# Performance optimizations, implementation, and verification of the SGI Challenge multiprocessor

Mike Galles and Eric Williams
Silicon Graphics Computer Systems
2011 N. Shoreline Blvd.
Mountain View, CA 94039-7311

**Abstract**

*This paper presents the architecture, implementation, and performance results for the SGI Challenge symmetric multiprocessor system. Novel aspects of the architecture will be highlighted, as well as key design trade-offs targeted at increasing performance and reducing complexity. Multiprocessor design verification techniques and their impact will also be presented. The SGI Challenge system architecture provides a high-bandwidth, low-latency cache-coherent interconnect for several high performance processors, I/O busses, and a scalable memory system. Hardware cache coherence mechanisms maintain a consistent view of shared memory for all processors, with no software overhead and minimal impact on processor performance. HDL simulation with random, self checking vector generation and a lightweight operating system on full processor models contributed to a concept to customer shipment cycle of 26 months.*

## 1 Introduction

The Silicon Graphics Challenge line of symmetric multiprocessing computers is designed to provide engineers and scientists with a powerful set of computational, design, and visualization tools at aggressive price performance levels. Scalability and balance across processor, memory, and I/O subsystems also makes these computers attractive to a wide range of high performance computing applications, from the creation of special effects for the entertainment industry to database management in the commercial sector.

This paper begins with an overview of the system parameters and configurations, then moves directly into presentation of the coherence protocol, latency reduction techniques, and bus efficiency mechanisms, with emphasis on performance impact. The I/O bus and subsystem is later described, followed up by a discussion of effective design verification techniques used to reduce the overall product cycle time. Finally, performance results are presented and summarized.

### 1.1 Overview of the coherent interconnect

The heart of any multiprocessor system is its interconnect. The interconnect provides each processor with a path to memory and I/O, and may also support cache-coherence and other features. All Challenge multiprocessors use the POWERpath-2 as a coherent interconnect. The POWERpath-2 is a fast and wide split transaction bus, which provides high-bandwidth, low-latency, cache-coherent communication between processors, memory, and I/O. Additional features include special transactions for efficient processor synchronization and I/O DMA transfers.

**Table 1 POWERpath-2 Overview**

- 1.2 Gigabytes per second sustained transfer rate
- 9.5 million transactions per second, sustained
- Snoopy write-invalidate cache-coherence maintained in hardware
- Multiple outstanding, variable duration split read transactions
- Independent 256 bit data, 40 bit address busses
- 47.6 MHz synchronous signalling (21 ns cycle)
- ECC protected memory and caches, parity protected data and address busses

### 1.2 Available system configurations

Given a flexible, high speed interconnect with fully interchangeable nodes, it is possible to build a wide variety of custom system configurations. Subsytems can grow in size along several axes, the limits of which are listed in the following table.

**Table 2 Max system configurations**

- 36 MIPS R4400 Processors, 2.7 GigaFlops peak
- 18 MIPS streaming superscalar TFP processors, 5.4 GigaFlops peak
- 16 Gigabytes main memory, 8-way interleaving
- 4 POWERchannel-2 I/O busses, each providing 320 MBytes/sec

- 32 fast-wide independent SCSI-2 channels
- 3.0 Terabytes disk (RAID) or 960 Gigabytes disk
- 4 HIPPI channels, 8 ethernet channels
- 5 VME64 expansion buses provide 25 VME64 slots
- 3 Reality Engine high performance graphics systems

## 2 POWERpath-2 Protocol

The POWERpath-2 protocol was designed with a RISC philosophy. The types and variations of transactions are small, and each transaction consumes exactly five cycles to transfer a single cache block of 128 bytes. Read transactions are split, and independent address and data transactions can occur simultaneously, creating a pipeline effect. This allows bus performance to be optimized for common transactions, without requiring additional hardware complexity to handle rare cases.

### 2.1 Cache Coherency

The cache coherency protocol is identical to the Illinois Protocol [1], except that cache to cache transfers are only used for dirty data. Each cache has four states; invalid, exclusive, dirty exclusive, and shared. Transition between cache states is caused by actions initiated by the processor or by coherent transactions appearing on the bus. In order to eliminate unnecessary cache contention between the processor and the bus snoopy mechanism, a duplicate set of cache tags [2] is maintained by the processor interface ASIC. A processor cache will only be accessed for coherency reasons if the data in question actually resides in that cache; bus traffic targeting lines not cached by the local processor will not affect the processor or its cache.

