling

To increase the global ring speed, the use of other signalling methods employing optical and other high-bandwidth communications schemes was briefly explored. Unfortunately, the complexity of the special devices required and the increase in latency they imposed was intolerable compared to the bandwidth they offered. An optical approach would have been particularly interesting because it would have allowed local rings to be placed at greater distances, possibly spanning different rooms or even buildings. However, it was easier to build a wider ring than to employ esoteric devices.

If the NUMAchine project had the resources to design custom chips, or if a redesign is to be done today, the choice of signalling schemes should be revisited. For example, current FPGAs support a multitude of high-speed signalling schemes such as LVDS which can make the IRI design much simpler.

4.2.2 Close Placement of Components

Since custom silicon design and high-speed signalling was not an option, the best remaining option was to remove the latency imposed by the cabling. If the entire global ring is constructed in as small a package as possible, propagation delay through the wires would be minimized. Constructing the global ring on one circuit board would shrink the propagation delay as much as possible. Then, clock-to-output and setup time delays of the registers and FIFO buffers are the major contributors to the critical path.

The clock-to-output was particularly important, since data could come from two sources: the FIFOs or the discrete registers. The FIFOs have the longest clock-to-output delays (8ns), so their output would have to be registered again. Accounting for driver fights, signal ringing, and data skew also caused concern.

4.2.3 Ring on One Chip

The best approach appeared to be moving the entire global ring onto one FPGA device to act as a router. This eliminates analog concerns (driver fights and signal ringing) because multiplexing would be done using internal logic. By doing this, we believed the critical path moved on-chip and could be thoroughly explored using MaxPlus+II before manufacturing any boards.

4.2.4 Dividing the Datapath

Since no FPGA at the time had enough pins or FIFO memory available, the design still had to be decomposed into multiple devices. Dividing the IRI datapath into a number of distinct bitslices, each 18-bits wide, was the best option. A number of other bitslice sizes were investigated, including selection of the best PLDs and external FIFO parts for each size. The 18-bit size was the best for a number of reasons: it used the fewest parts, it occupied the smallest amount of board space, and it was the least expensive in chip cost.

With the 18-bit size, a single low-cost FLEX6000 device is capable of implementing the global ring switching logic, and eight external 18-bit wide FIFOs buffer the data.[4] Of the $64 + 49 = 113$ packet signals, 108 signals are switched using 6 datapath bitslices. The remaining five bits contain routing information (the 4-bit filtermask and a sequence request bit) which are handled by a separate *routing controller*.

4.2.5 Distributing Control Information

The best way to make routing decisions switch with the routing bits was considered along with the datapath bitslices. One extreme is to have a centralized controller send signals to each bitslice. The other extreme is to replicate the routing bits such that there is one controller per bitslice. These options, and the area in between, were investigated.

First, consider the placement and area of the datapath bitslice logic. The external FIFOs must feed the routing FPGAs at the global ring speed, so they must be located close together in a cluster to reduce propagation delays. Each cluster contains 9 chips, so a board containing 54 primary chips would be quite large. Distributing the clock and the routing information across a board of this size would

[4] There are four local rings and each requires one upstream and one downstream FIFO buffer.

be difficult to do at very high speeds. This suggests having multiple, distributed controllers on the board and replicating the routing bits for each one.

Replicating the routing bits for each bitslice makes all routing decisions local, but it wastes FIFO space (some FIFO bits would be used to store the routing bits) and the number of bitslices would increase by 50% to nine (after storing the 5 routing bits, only 13 data bits per slice are available to switch the 108 signals). Sharing routing bits between 2 or more bitslices reduced the overhead slightly, but complicated physical board layout.

Ultimately, it was decided any replication or sharing scheme could also be problematic – if two routing controllers made a different routing decision, the resulting state would be unrecoverable. This would be difficult to debug such a board, since an intermittent error in one routing bit of one controller would be very difficult to diagnose. Hence, a centralized routing controller was chosen to make decisions and send control signals to each bitslice. The controller itself fit on a single CPLD and the routing bits were stored externally in FIFOs, but only the upstream direction is needed. To get around the large board design problem, it was determined that the bitslices would be constructed on daughtercards.

4.2.6 Daughtercard Design

To reduce the size of the global ring board, each bitslice was placed on a *datapath daughtercard* and the controller was placed on a connecting *backplane*. The logic connecting to a local ring was also large, so each ring connection was placed on a *local ring daughtercard* as well.

This daughtercard design offers a number of distinct advantages. First, the daughtercards are all identical, so a few spares could be manufactured in case of defects. Second, the cards can be rearranged to isolate bit failure errors. Third, datapath cards had to operate at high frequency, and the simple 4-layer[5] daughtercard allow simple, short, and widely-spaced connections to be formed. Fourth, a redesign of the backplane is possible while reusing the datapath cards, so the expensive FIFO buffers are not wasted. Lastly, the daughtercards take advantage of the 3rd dimension, significantly shrinking the distance the high-speed control signals would need to travel from the controller.

[5]There were two power layers and two signal layers, simplifying debug and improving ground coupling.

4.2.7 Overall System Impact

Physically, creating the global ring on a single chip is an unrealistic way to build large systems. Such a scheme confines the global ring to a small, central area, creating the need for longer cables to travel from various local rings. This cable length would adversely affect local ring clock rate in a larger system.

Fortunately, this packaging issue was not a problem for our prototype size. Four local rings (of four stations each) should fit in two standard racks, with the global ring placed in the very center. Although we welcomed the ability to use higher speeds on the global ring datapath, it falls short of what is achievable using high-speed signalling.

5 Monitoring Design

Due to time constraints, the global ring was manufactured before the monitoring designs were finalized. This section documents the design effort that went in to the ring monitor.

5.1 Design Overview

The global ring monitoring design consisted of separate designs for monitoring global ring and local ring traffic. The primary operation for the ring traffic monitors is to count the different types of traffic passing by a particular ring segment[6] and to keep track of the queue depths and ring utilization.

The local and global monitoring designs can both be decomposed in the same way. A master controller determines when interesting events to monitor take place, and triggers the counting subsystem. Parallel events may be counted by dedicated counters in the master controller, or sequential events may be counted by a large number of counters stored in SRAM. The SRAM counters need logic to control a read-increment-write operation with the data and to select the proper counter address.

