NUMAchine

NUMAchine Principles of Operation for System Programmers

**** 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 Introduction

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.

The NUMAchine station is shown in Figure 1.2. 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.

Each local memory implements a portion of the global shared memory, and the network interface includes a network cache, to cache data from remote memorys. A hardware cache coherence protocol is implemented in the local memory and network interface. This sequentialy consistant protocol automatically maintains valid copies of data in caches throughout the system in order to provide ease of programming.

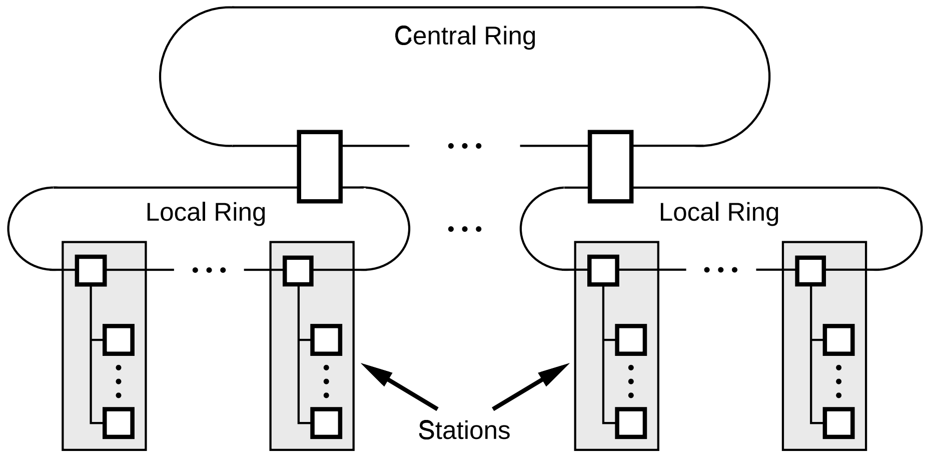

Figure 1.1: The NUMAchine hierarchy

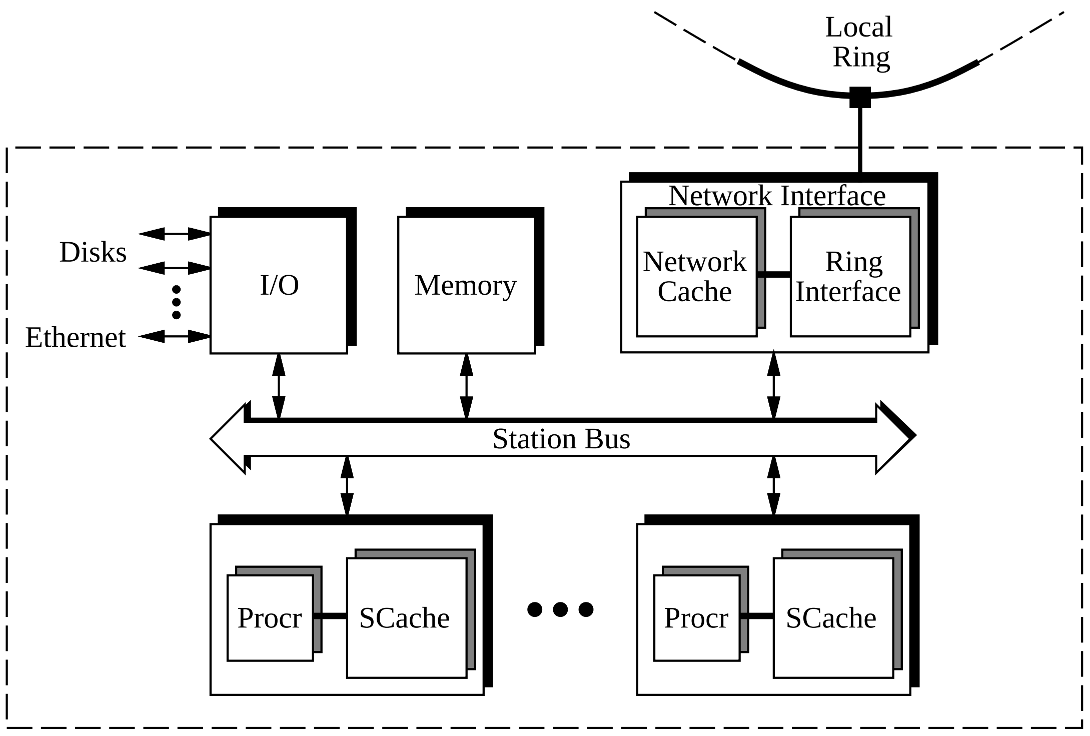

Figure 1.2: Components in a NUMAchine station

1.2 Processor Card

The processor card is illustrated in Figure 1.3. The principal components of the processor card are the MIPS R4400 microprocessor, the secondary cache, and the external agent. The processor card also contains FIFO buffers to and from the NUMAchine station bus, and a number of local resources accessible through a separate local bus interface. These resources consist of: (a) performance monitoring hardware, (b) dedicated registers for interrupts and barriers, (c) local I/O hardware, (d) EPROM memory, and (e) an interface to the *Gizmo* [PVL92], a separate single-board microcomputer.

1.3 Memory Card

The memory card is illustrated in Figure 1.4. The memory card consists of FIFO buffers to and from the NUMAchine station bus, DRAM for data storage, SRAM for the directory maintained by the hardware-implemented cache coherence protocol, and a special functions/monitoring unit.

1.4 Network Interface Card

The network interface card that includes the network cache is illustrated in Figure 1.5. As with all cards within a station, the network interface card also contains FIFO buffers to and from the NUMAchine station

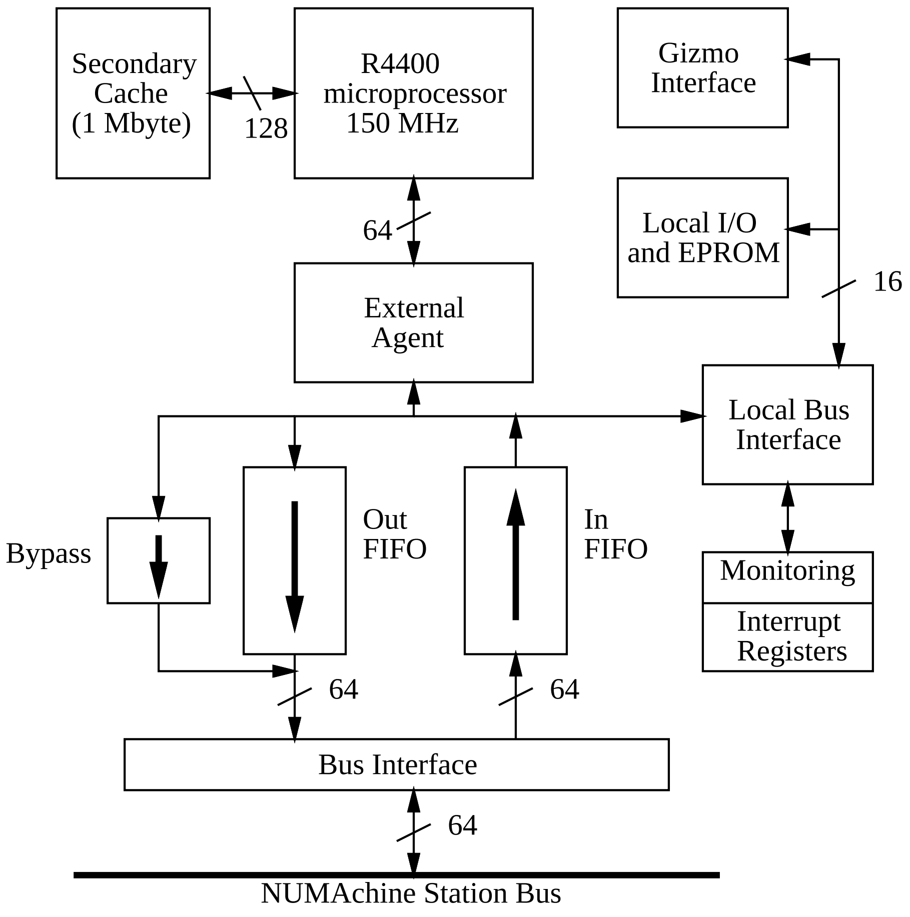

Figure 1.3: NUMAchine processor card

bus. There are two separate FIFO buffers to the station bus in order to satisfy the deadlock avoidance protocol. One FIFO buffer is for *sinkable* requests that cannot generate new requests when they arrive at their destination, and the other FIFO buffer is for *non-sinkable* requests that may generate new requests when they arrive at their destination. To prevent deadlock, these two classes of requests are handled separately to ensure that it is always possible to process all outstanding sinkable requests. Since sinkable requests do not generate any new requests, consuming them frees up any resources (e.g., buffer space) needed by outstanding non-sinkable requests. Note that this deadlock avoidance is performed *automatically* by the hardware for ease of programming.

The remaining components of the network interface card include: (a) SRAM for assembling packets coming from the ring, (b) DRAM for the network cache, and (c) the cache coherence controllers and directory SRAM for the network cache.

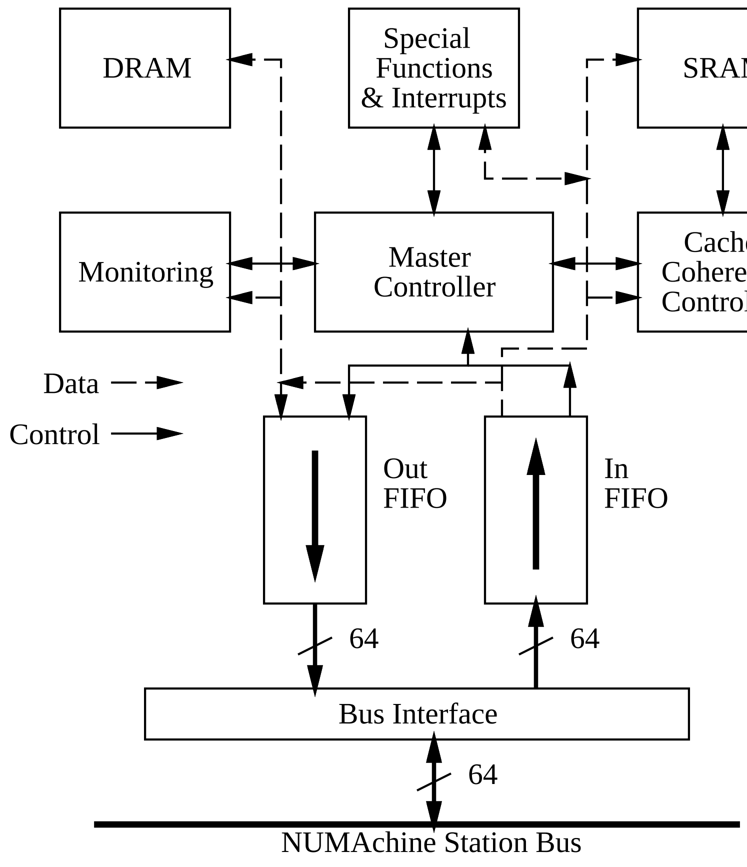

Figure 1.4: NUMAchine memory card

1.5 I/O Card

Each station can support up to two I/O cards. The I/O card includes a R4650 processor, on board memory and a PCI bus including 4 SCSI controllers.

1.6 Software

The software for the NUMAchine multiprocessor consists of application programs and system programs. From the point of view of application programs, NUMAchine presents a flat address space that is accessed in the same way using load/store instructions for both local and remote memory. The hardware-implemented, sequentially consistant, cache coherence protocol automatically maintains valid copies of both local and remote data throughout the system. However, system programs need to distinguish between local and remote accesses for data placement and management, and also must be able to override the cache coherence protocol for initialization or application-specific optimizations. In addition, system programs require control over low-level hardware features such as I/O.

