

Chapter 4

Hardware Implementation

In the previous chapter, numerous hardware features that are useful for performance monitoring of a multiprocessor were given. As an example of how the features can be implemented, this chapter will describe the hardware monitoring system being built into the NUMAchine processor card. The implementation uses FPGAs and CPLDs to keep prototyping costs low, but the reprogrammable nature of these devices is also advantageous for changing or improving the way data is collected.

Beginning with an introduction of the NUMAchine architecture, the context of the monitored environment is constructed. Following this, a description of the processor card and its monitoring subsystem are given. Then, the programmable configuration modes of the monitor are described. Finally, the implementation is evaluated based on cost and how well it realizes the recommended features. These results are summarized in tables at the end of the chapter.

4.1 Introduction to NUMAchine

NUMAchine is a distributed shared-memory multiprocessor which supports a sequentially-consistent memory model. The smallest NUMAchine unit is a station composed of four 150 MHz MIPS R4400 processors, memory, and I/O on a split-transaction bus. An example station is illustrated in Figure 4.1. Larger systems are built by adding a network cache and ring interface to each station and connecting a number of stations together in a slotted ring network. The network cache is necessary to provide an intermediate node in the coherence protocol, which will be discussed shortly, but it also provides data caching and reduces network traffic by combining remote requests. Still larger systems can be built by connecting multiple rings together with a larger ring and inter-ring switches; such a system is depicted in Figure 4.2. The NUMAchine prototype being constructed is limited to 64 processors, but the architecture is designed to be scalable to a few hundred.

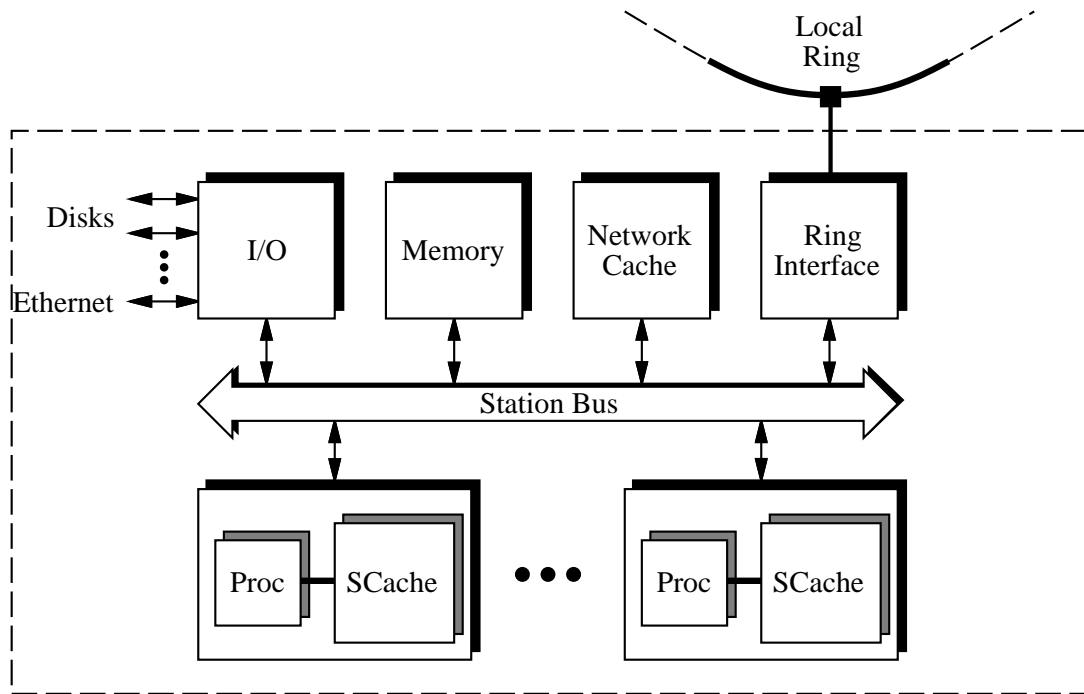


FIGURE 4.1. The NUMachine station.

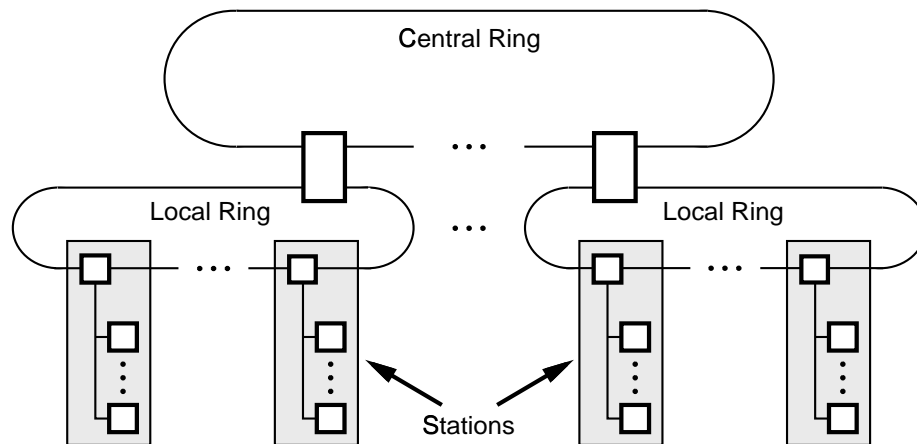


FIGURE 4.2. The NUMachine hierarchy.

NUMachine has a two-level cache coherence protocol: at the *network level*, between network caches and memory, and the *station level*, between processor caches and the network cache. Both levels employ a write-back, invalidate-based strategy, but flexibility in the protocol and hardware has been reserved for simultaneously supporting an update-based scheme. This flexibility allows a program to choose the best strategy for its different

types of data. Additionally, the R4400 processors force primary caches to maintain a strict subset of secondary cache contents, a property called *inclusion*, and follow a strict allocate-on-write policy while handling a write miss. In contrast, the network cache does not guarantee this inclusion (with respect to secondary caches) and it does not allocate a line when writing back a block if it misses. The two-level coherence scheme used by NUMA-machine is structured to match the hierarchy of the interconnect, so good performance is expected.

At the station level, processors keep cache lines in a variant of the well-known MESI (modified, exclusive, shared, or invalid) states. NUMAchine drops the use of the exclusive state so that memory is properly informed of a transition to the dirty (modified) state; normally, a processor simply makes the transition silently. If a processor wishes to obtain data in exclusive state, it is likely to modify the data. For this reason, NUMAchine places exclusively-read data into the dirty state immediately.

At the network level, the network caches and memory maintain state information about a memory block. Specifically, four primary states are defined: *local valid* (LV), *local invalid* (LI), *global valid* (GV), and *global invalid* (GI). These states indicate whether a local copy of the data exists on the station or if it exists remotely, and whether the current copy in the network cache or memory is outdated because a (local or remote) processor has a dirty copy.

Beyond the states already mentioned, the network and processor caches both have a logical *not in* (N) state to indicate that the memory block is not present. This is different from an invalid state because the latter implies a tag match. For further information about the states and requests causing transitions, the reader is referred to Figure 4.3. As well, a more detailed discussion about cache coherence policies can be found in [Vranesic95].