The local ring monitoring design was to be placed on the local ring daughtercards, and the global ring would have been placed on the backplane. Although the local ring daughtercards were constructed without monitoring at first, it would be possible to re-spin just these boards and add local ring monitoring ability. The

[6]This should always be the segment that acts as the sequencing point.

global ring monitoring required significant additional effort to be able to read back performance data, so it was abandoned at an earlier stage.

5.2 Accessing Monitoring Data

Ring monitors were memory-mapped in the NUMAchine address space so performance state could be initialized by writes and data could be read back. Writes to the monitor would enter the ring and be removed from the ring without descending. Reading was a more complicated matter, however.

To simplify the injection of response data into the network, it was decided that a processor can only read back monitoring data from its own local ring. As well, only processors on local ring 0 would be capable of reading back global ring monitoring data. Response packets were always to be injected into the local ring at the point where the downstream FIFO data is first read into a register before sending it along the local ring segment. When a response was ready, the local ring controller would take a packet of monitoring data in preference to regular downstream data. This should be infrequent enough to be fair, but a rate limiter enforced this to happen at most once every 8 cycles.

5.3 SRAM Counters

The local ring monitor heavily relies on having a large number of counters to count packets on every cycle. To do this, an SRAM counter subsystem using two banks of 32-bit high-speed synchronous SRAM was designed as shown in Figure 5. A monitoring controller (not shown) would determine that an interesting event had just taken place, including the possibility that a read or write to the SRAM was required. It would inform the *sramfsm* device of this and place the correct address (or counter number) on the read/write port address lines.

Data written to the SRAM comes from a 32-bit write port on the monitoring controller, but data read out from it is injected directly into the local ring (at the right time, of course). This way both SRAMs can be initialized at the same time with the same data, but they can be read out in parallel for increased performance. Note that monitoring events can occur every cycle and are directed to the proper bank by *sramfsm*. However, reads or writes to the SRAM may interrupt monitoring for a short while to complete the transaction.

A two-cycle read-modify-write operation is required to increment an SRAM counter, so the two banks are interleaved to count one event per cycle. On any particular cycle, only one counter can be selected so only sequential events can be

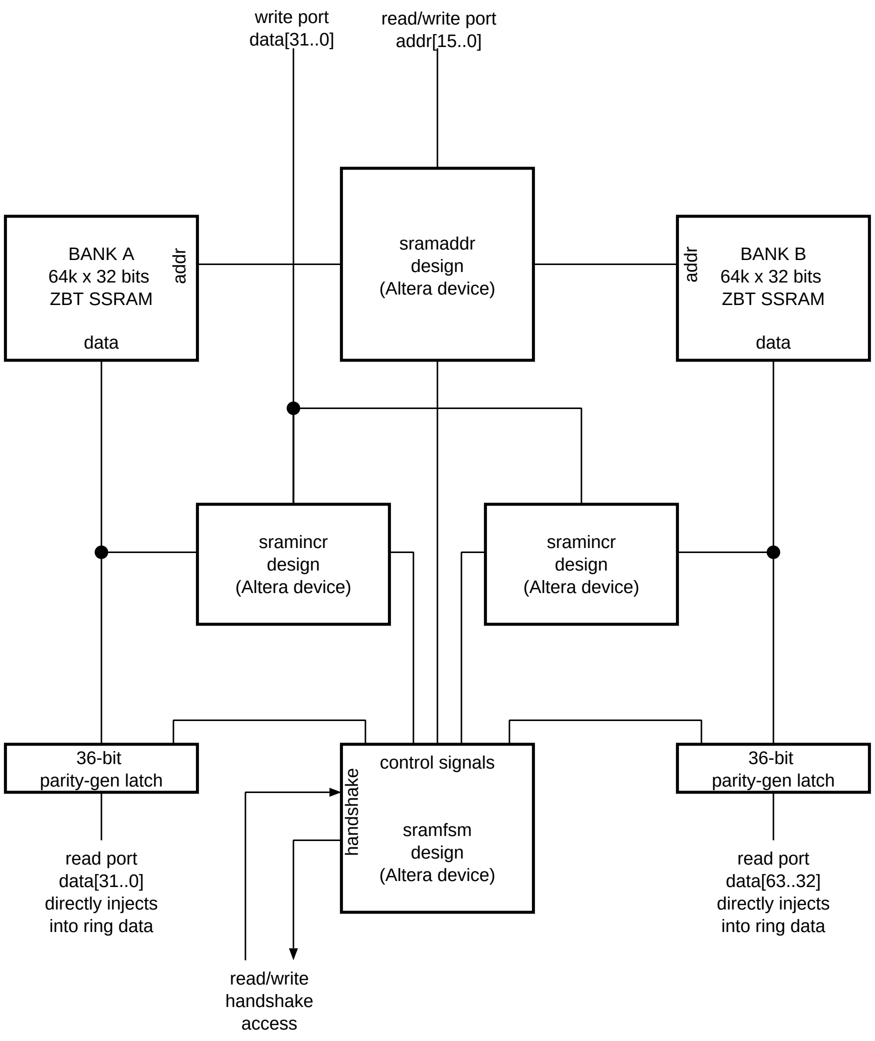

Figure 5: SRAM counter subcircuit used by the local ring monitor.

counted. When reading back SRAM data, both banks are accessed in parallel to form a 64-bit result.

The SRAM device must be carefully selected to support this two-cycle operation: most synchronous SRAMs contain an additional data output register to achieve higher clock rates, and other devices cannot do back-to-back read/write or write/read operations. This latter feature is usually called *ZBT* (Zero Bus Turnaround) or *NoBL* (No Bus Latency).

The increment logic generates an overflow signal so an appropriate reaction can be initiated by the master controller. Additional logic was designed in to support efficient initialization of all SRAM counters with a single NUMAchine write operation. As well, the SRAM could hold data to reprogram the monitoring FPGA. This gives software programmers a mechanism for instantiating a new monitoring circuit in the hardware.

5.4 Local Ring Monitor

A wide variety of events were considered important for the local ring monitor. A list interesting measurements are are shown in Tables 1 and 2. It isn't realistic to perform all of these measurements at once, so a sample local ring monitor was designed to keep a FIFO depth counter histogram. This measurement was a good choice as the first monitor circuit because it used all of the hardware components, required planning in two clock domains, and produced an interesting performance statistic.

The FIFO depth counter histogram circuit works as follows. Every cycle, the depth of the upstream and downstream FIFOs were measured. These two depths select a unique appropriate SRAM counter to be incremented. The SRAM data shows how much time each FIFO spent filled at every particular depth. This allowed data to be be correlated so that time spent at every upstream depth could be inspected only when the downstream FIFO was nearly full, for example.