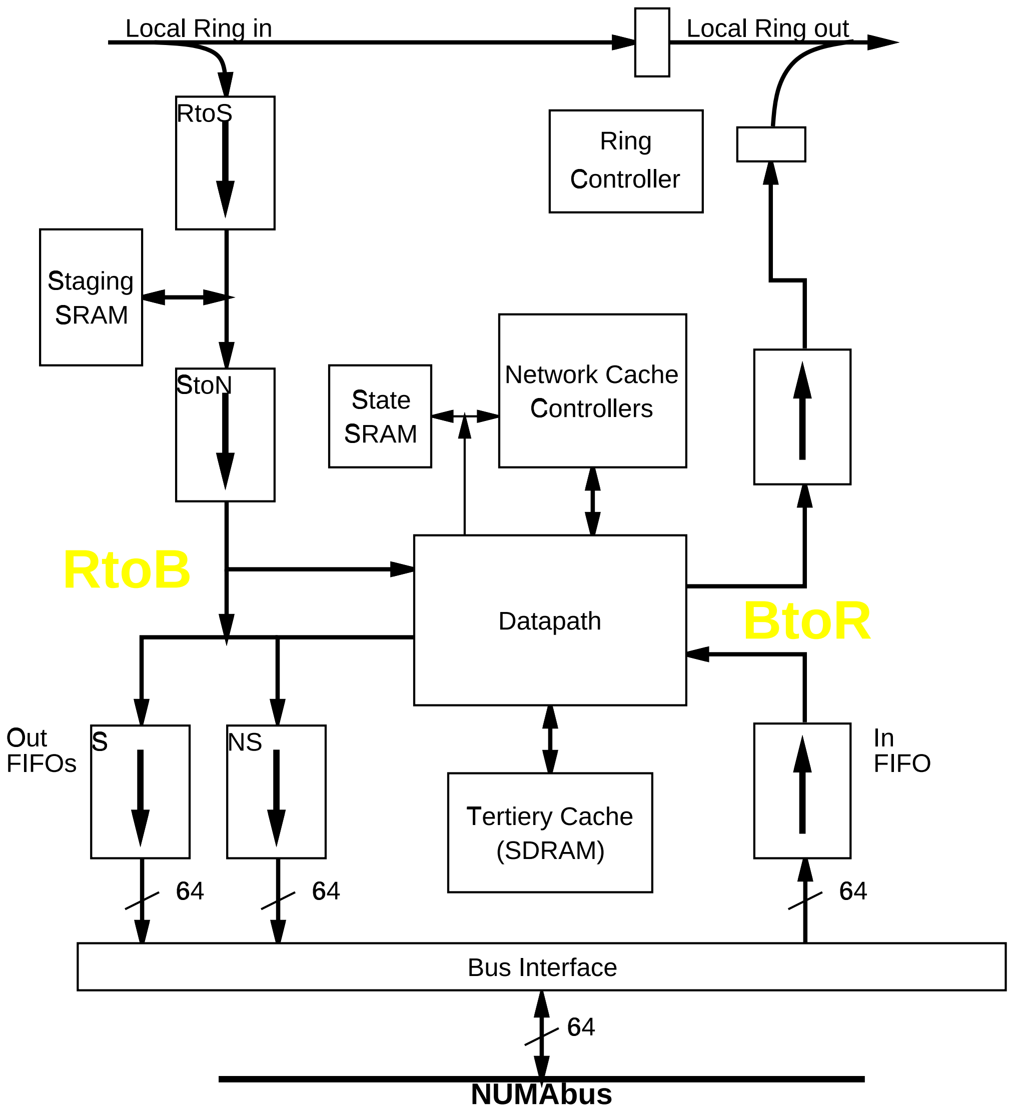

Figure 1.5: NUMAchine network interface card

1.7 Organization of this Document

The purpose of this document is to provide detailed hardware information for system programmers of the NUMAchine multiprocessor, and to describe issues relevant to system software.

In Part II of this document, the NUMAchine address space is defined (Chapter 2 on page 9), features of the processor card are described (Chapter 3 on page 16), and cache coherence hardware is discussed (Chapter 4 on page 25). I/O hardware is discussed (Chapter 5 on page 31), and details for monitoring are given (Chapter 6 on page 36)

In Part III of this document, configuration and startup of NUMAchine is described (Chapter 7 on page 45), testing is discussed (Chapter 8 on page 46), and other details of importance are discussed (Chapter 9 on page 47).

Further details on implementation for various components of the NUMAchine multiprocessor are found in a number of related documents:

- cache coherence protocol specification [Grb96]
- design of station bus and ring hierarchy [Lov96]
- implementation details of processor card, revision 1.0 [ZLC+96]
  (NOTE: this document is obsolete, but provides a useful overview that is the basis for later revisions of the processor card.)
- implementation details for the memory card, revision 1.0 [LGG+96]
  (NOTE: this document is obsolete, but provides a useful overview that is the basis for later revisions of the memory card.)
- implementation details for I/O card [Gus96]
- design of monitoring hardware [Lem96]
- overview of the Tornado operating system [O/S96]

Part II

Hardware Details

Chapter 2

Address Space

2.1 Address Space Definition

The address space for the NUMAchine multiprocessor has multiple views that are, in large part, determined by the MIPS R4400 microprocessor memory management and address translation, whose details are described elsewhere [Hei94]. In NUMAchine, the R4400 operates in 64-bit mode. The different views of the address space are detailed below.

NUMAchine application developer address space. a virtual global address space of 1 Tbyte (2^{40} bytes) that ranges from `0x00 0000 0000` to `0xFF FFFF FFFF`. This address corresponds to the *R4400 user process space*.

NUMAchine system programmer address space: a virtual global address space of 16 Ebytes (2^{64} bytes) that ranges from `0x0000 0000 0000 0000` to `0xFFFF FFFF FFFF FFFF`.

- This address space corresponds to the *R4400 supervisor and kernel space*.
- Not all of the 16-Ebyte space is usable by system programs; consult [Hei94] for details. In particular, the lowest 1-Tbyte portion of this space overlaps with the R4400 user process space to permit system programs to access the address space of the current user process.

R4400 physical address space: a global address space of 64 Gbytes (2^{36} bytes) that ranges from `0x0 0000 0000` to `0xF FFFF FFFF`. This is the maximum physical memory that can be addressed by the R4400, i.e., only 36 bits of the potential 64-bit address space are used.

NUMAchine physical address space: a global address space of 1 Tbyte (2^{40} bytes) that ranges from `0x00 0000 0000` to `0xFF FFFF FFFF`.

- This space is divided equally among 16 stations, with the high-order bits 39..36 selecting the subrange for each station. Outgoing 36-bit physical addresses generated by the R4400 are transformed by NUMAchine hardware to 40-bit NUMAchine physical addresses using the transformation shown in Figure 2.1 on the following page.
- In the current R4400-based implementation of NUMAchine, bits 31..28 are always zero after this transformation.[1]

[1] Bits 31..28 are zero in the current implementation to support a future implementation using the MIPS R10000 microprocessor [MIP94], that has a 40-bit address range; the transformation from 36-bit addresses to 40-bit addresses is only for the R4400.

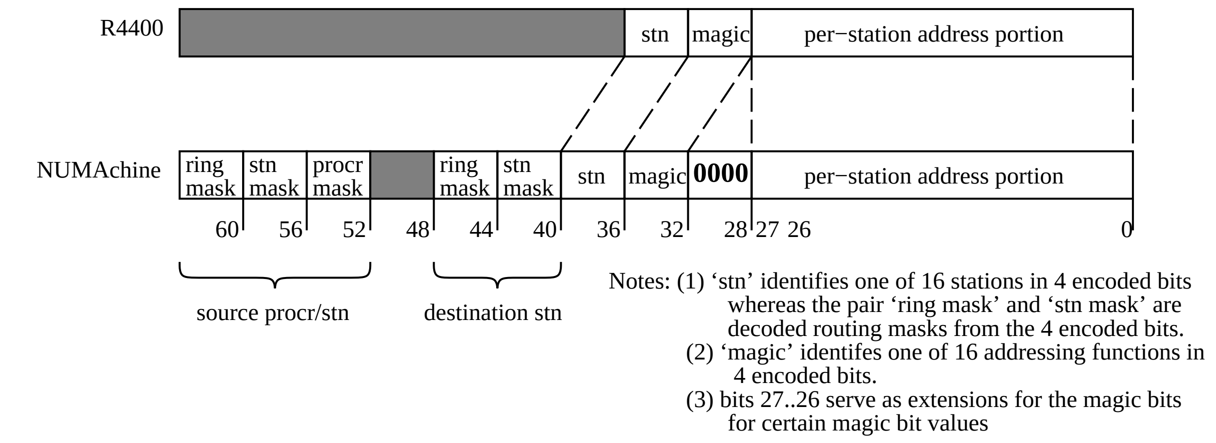
R4400
stn
magic
per−station address portion
NUMAchine
ring mask
stn mask
procr mask
ring mask
stn mask
stn
magic
0000
per−station address portion
60
56
52
48
44
40
36
32
28
27 26
0
source procr/stn
destination stn
Notes: (1) 'stn' identifies one of 16 stations in 4 encoded bits whereas the pair 'ring mask' and 'stn mask' are decoded routing masks from the 4 encoded bits.
(2) 'magic' identifes one of 16 addressing functions in 4 encoded bits.
(3) bits 27..26 serve as extensions for the magic bits for certain magic bit values

Figure 2.1: Assignment of address bits

- Since only bits 39..0 are used for the NUMAchine physical address, the remaining bits (63..40) in the 64-bit datapath are available for other purposes. Bits 63..40 are therefore used to hold routing information. Incoming 40-bit NUMAchine addresses are mapped back into 36-bit R4400 addresses by simply reversing the transformation. Since the R4400 ignores bits 63..36 of incoming addresses, these bits in the NUMAchine address are left unchanged by the reverse transformation. Details of the address transformation and generation of routing information are described in the hardware implementation documentation [ZLC$^+$96].

2.2 Magic Bits

Within the per-station subrange of the NUMAchine address (i.e., bits 35..0), bits 35..32 are called the *magic* bits. Their purpose is to specify the interpretation of the subrange of the address space given by the remaining bits (27..0). The per-station subrange of the address space identified by bits 35..32 and 27..0 is divided between physical DRAM memory for instructions and data, memory-mapped special functions, and special-purpose registers. The interpretation of the magic bits is given briefly in Table 2.1 on the next page, along with a reference in this document where more details are available.

Note that due to the address transformation shown in Figure 2.1, the magic bits in *R4400 addresses* are in bits 31..28. When system software constructs addresses, either for normal memory accesses or for special functions, the magic bits must be correctly specified in bits 31..28 of 36-bit R4400 addresses so that after transformation into 40-bit NUMAchine addresses, the magic bits are correctly shifted to bits 35..32.

Unless otherwise noted, all addresses in the remainder of this document are 40-bit NUMAchine addresses. The corresponding 36-bit R4400 address is obtained by performing the reverse of the transformation in Figure 2.1.

Table 2.1: Interpretation of magic bits

Magic Bits 35..32	Macro	Brief Description	Detailed Reference
0x0	MGC_LOCAL_MEM	Alias for *local* memory card DRAM accesses (*used only during exception processing*)	Section 4.2
0x1	MGC_PROCR	Access local resources on processor card (Gizmo board, reset controller, EPROM, etc.)	Chapter 3
0x2*	MGC_NC_WB	Network cache forced DRAM writeback by *address*	Section 4.3
0x3*	MGC_NC_DRAM_LOCK	*Uncached, atomic* network cache DRAM access by *index*	Section 4.3
0x4	MGC_MON	Access special functions and monitoring	Chapter 6
0x5*	MGC_MEM_UPDATE	Initiate *update* from memory card (uncached write)	Section 4.2
0x6*	MGC_NC_KILL	Network cache kill operation of cache line by *address*	Section 4.3
0x7*	MGC_NC_PREFETCH	Network cache prefetch of cache line by *address*	Section 4.3
0x8†	MGC_REM_INT_MON	Remote *writes* to interrupt registers or monitoring hardware on any processor card	Section 3.4
0x9*	MGC_NC_CMD	Network cache operations by *index*: forced DRAM writeback, uncached normal DRAM access, uncached normal SRAM access, uncached atomic SRAM access	Section 4.3
0xa	MGC_MEM_SRAM_LOCK	*Uncached, atomic* memory card SRAM access	Section 4.2
0xb	MGC_MEM_DRAM_LOCK	*Uncached, atomic* memory card DRAM access	Section 4.2
0xc	MGC_MEM_BCAST	Initiate *broadcast* from memory card	Section 4.2
0xd	MGC_MEM_SRAM	*Uncached* memory card SRAM access	Section 4.2
0xe	MGC_IO	I/O operations	Chapter 5
0xf	MGC_NORMAL	Normal memory or network cache DRAM access (cached or uncached) based on station bits in address	Sections 4.2, 4.3

*The target station is determined by the *filter mask register* in the external agent (see Section 3.2 on page 17).
†The target station is determined *either* by the filter mask register *or* the station bits in the address.