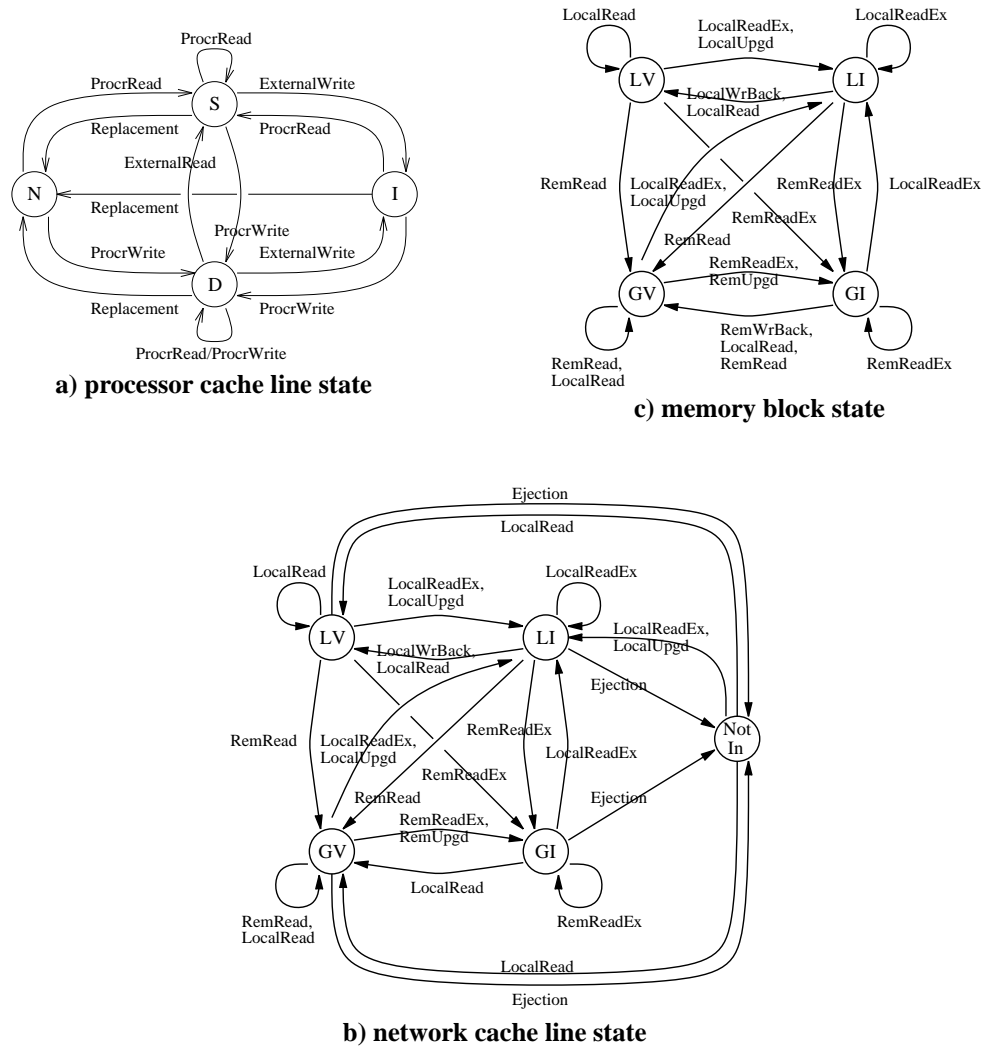


FIGURE 4.3. Memory consistency state transition diagrams.

4.2 Processor Card Organization

Each NUMachine station contains up to four processor cards on the NUMachine bus. The processor card datapath is organized according to Figure 4.4. The *Bus Interface* provides access to the station bus where the main memory, network cache, ring interface, primary I/O (disks, ethernet), and other processors reside. Data is passed into and out of the processor card through *FIFOs* to smooth flow control and decouple the processor card and bus clocks so they can run at independent speeds, if necessary. The *External Agent* performs two functions: 1) it acts as a bridge between the R4400 and the NUMachine bus,

and 2) it controls data flow between the R4400, Monitor and FIFOs. The *Monitor* is situated to observe traffic on the external agent bus as well as accept uncached reads and writes from the processor or uncached writes from the system. The latter allows for simpler initialization, synchronization, and reconfiguration of the monitor by using broadcast writes throughout the system. For convenience, the monitoring interface serves an additional role of providing *Local I/O* for the processor bootstrap code and a UART for debugging. Finally, the processor has a dedicated interface to the *Secondary Cache*, which can be split in half between instructions and data or provide a single unified cache. Typical operation will be done with a unified cache, but the split organization can be used in conjunction with some types of monitoring in which accesses to data and instructions should be isolated; this feature is recommended in [Singhal94].

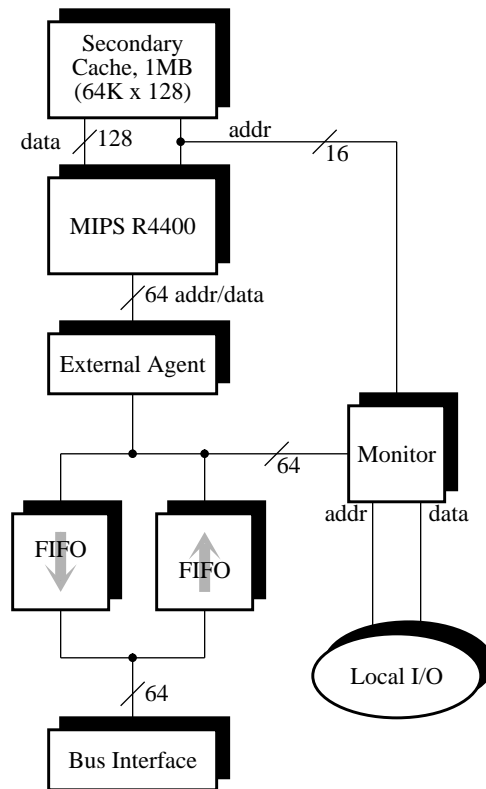


FIGURE 4.4. NUMachine processor card datapath organization.

The NUMachine bus and all of the logic on the processor card are designed to run at 50 MHz. Additionally, a small portion of the monitoring circuits connected directly to the

processor must run from a 75 MHz clock which is locked to one-half the processor speed. Such aggressive system speeds are difficult to achieve in the implementation technology (FPGAs) in a cost-effective manner and, consequently, have influenced many of the design choices.

4.3 Monitor Organization

The processor card performance monitor is positioned so that it can monitor traffic to and from the processor in a non-intrusive fashion. To do this, the monitor is situated on the external agent bus, between the FIFOs and External Agent. When a program chooses to consult with the monitor to read performance data, it uses normal load and store instructions to an uncached, but TLB-mapped, region of memory. By mapping the region, the operating system can protect the monitor from unprivileged user processes. For convenience, both word (32-bit) and doubleword (64-bit) accesses are supported. As previously mentioned, the External Agent also allows writes to the monitor from other processors.

The internal organization of the monitor is illustrated in Figure 4.5, but control signals are omitted from the figure for clarity. As shown, the external agent bus is split into the monitor bus and local I/O bus by the *Local Bus Controller*. From the monitor bus, transactions and configuration information travel to the *Configuration Controller*. Additionally, the monitor bus allows data in the *SRAM* or *Counters & Interrupts* circuits to be read or written. The *Latency Timer*, *Count & Increment*, *Pipeline Status*, *SRAM* and *Counters & Interrupts* circuits all operate with the cooperation of the Configuration Controller to provide the monitoring functions. The monitor bus, however, is under the control of the Local Bus Controller.