5.5 Global Ring Monitor

The events that were considered important for a global ring monitor are shown in Table 3. It was determined that it is not necessary to snoop on the type of traffic carried by the global ring, since this is always available at the local ring level as it goes upstream. Also, the high speed of the global ring would have made snooping the packet contents more difficult. As a result, the monitoring items listed in the table include only overview-type measurements.

Table 1: Local Ring Monitoring.

Difficulty	Description and Commentary
EASY	free-running counter that obeys start/stop commands
ring link utilization	
EASY	number of free/not_free slots
EASY	number of cycles ring is halted due to stop_up
EASY	number of cycles ring is halted due to stop_dn
packet counts	
EASY	number of UP packets
EASY	number of DN packets
EASY	number of THROUGH packets
EASY	number of free slots
EASY	filter packets based on:
	data packets, address/cmd packets
	Nack, REQ/RESP, R, RE, WB, INV, UPGD etc.
HARD	separate packet counts by:
	PhaseID (must keep state)
	sending station (taken from SMA)
count degree of multicasts:	
EASY	count total # of destinations for each packet (bits in FMASK)
	gives avg. degree of fanout going to GR for modelling
EASY	count # packets sent TO each station
	accurate if we count UP packets only
	otherwise, we only get a sample of LR multicasts
HARD	count # packets sent FROM each station
	accurate if we count UP packets only
	otherwise, we only get a sample of LR multicasts
	hard to know who the sender is (must keep state)
EASY	filter multicast packets based on:
	data packets, address/cmd packets
	Nack, REQ/RESP, R, RE, WB, INV, UPGD etc.
HARD	separate multicast packet counts by:
	PhaseID (must keep state)
	sending station (taken from SMA)

Table 2: Local Ring Monitoring (continued).

Difficulty	Description and Commentary
EASY	count # of slots not used due to use_free_slots (from rtor chip)
FIFO depth counting:	
Note: two FIFOs to monitor for each LR, UP FIFO and DN FIFO	
EASY	two ways of counting FIFO depths:
	use E/AE/HF/AF flags as markers (coarse grain counting)
	maintain head/tail pointers, take difference (fine grain counting)
EASY	store PEAK FIFO depth, and PhaseID when it occurs
HARD	count # cycles FIFO stays at each depth (histogram):
	important to compute TIME AVG of depth
	can use lots of counters, but readback mux grows big
HARD	count number of times each FIFO depth is reached (histogram):
	gives POPULATION AVG of FIFO profile
	can use lots of counters, but readback mux grows big
EASY	may wish to ADD the depths of UP and DN FIFO:
	if we don't add, we can't correlate whether the two FIFO depths
	are both high/low at the same time
	gives us idea of delays packets really see through GR
Measure latency through GR:	
EASY	periodically tag a packet going UP:
	up-bound packet doesn't arrive when wanted, must wait
	use period counter to evenly space samples
EASY	time how long it takes to come down
EASY	remember the LONGEST time to understand maximum latency
EASY	count # of packets tagged
EASY	measure cumulative time tagged packets are in GR:
	gives an AVERAGE response time
EASY	build histogram of GR response time in SRAM:
	links the length of up and down FIFOs (time correlation)
	includes effects of destination LR being busy
HARD	use PhaseID to isolate measurements to specific regions of code

Table 3: Global Ring Monitoring.

Difficulty	Description and Commentary
EASY	free-running counter that obeys start/stop commands
ring link utilization:	
EASY	total number of free/not_free slots
EASY	total number of slots needing to be sequenced
EASY	total number of cycles ring is halted
head of FIFO queue waiting for GR (UP path from each LR):	
EASY	total # cycles head is full / empty
EASY	total # cycles head is waiting for a free slot: already know total # of packets from LR, gives avg. # if cycles waiting for free slot
EASY	total # free slots that could be used, but the FIFO couldn't fill head in time (latency increased due to FPD implementation)
EASY	total # slots not used because of use_free_slots policy won't let us use a newly freed slot (latency increased due to fairness policy)
EASY	total # cycles head is waiting due to FLOW CONTROL

6 Final Design

The modular backplane/daughtercard design makes a compact, upgradeable global ring. One daughtercard is used for each local ring connection, and six datapath daughtercards are used as bitslices to switch the global ring traffic. Four local rings can be physically connected to the global ring, but logically they may be partitioned so that zero, two, three, or four of them are global ring participants. The participants are always the lowest numbered local rings. Despite the partitioning, the global ring always consists of four physical ring segments; the extra one or two cycles of latency are undesirable but they too difficult to work around.

6.1 Local Rings

The local ring daughtercards contain a local ring controller, called RINGCON, which is very similar to the RTOB/BTOR (ring-to-bus/bus-to-ring) controller in the NIC cards. In this analogy, the global ring is treated as the ‘bus’ side of the controller. The RINGCON logic is modified slightly so that only upstream packets are selected and written to the upstream FIFOs.

Hardwired pins in the LR slots number each local ring. A configuration protocol on the local ring numbers the stations on ring. The RINGCON is always defined to be the sequencing point of each local ring, whether or not this local ring is logically participating in the global ring.

To help with future error checking, the RINGCON always sets a *watchdog bit* for all downstream packets. This bit ensures packets are actually removed from the local ring in the presence of faulty stations which do not properly remove themselves from the filtermask. Any packet that is received by RINGCON with this bit set indicates a faulty or absent station, and the slot should be marked free. Although the current RINGCON design always sets this bit, it does not check its status.[7]

Non-participating local rings are prevented from writing data to the upstream FIFO, otherwise it may fill during operating system configuration probing and the ring would lock.

The local ring daughtercards receive an ECL clock from RJ-45 connectors on the global ring backplane. They convert this to a TTL signal and distribute local copies to on-card registers. They also send two copies off-card to the upstream

[7]Note: the RINGCON design also filters out absent stations from the fmask bits, so watchdog packets shouldn’t ever be created on the ring.

FIFOs, one on the backplane and the twelve on the six datapath daughtercards. This fanout has proven to be problematic and ad-hoc termination schemes haven't solved it reliably.

6.2 Global Ring Segment

Four global ring segments are hard-coded in a single control chip, the *master controller*. The LR slots each numbered in a hardwired configuration, with backplane slot labeled as local ring A defined to be the sequencer. The logic for each ring segment is identical, except for some compile-time options which modify the sequencing and filtering logic required to select and clear filtermask bits.