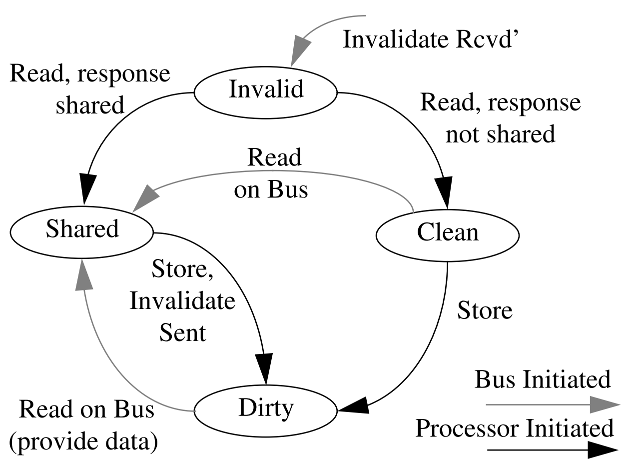



**Figure 1 Cache coherency state transitions**

Whenever a read request is satisfied by data from a processor's cache, the memory accepts the cache read response as a sharing writeback. This mechanism eliminates the need for a shared dirty state, as data is cleaned as it passes across the bus from cache to cache. [3]

Split read transactions complicate a coherency protocol, as they are not atomic operations. A read or read exclusive transaction affects the state and ownership of a line, but between the read request and read response the precise owner and state are ambiguous. To avoid this ambiguity, the POWERpath-2 bus protocol disallows coherent operations to cache lines which target a pending read. To enforce this restriction, each bus interface must track all pending reads, and avoid issuing conflicting requests. To keep the implementation tractable, a maximum of 8 pending reads are allowed at any given time.

Pending reads are tracked by being associated with read resources. When a read request is issued, it occupies the first available read resource. A pending read will occupy a read resource until a corresponding read response appears on the bus. If all read eight read resources are filled, future read requestors must wait until a read resource becomes available.

To indicate which read resources are occupied, the POWERpath-2 bus protocol specifies inhibit signals. There are eight inhibit signals; one for each read resource. When a bus node has not yet finished its cache snoop to satisfy a pending read, the node will drive the inhibit signal of the read resource which caused the snoop. If the snoop completes and indicates a clean cache line, the snooping node will simply drop its inhibit signal and allow the requesting node to accept memory's response. If the snoop indicated a dirty line, the node will request the bus and provide a read response, then drop its inhibit signal.

### 2.2 Bus Timing

Bus state sequencing, arbitration, and flow control techniques have a large impact on latency and achievable bandwidth. Technology restrictions play a key role in determining these aspects of bus design. The POWERpath-2 bus transceivers, for example, do not support high speed wired OR operations and require a one cycle turn around time between different drivers. Given these restrictions, a bus timing protocol was developed which incorporates multi-level arbitration, flow control, and complete address, command, and data transfer in a 5 clock transaction.

Every POWERpath-2 bus transaction consists of five clock cycles. System wide, bus controller ASICs execute the same five state machine synchronously: arbitration, resolution, address, decode, and acknowledge. This RISC approach to bus protocol simplifies the controllers and allows them to operate at maximum frequency with minimal design risk.

When no transactions are occurring, each bus controller drops into a two state idle machine. This allows new re-

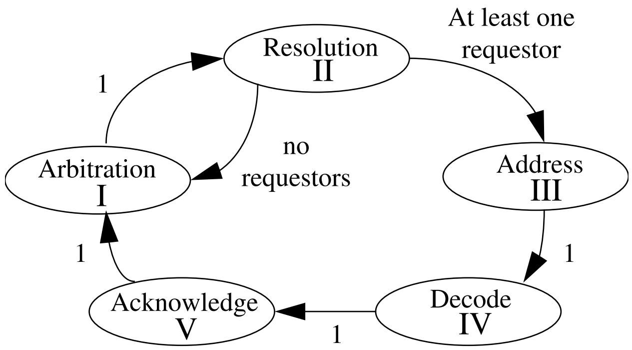



**Figure 2 Bus state transition diagram**

quests which appear on an idle bus to arbitrate immediately, instead of waiting for the arbitration cycle to arrive. Two states are required to prevent different requestors from driving the arbitration lines on subsequent cycles.

The address and data busses are arbitrated separately to accommodate split read transactions in a highly pipelined fashion. While one node is using the address bus to issue a new read request, another node can use the data bus to provide a read response to one of several pending reads. Under normal system loads, the memory system can provide a read response on the data bus 2 transactions after the read request appears on the address bus. For write requests, address and data appear simultaneously, while invalidate requests occupy only the address bus. With a sufficient number of requestors, the bus can sustain the peak transfer rate of 1.2GBytes/sec.