4.3.1 Local Bus Controller

To allow nonintrusive monitoring of processor activity, the Local Bus Controller (LBC) passively observes external agent bus transactions. It also gives processors direct access to

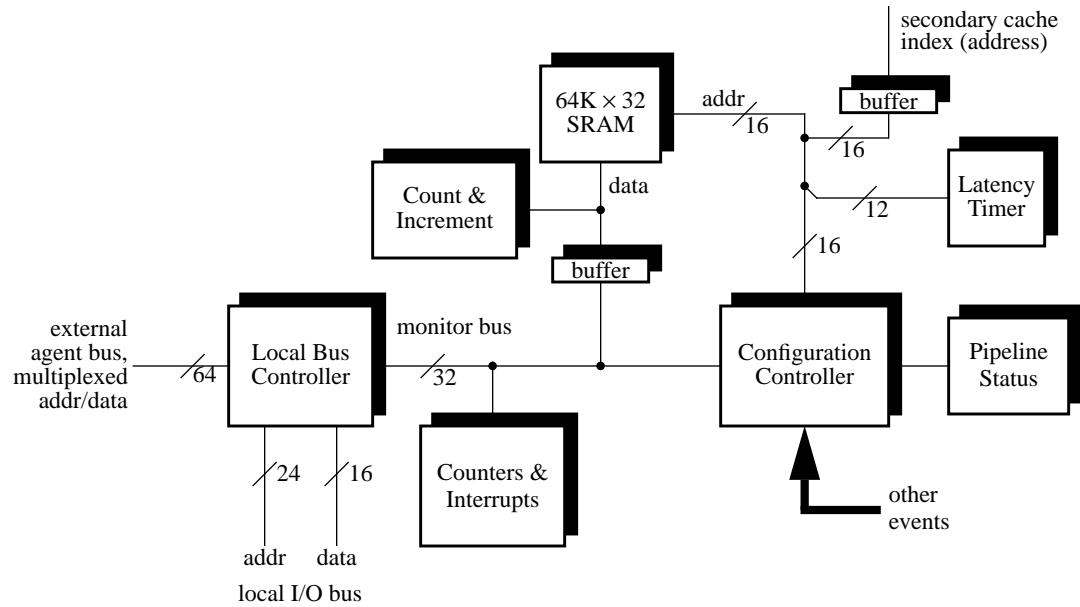


FIGURE 4.5. Processor card performance-monitoring subsystem.

the monitor bus and local I/O bus. The former is used so that a program can quickly obtain performance feedback, while the latter is incidental to NUMAchine test and development.

During the normal LBC operation, transactions on the external agent bus are captured in latches and placed onto the monitor bus. Conveniently, the monitor exploits two facts: 1) only the 64-bit address is needed from the transaction for monitoring, and 2) the external agent cannot process more than one transaction every two cycles. These facts are used to reduce the width of the monitor bus to only 32 bits by holding the upper-half address in a register and sending it on the monitor bus after the lower half is sent. This does not present a problem for doubleword writes to the monitor because the upper address bits can be safely discarded and overwritten as the upper data word is written during the second clock cycle. The upper address bits for doubleword writes to the monitor are not important because it only contains network transaction information, as shown in Table 4.1.

The LBC's second function is to allow reading and writing of the performance data and configuration of the monitor. While the processor or system is reading or writing the monitor, the External Agent is held busy to ensure that no traffic to be monitored will be

Address Bits	Name	Description
63..56	SRC Routing Mask	identifies source processor station of remote transactions
55..52	PID	processor identifier: which processor in station 'SRC' issued transaction
51..48	PhaseID	indicates phase of remote or local processor, but monitor has own copy
47..40	DST Routing Mask	identifies all destinations that were to receive this transaction
39..36	Station Address	indicates packed-encoding of DST when there is only one destination
35..32	Magic Bits	indicates special functions; the monitor has no special functions
31..28	Reserved	future physical address extension
27..0	Physical Address	normal memory address

TABLE 4.1. Breakdown of 64-bit address space in NUMachine.

lost. This should not be considered intrusive because accessing the monitor is inherently an explicit, intrusive decision on the part of the program. Additionally, writes may be used to reconfigure the monitor in two ways: either 1) a *hard configuration*, or 2) a *soft configuration*. The former involves changing the FPGA circuitry in the Configuration Controller (CC) and Counters & Interrupts circuits and the latter simply writes a new value into the configuration registers implemented in the CC.

A third function of the Local Bus Controller is to give the R4400 access to boot ROM and a debugging UART via a 68000-style local I/O bus. During testing, a Motorola 68000-based computer is situated on this local I/O bus to provide scratch memory and Ethernet to the processor card. The LBC is designed so that a microcontroller situated on this bus can also configure the monitor or access the performance data. Beyond its usefulness for testing, this interface can be used by a remote workstation to continuously and dynamically instrument, analyze, and display performance data without intruding upon a running program.

4.3.2 Pipeline Status

In Section 3.2, the importance of processor efficiency was described. Although most recent processors already have two or more performance counters for this task, the MIPS R4400 does not. Instead, it outputs pipeline status bits so that external hardware can mea-

sure this performance. The Pipeline Status (PS) circuit is used to help count the MIPS R4400 pipeline states listed in Table 2.3.

The 150 MHz processor produces two sets of status signals once every 75 MHz cycle to indicate the pipeline events in the last two processor cycles. Consequently, the PS circuit must operate at 75 MHz. Ideally, the PS would contain 15 counters to measure each possible event separately, but this is not economical because FPGAs combining high-speed and high-density are very costly. Instead, the PS circuit is used to decode and select the events of interest so they may be counted by four slower (50 MHz) general-purpose counters in the Counters & Interrupts circuit.

To account for the speed difference between the pipeline status changes and the counters some special preprocessing is needed. The PS produces two bits every cycle to indicate whether zero, one, or two events appeared in the last two processor cycles. These two bits are then added into a 2-bit accumulator that is hidden from the user. Every time the 2-bit accumulator overflows, it triggers circuitry that synchronizes the overflow to a 50 MHz clock and then informs the counter of the event. As a result of this action, the general-purpose counters are only incremented after every fourth event. This loss of accuracy is minimal and should be acceptable for almost all applications.

To specify which pipeline events to count, it would be most convenient to use a 15-bit control word to enable individual events from Table 2.3. However, this is not economical because the resulting circuit is large and would require a fast, expensive FPGA. On the other hand, using a 4-bit control word to count only one event would require four runs of a program to count all 15 events in the four counters. This is not reasonable in the cases where full detail is not required. To permit greater flexibility while counting, but still keep the circuit small, a 7-bit control word is used instead. The resulting circuit is more flexible, yet it is small enough to fit in two inexpensive Altera MAX7064 CPLDs which easily meet the timing requirement. These seven bits, named PS[6..0], are realized in the Configuration Controller as a portion of the control word, *GPCM_x*, for each general-purpose counter. The function of these bits will be described below.

The PS[6..0] bits are divided into two mode bits in the upper portion, and five selection bits in the lower portion. The two most-significant bits indicate one of the three counting modes listed in Table 4.2. The single event mode monitors one specific pipeline state and the other two modes allow certain states to be counted together, or *merged*, in one counter.

PS[6]	PS[5]	Monitoring Mode
0	0 or 1	counts <i>single event</i> specified by PS[3..0]
1	0	counts <i>running cycles</i> , multiple events are selected by setting the appropriate bits in PS[2..0]
1	1	counts <i>idle cycles</i> , multiple events are selected by setting the appropriate bits in PS[4..0]

TABLE 4.2. Pipeline Status bits PS[6..5] specify one of three operating modes.

The lower five bits select which specific event or events are of interest according to Table 4.3. In the single event mode, PS[3..0] specifies one pipeline state using the same encoding shown in Table 2.3. The other two modes use the lower bits to merge multiple pipeline states together by setting one or more bits in PS[4..0]. The specific events merged are listed in Table 4.3.