The control logic for a single global ring segment is shown in Figure 6. The shaded box in this figure also illustrates the simple datapath logic used in the datapath bitslices. All logic for the next ring segment is predecoded in the current ring segment and stored in output registers. The behaviour of some of this logic is altered by the four control signals which are indicated with boxed outlines in the figure.

The master controller uses four external upstream FIFOs to hold the routing information, consisting of eight filtermask and one sequence_request bits.[8] Note that the master controller does not require downstream FIFOs, but it must generate the proper control signals for each datapath bitslice. All data which must be switched and sent downstream, including a copy of the four station filtermask bits, is sent entirely through the datapath daughtercards.

The external FIFO components used for the global ring were the largest (512 entries deep by 18 bits wide) of the fastest (claimed 10 ns cycle times) available that were graciously donated by Cypress Semiconductor. Although we chose the fastest FIFOs available, the clock-to-output time is still on the critical path. To mitigate this delay, their output is latched immediately on-chip using fast input registers. This introduces a dead cycle between issuing a read-enable signal and actually being able to use the data, which is problematic since the controller only knows about one free slot for the next immediate cycle. Hence, an on-chip 2-deep *readahead FIFO* is necessary. This allows data to be streamed continuously from the FIFO without losing data from a read overrun.

The readahead FIFO can be controlled by a finite state machine on the master controller, or in another mode it can be controlled by the external FIFOCTL

[8]The four station filtermask bits are not essential, but are included for monitoring and future use, *e.g.*, if the global ring is used as a local ring replacement.

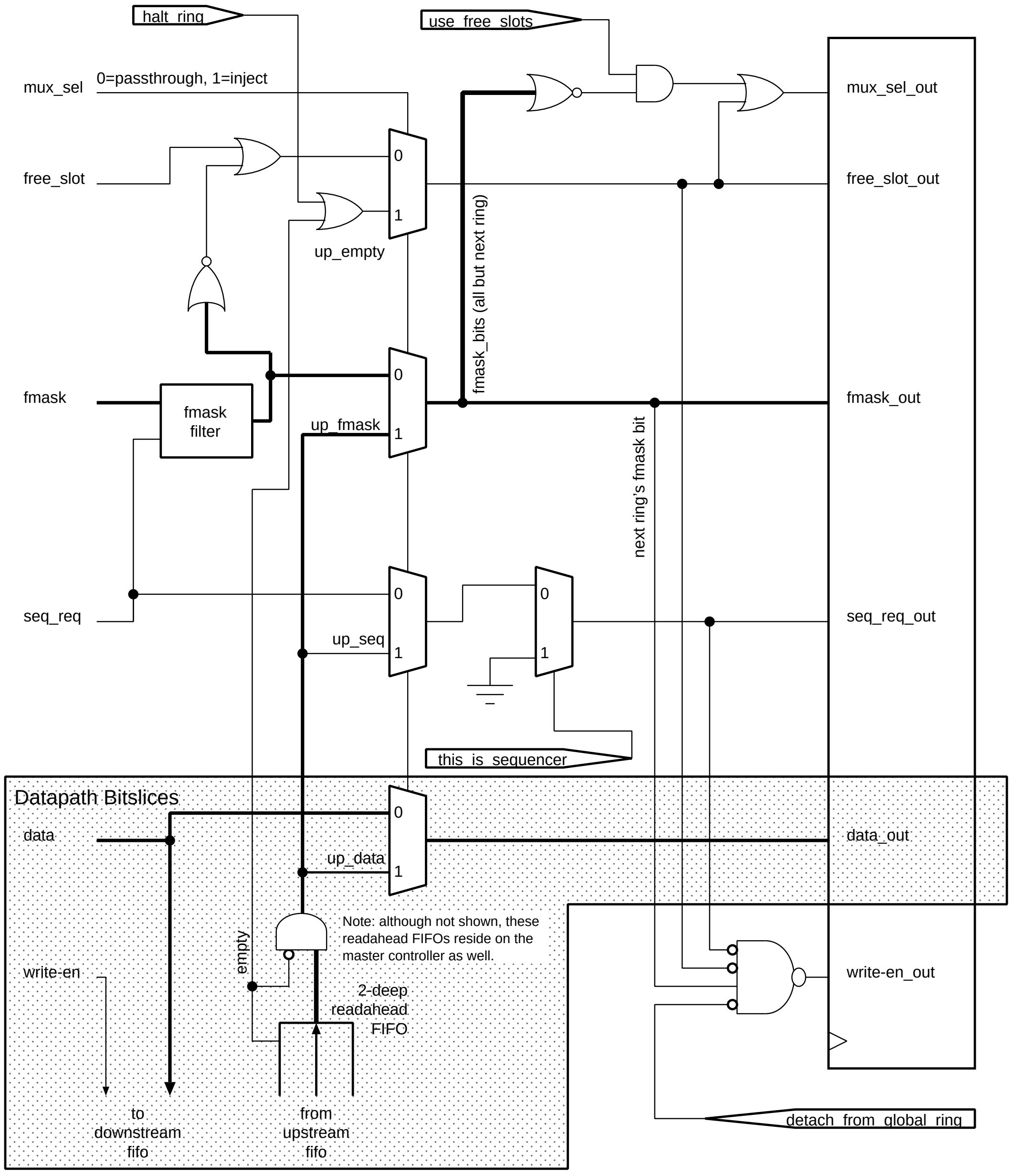

Figure 6: Control and data flow for one segment of the Global Ring.

chips dedicated to each ring segment. A compile-time option ‘ACTIVE’ alternates between these two modes since it is not apparent which way works better for timing. The FIFOCTL chips are essential because the master controller does not have enough logic, pins, or speed to control the FIFOs on all of the datapath daughtercards. As a result, the FIFOCTL chips produce six copies of the upstream read-enable signals, one per datapath daughtercard. Because of this fanout restriction, the logic on the datapath bitslices actually operates one or two clock cycles behind the master controller.

The master controller uses the routing information to determine where to send packets. It precomputes the free-slot and write-enable signals going to the next ring segment. This way, the data can be immediately written to the downstream FIFOs in the next segment without logic or buffering. To limit fanout, three copies of the ring mux control signals and downstream FIFO write-enables are sent to the six datapath daughtercards. A fourth copy of these signals is generated for observation by monitoring.

An external device, FIFOFULL, monitors almost-full flags from the downstream FIFOs to determine when to halt the global ring. A hysteresis counter is used to keep the ring halted for at least eight cycles. This ensures stability of the ring and makes it easier to debug/observe ring halting.