Because they can act independently, the address and data busses must be arbitrated for separately. During the arbitration cycle, the middle third of the address and command signals is used for address arbitration and the lower third of the address and command signals is used for data arbitration. The upper third is used to make *URGENT* (high priority) requests. Each bus node asserts the appropriate signals to indicate which type of request it is making. The arbitration vector is then read into every bus controller ASIC, where bus winners are determined based upon a common algorithm and shared bus state. Whenever possible, requests for the address and data buses are combined for greater bus efficiency. Read requests and responses are given higher priority to reduce read latency. Urgent requests are granted the bus quickly to prevent starvation. When there are multiple requestors at the same priority level, a round robin algorithm is used to ensure fairness.

This scheme of distributed arbitration reduces the overall time required to determine a bus winner, and thus reduces latency. A central arbitration scheme requires that all requests be issued to a single chip, which determines a winner and grants the bus. This requires one additional transaction over the distributed arbitration scheme, where the knowledge of which node won the bus is immediately available to all nodes. Although distributed arbitration does require more logic to implement, higher gate counts available in today's ASICs make this cost low compared to the benefits of saving clock cycles

The figure below shows a generic timing diagram for POWERpath-2 bus transactions. The arrows indicate data bus arbitration in state I, identification through read resource tagging in state III, and flow control information transferred in state V. During state V (the acknowledge cycle), a node is allowed to drive the same 3 address and command lines it uses for arbitration, allowing the acknowledge and arbitration cycles to occur adjacently without violating the one cycle turn around time required by the driver technology. These 3 lines are used for address acknowledge, data acknowledge, and data state (shared versus exclusive.).

## 2.3 Minimizing read latency

Memory read latency has one the most direct impacts on system performance of any design parameter. To achieve scalability in multiprocessor systems, it is particularly important that latencies remain low while larger numbers of processors and IO devices saturate the bus with memory requests.

Latency reduction techniques implemented in the POWERpath-2 can be divided into two broad categories; techniques which minimize latency for an single processor to achieve high performance for an individual node, and techniques which minimize overall system latency to support scalable multiprocessing.

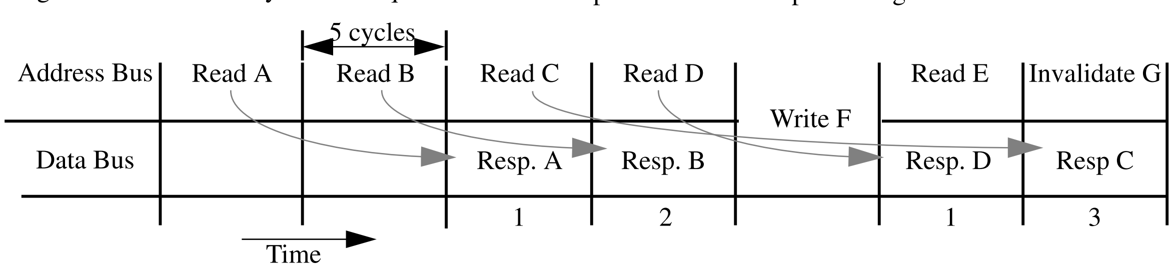



**Figure 3 Split read pipelining with write and invalidate traffic**

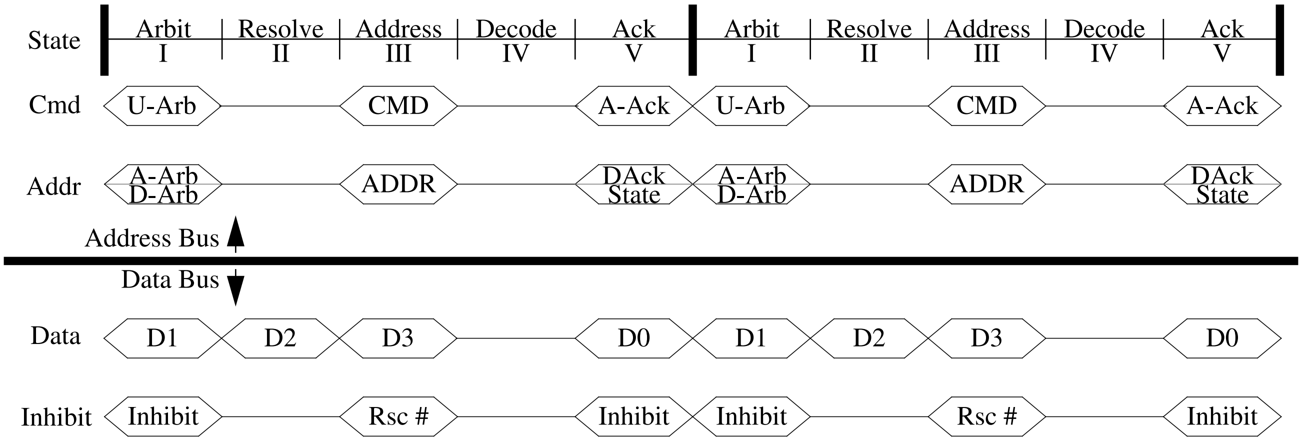



**Figure 4 POWERpath-2 bus timing diagram (2 transactions shown)**

### 2.3.1 Memory read response latency

Read latency to an individual processor is minimized by a number of design features in the processor interface implementation, in the bus protocol design, and in the memory system design. The numbered features below refer to the following figure.