2.3 Magic Extension Bits

For some magic bit values, additional information is needed. To convey this additional information, address bits 27..26 are used as *magic extension bits*. The interpretation of the extension bits is specific to each magic bit value that requires them; consult the references given in Table 2.1 on the page before.

2.4 Maximum Physical DRAM Memory

The local DRAM memory in each station is accessed with the magic bits set either to 0x0 or 0xf (see Table 2.1 on the preceding page). Since the per-station address range for physical DRAM memory is limited to 28 bits (when using the R4400), the maximum physical DRAM memory per station is 256 Mbytes (2^{28} bytes). For 16 stations, the total DRAM capacity is therefore 4 Gbytes (2^{32} bytes).[2]

Although the memory card is designed for the full capacity of 256 Mbytes, it has only been tested with 16 Mbyte SIMMs, for a total of 128 Mbytes of memory. Two memory cards can be used on a station bus, however.

2.5 Constructing Addresses in System Software

System software must construct appropriate addresses in order to access memory or perform special functions. To aid in this task, a number of convenience macros are provided for system programmers to set the contents of individual fields of an address (see Appendix A on page 49 for more details on these macros). Addresses are constructed by ORing together the desired field contents. For example, to read the memory card SRAM contents for a cache line whose home memory is in station 0x5, the magic bits must be set to 0xd (see Table 2.1 on the preceding page) and the station bits must be set to 0x5. Assuming for illustrative purposes that the offset for the cache line in the home memory is 0x3a45f00, the following **C** code will then construct and access the appropriate address, placing the result in the variable `SRAM_data`:

```
{
    long SRAM_data;
    long *SRAM_data_addr;

    SRAM_data_addr =
        (long *) (NUMA_STN (0x5) | NUMA_MGC (0xd) | 0x3a45f00);
    SRAM_data = *SRAM_data_addr;
}
```

Although a macro is provided for setting the magic field to an arbitrary value, symbolic macros are also provided as a mnemonic device to avoid having to consult Table 2.1 on the page before. For memory SRAM access, the macro `MGC_MEM_SRAM` generates the desired magic bits. The above code could be modified as follows:

```
    SRAM_data_addr =
        (long *) (NUMA_STN (0x5) | MGC_MEM_SRAM | 0x3a45f00);
```

[2]In an implementation using the MIPS R10000, bits 31..28 can also be used to address DRAM memory, increasing the maximum physical memory per station to 4 Gbytes (2^{32} bytes), and the total for 16 stations to 64 Gbytes (2^{36} bytes).

The macros shift the given values to the appropriate position in the 36-bit R4400 address, i.e., the resulting address in `SRAM_data_addr` will be `0x5 d3a4 5f00`. This address will then be converted automatically to a 40-bit NUMAchine address when it leaves the R4400, and will also have the required routing information embedded in the high-order bits 63..40.

By using these convenience macros, system software need not be concerned with the precise bit positions of the various fields. The macros hide these details to simplify the development of systems software and allow for future portability.

2.6 Interconnect Routing Protocol and the NUMAchine Address Space

The address space of the NUMAchine multiprocessor presents application programmers with a coherent, shared-memory programming model supported entirely in hardware. Although system-level programmers can also take advantage of this hardware-supported model, they may also exercise more direct control over the routing of requests through the interconnection network. In order to exercise more direct control, system-level programmers must have sufficient knowledge of the interconnect protocol that governs the transmission of requests and responses. To this end, relevent aspects of the interconnect protocol are presented in this section for the benefit of system-level programmers. Further details are available in *NUMAchine Hardware Reference and Maintenance Manual* [CGG+97].

2.6.1 Packet Format

The various hardware components of the NUMAchine multiprocessor communicate by exchanging information in units of *packets*. The format of the basic NUMAchine packet is given in Table 2.2. Each packet contains a set of command bits, and 64 address/data bits. Parity/ECC bits are also provided for error detection. There are two types of packets:

- a *header* packet (Address and command),
- and a *data* packet (Data and data ID).

Table 2.2: Basic NUMAchine packet format

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

2.6.2 Packet Sequences

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 2.3 on the next page.

The header packet in each sequence contains the address of the data in question, specifically a 40-bit *NUMAchine* address augmented with routing information, as highlighted by the bold boxes shown in Figure 2.2 on the following page.

Table 2.3: 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.

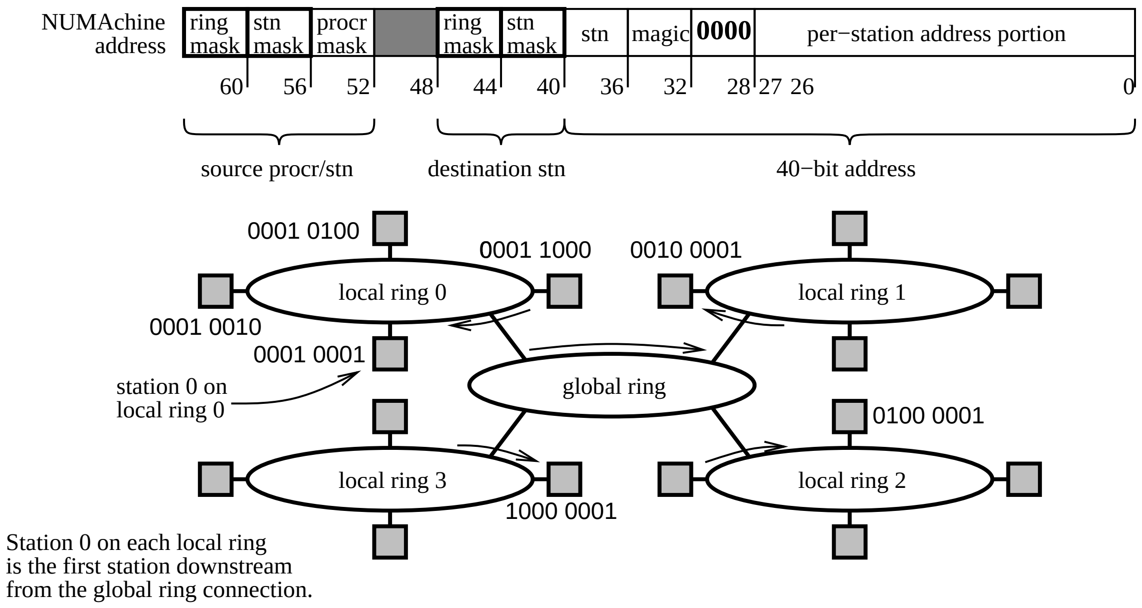

Figure 2.2: Assignment of unique station and ring addresses

2.6.3 Control Over Packet Routing

All component packets in an *outgoing* packet sequence from the R4400 are routed together, according to the information in the header packet.

- The destination station for normal memory requests (e.g., magic=0xf) is determined automatically by the hardware based on the contents of bits 39..36 of a NUMAchine address in the header packet. These bits are determined by the virtual-to-physical memory translation within the R4400, followed by the translation of the 36-bit R4400 physical address into a 40-bit NUMAchine address; the later operation is performed by the external agent hardware associated with each processor.

- The 4-bit station identifier in bits 39..36 is decoded into an 8-bit filter mask that is placed in routing bits 47..40 of the NUMAchine address. The source identifier bits 63..52 are also set. Both operations are performed by the external agent hardware associated with each processor.

However, some requests utilize the *filter mask register* to explicitly specify one or more destinations for an outgoing request.

- Each processor's external agent has a separate filter mask register whose contents can be written by system software.
- For certain magic bit combinations in requests issued by the processor, the attached hardware automatically substitutes the contents of the filter mask register into the routing bits 47..40 of a NUMAchine address.
- The substitution may also be enabled continously under software control for all outgoing addresses.
- This override capability involves a two-step process:
  (a) the desired bit values are written to a special mask register in the external agent hardware,
  (b) a special magic bit combination for write requests instructs the external agent to use the contents of the mask register for the destination routing bits.
- Note that the *source* routing information in bits 63..52 cannot be overridden under any circumstances; this information is crucial for proper routing of *responses* to outgoing requests.

Further details regarding the filter mask register are described in Section 3.2 on page 17.

To use the override capability with the filter mask register, it is necessary to understand the hierarchical routing protocol used in NUMAchine.

- The 64-processor NUMAchine multiprocessor prototype consists of a hierarchical arrangement of up to four local rings connected to a global ring, as shown in Figure 2.2 on the preceding page.
- On each local ring, there are up to four stations, each with up to four processors.
- In order to transfer data across the hierarchy of the NUMAchine multiprocessor, every station on every ring needs a distinct address. A station is uniquely identified by the combination of its ring and station number to form a ring/station mask that is used to identify both source and destination in the NUMAchine address, as shown in Figure 2.2 on the page before. A distinct station and ring address is automatically assigned to each station at powerup.

Chapter 3

Processor Card

*** THE MATERIAL IN THIS CHAPTER IS SUBJECT TO CHANGE. ***

The behavior of the hardware has not been finalized.

The address range selected with magic=`0x1` or 'MGC_PROC' is used to access resources which are local to each processor card. The magic extension bits 27..26 are used to select a specific resource and function for read or write operations. Table 3.1 provides a brief description for each possible bit combination and gives references where more details are available.

Table 3.1: Address space for processor card

Magic Bits 35..32	Address 27..26	Op	Brief Description	Detailed Reference
0x1	00	R	Disable incoming FIFO (uses read to give processor acknowledgement)	Section 3.1
		W	Write the filter mask (FM) register and FM enable bit, and enable incoming FIFO	Section 3.1
	01	R	Disable parity checking for incoming data	Section 3.3
		W	Write the filter mask (FM) register and FM enable bit, and enable parity checking for incoming data	Sections 3.2, 3.3
	10	R/W	Access interrupt registers and local monitoring	Sections 3.4, 6.2.1
	11	R/W	Access local bus (includes Gizmo bus)	Section 3.5

3.1 Controlling the Incoming FIFO Buffer

A read with magic=0x1 and bits 27..26=00 disables the incoming FIFO buffer to the processor. This feature is intended to guarantee that no external invalidations or interventions modify the processor cache contents while system software performs a special function in the cache. The response to this read operation does not contain valid data; it is used only as an acknowledgement indicating that the FIFO has been disabled.

A write with magic=0x1 and bits 27..26=00 re-enables the incoming FIFO for normal operation. There is no acknowledgement in this case. The write simultaneously modifies the FM register (see Section 3.2).

Note: Issuing a read request that misses in the cache before the FIFO is re-enabled results in *undefined* behavior and should be avoided. Disabling of the FIFO is intended only for short intervals where it is necessary to guarantee that the cache contents are not affected by external requests.

3.2 Filter Mask Register

In normal operation, the destination of outgoing requests from the processor is determined automatically by hardware from the station bits in the address, i.e., the routing information in bits 47..40 of the NUMAchine address is derived from bits 39..35 (see Figure 2.1 on page 10). The filter mask (FM) register associated with each processor permits system software to override the destination of individual outgoing requests (this is done automatically for certain magic bit values), or for all outgoing requests (this must be enabled/disabled explicitly). The contents of the FM register are copied into bits 47..40 in outgoing addresses when this override feature is enabled either automatically or explicitly.

The contents of the FM register are modified by an uncached write with magic=0x1, and bits 27..26=00 or 01. Bits 47..40 in the data for the write must contain the new filter mask. To enable the substitution of the FM register contents into *all* subsequent outgoing writes (for normal memory accesses, i.e., magic=0xf), the FM enable bit (address bit 25) must be set when writing the FM register. *Note that it is not possible to set the enable bit without setting the FM register. The contents of data bits 47..40 are* always *written into the FM register when magic=0x1 and address bits 27..26=00 or 01.* As long as the FM enable bit is set, *all* outgoing addresses (with magic=0xf) have the FM register contents substituted in bits 47..40. The FM enable bit is cleared with an uncached write to the FM register with address bit 25=0. Normally, the FM enable bit would not be set, and the FM register substitution is done automatically for *individual* addresses based on the magic bits.

For example, if the mask 0x11 is to be written to the FM register, and FM register substitution is to be enabled for all outgoing requests, system software must construct a NUMAchine address for an uncached write such that:

- address bits 35..32 = 0x1 (magic bits),
- address bits 27..26 = 00 (or 01),
- address bit 25 = 1.

The data (i.e., the new filter mask) for the write must have bits 47..40=0x11.