An error bit is set if a packet is ever destined for a local ring that has been logically disconnected. A protection circuit prevents that downstream FIFO from accepting the packet, otherwise it could fill and flow control would halt the global ring. This is particularly important for multicasts which are over-specified, such as when the operating system is probing the system configuration. The master controller actually allows 3 such over-specified packets to be sent, but the fourth one sets the error bit and drives the red ERROR LED on the mainboard.[9]

A global ring clock is generated on-board and distributed as ECL to the datapath daughtercards. An on-board ECL-to-TTL conversion device clocks the master controller and other devices (upstream FIFOs, FIFOCTL, and FIFOFULL designs) on the backplane.

6.3 Datapath Daughtercards

The datapath cards are used exclusively for switching data and make no independent control decisions on their own. Mux control signals and downstream FIFO

[9]Note that a watchdog bit was not implemented in the global ring because the logic would not fit on the master controller device. It was intended that a watchdog error also set this error bit.

write-enables are generated by the master controller. Upstream FIFO read-enables are received from the FIFOCTL device. All of these control signals are issued one cycle or two cycles after the actual switching decision is made in the master controller, depending on the 'ACTIVE' mode. This lag isn't a problem because all downstream data is already in the datapath cards; there is no need to remain synchronized with data in the master controller.

The daughtercards receive their own copy of a high-speed ECL clock. A conversion chip distributes TTL versions to all eight FIFOs and the datapath switch.

6.4 Front Panel and Reset Features

A front panel connector on the backplane was designed to give a user physical buttons to control different system features. Pressing physical buttons would issue commands over a slow (1–4MHz) 8-bit front panel bus, which in turn control the following features of the global ring:

1. set the ring speed,
2. set the logical partitioning of local rings on the global ring,
3. issue system resets, and
4. drive the status LEDs.

The front panel bus operates as follows. Bus arbitration is done in priority order among the following devices: the front panel device and the four local ring monitors (on rings 3 to 0). If no device requests the bus, the reset controller gains control and drives the front panel LEDs with a stored pattern, then bus arbitration repeats. An address strobe indicates one of the actions listed in Table 18 is to be done with the subsequent 8 bits of data.

A reset controller connected to the front panel bus can issue a reset to individual, disconnected local rings or to all global ring participants. This controller also chooses which two different configuration bitstreams are sent to the datapath FIFOs (the top or bottom half of the EPROM data). The choice of which bitstream to send must presently be preprogrammed in the device.

6.5 Critical Path and Latency

The critical path of the global ring design is formed by the clock-to-output of the upstream FIFOs and the setup time of the readahead FIFO in the master controller.

Table 4: Estimate of Global Ring Critical Path Delay.

FIFO T_{co}	Board Delay T_p	Data & Clock Skew T_{skew}	CPLD Setup T_{su}	TOTAL T_{crit}
8.0 ns	1 ns (estimate)	1 ns (estimate)	3.0 ns	13 ns

If the global and local rings are running at the same clock rate, a latency through the global ring of between 12 to 16 cycles from end to end was observed during simulation. The variation depends on the ACTIVE mode and whether data was already being streamed through the readahead FIFOs. If the global ring is running at a different rate, additional latency may be encountered due to FIFO clock mismatch and synchronization.

An approximate breakdown of the global ring latency is as follows. Five or six cycles of latency come from entering the upstream FIFO, activating the readahead FSM, and advancing the data to the head of the global ring queue. One or two additional cycles are lost because the datapath runs slightly behind the controller in time. Two cycles are needed for the data to travel across half the global ring, and another five or six cycles are needed to leave the downstream FIFOs and advance to the head of the local ring queue.

The global ring supports full-bandwidth streaming from the FIFOs, but the local ring will only drain the global ring at half-bandwidth.

7 Correctness and Construction

This section discusses the practical strategies used to design, construct, and test a correctly working system.

7.1 Simulation Strategies

The global ring was probably the most thoroughly simulated module of the NUMAchine system. After all of the components were designed and entered into Cadence, module-level testing was done using the extracted Verilog code. Script files were written to simulate insertion of data packets from the four local rings, and all output activity was logged.

Various types of activity was simulated, including: single point-to-point transactions in an unloaded network; carefully constructed cases involving contention; and random, high volume traffic using a mixture of unicasts, multicasts and sequencing.

The global ring was simulated at different operating frequencies, both with and without the use_free_slots policy. During the high-traffic random simulation, using the free slots required 5–10% fewer clock cycles to switch the same traffic.

Verification scripts were written in awk to ensure that the correct data packets were delivered in the correct order for each destination ring. This was made easier by stuffing the data payload with special tag information to indicate where the packet came from, where it was going to, and its own unique identifying number.

Use of these simulation strategies gave a very high degree of confidence in the logic design prior to manufacturing the backplane.

7.2 Other Practical Considerations

7.2.1 High speed signals

The design of the global ring involved high speed signals which required careful attention during layout. To make these signals easier to find and check, every high speed signal name in the board design is prefixed with the designator "hs_". A node name search in the Allegro layout tool can easily find and highlight all of these wires.

Some high speed control signals travelled over connectors from the backplane to the datapath daughtercards. Fanouts of these signals were limited to 2 by having the master controller and support CPLDs generate multiple copies of each signal.

7.2.2 Clock Distribution

Clocks were distributed as differential ECL until they were as close as possible to the logic devices. At that point, ECL-to-TTL converters generated copies of the TTL clock for delivery to multiple chips.

The clock distribution network could not be properly simulated because of the use of differential signals. Instead, a special testing jig was constructed for simulation to bypass most of the clock distribution network. This introduced an error into the circuits during board layout where only one wire of the ECL pair was actually connected. To eliminate skew concerns, the onboard traces were cut and twisted-pair patchwire was installed to correct this.

7.2.3 Testability

Designing the global ring for testability proved difficult. Creating an artificial input-stimulus circuit was briefly considered, but verifying the generated test vectors was deemed too difficult. Instead, many debugging connectors were placed on the backplane and daughtercards. Some of these pins were hard-coded, but others connect to programmable FPGA/CPLD pins so any signal can be extracted.

The use of a daughtercard design was helpful because it allowed cards to be swapped and rearranged to isolate faulty cards.