	Single Events	Running Cycles	Idle Cycles
PS[4]	reserved	reserved	integer + floating-point pipeline slips
PS[3]	see Table 2.3	reserved	instructions killed due to exception + branches
PS[2]	see Table 2.3	other integer + other floating-point instructions	multiprocessing + other stalls
PS[1]	see Table 2.3	taken + not-taken branch instructions	secondary cache stalls
PS[0]	see Table 2.3	load + store instructions	primary instruction + data cache stalls

TABLE 4.3. Pipeline Status bits PS[4..0] select which events to monitor.

It should be noted that the Pipeline Status circuit uses a fairly complex method to keep FPGA costs low but still maintain reasonable flexibility. However, since full-custom VLSI can easily implement fast, area-efficient counters [Vuillemin91], it would be better to include a large number of counters directly on a processor.

4.3.3 Counters & Interrupts

The Counters & Interrupts (C&Int) circuit is constructed to hold up to four general-purpose 32-bit counters and, for convenience, a barrier register and two interrupt registers. The interrupt registers are necessary for NUMAchine interrupt processing and the barrier register is an experimental hardware synchronization mechanism used to accelerate performance, but neither are essential for monitor operation. The counters, however, are essential for monitoring high-speed or overlapping events such as the pipeline states. The selection of which event to count is governed by the Configuration Controller.

The counters have been designed to produce a maskable interrupt on overflow and to automatically reset if read from a specific address. By preloading a negative number, a program can wait for a precise number of events. This allows software to be reactive to an excessive number of cache misses, for example. Also, overflow interrupts allow system software to create the illusion of a larger counter, if desired¹. Additionally, the automatic reset-on-read gives software a useful atomic fetch-and-clear option. For more information about the implementation of the counters, see [Zilic95].

The C&Int components are interconnected so that they may be read or written from the monitor bus, as depicted in Figure 4.6. The read path is obvious but the write path is unusual so it is highlighted by the shaded arrow. This strange path is an example of how the FPGA architecture has influenced the design: rather than using separate read and write paths, they are merged so that fewer FPGA logic blocks are required. In the merged path, the *data hold* multiplexer serves two purposes. First, during a counter read it captures the count in one cycle and holds it for as long as is necessary for the weak FPGA pins to drive the monitor bus. Second, during a counter write it holds the new counter value for multiple cycles while the counter's complex carry chain stabilizes. Without this organization, a larger and more expensive FPGA would be necessary.

1. The 32-bit width is sufficient to limit the counter overflow interval to approximately 1.4 minutes. This is considered infrequent enough to be nonintrusive.

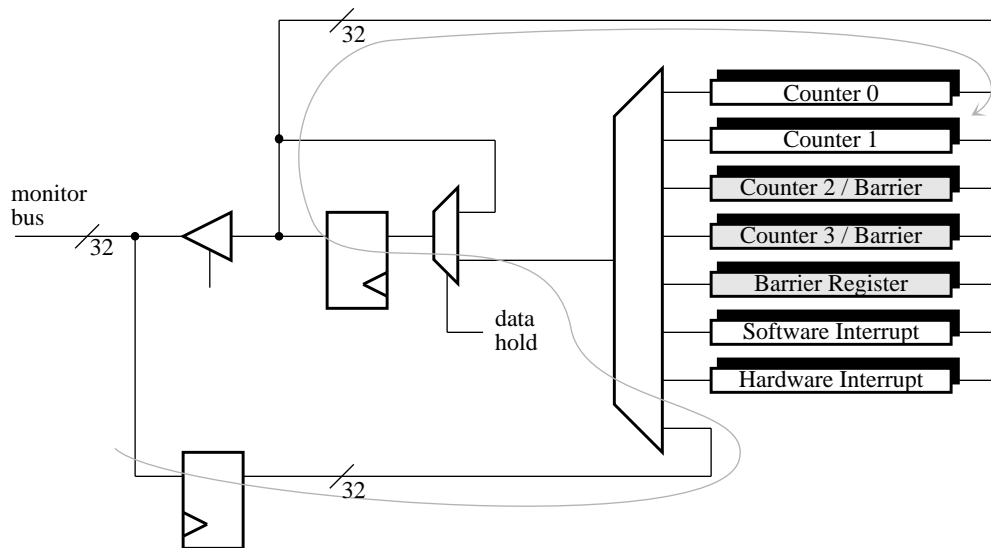


FIGURE 4.6. Counters & Interrupts datapath, with write path highlighted.

Economics also play a role in the number of counters and barrier registers. The design in the figure does not currently fit into the target FPGA, an Altera FLEX8636, because of routing constraints. Consequently, a simple design which does not include the shaded registers, hence is easier to route, is currently implemented. Designs with an additional counter or barrier register have also been realized, and these may be programmed into the FPGA via a hard reconfiguration of the C&Int device. Also, future experimentation may eventually realize a design with all the features because there is considerable flexibility in the counter design to trade-off logic cell use for routability.

Because of these routing constraints, it was decided to fix part of the C&Int architecture (the unshaded registers) and allow the implementation of the shaded portions to float. This allows a program to reconfigure the C&Int device with more barrier registers or more counters, depending upon its requirements and the latest developments in fitting the circuit in the FPGA.

4.3.4 SRAM Memory

The processor card includes fast SRAM that can be used for a variety of functions. The primary use of the SRAM is to store the state for a large number of infrequently-used counters. For this use, the depth of the SRAM (64k) is chosen so that it can be used to pro-

file secondary cache accesses as suggested in Section 3.3; this is the purpose of the ‘secondary cache index’ buffer in Figure 4.5. Like the general-purpose counters, the SRAM is wide enough (32 bits) to limit overflow frequency.

The exact function of the SRAM is governed by the Count & Increment, Latency Timer, and Configuration Controller devices which will each be described below. However, the SRAM can also be used as scratch memory by the processor for any purpose. One such use is for storing the FPGA configuration data before it reprograms them.

4.3.5 Count & Increment

The Count & Increment (C&Inc) circuit operates on data from the SRAM. Its main operation is to fetch a word from SRAM, add one to it, and write it back. Each operation requires one clock cycle. Alternatively, the write back cycle may be extended for multiple cycles. During this time, the counter is incremented on each cycle if a count-enable signal is asserted, and the latest count is continuously written back to the SRAM. An example of this use is to separately count stalled cycles for each basic block. The final use of C&Inc is as an accumulator for timing basic blocks. In this mode, the C&Inc is initialized by a word write from the processor and then the contents of the SRAM are added to it before being written back. However, this mode is not yet implemented in the current design.

The C&Inc circuit is simple enough to be implemented in the smallest Altera CPLD, a MAX7032.

4.3.6 Latency Timer

Although cache misses are important to count, Section 3.3 motivated the significance of memory latency. To time the memory response to a processor read, the Latency Timer (LT) is used. This timer is reset as the first part of the transaction is written from the external agent into the outgoing FIFO and it increments every cycle until the response or a BUS_ERROR is returned. The BUS_ERROR occurs after a time-out of 2^{12} (4096) cycles; this determines the size of the LT counter. The timer value is used to produce a histogram

of memory access times in the SRAM. Additionally, it will be shown later that the histograms can be separated by a 4-bit PhaseID, for example.

In addition to the normal 12-bit time representation, the LT can compact the time value into a 7-bit floating-point representation. The seven bits are divided into a 3-bit exponent (high bits) and 4-bit mantissa (lower bits). The significand contains an implied leading '1' unless the exponent is zero, in which case a denormal representation is used. The exact time implied by this is best explained using the following pseudocode:

```
if( exponent == 0 ) time = 0.mantissa * 2^5;
else if( exponent > 0 ) time = 1.mantissa * 2^5 * 2^(exponent-1);
```

The advantage of this format is it uses fewer bits to represent the latency by grouping longer latency measurements together into larger histogram buckets. The idea is not new; it was also used by SUPERMON to compact addresses. The bits saved will be used to separate the histograms more; for example, PhaseID can be extended to 9 bits.