NOTE: There is a small problem in that in order to write the FM register while keeping the incoming FIFO in the disabled state, it is necessary to know whether parity checking is currently disabled. This is because address bits 27..26 must be 01 to keep the FIFO disabled (a write with address bits 27..26=00 enables the FIFO). However, a write with bits 27..26=01 enables parity checking in addition to updating the FM register. To disable parity checking again, a read with bits 27..26=01 must immediately follow the write. Normally, parity checking is disabled only when debugging, hence writing the FM register with bits 27..26=01 is safe, since it keeps parity enabled.

3.2.1 Initiating Cache Line Broadcasts from the Processor

The FM register permits the generation of cache line broadcasts directly from the processor (broadcasts through the memory card are described in Section 4.2 on page 26). This special operation should normally performed only by system software using the following procedure:

1. The cache line to be broadcast is loaded into the processor cache.
2. The FM register is written with a filter mask which specifies more than one destination. In the same write operation, the FM enable bit is set (see Section 3.2 on the preceding page).
3. Using the CACHE instruction of the R4400 [Hei94, p. A-44], the cache line in question is forced to be written back. As the cache line leaves the processor and enters the *outgoing* FIFO, the FM register contents are substituted into the outgoing address.
4. The FM enable bit is cleared with a write to the FM register to return to normal operation.

Note: It is possible for the cache line in question to be ejected from the cache before it can be written back explicitly. The CACHE instruction is then applied to whatever cache line resides in the cache location corresponding to the address in question, hence there may be a broadcast of some other cache line.

Note: If an uncached write is performed with magic=0xf, and the FM register is active, the uncached writes will be broadcast to the locations specified by the contents of the FM register.

3.3 Controlling Parity Checking for Incoming Data

A read with magic=0x1 and bits 27..26=01 disables parity checking for incoming data in the FIFO buffers that are external to the processor.[1] The response to the read does not contain valid data; it serves only as an acknowledgement. *This feature is intended only for debugging DRAM and SRAM accesses to the memory card or network cache. By disabling parity checking, no interrupt is generated on parity errors, which permits debugging software to examine data as it arrives.*

A write with magic=0x1 and bits 27..26=01 enables parity checking. There is no acknowledgement in this case. The write simultaneously modifies the FM register (see Section 3.2 on the preceding page).

3.4 Interrupt and Barrier Registers

With magic=0x1 and bits 27..26=10 in an address, a processor may read or write its interrupt registers. With magic=0x8, the registers of other processors may be written (but not read). The following sections provide details on access to the interrupt registers.

3.4.1 Register Addresses and Operation

There are two interrupt registers which are separately addressable. The addresses are given in Table 3.2 on the next page; address bits 21..6 and 2..0 are not used and should be set to zero. The registers may be read destructively or nondestructively (i.e., contents cleared or preserved). Note that remote accesses are limited

[1]Parity checking by the R4400 is determined at boot time from the dedicated serial configuration ROM. The details for R4400 configuration are described elsewhere [Hei94]. Section 3.5.2 on page 21 of this document discusses how system software may control R4400 configuration.

Table 3.2: Interrupt and barrier register addresses

Register	Magic 35..32	Address 27..26	Address 25..22	Address 5..3
Hardware Interrupts	0x1 (local; write)	10	000X	000
	0x1 (local; destruct. read)	10	000X	000
	0x1 (local; nondestruct. read)	10	1110	000
	0x8 (remote; write)	see Table 3.5 on page 21	000X	000
Software Interrupts	0x1 (local; write)	10	000X	001
	0x1 (local; destruct. read)	10	000X	001
	0x1 (local; nondestruct. read)	10	1110	001
	0x8 (remote; write)	see Table 3.5 on page 21	000X	001
Barrier	0x1 (local; write)	10	000X	010
	0x1 (local; destruct. read)	10	000X	010
	0x1 (local; nondestruct. read)	10	1110	010
	0x8 (remote; write)	see Table 3.5 on page 21	000X	010

Note: remote reads are *undefined*.

Table 3.3: External access to interrupt registers

Interrupt Reg.	Bits	Accessible from NUMAchine bus?
Hardware	31..16	no; any data supplied in these bits is ignored; bits 20..16 are dedicated lines from procr. card
	15..0	yes; write only
Software	31..0	yes; write only

to writes only; remote reads are undefined and should be avoided. The *hardware* interrupt register contains 32 bits. The *software* interrupt register has a minimum of 32 bits and may be extended to 64 bits. Accesses to either interrupt register must use a data size of either 32 bits or 64 bits, with proper address alignment to 32-bit or 64-bit word boundaries. For 64-bit accesses, the lower 32 bits contains valid data when accessing 32-bit interrupt registers.

The local processor may read or write all bits of both registers. Table 3.3 indicates which bits are writable by external devices from the NUMAchine bus. When writing to the hardware interrupt register from the NUMAchine bus, data bits 31..16 are ignored.

Barrier registers are 32 bits (extendible to 64 bits). They do not generate interrupts. Hence, they must be polled by the processor to determine if bits are set. The contents of barrier registers are normally set with broadcast uncached writes. The addresses are given in Table 3.2.

Writing *any* value to either the hardware register or software register triggers an interrupt for the local processor. Writing a ‘1’ to a specific bit position is used to indicate the source of the interrupt.

When an interrupt service return is invoked, it should normally read the interrupt registers. Reading an interrupt register performs three functions: (a) the current interrupt is cleared, (b) the contents of the interrupt register are returned as the read response, and (c) the register is cleared to zero after its contents are read. Note that *both* registers must be read to determine the source of the interrupt. The interrupt is cleared when the first register is read.

Table 3.4: Hardware interrupt register bit assignments

Bits	Mask (Hex)	Description
31..26		*reserved*
25	0x2000000	monitoring counter3 overflow
24	0x1000000	monitoring counter2 overflow
23	0x0800000	monitoring counter1 overflow
22	0x0400000	monitoring counter0 overflow
21	0x0200000	monitoring counter SRAM overflow (not yet implemented)
20	0x0100000	UART (with connector across bus) has pending request; this interrupt should be maskable so that only one processor per station is interrupted.
19	0x0080000	Too many NACKs or retries for bus request
18	0x0040000	Bus access timeout
17	0x0020000	Interrupt received either from Gizmo, on-board DUART, or Reset controller (see Section 3.5.2 on the following page). Read reset controller general-purpose register to determine source of interrupt (see Table 3.8 on page 23).
16	0x0010000	"Gizmo dead" interrupt if no read response from Gizmo interface before timeout occurs
15..3	0x000fff8	Software-programmable interrupts for Mem. special functions, Monitoring, and I/O
2	0x0000004	Ring Interface error packet received
1	0x0000002	Network cache coherence error
0	0x0000001	Memory card coherence error

3.4.2 Bit Assignments

The hardware interrupt register is 32 bits. The upper 16 bits are reserved for on-board hardware interrupts and accessible only from the local processor. The lower 16 bits are writable from the NUMAchine bus. The bit assignments are detailed in Table 3.4.

There are no fixed bit assignments for the software interrupt register; it may be used in any manner.

There are no fixed bit assignments for the barrier register; it may be used in any manner.

3.4.3 Remote Writes

The interrupt registers for other processors may be *written* with magic=0x8. Address bits 27..26 select a specific function, as shown in Table 3.5 on the following page. Remote reads are not permitted. Note that writes to monitoring hardware of other processors are performed using the same magic bit value; details are given in Chapter 6 on page 36.

For example, if processor 1 writes to the interrupt registers of processors 2 and 0 in the same station, the required NUMAchine address must consist of:

- bits 39..36 = 0101 ('1' in the bits for processors 2 and 0),
- bits 35..32 = 0x8 (the magic bits set for remote access),

Table 3.5: Remote writes of interrupt registers or monitoring of other processors
**** IS THIS TABLE CORRECT ????? ****
** IS THERE A DIFFERENCE BETWEEN 10 AND 11 FOR REMOTE WRITES? ***

Magic Bits 35..32	Address 27..26	Op	Brief Description
0x8	0X	W	Write to interrupt registers or monitoring of a processor *in the same station*. The target processor is selected by bits 39..36 in the NUMAchine address. (Remote access to monitoring is detailed further in Chapter 6 on page 36).
	1X	W	Write to interrupt registers or monitoring of a processor in another station. The target station is determined by the current FM register contents, and the processor within the station by bits 39..36 in the NUMAchine address. (Remote access to monitoring is detailed further in Chapter 6 on page 36).

- bits 27..26 = 00 (access register in same station).

Note that a processor may write to its own register by selecting itself in bits 39..36.

To write registers of processors in other station(s), the FM register must first be written with the desired routing mask for one or more stations (see Section 3.2 on page 17); the FM enable bit need *not* be set, since the magic bits will automatically select the FM register contents. Normally, when more than one station is selected, *all processors* in each station are selected since this is typically a global interrupt. In this case, bits 39..36 would be 1111. When selecting a specific processor in another station, the routing mask should uniquely specify a single remote station, and only one bit should be set in bits 39..36.

3.5 Local Bus Address Space

With magic=0x1 and bits 27..26=11 in an address, the *local bus* is accessed. A number of devices are connected to the local bus, and bits 23..20 dictate which device is accessed, as detailed in Table 3.6 on the next page. Address bits 25..24 are not used for local bus accesses. The local bus is accessed with *uncached* reads or writes.

The Gizmo interface and the reset controller are discussed in the following sections. Monitoring is discussed in Chapter 6 on page 36, and the local UART is discussed in Chapter 5 on page 31.

3.5.1 Gizmo Interface

3.5.2 Reset Controller

When the reset controller is selected, address bits 4..1 select the desired function, as specified in Table 3.7 on the following page. When accessing the General Purpose Registers (i.e., address bits 4..1=1000 or 1010), Tables 3.8 on page 23 and 3.9 on page 24 provide the interpretation of the data bits for uncached reads/writes. This interface supports *only* 16-bit, 32-bit and 64-bit accesses. *The data is in the UPPER 16 bits of 32-bit words for both reads and writes.*

In Tables 3.8 on page 23 and 3.9 on page 24, setting a bit performs the specified function on writes, and a set bit on reads indicates that the corresponding condition is true. When overriding the processor configuration boot ROM values, Table 3.8 on page 23 indicates the interpretation of bits 7..1. Note in Table 3.8 that

Table 3.6: Local bus address space

Magic bits 35..32	Address 27..26	Address 23..20	Description
0x1	11	0000	Selects Gizmo if present, otherwise local EPROM
		0001	Selects Gizmo if present, otherwise local EPROM
		001X	Selects Gizmo if present, otherwise local EPROM
		0100	Selects reset controller
		0101	Selects FLEX programmer for Altera devices
		011X	Selects monitoring SRAM
		1000	Selects local EPROM
		1001	Selects FLEX programming EPROM
		101X	Selects local UART
		110X	Selects Gizmo, if present, otherwise raise “Gizmo-dead” interrupt with garbage data returned for reads
		111X	*undefined*

Table 3.7: Reset controller address space

Magic bits 35..32	Address 27..26	Address 23..20	Address 4..1	Description
0x1	11	0100	00XX	reserved for boot ROM programming
			0100	optional gas plasma display
			0101	*undefined*†
			011X	*undefined*†
			1000	General Purpose Register #1
			1001	*undefined*†
			1010	General Purpose Register #2
			1011	*undefined*†
			1100	lower 2 digits of 7-segment display
			1101	upper 2 digits of 7-segment display
			111X	*undefined*†

†Writes are ignored; reads return garbage data.
Requred because only 32-bit and 64-bit accesses supported.

writes with bit 0=0 and bit 5=1 duplicate the function of writes with bit 13=1. This feature is provided for backward compatibility only and should not be used for new designs.

The reset controller directly controls one of the hardware interrupts (bit 17 in Table 3.4 on page 20). This interrupt can be set in one of the following ways:

- DUART interrupt (GP register#1, bit 13 is set),
- configuration error (GP register#1, bits 14 and/or 9 set),
- interrupt sent from Gizmo (write GP register#1, bit 13; a read of GP register#1 bits 9, 13, and 14 is

Table 3.8: Reset Controller General Purpose Register #1

Data bits	Description	
	Read	Write
15	procr. scndry cache line size=128 bytes	make an NMI request
14	bus CLine size != procr. CLine size	—
13	DUART interrupt is present	send interrupt from Gizmo
12	control DUART interrupts; 0→disable, 1→enable	
11	control LBI interface access for R4400; 0→disable, 1→enable	
10	procr_use_ECC	system reset request
9	procr_use_ECC != bus_use_ECC	bus reset request
8	soft reset cause (0→NMI,1→warm reset)	processor warm reset request
7..1	R4400 processor configuration boot ROM override values (defaults may be changed by reprogramming hardware)	
	bits 7..5: processor-to-system-interface clock speed divisor; default is 001 which means divide-by-3 (*Note:* a write with bit 0=0 and bit 5=1 performs the same function as bit 13=1, i.e., interrupt from Gizmo. Provided for backward compatibility only.) bit 4: most sig. bit of secondary cache read time bit 3: most sig. bit of secondary cache write time bit 2: ECC/parity select (0→ECC, 1→parity) bit 1: secondary cache line size (0→64 bytes, 1→128 bytes)	
0	control R4400 processor configuration boot ROM override; 0→disable, 1→enable (Normally, this feature should be *disabled*; should only be enabled immediately prior to a reset operation to reconfigure processors)	

zero); this interrupt is also caused by the terminal keyboard connected to the Gizmo.