8 Hardware Manufacturing and Bugs

Due to their simplicity, the datapath daughtercards were constructed first. During testing it was discovered that the incoming high-speed ECL clock signal was left unterminated, so patchwork was required. As well, the ECL-to-TTL conversion chips were not specified to operate beyond approximately 60 MHz, so they had to be removed and replaced.

The backplane was manufactured next, after which it was discovered that the ECL clock distribution from the backplane to the datapath cards was faulty. The use of the simulation jig resulted in only one of the differential signals to be routed to each daughtercard. Twisted-pair patchwire was soldered directly from the ECL distribution chip to the connector for each daughtercard. To eliminate skew, these wires were cut to the same length. This patch was ugly and, due to the sensitive nature of clocks, may have been a source of future faults. Also, some superfluous passive components were removed from the backplane.

The local ring daughtercards were also constructed, but they did not require any board modifications.

9 Testing

The global ring was first tested with two local rings containing a single station each. Continuous read and write traffic from one processor to the remote memory operated correctly using a variety of clock rates.

Although simple tests worked, ones involving multiple local rings would not work for longer than a minute or two. Occasionally packets (or fragments thereof) were dropped or duplicated, and often the cause could not be determined. Errors introduced from faulty datapath daughtercards sometimes complicated things, but could be solved by swapping them. Also, since cache transfers are broken up into multiple packets, it is very difficult to observe what is happening in the overall system and determine the cause. Finding the erroneous situation on the logic analyzer is even more difficult because the analyzer buffers are not deep enough to correlate the error with the cause.

The type of error could be spotted by introducing predictable data payloads (setting all hex digits to 1, then 2, etc) and printing the failed data. When a particular datapath slice skipped or duplicated a packet, it was visually apparent in the data. Unfortunately, the errors were not routinely occurring in any particular slice at any particular time.

Whether the rings were running at a high speed or a very slow one, the nature of the errors did not change. The problem could have been caused by ringing on the control signals (or data) – eventually waiting out the oscillations by slowing up the clock should have fixed this. The suspected faults were either a logic design error, a significant setup time violation, crosstalk between control signals or the clocks, or clock ringing, where multiple clock edges are sometimes observed when only one was intended.

Extensive simulation work did not indicate any logical errors, and further investigation did not show any possible errors. Likewise, a careful inspection of all setup times along all signal paths did not show any anomalies there either. This left the sources of error as being a crosstalk or clocking problem.

Investigating crosstalk is difficult, and a solution would have required slowing the edge rates of the problematic signals. To solve this, termination on some control signals was introduced, but the results were indeterminate. Also, slowing up the clock should have fixed most types of crosstalk errors, but they did not go away.

Among all error sources, clock ringing was the prime suspect. Scope results showed that some TTL clocks did indeed overshoot. Double-clocking a device such as a FIFO would duplicate data on writes, and drop data on reads. Terminating clock lines differently seemed to alter the behaviour slightly, but did not solve it.

My personal suspicion is that the local ring clocks are the source of the errors. The LR clock is converted to TTL on the LR daughtercard, where a distribution chip sends copies to different devices on the LR card itself. However, a single copy of the TTL clock is sent across the backplane to the six datapath daughtercards (connecting to two FIFOs on each one) and to the upstream FIFOs on the backplane. Although this clock only operates at 20ns, such a high fanout may cause ringing or it may be excessively damped. If the former, termination would be difficult without overdamping. However, if the clock is overdamped, it would always arrive late and create a hold time problem with the data (which has a single fanout). To test this theory, extensive patchwire hacks to the daughtercards and backplane are necessary. To be reliable, the LR daughtercard and the backplane would need to be redesigned.

Manufacturing a new backplane is expensive and there is no guarantee of success. If such an effort is to be done, it would be worth redesigning the clocking and data flow from the local ring to the global ring.

10 Lessons Learned and Conclusions

Designing the NUMAchine global ring was a challenging task because of the desire for a high-bandwidth, low-latency, modular system. The daughtercard approach was good because it satisfied the need to package the ring in a small space to attain higher clock rates.

Although swapping daughtercards was useful for debugging, signals on different datapath cards sometimes required different types of termination to create a clean signal. This might have been avoidable by more careful backplane design, but ultimately we had to lock down which card was in which slot.

It is very important to pay attention to clock distribution during schematic generation and board layout, particularly where high-speed clocks are used. Although care was taken during the design process, time pressures inevitably caused errors to pop up. Proper termination of these signals is essential.

It is not reasonable to expect data and control signals to travel far distances in a single clock cycle. We expected a signal to leave the local ring daughtercard, travel across the backplane, and enter a datapath daughtercard without latching or buffering. This was done for a number of local ring signals, which was not as high a speed as the global ring but nevertheless remained problematic. Registering things on the backplane would have been nice, but it would have increased board area significantly.

Despite efforts to make a global ring which would be modular and upgradeable by manufacturing new daughtercards, it is probably best to start any new design from scratch. Current FPGAs come with a plethora of new features such as high-speed signalling, on-chip memory, high logic capacity, and high pincounts, all of which change the design scenario from just a few years ago.

APPENDIX

A Mainboard Layout

- Table 5. Mainboard jumpers.
- Table 6. Mainboard LEDs.
- Figure 7. Mainboard component floorplan.

Table 5: Mainboard jumpers.

Jumper	Description
JH4	use_free_slots (see also D5 in Table 6, amber LED)
JH5	V_{cc} sense ***DO NOT SHORT!!!!***
JH6	disable clocks (for JTAG programming reliability)
DIPSWITCH	global ring frequency selection, see formula on backplane

Table 6: Mainboard LEDs.

LED	Colour	Description
D1	red	global ring controller error
D2	red	datapath FPGA programming active
D5	amber	use_free_slots (see also jumper JH4 in Table 5)
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