Implementing the compaction requires a barrel shifter, an exponent generator, and a state bit for denormal support. Despite the seemingly complex behaviour, this circuit can also be implemented in the smallest Altera CPLD, a MAX7032.

4.3.7 Configuration Controller

The most complex portion of the monitor is the Configuration Controller (CC). It is the command centre of the monitor that controls what is to be monitored and, in some cases, determines whether an event being monitored has just occurred. It also controls whether an interrupt should occur when a counter overflows. Due to the detail and complexity involved, the next subsection will describe the CC in greater detail.

4.4 Programmable Configuration

The Configuration Controller (CC) is implemented in a reprogrammable FPGA so that its function may be changed to collect new data or change the conditions of collection. However, for typical uses a Master CC (MCC) circuit was designed to provide most of the flex-

ibility a user will require. This is done by *soft configuration* of the circuit with simple writes to the user configuration registers listed in Table 4.4. The operation of the registers will be explained below.

4.4.1 PhaseID Register and PhaseID Watch

The PhaseID Register (PR) is a 16-bit implementation of the PhaseID recommended in Chapter 3 to easily partition the collected performance data. By writing a new PR value², the address to the SRAM is changed and a different counter is selected. As will be shown later, some SRAM counting modes may use only a portion of PhaseID. In these cases, a new PR value will select a different bank of counters.

The lower 4 bits of the PR have an additional purpose. They are brought outside of the processor card monitor and attached to some bits in the outgoing FIFO so that they are attached to all memory transactions initiated by the processor. Performance monitoring hardware in the memory card and network can use these 4 bits to demarcate transactions originating from different phases of a program.

The PR can also selectively enable the SRAM or C&Int counters. A constant, called PhaseID Watch (PW), is compared against the PR. The result of this comparison drives a counter-enable circuit which will be described later. The primary use of this feature is to enable a C&Int counter during one specific phase.

4.4.2 Command Watch and Command Filter

The Command Watch (CW) and Command Filter (CF) registers are used to restrict counting to only certain types of transactions. Every NUMAchine transaction is composed of multiple network *packets*, each containing a 13-bit Command identifier to indicate the transaction type. For example, it can identify whether the packet is a cache line read, a request or response, or whether it contains an address or data³.

2. In Chapter 3, two other methods of changing PhaseID were suggested which involved encoding the new value into the address during a read or a write. These alternative methods have not yet been implemented, but they are easy to add.

The CF and CW registers are used in conjunction with the Command Register (CR). The CR is automatically updated with every packet's Command as it is passed to and from the External Agent. The Command Filter (CF) register is used as a bitmask to remove uninteresting bits, and the result is compared against the Command Watch (CW) register⁴. Again, the comparison result is used to drive a counter-enable circuit.

By carefully selecting proper CW and CF values, it is possible to specify multiple events to be monitored at once. For example, cache line read and read-exclusive responses can be counted together.

4.4.3 Address Watch and Address Filter

Similar to CW and CF, the Address Watch (AW) and Address Filter (AF) registers can be used to watch accesses to a region of memory. Because of the 32-bit width of the monitor bus, the upper and lower portions of the AW and AF registers must be written separately. This can be done by software with two word writes or with a single doubleword write in big-endian data-word order.

The AW and AF registers are used in the following manner. First, an Address Register (AR) is automatically updated with the most recent physical address passed to or from the External Agent. Then, AF is applied to AR as a bitmask and the result is compared to AW. With this setup, a contiguous region of memory which is aligned and sized to a power-of-2 can be monitored. However, for this to work properly the operating system must allow a program to allocate a contiguous portion of memory. Additionally, the operating system should lock these memory pages down so they don't migrate and aren't demand-paged to disk.

3. Although packets forming a transaction are always placed contiguously on a NUMAchine bus, they may be separated and interleaved with other transaction packets on a ring.

4. Software must ensure that a new CW value is compatible with CF by masking CW first, or the comparison may never match.

Address	Acronym	Name	Bits	Notes
0	PR	PhaseID Register	15..0	PhaseID Register
1	PW	PhaseID Watch	15..0	compared against PR
2	CW	Command Watch	12..0	compared against filtered command
3	CF	Command Filter	12..0	removes unwanted command bits
4	AWhi	Address Watch High	7..0	forms AW bits 39..32
5	AWlo	Address Watch Low	31..0	forms AW bits 31..0
6	AFhi	Address Filter High	7..0	forms AF bits 39..32
7	AFlo	Address Filter Low	31..0	forms AF bits 31..0
8	GPCM0	General-Purpose Counter 0 Mode	23 22..16 15 14..11 10 9..8 7..6 5..4 3..2 1 0	counter enable pipeline status configuration bits PS[6..0] enable interrupt on overflow event select (count one of 16 events) count cycles high or low-to-high transitions enable masks below .. invert mask sense enable PW compare .. invert compare sense enable AW compare .. invert compare sense enable CW compare .. invert compare sense enable sending mask, SM enable receiving mask, RM
9, A, B	GPCM1, GPCM2, GPCM3	General-Purpose Counter 1, 2, 3 Mode	23..0	same as GPCM0
C	SCM	SRAM Counter Mode	22 21..20 19 18 17 16 15 14..11 10..0	enable interrupt on overflow counter mode muxMISS — uses miss type when set muxRSR — uses RSR when set muxPRhi — uses upper 5 bits of PR when set muxPRmid — uses middle 7 bits of PR when set muxPRlo — uses lower 4 bits of PR when set event select (count one of 16 events) same as GPCM0 bits 10..0
D	reserved	reserved	n/a	reserved
E	GPCE	Master General-Purpose Counter Enable	0	master enable for all four general-purpose counters
F	SRAMCE	Master SRAM Counter Enable	0	master enable for SRAM counters

TABLE 4.4. Master Configuration Controller user configuration registers.

4.4.4 General-Purpose Counter Modes

The general-purpose counters in the Counters & Interrupts circuit can count from a number of events and have a number of different operating modes. The operation of these counters is governed by the four GPCMx registers; each register is identical except that it governs a different counter. The role of the various bits in these registers is described below.

The lower 10 GPCMx bits are invert and enable bits for the count-enable circuit shown in Figure 4.7. To limit the circuit size but maintain some flexibility, the counters share comparators that feed individual count-enable circuits. One way this can be used, for example, is to enable one counter on address matches while another is enabled for all single-word reads. Another use is to AND together multiple comparators by enabling them simultaneously to form a compound condition for enabling a counter. Alternatively, the invert bits can be used to reverse the sense of the comparison (*i.e.*, not-equals) or to merge multiple comparisons in an OR fashion. Finally, the count-enable circuits can also be enabled while transactions are sent or received (or both) by the external agent.

Bits 11 through 14 control a multiplexer that selects which event count according to the list in Table 4.5. When bit 10 is clear, the counters will count time by incrementing for every cycle the event is asserted. However, it makes sense to set bit 10 on events 3, 4, 5, 6, 7, 8, A, D and F so that only low-to-high transitions of the event, *i.e.* *event occurrences*, are counted instead of latency; these event occurrences are indicated with {braces} in the table. Additionally, when the counter is configured for pipeline status events, the PS configuration bits, GPCMx[22..16], are used to specify the proper pipeline event. The function of these bits was already discussed in Section 4.3.2. The counters have a master enable, bit 23, and can be configured to trigger an interrupt when they overflow by setting bit 15.