Table 3.9: Reset Controller General Purpose Register #2

Data bits	Description	
	Read	Write
15..14	num_processors[1..0] − 1	—
13..12	num_memories[1..0] (one bit is for presence detect., other bit is #mem_cards − 1)	—
11..9	Revision number	—
8	—	—
7	UART Poll Request Out	—
6	I/O presence detect	—
5	Bus_CL_size	—
4	Bus_use_ECC	—
3	external clk_sel (1→75 MHz, 0→50 MHz); internal clk is double	—
2	poll_req	—
1	UART Flow Control Out (R/W); default=0	
0	UART Flow Control In	—

Chapter 4

Cache Coherence Hardware

The R4400 processor used in NUMAchine permits both cached and uncached data accesses on a per-memory-page basis as indicated in its TLB. For cached accesses, the R4400 further permits accesses to be coherent or non-coherent. NUMAchine implements a directory-based cache coherence in the hardware external to the R4400 [V+95]. The directory information is maintained for each cache line in high-speed SRAM memory which is separate from the DRAM used for instructions and data. Normally, only the hardware implementing the coherence protocol accesses the directory SRAM. However, the directory SRAM is accessible by system programs for initialization, as well as for implementing application-specific coherence policies. The contents of the directory may be examined or modified with *uncached* reads and writes from the R4400, with magic bits set appropriately to indicate that it is the SRAM, and not the DRAM, which must be accessed.

4.1 Cache Coherence, Cache Consistency, and Uncached Accesses

Cache *coherence* is a property which applies to an individual cache line and means that the cache line is guaranteed *systemwide* to be in exactly one of the following two states:

- *either* there is only exactly one valid copy of the cache line, located in either a processor cache or a network cache (the *exclusive* state),
- *or* there are one or more *identical* valid copies of the cache line, one of which *must* be in its home memory (the *shared* state).

Note that a cache line in the exclusive state is *not* valid in its home memory.

State transitions for a given cache line follow a strict discipline:

- *Shared*$\rightarrow$*exclusive*: this transition is initiated with one processor wishes to modify the contents of a cache line, and requires that *all* other copies, including the one in the home memory, are invalidated prior to performing the modification.
- *Exclusive*$\rightarrow$*shared*: this transition is initiated either when the only valid copy is returned from a processor or network cache to the home memory, or a request to read the cache line by a processor would require that an additional copy be made for another cache. The transition to shared is completed prior to fulfilling the new request.
- *Shared*$\rightarrow$*shared*: in response to a read request, a copy of the cache line is provided from wherever a valid copy exists, i.e., from the home memory, or from a network cache.

- *Exclusive→exclusive*: this transition is initiated either when there is only one valid copy of a cache line in a processor cache, and another processor wishes to modify it, or when there is only one valid copy in a network cache and a processor wishes to modify it. The valid copy is moved to the cache of the processor requesting it, and invalidated at its previous location to satisfy the requirement for exactly one valid copy. This transition is also initiated when the only valid copy is ejected from a processor cache, and is moved to a network cache (if it cannot be moved to the network cache and must be returned to its home memory, then the state becomes shared; see the description in an earlier bullet).

The R4400 and the coherence hardware implemented in NUMAchine together ensure cache coherence systemwide for a given cache line provided that *all* processors using the cache line access it in the *cached, coherent* mode of operation.

Cache *consistency* is a property which applies to multiple cache lines and characterizes the *observed order by different processors* of modifications to those cache lines. The key words in this property are "*observed*" and "*different* processors." For example, if a given cache line is modified four times, but a given processor only reads the cache line once, it observes only one modification. However, what matters is the relationship between this observation and modifications of *other cache* lines. Furthermore, it is the comparison of the observations across different processors which determine the nature of cache consistency.

If it can be shown that all processors observe the same order of cache line modifications from any valid concurrent execution on multiple processors, and that this order could also be achieved by any valid serial execution of the same code on a single processor, then the multiprocessor execution is said to exhibit *sequential consistency*.

The R4400 and the coherence hardware implemented in NUMAchine together ensure sequential cache consistency systemwide for a given multiprocessor execution provided that *all* processors access cache lines in the *cached, coherent* mode of operation.

Uncached accesses bypass caches and directly access memory. The NUMAchine coherence hardware does not perform any state transition when servicing uncached accesses. Hence, the memory contents of a given cache line may be modified, which may then no longer match copies of the cache line throughout the system.

Furthermore, buffering of outgoing read and write requests in the external agent associated with each processor does not guarantee that an uncached write reaches memory before an uncached read for the same address issued later by the same processor. This is due to the fact that the outgoing buffers provide a bypass path for all read requests which can permit an uncached read to reach memory before an earlier uncached write.

The R4400 and the coherence hardware implemented in NUMAchine *do not* provide either cache coherence or sequential consistency if uncached accesses are made to a cache line in memory. In particular, sequential consistency is not ensured even for a single processor.

4.2 Memory Card

The memory card contains much of the cache coherence hardware in addition to providing the DRAM memory for instructions and data. Application programs normally access only the DRAM memory. System programs, however, may also access the SRAM memory used to store state information for the cache coherence

Table 4.1: Magic bits combinations for accessing Memory Card

Magic Bits 35..32	Macro	Description
0x0	MGC_LOCAL_MEM	Alias for *local* memory card DRAM accesses (*used only during exception processing*)
0x4	MGC_MON	Access special functions and monitoring
0x5*	MGC_MEM_UPDATE	Initiate *update* from memory card (uncached write)
0xa	MGC_MEM_SRAM_LOCK	*Uncached, atomic* memory card SRAM access
0xb	MGC_MEM_DRAM_LOCK	*Uncached, atomic* memory card DRAM access
0xc	MGC_MEM_BCAST	Initiate *broadcast* from memory card
0xd	MGC_MEM_SRAM	*Uncached* memory card SRAM access
0xf	MGC_NORMAL	Normal DRAM access (cached or uncached)

*The target station is determined by the *filter mask register* in the external agent (see Section 3.2).

protocol. System programs may also exploit special functions implemented in the memory card to perform specialized tasks such as initialization of the DRAM or SRAM. Table 4.1 provides the magic bit combinations that are used to access various features of the memory card. The remainder of this section describes the details of accessing DRAM, SRAM, and special functions on the memory card.

4.2.1 Normal DRAM Access

Processors may read or write the contents of the DRAM memory by issuing addresses containing the magic bit combination MGC_NORMAL and (see Table 4.1).

When used for *uncached* accesses, MGC_NORMAL allows up to 8 bytes to be read or written at one time. Uncached accesses *do not* alter the cache line state since they do not require enforcement of coherence.

When used for *cached, coherent* accesses, MGC_NORMAL reads or writes an entire cache line of either 64 bytes or 128 bytes. For coherent accesses, the cache line state is affected, and additional action may be taken by the hardware to enforce coherence.

When used for *cached, noncoherent* accesses, MGC_NORMAL also reads or writes an entire cache line of either 64 bytes or 128 bytes. For reads, the requesting station is added to the filter mask for the cache line, *but no coherence-enforcement actions are taken* since the access is noncoherent. The updating of the filter mask is provided merely as a convenience for the benefit of system software, since noncoherent access are most likely to be used for application-specific software cache coherence. System software may access the state of a given cache line to determine which stations have made noncoherent requests for the cache line.

4.2.2 Aliased Local DRAM Accesses

During exception processing in system software, the magic bit combination MGC_LOCAL_MEM is used to ensure that accesses are made to the *local* memory on the same station as the requesting processor. Apart from this guarantee of local memory access, the functionality of MGC_LOCAL_MEM is otherwise identical to MGC_NORMAL. The reason for a dedicated magic bit combination for local memory access is to permit the use of common exception code that is independent of the station on which it is executed.

4.2.3 Atomic DRAM Accesses

Processors may *atomically* access the contents of the DRAM memory by issuing addresses containing the magic bit combination MGC_MEM_DRAM_LOCK and (see Table 4.1). Atomic access to the memory is *only* available through uncached accesses, hence the maximum amount of data that may be accessed atomically is 8 bytes. Atomic access for larger data quantities through processor caches requires the use of critical sections with an exclusive lock protecting the affected data structures.

Use of MGC_MEM_DRAM_LOCK requires adherence to a specific two-stage protocol. The first stage is *read-and-lock* and the second stage is *write-and-unlock*. A processor wishing to make an atomic DRAM access performs the read-and-lock by issuing an uncached read request with the magic bits set to MGC_MEM_DRAM_LOCK. When the request is received at the target memory, the hardware first locks the cache line, then reads and returns the requested data. Once the cache line is locked, subsequent normal read or write requests (cached or uncached) from other sources are rejected with a negative acknowledgement.

When the response data arrives at the processor that issued the read-and-lock request, the locking of the cache line ensures that response data agrees with the data in memory. The requesting processor may then perform the write-and-lock by issuing an uncached write with the magic bits set to MGC_MEM_DRAM_LOCK, accompanied by the data to be written to the memory location. When the write-and-lock request arrives at the target memory, the DRAM contents are written, then the cache line is unlocked to permit normal data accesses.

4.2.4 Update Operations

The R4400 processor supports an update operation for maintaining cache coherence [Hei94]. However, NUMAchine does not support the update operations issued directly by the R4400. Instead, NUMAchine requires that a processor perform an uncached write with magic=0x5 to the home memory of the data for the update. The memory then uses the directory information to identify the stations that may have a copy of the cache line to be updated. The update request is then issued by the memory as a multicast to one or more stations. *NUMAchine further restricts update operations to exactly 64 bits that must be properly aligned.* The per-station bits of the address issued with magic=0x5 identify the cache line to be updated. The individual 64-bit word of the cache line to be updated is determined by the lower address bits.

4.2.5 Cache Line Broadcast Operations

4.2.6 Normal and Atomic Accesses of SRAM Directory Contents

To access the SRAM directory contents maintained in the memory card of each station, the magic bits must be set to MGC_MEM_SRAM_LOCK or MGC_MEM_SRAM. In either case, an *uncached* read or write must be used to access the directory contents. *Note that for uncached accesses to the directory, all addresses MUST BE aligned with cache line boundaries. Undefined behavior results if addresses are not properly aligned.*

MGC_MEM_SRAM_LOCK allows atomic access to the directory; an uncached read *locks* the cache line in addition to returning its directory contents, while an uncache write *unlocks* the cache line in addition to replacing its directory contents. MGC_MEM_SRAM reads or writes the directory contents without locking and unlocking.

The remaining bits of the per-station address are used to determine which cache line SRAM state is accessed. The hardware determines the index for the cache line in the SRAM memory from the address. For a maximum of 256 Mbytes (2^{28} bytes) of DRAM, the index is given by bits 28..6 for 128-byte cache lines, and bits 28..5 for 64-byte cache lines.

The format of the data transferred with the uncached read or write corresponds directly to the SRAM directory contents. The interpretation of the directory contents for the memory card is given in Table 4.2.

Table 4.2: Interpretation of directory contents for memory card (R/W in data portion)

Bits	Mask (Hex)	Description
63..17		Unused
16..9	0x1fe00	Filter mask (bits 16..13 for ring, bits 12..9 for station)
8	0x00100	Reserved
7..4	0x000f0	Processor mask (bit 7 is for processor 3, bit 4 is for processor 0)
3	0x00008	Valid (1) or invalid (0)
2	0x00004	Locked (1) or unlocked (0)
1..0	0x00003	Reserved

Table 4.3: Magic bits combinations for accessing Network Cache

Magic Bits 35..32	Macro	Description
0x2*	MGC_NC_WB	Force DRAM writeback by address from network cache
0x3*	MGC_NC_DRAM_LOCK	Uncached, atomic network cache DRAM access by index
0x6*	MGC_NC_KILL	Network cache kill operation of cache line by index
0x7*	MGC_NC_PREFETCH	Network cache prefetch of cache line by address
0x9*	MGC_NC_CMD	Network cache operations by index: forced DRAM write-back, uncached normal DRAM access, uncached normal SRAM access, uncached atomic SRAM access

*The target station is determined by the *filter mask register* in the external agent (see Section 3.2).