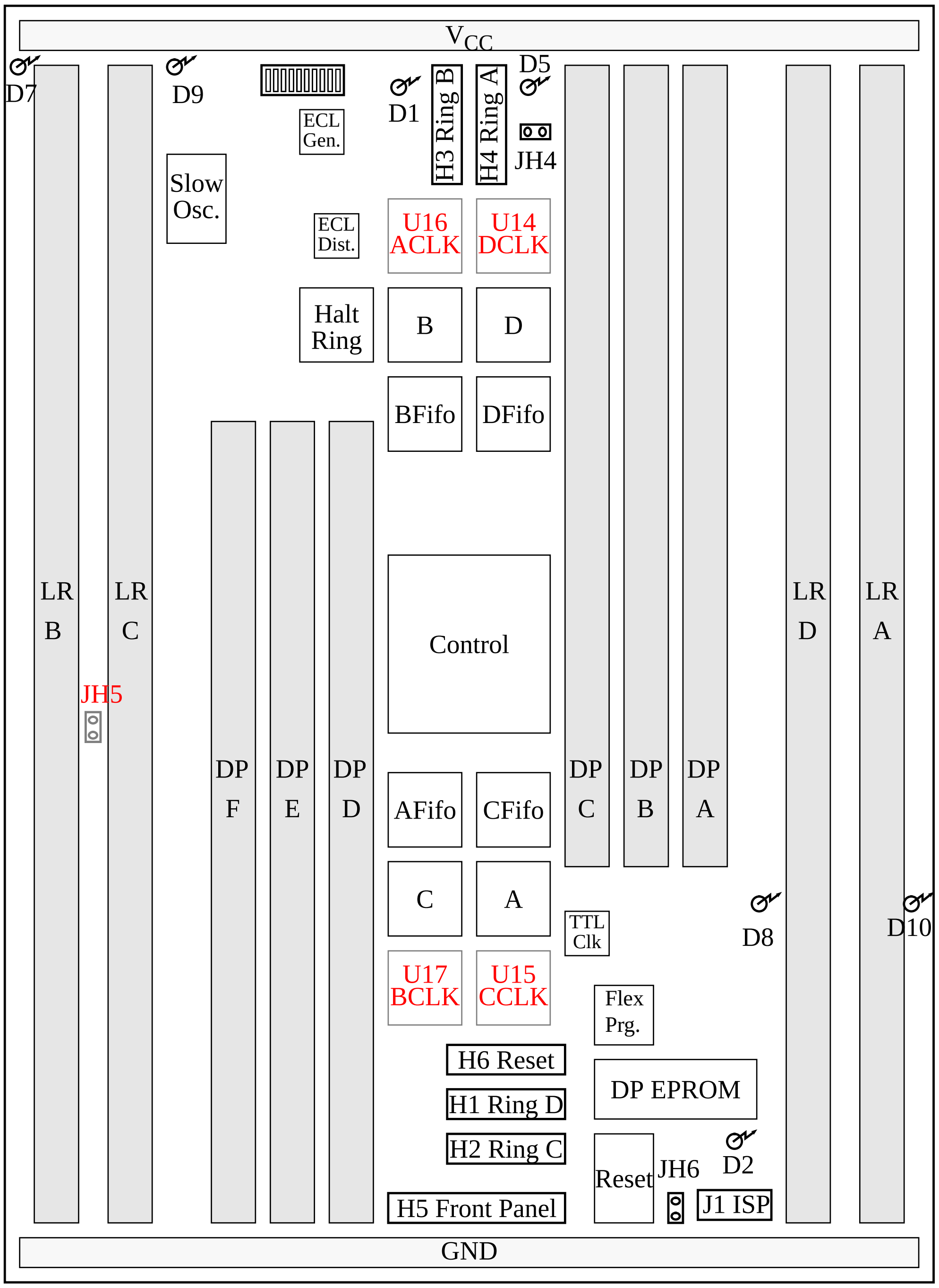

Figure 7: Mainboard component floorplan.

B Post-Manufacturing Modifications

B.1 Mainboard.

- Correctly terminate ECL clock signal (see Figure 8).
- Rewire all ECL clock signal pairs, one copy going to the onboard PECL/TTL converter and six copies going to the datapath daughtercards.
- Replace PECL/TTL converter with a faster device.
- Install 33 or 55 ohm pullups on R54, R55, R56 and R59. These number the local ring slots with a hardwired value.
- Add terminators on the read and write signals going to the datapath FIFOs.
- Install terminator on local ring clock signal of datapath daughtercard Slot F. Note: this modification is done to one specific daughtercard which must be installed in the correct slot.

B.2 Datapath Daughtercard.

- Correctly terminate ECL clock signal (see Figure 8).
- Replace PECL/TTL converter with a faster device.
- Remove R31 since it is an incorrect ECL terminator.
- Note: see last item of the mainboard modifications.

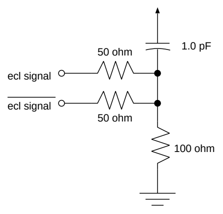

Figure 8: ECL signal termination.

C Connector Pinouts

Table 7: 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 8: 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	$\overline{\text{hs\_dn\_we\_d}}$
8..0	hs_debug_d<8..0>
clk	unconnected

Table 9: 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	$\overline{\text{hs\_dn\_we\_c}}$
8..0	hs_debug_c<8..0>
clk	unconnected

Table 10: 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	$\overline{\text{hs\_dn\_we\_b}}$
8..0	hs_debug_b<8..0>
clk	unconnected

Table 11: Mainboard debug connector H4, *ring A*.

Signal Position	Description
15	use_free_slots
14	hs_halt_ring_l
13	hsmon_sending_lrtogr_a
12..11	hs_a_muxsel<1..0>
10	hsmon_freeslot_a
9	$\overline{\text{hs\_dn\_we\_a}}$
8..0	hs_debug_a<8..0>
clk	global ring clock

Table 12: 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 13: 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 14: Mainboard clock connectors for local ring (RJ-45).

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

Table 15: Local ring daughtercard debug connector H1.

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

Table 16: Mainboard bitslice assignments for datapath daughtercards. This assignment actually depends on the wiring of the local ring daughtercards.

Datapath Slot	Description
Slot A	FMASK_STN<3..0> (0 is LSB) SMA<10..0> TAG bit (GLOBAL_MON_GTOL or _LTOG) EXTRA<2..1> (unused)
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> (71 is MSB)

Table 17: Datapath daughtercard debug connector H1.

Pin	Description
15..12	almost empty flag D..A
11..8	empty flag D..A
7..0	debug<7..0>
clk	global ring clock

D Address Space and Bitslice Assignment

- Table 18. Front panel bus address space.

Table 18: Front panel bus address space.

Address	**Function**
00xxxxxx	Reset controller address space.
00000000	Issue reset: data[7..4] indicates which LR to reset data[3..0] indicates which LR to exclude from GR. Note: GR may only contain consecutive local rings, so excluding just LR1 forces 4 separate local rings, for example.
00000001	Load GR clock register M[7..0] set to data[7..0]
00000010	Load GR clock register M[8] set to data[0] N[1..0] set to data[2..1] N[3..2] set to ‘01’
others	Undefined.
01xxxxxx	Monitoring address space. Unused.
1xxxxxxx	Front panel address space.
10000000	Update status LED display using data[7..0]. Only bits 3..0 actually affect the LEDs.
others	Undefined.