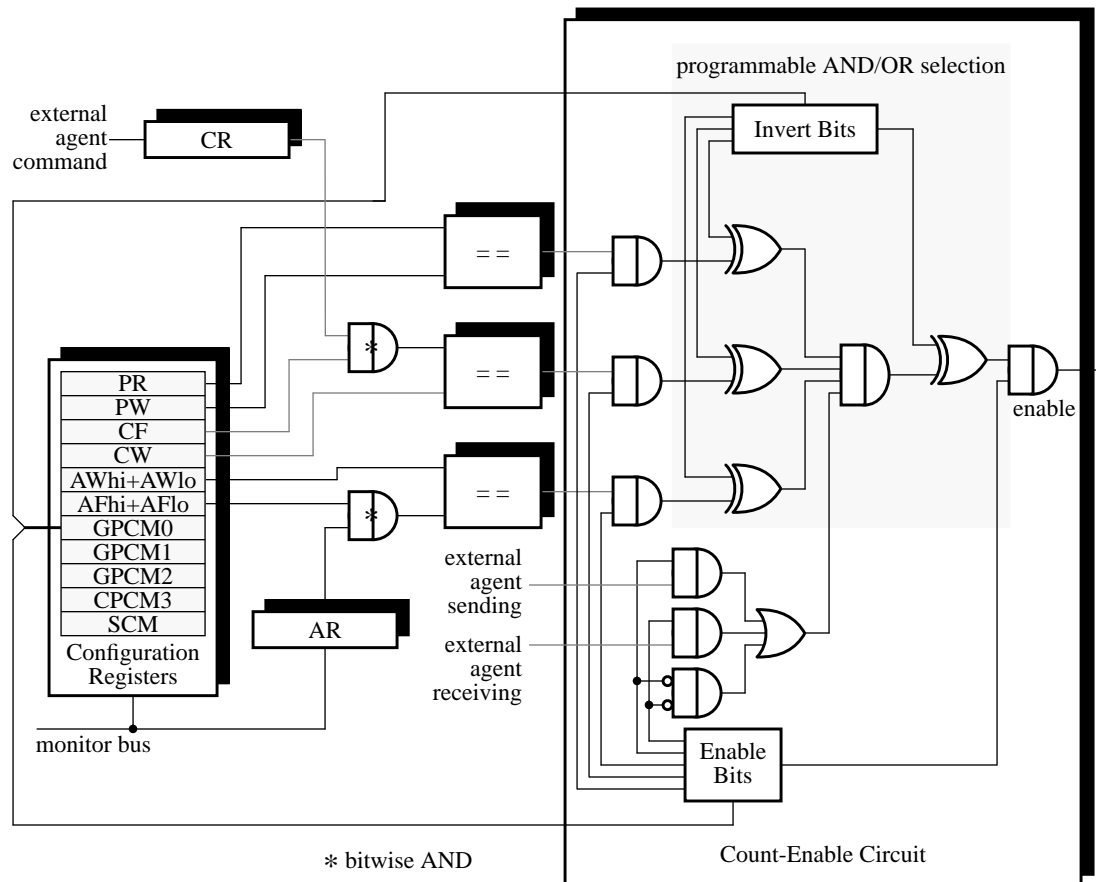


FIGURE 4.7. MCC registers with one Count-Enable Circuit.

General-Purpose Counter Event Details

A number of events in Table 4.5 require some additional explanation. First, the latency or number of *successful* ownership misses are measured with event 7; an unsuccessful miss is caused by a time-out, so a bus error exception is taken and may be counted by software. This must be contrasted to cancelled ownership misses, which occur because a competing processor ‘won’ the ownership first, that are counted by event 8. The total number of ownership misses is the sum of the successful, unsuccessful, and cancelled ownership misses.

Second, some transactions such as a read request may be negatively acknowledged (NACKed) and returned by a device when it cannot service the transaction, so the External Agent must retry these requests. This may happen if memory temporarily has a block locked, for example, and it may be NACKed several times before a response arrives; this is measured by event 9. Also, the total cycles spent from the first retry to the final response

GPCMx[14..11]	Event
0	count nothing (disable)
1	count always (high-resolution cycle counter)
2	pipeline status events
3	interrupt request latency {interrupt requests}
4	cache line read latency {cache line reads}
5	cache line read-exclusive latency {cache line read-exclusives}
6	cache line read latency + read-exclusive latency {cache line reads + cache read-exclusives}
7	successful ownership miss latency (upgrades + updates) {successful ownership misses}
8	cancelled ownership miss latency (upgrades + updates) {cancelled ownership misses}
9	negative acknowledgement retries (may be >1 per request)
A	negative acknowledgement cycles {transactions with one or more negative acknowledgements}
B	latency of most recent (cached or uncached) request, eg: read, read exclusive, upgrade
C	external agent bus activity (<i>i.e.</i> , utilization)
D	bus request latency (until a bus grant is given) {bus requests}
E	external invalidations that hit in the cache (estimate)
F	invalidation miss latency (estimate) {invalidation misses}

TABLE 4.5. General-purpose counter events.

is represented in event A; counting the low-to-high transition of this event counts the number of requests that received one or more NACKs.

Third, the total latency of the most recent transaction is measured by event B. Here, the counter is reset every time a request is issued and stopped whenever a response is returned. Fourth, every cycle the external agent bus is used will be counted with event C; measuring transitions of this corresponds to counting the total number of transactions. Fifth, event D measures the response time of the bus, which may be slow due to contention. Sixth, invalidations that originate from outside the processor and hit in the secondary

cache are measured with event E and the number of invalidation misses are reflected by event F.

To measure external invalidation hits, the state machine in Figure 4.8a is used. The invalidate starts a cache-watching state which waits for a secondary cache probe (from the processor) that maps to the same address as the invalidate; a successful mapping is called an *index match*. If the state of this line is valid and subsequently changed to invalid before a different cache line is accessed, the state machine passes through the shaded state in the figure and the event is counted. Otherwise, the state machine returns to the idle state. This algorithm is only approximate because the MCC does not do the same tag comparison the processor does; there are not enough pins left on the FPGA to monitor the tag SRAM. Despite this, it should still be accurate. The state machine will only over-count if the processor intentionally invalidates a block which maps to the same line within an approximately 34-cycle time window (the external invalidate is guaranteed service within this time by the processor design). This is unlikely to occur because the processor seldom intentionally invalidates a line (only explicit cache flushes will do this). Also, it only under-counts if the processor has 2 consecutive primary cache misses within this window and the first maps to the same line as the invalidate. This, too, is unlikely because of high primary cache hit rates and the improbable index match. Finally, ownership misses which are cancelled will be included in the external hit count, but they can also be measured by a performance counter and subtracted out, if desired. Thus, the approximation should be sufficient for most uses of the data.