Because bits 0..16 are used, uncached accesses must be for at least 3 bytes, with the starting address aligned to a cache line boundary.

4.2.7 Special Functions

Details for special functions supported by the memory card are provided in Section 6.3.1.

4.3 Network Cache

The network cache maintains local copies of data from remote stations. The cached local copies are accessed transparently; the hardware automatically performs remote accesses only when there is no cached local copy. Application programs normally access only the DRAM in the network cache. System programs, however, may also access the SRAM used to store state information for the cache coherence protocol. System programs may also exploit special functions implemented in the network cache. Table 4.3 provides the magic bit combinations that are used to access the network cache. The remainder of this section describes the details of accessing DRAM, SRAM, and special functions on the memory card.

Table 4.4: Interpretation of directory contents for network cache (R/W in data portion)

Bits	Mask (Hex)	Description
63..40		Unused
39..23	0xffff800000	Tag for cache line (magic bits 35..32 not checked by network cache)
22..16	0x00007f0000	Unused
15..14	0x000000c000	Reserved
13	0x0000002000	Data bit (1=data received for interventions)
12..9	0x0000001e00	Processor counting mask for interventions
8	0x0000000100	Assurance bit (1=processor mask is exact)
7..4	0x00000000f0	Processor mask (bit 7 is for processor 3, bit 4 is for processor 0)
3	0x0000000008	Valid (1) or invalid (0)
2	0x0000000004	Locked (1) or unlocked (0)
1	0x0000000002	Local (1) or global (0)
0	0x0000000001	Not-in bit (1=data for cache line not in station)

INITIAL VALUES: tag is set for a local cache line, assurance bit is set, all other bits are zero. The initial values must be set by system software for correct operation.

4.3.1 Normal and Atomic DRAM Data Accesses

4.3.2 Forced Writebacks

4.3.3 Cache Line Kill Operations

4.3.4 Cache Line Prefetch Operations

4.3.5 Normal and Atomic Access of SRAM Directory Contents

To access the directory contents maintained in the network cache of each station, the magic bits must be set to $0x9$. An *uncached* read or write must be used to access the directory contents. *Note that for uncached accesses to the directory, all addresses MUST BE aligned with cache line boundaries. Undefined behavior results if the address is not properly aligned.*

The remaining bits of the per-station address in the uncached operation are used to determine how the directory is accessed and which cache line is accessed. The address bits 27..26 in the address dictate whether or not the directory is to be accessed atomically (10=atomic, 11=normal). The cache line is selected by *index* in the network cache, i.e., using the low-order bits corresponding to the size of the network cache (excluding the bits specifying the offset within a cache line). With an 8-Mbyte network cache, the index is given by bits 22..6 for 64-byte cache lines, and by bits 22..7 for 128-byte cache lines. Bits 39..23 form the tag.

When the network cache directory is written, the tag portion of the directory contents is located in the data packet in the same position it would appear in the 40-bit NUMAchine address. When reading the directory, the tag portion is returned as part of the response data along with the remainder of the directory contents for the cache line.

The format of the directory data contents which are transferred with the uncached read or write is given in Table 4.4. Since bits 39..0 are used for reads, uncached reads must access at least 5 bytes. Since bits 15..0 are used for writes, uncached writes must access at least 2 bytes. In both cases, the addresses must be aligned with cache line boundaries.

Chapter 5

I/O Hardware

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.

5.1 Address Space

The R4400 processors may invoke I/O operations by issuing addresses with magic=0xe, as shown in Table 5.1. The destination station for an I/O request may be selected by setting the station bits 39..36 appropriately in the NUMAchine address. Because there may be up to two I/O cards in each station, address bit 27 provides the means of selecting the appropriate card.

Table 5.1: NUMAchine address space for I/O operations

Magic Bits 35..32	Address† 27	Op	Brief Description
0xe	0=I/0 card#1 1=I/0 card#2	R/W	Routes commands/data to I/O hardware. (Dest. station based on station bits 39..36)

†Checking of address bit 27 is currently disabled addressing of multiple I/O cards must be done through software

Note: Table 5.1 does not define any specific I/O behavior because it is determined entirely by *software that executes locally on each I/O card.* The external agent hardware for each R4400 recognizes commands with magic=0xe and routes them to the appropriate I/O card. The I/O card hardware accepts and buffers these commands. The software executing on the I/O card is then entirely responsible for all aspects of processing these commands, communicating with I/O devices, formatting appropriate responses for the requesting NUMAchine processors, and sending these responses back to the requesters.

To format appropriate responses, system programmers writing code for the I/O card have access to the *Command* field of header and data packets (see Table 2.2 on page 13). This capability is also used to allow

NUMAchine I/O hardware to directly access the Memory Card in local or remote stations. *This access is provided only for packets generated from the I/O card*; normal system software running on the R4400 processors cannot access the Command field.

Full details regarding the bit assignments and encodings for the *Command* field in header and data packets are available in *NUMAchine Hardware Reference and Maintenance Manual* [CGG+97].

5.2 I/O Card Organization

A block diagram of the I/O card is shown in Figure 5.1 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.*

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 and shouldn't be adjusted without first being tested.

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.*

5.2.1 Local Address Space on I/O Card

The R4650 on each I/O card has its own local address space. *The local address space on each I/O card is completely independent of the NUMAchine address space and completely isolated from it.* The only communication between address spaces is by transferring data through the FIFO buffers.

Table 5.2: Local address space on I/O card

Virtual address	Local Address	Size	Description
0xBFC00000	0x1FC00000	4 MB	Flash EPROM (BOOTCS)
0xBF000000	0x1F000000	12 MB	DUART (CS3)
0xBD000000	0x1D000000	16MB	Monitoring (CS2)
0xBC800008	0x1C800008	8 MB	R/W Data FIFO to bus (force parity CS1)
0xBC800000	0x1C800000	8 MB	R/W Data FIFO to bus (CS1)
0xBC400000	0x1C400000	8 MB	Status register only (CS0)
0xBC000000	0x1C000000	8 MB	CMD FIFO and status register (CS0)
0xB4000000	0x14000000	? MB	GT internal registers
0x14000000	0x14000000	32MB	Direct mapped PCI memory space
0x10000000	0x10000000	32MB	Direct mapped PCI IO space
0x00000000	0x00000000	32 MB	Local DRAM

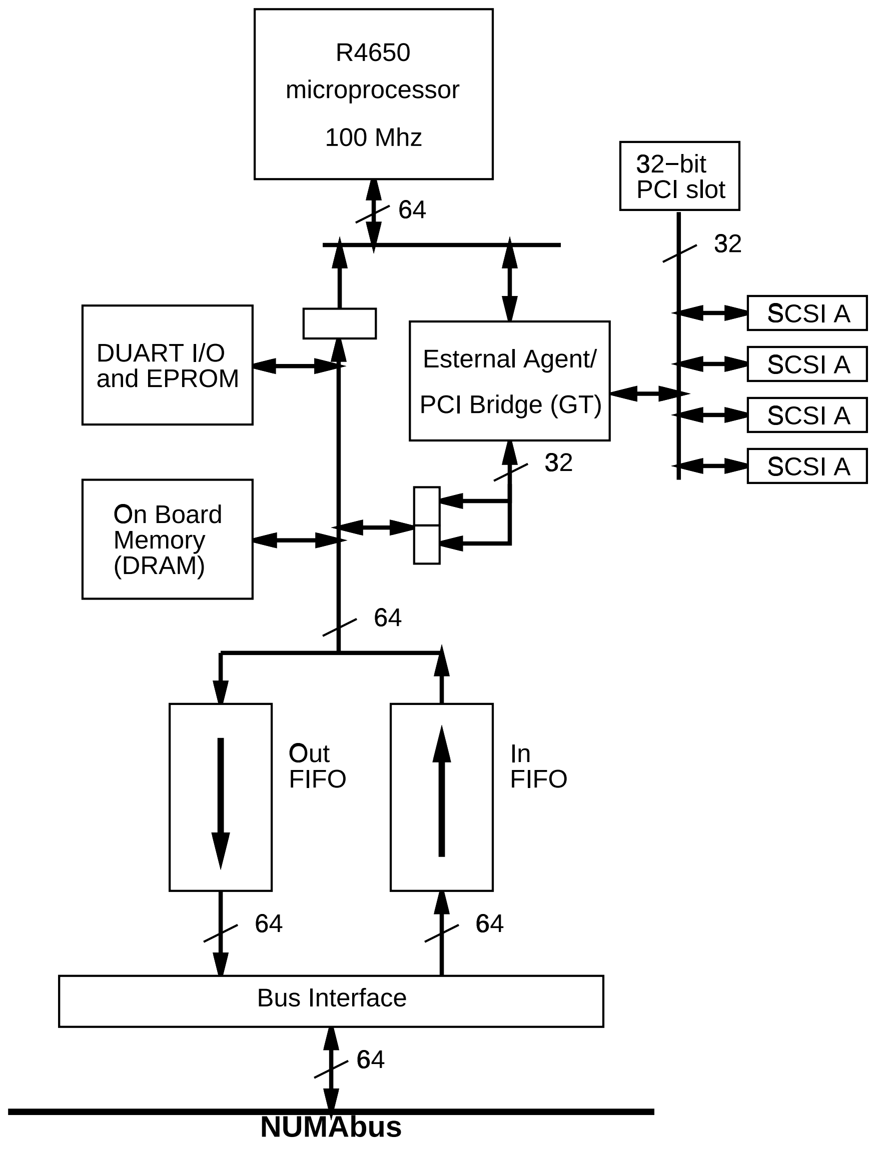

Figure 5.1: NUMAchine I/O card (Rev. 2)

Table 5.2 on the page before describes the local address space for the R4650 on each I/O card. A number of constants have been defined in numaio.h. The address for writing the data FIFO which forces parity should always be used for addresses to ensure that outgoing address packets have correct parity independent of the mode of the system (ECC/parity).

5.2.2 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 Table 5.3

Table 5.3: Contents of the status register on the I/O card

Bits	Active	Description	Bit Mask Macro
35..32		SCSI phase device 0 (GPIO[4..1])†	
37..36		SCSI phase device 1 (GPIO[4..1])†	
43..40		SCSI phase device 2 (GPIO[4..1])†	
47..44		SCSI phase device 3 (GPIO[4..1])†	
51..48	Low	SCSI interrupts	IO_STAT_SCSI_IRQ
52	Low?	Out data FIFO almost full	IO_STAT_DAT_AF
53	Low?	In data FIFO empty	IO_STAT_DAT_EF
54	Low?	Out command FIFO almost full	IO_STAT_CMD_AF
55	Low?	In command FIFO empty	IO_STAT_CMD_EF
56	High	In FIFO read on empty error	IO_STAT_RD_ERR
57	High	Out FIFO write on full error	IO_STAT_WR_ERR
58	High	Poll request	IO_STAT_UART_POLL
59	Hight	ECC/Parity error detected	IO_STAT_ECC_ERR
60	Low	DUART interrupt	IO_STAT_UART_INT
61		DUART op3	
63..62		Spare	

†GPIO[] is a group of programmable I/O pins on the 53C825A SCSI controller. GPIO[0] is connected to a red led on the I/O card, which will turn on if the I/O pin drives a low.

5.2.3 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. Table 5.4 on the next page details the use of the dedicated input pins ip[] and the dedicated output pins op[]. The include file for use with the I/O card DUART is numaioduart.h.

5.2.4 Boot-time Protocol to Down-load Local I/O Software

5.2.5 Details on Bus Interface Control

The bus interface is connected to the dedicated I/O processor through two logical Queues. The *In FIFO* and the *Out FIFO*. In reality there are actually two FIFOs in each direction, one containing address or data (Data FIFO) and the other containing the corresponding command and associated routing bits (CMD FIFO).