E AHDL Design Files

All of the AHDL design files are located in /nfs/eecg/numa/4/caranci/iri/altera/*. A description of the individual files is given in Tables 19 through 23.

- Table 19. Description of Altera design files for *primary* backplane circuits.
- Table 20. Description of Altera design files for *auxilliary* backplane circuits.
- Table 21. Description of Altera design files for daughtercard circuits.
- Table 22. Description of Altera design files for monitor circuits.
- Table 23. Description of Altera design files for backplane circuits.

Table 19: Description of Altera design files for primary backplane circuits.

Design File	Description
control/control.tdf	Top-level design containing four ring segments.
control/ringseg.tdf	Logic for one global ring segment.
control/headreg.tdf	A small 2-deep FIFO to hold last data read from external FIFO. This is needed to support streaming every cycle.
control/advfree.tdf	Determines whether a free slot will be generated.
control/maskfilt.tdf	Removes the bits set for this local ring from the filtermask.
fifoctl/fifohead.tdf	Top-level design on backplane containing upstream FIFO managers for one segment on the global ring. Reads data from FIFO, sometimes prefetching to support streaming every cycle. Sends duplicated read signals to the upstream datapath FIFOs 1 cycle later. May send the read signal to the upstream routing FIFO (or the control chip may do this, depending on which is ACTIVE).
fifofull/fifofull.tdf	Top-level design on backplane to monitor all downstream FIFO flags and activate GR flow control when any is almost full.

Table 20: Description of Altera design files for auxiliary backplane circuits.

Design File	Description
gr_panel/gr_panel.tdf	Top-level design containing the front panel interface. Issues resets and changes LRs included in the GR. Drives tri-colour LEDs to show when a RESET (red) is issued and which LR are in the GR (green) and which are not (yellow).
gr_panel/status_leds.tdf	Simple tri-colour LED driver.
gr_panel/tribus.tdf	Simple tristate bus macro.
iri_rst/iri_rst.tdf	Top-level design containing reset controller for the global ring backplane. It reprograms the datapath FPGAs on reset/power-up and controls the front panel bus. It specifies the number of local rings on the global ring, sets the global ring frequency, and sends system reset signals to specific local rings (or all those on the global ring) according to commands issued on the front panel bus.
iri_rst/flex_prg.tdf	This circuit programs the datapath FPGAs.

Table 21: Description of Altera design files for daughtercard circuits.

Design File	**Description**
datapath/datapath.tdf	Top-level design containing the global ring datapath slice.
datapath/headreg.tdf	A small 2-deep FIFO to hold last data read from external FIFO. This is needed to support streaming every cycle.
dpmisc/dpmisc.tdf	Top-level design on datapath cards to monitor FIFO flags and update LEDs.
ringcon/ringcon.tdf	Top-level design for the local ring controller on the local ring daughtercard. The monitoring support has been commented out.
ringcon/expand.tdf	Converts a 2-bit number (e.g., ring number) into the unencoded filtermask bit.
ringcon/contain.tdf	Determines whether one filtermask contains another as a destination.
ringcon/fifoctl.tdf	Downstream FIFO readout FSM. It can read a maximum of one packet every other cycle.

Table 22: Description of Altera design files for monitor circuits.

Design File	**Description**
monitor/sramincr/sramincr.tdf	Top-level design for SRAM data +1 increment logic.
monitor/sramfsm/sramfsm.tdf	Top-level design for SRAM incrementing FSM, skeleton for fsm1032 and fsm1333.
monitor/sramfsm/fsm1032.tdf	Simple FSM for one bank of SRAM (with NoBL support). Some monitored events will be missed.
monitor/sramfsm/fsm1333.tdf	Complex FSM for two banks of SRAM (with NoBL support).
monitor/sramaddr/sramaddr.tdf	Top-level design for Address latches shared by both banks of SRAM. Includes FSM to reprogram the monitor FPGA or initialize SRAM to all same data. Addresses can come one per cycle from the master controller, or can be generated from counters (enabled by the master controller).
monitor/sramaddr/addrcnt.tdf	Simple counter for address latches.
monitor/megaproj/	Top-level design to simulate sramincr, sramfsm, sramaddr. This was not intended to be programmed in a device.
monitor/grmon.tdf	Skeleton for a global ring monitor. Contains pinout but no function.
monitor/lrmon/	See lrmon details in Table 23.

Table 23: Description of Altera design files for local ring monitor circuits.

Design File	**Description**
lrmon/lrmon.tdf	Top-level design for the sample local ring monitor. It keeps track of FIFO depths.
lrmon/cntrs/cntrgrp.tdf	Four 32-bit counters. A novel dataflow scheme eliminates the need for a 4:1 readback mux by constantly advancing the counter data in a ring. To read a specific counter, the user waits for the right moment in time for the counter value to travel past the read port.
lrmon/cntrs/cntr.tdf	32-bit counter composed of four CNTRBYTEs.
lrmon/cntrs/cntrbyte.tdf	8-bit counter with optional write port. This is used as a building block for larger, routable counters. The carry-out is buffered to break the FPGA carry chain and make these more routable when chained.
lrmon/fifocnt/fifocnt.tdf	FIFO counter keeping head/tail pointers as Gray-coded values. Use this design if the fifo-depth must be observed according to a 3rd clock.
lrmon/fifocnt/dnfcnt.tdf	FIFO counter, head pointer is Gray-coded, tail pointer is binary-coded. Use this when the fifo-depth is observed using the FIFO read port clock (e.g.: downstream FIFOs).
lrmon/fifocnt/upfcnt.tdf	FIFO counter, tail pointer is Gray-coded, head pointer is binary-coded. Use this when the fifo-depth is observed using the FIFO write port clock (e.g.: upstream FIFOs).
lrmon/fifocnt/graycnt.tdf	Gray-code counter of programmable length. This is required when FIFO writes (up-count) and reads (down-count) occur at different clock rates, eliminating glitches so counts are be off by at most one. Gray code arithmetic requires two carry chains, so it is better to convert back to binary prior to computing FIFO depths.

F Schematics

The last pages contain the schematic diagrams for the global ring as drawn in Concept. The first drawing shows the overall system, which is composed of three primary parts. These schematics are presented in the following order:

- GRMAIN, the backplane,
- LR, the local ring daughtercard, and
- GRSLICE, the datapath daughtercard.