Similarly, invalidation misses are detected by the state machine in Figure 4.8b. It waits for an access which reads an invalid state from the secondary cache. If the next read or read-exclusive transaction emitted by the External Agent maps to the same line, the event is counted. If the index doesn't match, or an uncached read is performed, the state machine resets. However, if a write or writeback transaction is encountered instead, it must still wait for an External Agent read transaction because the write may have been generated earlier than the invalidation miss. Again, this circuit only forms an estimate of invalidation

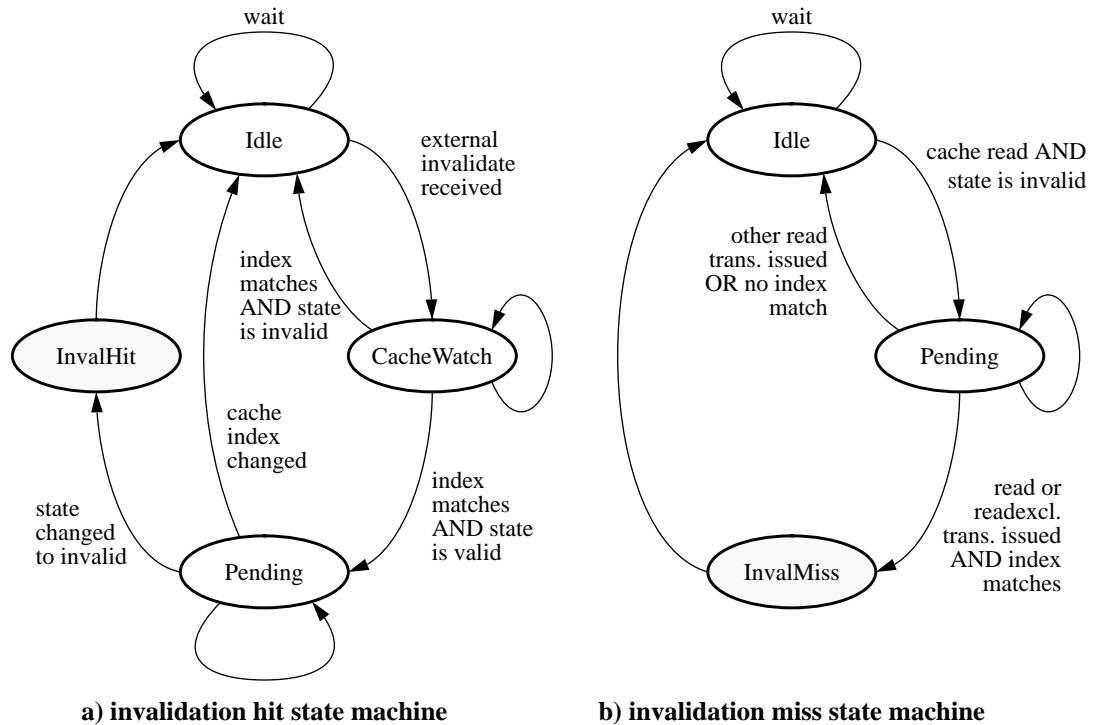


FIGURE 4.8. State machines to detect invalidation hits and misses.

misses because the cache tag is unknown and unchecked. Hence, it may incorrectly count some other types of misses as invalidation misses, but the error is expected to be small.

4.4.5 SRAM Counter Mode

The purpose of the SRAM Control Mode (SCM) register is to: 1) control the count-enable signal for the C&Inc device, 2) select the events to count, and 3) choose the address supplied to the SRAM. The count-enable circuit, which is identical to that used for the general-purpose counters, is controlled by the lowest 11 bits. The event to count is selected using the next 4 bits, SCM[14..11], as shown in Table 4.6. Not all possible events have been defined in the table, so room is left for future expansion. However, the few events that are present can be combined with the count-enable circuit to collect a wide variety of data. This will become apparent below.

The address supplied to the SRAM counters is determined by the next group of seven SCM bits. The lower five of these, *i.e.* bits 15 through 19, control a collection of multi-

SCM[14..11]	Appropriate Mode	Event
0	any	count nothing (disable)
1	any	count always
2	0	count secondary cache accesses
3	0	count secondary cache reads
4	0	count secondary cache writes
5	0	count secondary cache misses
6	1, 2, or 3	cache line reads + read exclusives + upgrades
7 to F	any	reserved

TABLE 4.6. SRAM counter events.

plexers in the MCC that form the SRAM address. This circuit is shown in Figure 4.9; refer to Table 4.4 for definitions of the various signals in the figure. The two upper bits control the tristate-enables in the figure and are encoded so that software cannot mistakenly enable competing drivers; the encoding is shown in Table 4.7. Furthermore, the multiplexer and tristate controls can be overridden by the muxAR control which is generated automatically when the R4400 attempts to read or write the SRAM counters.

SCM[21..20]	Mode	SRAM Address Source
0	Secondary Cache Mode	secondary cache index
1	Latency Timer Mode	latency timer (all 12 bits) plus muxPRlo data
2	MCC Mode	all MCC sources (muxPRlo, muxPRmid, and muxPRhi)
3	Latency Timer Compact Mode	latency timer (lower 7 bits, compressed format) plus muxPRlo and muxPRhi data

TABLE 4.7. SRAM counter modes.

The Secondary Cache Mode allows the SRAM counters to count secondary cache accesses, reads, or writes. Because all secondary cache accesses are caused by a primary cache miss, the number of primary cache misses can be monitored. By counting the number of loads and stores with the general-purpose counters, primary cache miss rates can easily be determined. Similarly, secondary cache miss rates can be measured. Note that the miss counts are per cache line, so small regions that suffer from conflict or invalidation misses can be determined. By initializing all of the SRAM with a suitable initial count and enabling interrupts on overflow, software can be informed when a cache line receives too

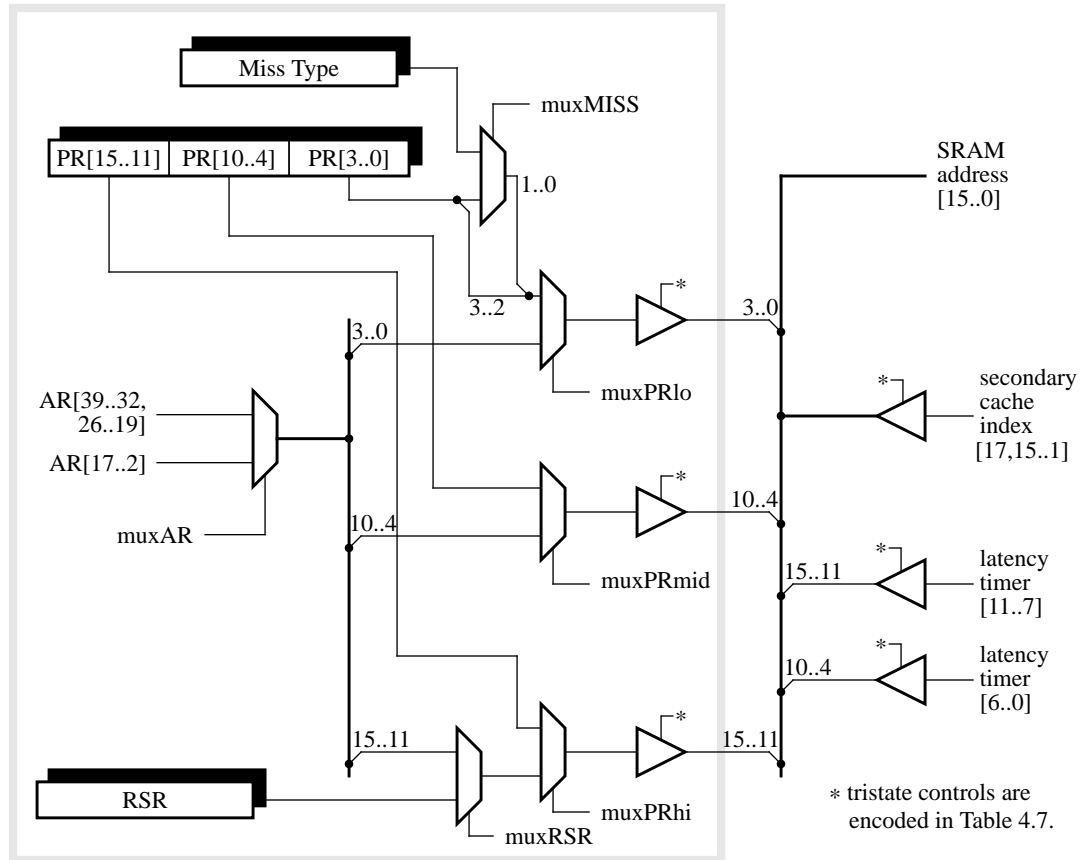


FIGURE 4.9. SRAM address generation. The outlined portion is inside the MCC.

many misses. Additionally, the R4400 can optionally split the secondary cache by placing instructions in the upper half (index bit 17 is set) and data in the lower half. By using the split mode, separate statistics on instructions and data can be collected.