Table 5.4: DUART I/O pin connections on the I/O card

Name	Description	Bit Mask Macro
ip0	A_CTS	DUART_IPR_CTSA
ip1	UART_CTS_N	DUART_IPR_CTSN
ip2	Revision number - 1	DUART_IPR_REVNUM
ip3	UART_POLL_REQ in	DUART_IPR_POLLIN
ip4	Only one I/O card	DUART_IPR_ONEIO
ip5	I/O card number	DUART_IPR_CARDNUM
ip6	cache line size (1 = 128 bytes)	DUART_IPR_CLSIZE
op0	A_RTS	
op1	UART_RTS_N	
op2	UART_POLL_REQ out	DUART_POLL_REQ
op3	DUART controlled LED	
op4	A_RX_RDY (NC)	
op5	System Reset Request (Jumper JH1 must be installed)	
op6	A_TX_RDY (NC)	
op7	PCI reset request (Jumper JH2 must be installed)	

A read removes an entry from the corresponding *In FIFO*, and a write places and entry in the corresponding *Out FIFO*. The addressing for these FIFOs is detailed in Table 5.2 on page 32.

A transaction on the NUMAchine bus is broken down into multiple packets. The simplest transaction class, referred to as a short transaction contains only one packet. This packet has two parts, the address, and the corresponding command. This is the format used for read requests. The next class is referred to as medium transactions which comprise 2 packets. The first packet always contains the address and related command, and the second packet contains 64 bits of data and a data identifier. The third class is referred to as long transactions. They contain one address packet (Address + CMD) and a cache line worth of data packets (Data + Data ID) which is either 8 or 16 depending on the cache line size in use in the system.

To simplify processing of transactions by the I/O card all data ID's are stripped off on incoming transactions, and the data ID's are automaticly generated for outgoing transactions. A means of modifying the Data ID exists, but that will be discussed later. As a result the I/O processor needs only to read or write the CMD FIFO once per transaction, independent of the size of the transaction.

It is important that the number of requests that the I/O card has in the system is limited. The hardware places a upper bound of 8 outstanding requests per logical device. The I/O card(s) are classified as one logical device, so to simplify control it is recommended that there be only 4 outstanding requests allowed per I/O card. Three of the upper bits in the command are referred to as the request number. It is recommended that these be set as follows. CMD[12..10] = {(IO_card_number),(next_req_num[1..0])} It is the responsibility of the I/O card to ensure that only one request number is in use at any time for requests. The value of request number doesn't matter for write operations.

How to modify outgoing data IDs **[ADD HERE!]** $\Longleftarrow \star$

5.2.6 Notes on PCI

Chapter 6

Monitoring and Special Functions

MANY OF THE FEATURES DESCRIBED IN THIS CHAPTER HAVE NOT BEEN TESTED.

6.1 Address Space

Table 6.1 provides the magic bit combinations for monitoring and special functions.

Table 6.1: Address space for special functions and monitoring

Magic Bits 35..32	Address 27..26	Op	Brief Description	Detailed Reference
0x1	10	R/W	Processor card monitoring	Sections 6.2.1
0x4	00	R/W	Bus monitoring/trace hardware	Section 6.2.2
	01		Inter-ring interface (IRI) monitoring and special functions *(No monitoring hardware is currently installed on the global ring)*	Section 6.2.5,**??**
	10		Network interface (NI) monitoring and special functions (includes RI error registers)	Section 6.2.4,6.3.2
	11		Memory card monitoring and special functions	Sections 6.2.3,6.3.1
0x8	Remote access to processor monitoring; *write-only* (see Table 3.5,6.2)			Section 6.2.1

6.2 Monitoring

6.2.1 Processor Card Monitoring (Untested)

Table 6.2 on the following page provides the monitoring address space on the local bus of the processor card. Note that monitoring requests may originate from a remote processor, but all requests in this address space ultimately appear on the local bus of a processor card. This address space is primarily used to access configuration registers and counters. For some of the cases in Table 6.2 on the next page, there is a distinction

Table 6.2: Monitoring address space on local bus of processor card (requests may originate remotely)
*** IS THERE A DIFFERENCE BETWEEN 10 AND 11 FOR REMOTE WRITES? ***

Register	Magic 35..32	Address 27..26	Address 25..22	Address 6..3
Counter SRAM	0x1 (local; read/write)	10	1011	(bits 17..3 go to SRAM)
(not impl.)	0x8 (remote; write)	see Table 3.5	1011	(bits 17..3 go to SRAM)
Monitoring	0x1 (local; write)	10	1101	see Table 6.3
Configuration	0x1 (local; read)			unsupported
Registers	0x8 (remote; write)	see Table 3.5	1101	see Table 6.3
Counter[3..0]	0x1 (local; write)	10	1110	mask (6=cntr3,...,3=cntr0)
	0x1 (local; destruct. read)	10	**not impl.**	01XX (XX=counter#)
	0x1 (local; nondestruct. read)	10	1110	01XX (XX=counter#)
	0x8 (remote; write)	see Table 3.5	1110	01XX (XX=counter#)
RESERVED	0x1 (local; read/write)	10	1111	Gizmo bus alias
	0x1 (local; read/write)	11	1111	R4400 access to Gizmo bus
	0x8 (remote; write)	10	1111	Gizmo bus alias
	0x8 (remote; write)	11		undefined

Note: remote reads are *undefined*.

between reads that are destructive and reads that are nondestructively (i.e., register contents are cleared or preserved).

For counters, mask is write-enable; data written to selected counters.

CHECK CORRECTNESS OF TABLE 3.5 on page 21

Table 6.3 on the next page provides the details for the monitoring configuration registers on the processor card. The data bits indicated in the table are for the data portion of the uncached access.

The current processor cards do not support any of the processor monitoring features. The hardware is installed on the circuit board, but the monitoring FPGA is programmed by default to support only the basic interrupt features. The monitoring design is currently being re-worked, and should be at least partially tested by the end of Q1 1998.

More details to be added as features are implemented.

......

6.2.2 Bus Monitoring

No dedicated bus monitoring hardware has been implemented, but there is a piece of the address space reserved for bus monitoring. A small monitoring circuit has been implemented on the I/O card which monitors the bus arbiter when the I/O card is installed in the arbiter slot. This monitoring is accessed via the embedded processor on the I/O card, and can not be directly accessed from the main processors. For more details please refer to the appropriate I/O documentation.

6.2.3 Memory Card Monitoring

Memory card monitoring had been implemented, but not yet documented or tested. The monitoring components such as the monitoring SRAM are functioning correctly. The details on how the monitoring functions will be added as they are tested.

Table 6.3: Monitoring configuration registers on processor card

Addr. 6..3	Mnemonic	Name	Data Bits	Notes
0x0	PR	PhaseID Register	15..0	
0x1	PW	PhaseID Watch	15..0	compared against PR
0x2	CW	Command Watch	12..0	compared against filtered command
0x3	CF	Command Filter	12..0	removes unwanted command bits
0x4	AWhi	Address Watch High	7..0	forms AW bits 39..32
0x5	AWlo	Address Watch Low	31..0	forms AW bits 31..0
0x6	AFhi	Address Filter High	7..0	forms AF bits 39..32
0x7	AFlo	Address Filter Low	31..0	forms AF bits 31..0
0x8	GPCM0	General-Purpose Counter 0 Mode	23	counter enable
			22..16	pipeline status configuration bits PS[6..0]
			15	enable interrupt on overflow
			14..11	event select (one of 16 events)
			10	count cycles high, or low$\rightarrow$high transitions
			9..8	enable masks below...invert mask sense
			7..6	enable PW compare...invert compare sense
			5..4	enable AW compare...invert compare sense
			3..2	enable CW compare...invert compare sense
			1	enable sending mask (SM)
			0	enable receiving mask (RM)
0x9 0xa 0xb	GPCM1 GPCM2 GPCM3	Gen.-Purpose Counter 1,2,3 Modes	23..0	Same format as GPCM0
0xc	SCM	SRAM Counter Mode	22	enable interrupt on overflow
			21..20	counter mode
			19	muxMISS—uses miss type when set
			18	muxRSR—uses RSR when set
			17	muxPRhi—uses upper 5 bits of PR when set
			16	muxPRmid—uses middle 7 bits of PR when set
			15	muxPRlo—uses lower 4 bits of PR when set
			14..11	event select (one of 16 events)
			10..0	same as GPCM0 bits 10..0
0xd	HWIRQ	Hardware IRQ Enable	20..16	enable bits for hardware IRQ
0xe	GPCE	Master Gen.-Purpose Counter Enable	0	master enable for all four general-purpose counters
0xf	SRAMCE	Master SRAM Counter Enable	0	master enable for SRAM counters

Table 6.4: Address space for Memory Card Monitoring and Special Functions

Magic Bits 35..32	Address 27..24	23	Description
0x4	1100	X	Special Function and interrupt controller
0x4	1101	X	Block Label SRAM
0x4	1110	X	Monitoring SRAM
0x4	1111	0	Initialize all monitoring SRAM
0x4	1111	1	Reprogram SF_mon controller from the monitoring SRAM

6.2.4 Network Interface Monitoring

There is no dedicated monitoring hardware on the Network Interface. There are though, two Ring Interface registers. The primary purpose of these registers is to capture the first error packet that comes off the ring.

Table 6.5: Address space for Ring Interface Error registers

Magic Bits 35..32	Address 27..24	Op	Description
0x4	1000	R	Ring interface Status register (contains the Command and SMA of the error, and some status)
0x4	1010	R	Ring Interface error register (contains the Add/Data packet of first local ring error)

A ring error occurs when a ring packet is not correctly removed from the ring. This can occur in two ways. The the packet may address a non existent station, or the routing bits in the packet may have been corrupted. The first case is caused by a software bug/oversight, and the second case is a fatal hardware error and should not occur. These two registers are intended to assist in the debugging of related software as well as communicate some of the system configuration information such as the number of stations connected to this local ring, and if the IRI is connected.

When a ring error occurs the faulty packet is removed from the local ring and stored in the error registers. The command, SMA and add/data are stored in these registers. When an error is captured the network cache sends a hardware interrupt to all local processors with bit 2 set (ie. 0x04 in the hardware interrupt register). All subsequent error packets are discarded until the Ring interface error register is read, at which point the controller is reset and a new error can be captured.

6.2.5 Inter-ring Interface Monitoring

Inter-ring interface monitoring is not yet implemented. Hooks have been added to the Global Ring main board which allow monitoring to be added there in the future. In order to do this the Local Ring board needs to be re-spun with additional hardware added. Until that is done the IRI address space should not be used, since future functionality us undefined.

Table 6.6: Detailed description of Ring Status register

Bits	Description
10..0	SMA of error packet
13..11	Reserved [Currently Undefined]
15..14	Error packet command parity
31..16	Error packet command
33..32	My station number (0-3)
35..33	My ring number (0-3)
37..36	Number of stations on this local ring minus one (0-3)
38	IRI present
39	FIFO from the Ring to the NC is half full
47..40	Reserved [Other info, currently undefined]
63..48	Reserved [Currently undefined]

6.3 Special Functions

6.3.1 Memory Card

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 optional 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 and sets the state to locked local valid)
- block memory move (must do obtain copy first)
- block broadcast with invalidate (The region to be broadcast must be locked local valid)
- block zero region

A preliminary library has been set up which implements all of these functions except block zero region. There are two versions of the library, sf_lib.c for code running on the processors, and sf_iolib.c. The include file numasf.h has a number of constants defined which are useful when using the special functions unit. The library has not been tested on remote stations, and will likely need modifications to make it work correctly.

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, or after the master controller has processed 8 incoming commands.

The special functions unit contains a number of registers as outlined in Table 6.7 on the following page. The address for Memory card special functions is currently AD[31..24] = 0x4c from the processor, or AD[35..24] = 0x40c from the I/O card.

Table 6.7: Registers for special functions and monitoring in the memory card

Addr	Registers			
11..0†	63..48	47..32	31..17	15..0
000	SF status register [R]			monitoring config. [R/W]
100	Special Functions Control Register†† [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 intr. target [R/W]	Intr_vector1 [R/W]
			station | device(s)	(special functions)
700	——		Mon. intr. target [R/W]	Intr_vector2 [R/W]
			station | device(s)	(monitoring)
800	HW Special Function lock register, returns the same data as 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
††Writing this register starts the special function.

The contents of the special function status and monitoring config register are detailed in Table 6.8 on the next page. The Lock owner identifies which hardware device is currently using the special functions unit. Any device wishing to use the special function unit must first lock it before modifying any of its internal registers. The hardware lock is set by performing a read of the lock register at address 0x800. If the value returned in the lock owner fields matches the requesting devices ID it is the lock owner, otherwise another device owns the lock. The unit is automatically unlocked when a special function iteration completes. There is no other way of unlocking the special functions unit.

Once the special function unit is locked the SF registers can be written. The special functions unit only allows writing of the Special Functions Control Register if the hardware lock is set, but it doesn't check if the lock owner corresponds to the device writing the register. Software must ensure that only the device having the lock may write it. The Special Functions Control Register should be written last since it initiates the special function.

The special function unit performs all special functions in phases. The phases are defined as follows:

- Phase 0 - Perform second last special function iteration
- Phase 1 - Perform last special function iteration
- Phase 2 - Send special functions interrupt
- Phase 3 - Send coherence error interrupt or monitoring interrupt

When a special function is initiated the value of start_phase indicates which phase the special unit will start in. Start_phase should be set to either 0 or 1. Unknown operation will occur with other settings. Typically a single part special function operation will start in phase 1 (eg. block write SRAM), and a two part special function operation will start in phase 0 (eg. block move).

For every phase there are multiple iterations. The phase count indicates the number of iterations to be performed. If the number of iterations is set to zero undefined behavior will occur if the corresponding phase is started. The maximum count allowed is 0xFF for this special functions unit.

Table 6.8: Special Functions Status and Monitoring Config Register (0x000)

Bit	Description	Macro shift amount
15..0	Monitoring configuration	
17..16	SIMM Size (00=4Mb, 11=8Mb, 10=16Mb, 01=32Mb)	
18	Cache line size as seen by memory card	
19	Number of memory cards - 1	
20	Interrupt pending (Mon, SF or Error)	
21	Error Interrupt pending	
22	Error occurred	
23	SF unit Enabled	
27..24	SF controller revision	
31..28	Board revision	
35..32	My station address (encoded)	MEMSF_STATION_SHIFT
39..36	Lock owner PID (compressed)	MEMSF_LOCK_SHIFT
47..40	Lock owner station (filter mask)	
49..48	SF state-machine Phase	
50	SF interrupt pending	
63..51	CMD[11..0] present when error detected	

Table 6.9: Special Functions Control Register (0x100)

Bits	Name	Description
7..0	Phase 0 count	The number of iterations to be performed in phase 0.
15..8	Phase 1 count	The number of iterations to be performed in phase 1.
23..16	SF Rselect	The response select used for the special functions
26..24	Start_phase	Phase to start processing the special functions at (either 0 or 1)
43..32	SF_CMD1[11..0]	Special functions command 1
44	PartB1_inc	Specifies whether the Add/Data portion of part B is incremented during phase 0
45	PartB1_zero	Specifies that the Add/Data portion of part B is set to zero
46	PartB1_add	Specifies whether SF_REG1(=0) or SF_REG2(=1) is used
47	PartB1_cmd	Specifies whether SF_CMD1(=0) or SF_CMD2(=1) is used
59..48	SF_CMD2[11..0]	Special functions command 2
60	PartB2_inc	Specifies whether the Add/Data portion of part B is incremented during phase 1
61	PartB2_zero	Specifies that the Add/Data portion of part B is set to zero
62	PartA2_add	Specifies whether SF_REG1(=0) or SF_REG2(=1) is used
63	PartA2_cmd	Specifies whether SF_CMD1(=0) or SF_CMD2(=1) is used

Each iteration is made up of two parts as follows:

- Part A - Deposit the special functions command and corresponding address (must be cache line aligned).
- Part B - Deposit the data portion if one is used, or deposit the target command and address for a move

instruction.

The command and address used for these two parts can come from SF_CMD1 or SF_CMD2, and the address/data portion can come from SF_Reg1 or SF_reg2. There are some restrictions on source depending on the current phase. In phase 0 the following combinations are valid:

- Part A1: SF_CMD1, SF_REG1
- Part B1: [SF_CMD1, SF_CMD2],[SF_REG1,SF_REG2,ZERO]

Once Phase0_count iterations of phase 0 have been completed the SF unit starts phase 1. In phase 1 the valid combinations are:

- Part A2: [SF_CMD1, SF_CMD2],[SF_REG1,SF_REG2]
- Part B2: SF_CMD2, [SF_REG2,ZERO]

The control variables included in table 6.9 on the preceding page called PartB1_zero etc are used to specify which source field is used when there are options. The increment options only effect Part B. The address used in part A is always incremented by a cache line. If a command is used in part A which has data, then the command used in part B must be a data identifier with the EOD bit set.

When the special functions unit completes phase 1 (Phase1_count is reached), it moves on to phase 2. In this phase a hardware interrupt is sent to the device(s) specified. A 16 bit vector, Intr_vector1 is sent to the SF int. target. The top 8 bits of the SF int. target register are a filter-mask indicating which station(s) should receive the interrupt. If the filter-mask doesn't match the local station than the interrupt is sent to the ring and is routed to the station(s) specified. Bits 7..4 of the SF nit. target register specify the selected processor for remote interrupts, and is the bits are all zero the I/O card is selected. if on the other hand the filter-mask matches the station number than the lower 8 bits of the SF int. target register is used as the select bits for the interrupt. Please refer to the System Interconnect chapter of *NUMAchine Hardware Reference and Maintenance Manual* [CGG+97] for the select bit and Rselect bit encodings.

Phase 3 is entered only when a memory coherence error occurs, or a monitoring interrupt is requested. It is not really a part of a special function operation, but it defined as phase 3 for implementation reasons. In phase 3 the special function unit broadcasts an interrupt to all local processors in the case of an error, or sends a monitoring interrupt as specified in the monitoring interrupt register (0x700).

Further details are available in *NUMAchine Hardware Reference and Maintenance Manual* [CGG+97].

6.3.2 Network Interface Special Functions

There are no block special functions implemented on the Network Interface. For information on how to access special Network Cache operations refer to the Cache coherence section.

Part III

System Software Issues

Chapter 7

Configuration and Startup

7.1 Configurable Features

A number of features are configurable via sofware, through overrides to settings in the serial boot PROM.

7.1.1 Clock Speed Selection

- The bus clock frequency is not currently selectable via software.
- The ring clock frequency is not currently selectable via software.
- The processor interface clock divisor is not currently selectable via software. **[Is this true?]** $\Longleftarrow \star$

7.1.2 Cache Line Size Selection

7.2 The Bootstrapping Process

7.2.1 Booting through the Gizmo Interface

7.2.2 Booting from EPROM

Chapter 8

Testing

Chapter 9

Other Important Facts

The R4400 performs uncached read and write operations in a certain way which the hardware must satisfy. This has implications for software in that the data must be properly aligned.

Bibliography

[CGG$^+$97] Steve Caranci, Alex Grbic, Robin Grindley, Mitch Gusat, Orran Krieger, Guy Lemieux, Kelvin Loveless, Naraig Manjikian, and Zeljko Zilic. *NUMAchine Hardware Reference and Maintenance Manual*. 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.

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

[LGG$^+$96] Kelvin Loveless, Mitch Gusat, Alex Grbic, Sinisa Srbjlic, Steve Caranci, Robin Grindley, Guy Lemieux, Zeljko Zilic, and Naraig Manjikian. *Implementation of the Memory Card for the NUMAchine Multiprocessor*. Dept. of Electrical and Computer Engineering, University of Toronto, Ontario, Canada, 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.

[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.

[ZLC$^+$96] Zeljko Zilic, Kelvin Loveless, Steve Caranci, Robin Grindley, Mitch Gusat, Guy Lemieux, and Naraig Manjikian. *Implementation of the Processor Card for the NUMAchine Multiprocessor*. Dept. of Electrical and Computer Engineering, University of Toronto, Ontario, Canada, 1996.

Appendix A

Convenience Macros for Address Construction

There are many more macros available than are currently documented in this file. If this stuff should be included here at all, then the header files should probably be ground through a pretty-printer and included. Otherwise, the info stands a very good chance of going stale very quickly.

The header file `<numasys.h>` defines a number of convenience macros that are useful when constructing addresses in system software. By using these macros, system software need not make explicit references to bit positions. Furthermore, the macros automatically shift the desired values to their correct positions for either the MIPS R4400 or MIPS R10000.

The first two macros set the magic and station fields in the address:

- `NUMA_MGC(x)`: This macro sets the four magic bits to x, where $0 \leq x \leq 15$.
- `NUMA_STN(y)`: This macro sets the four station bits to y, where $0 \leq y \leq 15$.

The following macros provide a symbolic name for each valid magic bit combination:

```
MGC_LOCAL_MEM     /* access local DRAM memory */
MGC_PROCR         /* access procr card resources */
MGC_NC_WB         /* force wback from NC */
MGC_NC_DRAM_LOCK  /* uncached,atomic NC DRAM */
MGC_MON           /* bus monitoring and trace h/w */
MGC_MEM_UPDATE    /* initiate updated from memory */
MGC_NC_KILL       /* kill cache lines in NC */
MGC_NC_PREFETCH   /* prefetch into network cache*/
MGC_REM_INT_MON   /* remote interrupt & monitoring */
MGC_NC_CMD        /* assorted NC accesses */
MGC_MEM_SRAM_LOCK /* uncached, atomic mem SRAM */
MGC_MEM_DRAM_LOCK /* uncached, atomic mem DRAM */
MGC_MEM_BCAST     /* initiate broadcast from mem */
MGC_MEM_SRAM      /* uncached mem SRAM access */
MGC_IO            /* I/O operations */
MGC_NORMAL        /* normal mem or NC DRAM */
```

For those magic bit combinations which require extensions in address bits 27..26, the following macros define the valid extensions which must be ORed with the magic bits:

```
/* for MGC_PROCR */
MEXT_PROCR_DISABLE_FIFO      /* on read */
MEXT_PROCR_FMREG_ENB_FIFO    /* on write */
MEXT_PROCR_DISABLE_PARITY    /* on read */
MEXT_PROCR_FMREG_ENB_PARITY  /* on write */
MEXT_PROCR_LCL_INT_MON       /* read/write */
MEXT_PROCR_GIZMO             /* read/write */

/* for MGC_MON */
MEXT_MON_BUS
MEXT_MON_RI
MEXT_MON_NC
MEXT_MON_MEM

/* for MGC_REM_INT_MON */
MEXT_INT_LCL_STN
MEXT_INT_REM_STN

/* for MGC_NC_CMD */
MEXT_NC_WBACK_INDEX
MEXT_NC_DRAM
MEXT_NC_SRAM
MEXT_NC_SRAM_LOCK
```

Finally, for those magic bit combinations requiring extensions, a single symbol is also defined to combine the magic bits with the extension bits:

```
/* for MGC_PROCR */
MGC_PROCR_DISABLE_FIFO
MGC_PROCR_FMREG_ENB_FIFO
MGC_PROCR_DISABLE_PARITY
MGC_PROCR_FMREG_ENB_PARITY
MGC_PROCR_LCL_INT_MON
MGC_PROCR_GIZMO

/* for MGC_MON */
MGC_MON_BUS
MGC_MON_RI
MGC_MON_NC
MGC_MON_MEM

/* for MGC_REM_INT_MON */
MGC_INT_LCL_STN
MGC_INT_REM_STN

/* for MGC_NC_CMD */
MGC_NC_WBACK_INDEX
MGC_NC_DRAM
```

```
MGC_NC_SRAM
MGC_NC_SRAM_LOCK
```

The following are examples of using these macros.

```
long  ReadMemorySRAM (long offset)

{
    long SRAM_data;
    long *SRAM_data_addr;

    SRAM_data_addr = (long *) (STN (0x5) | MGC_MEM_SRAM | offset);
    SRAM_data = *SRAM_data_addr;

    return SRAM_data;
}

void  DisableProcessorFIFO (void)
{
    long dummy;
    long *address;

    address = (long *) (MGC_PROCR | MEXT_PROCR_DISABLE_FIFO);
    dummy = *address;
}

/*
 * this function is used below
 */
long StationToFMASK (int station_id) /* station_id = {0,...,15} */
{
    long   fmask;

    /* upper 4 bits identify ring */
    fmask = (1 << (station_id / 4)) << 4;
    /* lower 4 bits identify station on ring */
    fmask |= (1 << (station_id % 4));

    /* shift decoded station into proper position for FM register */
    return (fmask << 40);
}

void WriteFMRegAndEnableFIFO (int station_id) /* station_id = {0,...,15} */
{
    long filter_mask = StationToFMASK (0x5);
    long *address;
    address = (long *) (MGC_PROCR_FMREG_ENB_FIFO);
    *address = filter_mask;
}
```

Index