The two Latency Timer Modes construct a histogram of memory access latencies. The histogram from the ‘full width’ timer shows exactly how many read requests took 0, 1, 2, ..., 4095 cycles to obtain a response. The compact timer divides these times into increasingly-sized bucket intervals as follows:

```

0-1, 2-3, ..., 62-63,
64-67, 68-71, ..., 124-127,
128-135, 137-143, ..., 248-255,
256-271, 272-287, ..., 494-511,
512-543, 544-575, ..., 992-1023,
1024-1087, 1088-1151, ..., 1984-2047,
2048-2175, 2176-2303, ..., 3968-4095.

```

From either of these histograms, a memory latency distribution, including characteristics such as average and variance, can be constructed.

The statistics generated with the Latency Timer Modes can be also be separated by the PhaseID Register, a *Miss Type Register (MTR)*, and a 5-bit *Response State Register (RSR)*. The PhaseID has been described previously, but MTR and RSR are new registers that describe a cache miss. The MTR is a 2-bit quantity that describes the type of cache miss according to Table 4.8. In contrast, the RSR contains the 3-bit memory state indicator (MSI) in the upper half and two bits in the lower half to indicate the request type.

Value	Description
0	successful ownership misses
1	cancelled ownership misses
2	invalidation misses
3	other misses

TABLE 4.8. Miss types encoded in the Miss Type Register.

The MSI bits returned with every NUMAchine memory response indicate whether data was returned from the network cache (a network cache hit) or memory and one of the four memory states: LV, GV, LI, and GI. In the case of a network cache hit, the state of the network cache line is returned; in all other cases the state at the home memory module is returned. The additional two bits in the RSR indicate whether the access was to local or remote memory and whether the request was a read or a read exclusive/upgrade. Together, these five bits give details about how far the request had to travel, the effectiveness of the network cache, and the amount of coherence traffic that was required to respond. *When using the RSR, a user must be careful to use the receiving (incoming) mask lest the SRAM counters be incorrectly updated before the RSR arrives.* The encoding of the RSR is shown in Table 4.9.

Value	RSR[4]	RSR[3]	RSR[2]	RSR[1]	RSR[0]
0	Network Cache Miss	Local State	Invalid	Local Address	Read
1	Network Cache Hit	Global State	Valid	Remote Address	Read Exclusive or Upgrade

TABLE 4.9. Response State Register encoding.

The final SRAM counter mode places the SRAM address completely under MCC control. This means the entire SRAM can be addressed by PhaseID register, part of it can come from the RSR or MTR, or portions can be constructed from the Address Register. The flexibility comes from the ability to specify the individual multiplexer control signals. One particular configuration, when these controls are all set to zero, uses the AR[39..32,26..19] bits to provide the SRAM address. In this ‘Address Mode’, all references to secondary-cache sized blocks (1 MB) are counted. This is an experimental feature to count how widespread the cache-miss access patterns of the application are. In particular, remote memory is counted distinctly from local memory, so a measure of locality can be constructed. By counting access latency, this mode can also help identify whether a particular memory module or network connection is more congested than others.

4.4.6 Master Counter Enables

Two master counter enables are provided so that monitoring can be easily stopped or started without affecting the configuration of the counters. Separate enables are provided for the SRAM and the general-purpose counters, but both may be enabled (or disabled) simultaneously with one doubleword write. Furthermore, multiple processor card monitors can be enabled using a NUMAchine multicast write.

4.5 Summary

The NUMAchine monitor described above is capable of measuring many events, but it may not clear to the reader whether it satisfies the measurements proposed in Chapter 3. A summary of the 23 specific features recommended to monitor and how the implementation meets these requirements is shown in Table 4.10. From this table, it can be seen that only five items cannot be monitored, of which three are not applicable to the R4400. The other two items, memory and network queue measurements and basic block timing support are left as future implementation items. Of these, the memory and network queues should be given higher priority because they are relevant for evaluating NUMAchine architecture performance.

	Recommended Item	Implemented?	Notes
Section 3.2	1 dynamic instruction count	y	pipeline counters form groups of instruction counts
	2 cycles lost due to NOPs and pipeline slips	y	pipeline counters can measure slips NOPs can be measured via basic block counts
	3 cycles lost due to stalls	y	pipeline counters allow flexible grouping of different stalls
	4 pipeline flushing and restarting	partial	pipeline counters can measure pipeline flushes, processor support required for counting restarts
	5 TLB faults	y	can be monitored in software
	6 branch prediction	n	n/a — processor does not use branch prediction
	7 special hardware features	y	network cache hits
Section 3.3	8 informing memory operations	n	n/a — processor must have special support for this
	9 phaseID register	y	varies between 4-bits and 16-bits
	10 profile cache activity	y	SRAM can count primary cache misses, secondary cache misses, etc.
	11 cache-profile watch-points or thresholds	y	can preload SRAM with negative-threshold and interrupt on overflow
	12 invalidation miss count	y	estimate only, need more FPGA pins to form an accurate count
	13 external invalidation hits	y	estimate only, need more FPGA pins to form an accurate count
	14 ownership misses	y	estimate only, need more FPGA pins to form an accurate count
	15 memory and network queue performance	n	future monitoring implementation
	16 measure locality of references	y	the RSR indicates a local/remote request; address-region monitoring can watch memory that is local/remote/both
	17 memory state indicator	y	result returned from memory indicates network cache hits and local/global or valid/invalid states
	18 measure total miss cycles	y	pipeline counters can be merged to count all cache miss stall cycles
19 measure apparent miss cycles	n	n/a — R4400 has a simple pipeline with no latency-hiding mechanisms	
Section 3.4	20 timestamp counter	y	provided by R4400
	21 local process counter	y	a general-purpose counter can be used for this
	22 basic block counting SRAM	y	supported by using entire 16-bit PhaseID
	23 basic block timing SRAM	n	accumulate function not implemented, possible future extension; SRAM must be shared with basic block counting item

TABLE 4.10. Comparison of recommended versus implemented features.

One of the primary objectives of this thesis is to maintain a cost-effective focus. In this regard, several design decisions for the NUMAchine processor card monitor, such as what size FPGA to use, were influenced by minimizing the cost of the hardware. To illustrate the costs, Table 4.11 shows the cost of the major performance monitor components described in this chapter. The total price of \$345 is slightly inflated because two of the more expensive components, namely the Local Bus Controller and Counters & Interrupts circuits, contain not only monitoring circuits, but also other non-monitoring functions that are required by NUMAchine.

Circuit	Device	Quantity	Approximate Price (\$CAD)
Local Bus Controller	FLEX8452A-5, 160-pin QFP	2	\$90
Pipeline Status	MAX7064-10, 44-pin PLCC	2	\$35
Counters & Interrupts	FLEX8636A-5, 84-pin PLCC	1	\$70
SRAM	64k x 16	2	\$30
Count & Increment	MAX7064-15, 84-pin PLCC	1	\$15
Latency Timer	(shared with C&Incr)		
Configuration Controller	FLEX8636A-5, 160-pin PLCC	1	\$75
buffers	latching, tristate	3	\$15
other control	MAX7064-15, 84-pin PLCC	1	\$15
		TOTAL	\$345

TABLE 4.11. Approximate cost of monitoring components.