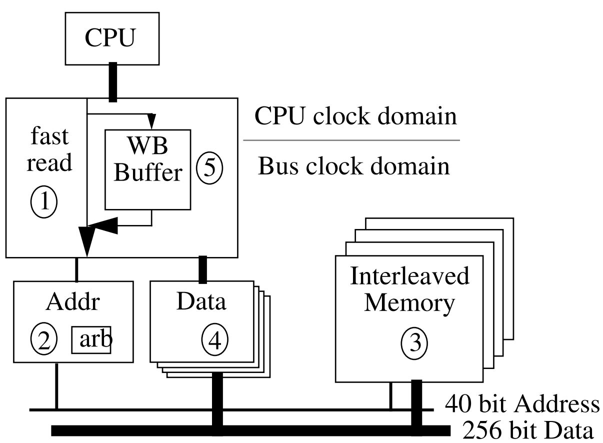



**Figure 5 Read latency reduction techniques**

1. The logic in the processor interface chips is designed to anticipate and accelerate read requests. Other transactions, such as writes, suffer increased latency in this design, but have minimal impact on performance.
2. Arbitration for the POWERpath-2 bus favors read requests over other requests with less sensitivity to latency. Since the POWERpath-2 is designed with split address and data busses, read requests and read responses for independent transactions can occur simultaneously, as these paths share no common resource.
3. The Challenge memory system uses high speed buffers to fan out addresses to a 576 bit wide DRAM bus. Fast page mode accesses allows an entire 128 byte cache line to be read in two memory cycles, while data buffers pipeline the response to the 256 bit wide POWERpath-2 data bus. Twelve clock cycles after a read address appears on the address bus, response data appears on the data bus.
4. As the data crosses the asynchronous interface into the processor clock domain, a programmable trigger register determines when the data buffers can begin streaming into the processor. Thus, for any arbitrary processor speed, the trigger register can be set to accept data in the minimal time.
5. The processor interface contains a writeback shadow buffer. When the processor encounters a cache miss, it initiates a read request followed directly by a write request if the replaced cache line is dirty. While the read is taking place, the processor interface ASIC accepts and stores the write data in a special buffer, where it is kept until the read completes. When the writeback is finally sent to the bus, it does not interfere with processor read requests.

### 2.3.2 MP System latency reduction

The POWERpath-2 protocol implements a number of design features to maintain low-latency across a large system under heavy load.

- Each processor interface ASIC maintains a complete set of duplicate cache tags which hold state information for each cache line. When a coherent request appears on the bus, the processor interface ASIC checks the state in the duplicate tag store. If the state is invalid or shared, a proper response is made on the bus and no request need be sent to the local processor. The duplicate tags not only prevent processors from receiving unnecessary external requests, they also lower the overall read latency because the duplicate tag lookups to remote processors is much faster than issuing external snoop requests to those processors.

- When a read request appears on the bus, memory immediately initiates a fetch of that cache line. Concurrently, any processor interface ASICs which discover duplicate tags indicating an exclusive cache block will initiate an intervention request to that processor's cache. By initiating both memory and, if necessary, remote cache accesses simultaneously, it is guaranteed that the response will arrive in the shortest amount of time possible, regardless of where the data resides.
- Whenever a read request is satisfied by data from a processor's cache, the memory system accepts the processor's read response as if it were a write request. Sharing writebacks cause dirty cache blocks to be cleaned as they move between processor caches. Any processor receiving a read response from another processor's cache will load the data in a clean state, eliminating the need for additional writebacks of that block.
- The operating system can mark text pages with a special attribute. When a processors encounters a miss to a text page, the processor interface ASIC sees the text attribute in the read request. Upon receiving the read response to a text miss, the processor interface ASIC always loads the data in the shared state. This eliminates the exclusive state for text pages, and avoids future intervention requests to the text line, as that line will never enter the exclusive state.
- If two or more processors issue read requests for the same cache block, the POWERpath-2 bus protocol allows them to piggyback on the read response. This means that even though a single read request is issued to the bus and a single read response is provided, any number of processors may participate in the transaction by accepting the read response as their own and indicating that the cache block should be treated as shared. This feature is targeted at parallelizing compilers, and can significantly reduce read latency during synchronization since processors requesting the block after the first request is made need not wait for arbitration or memory access delays. Overall bus bandwidth is also conserved, since several requests are serviced by a single transaction.
- Each processor interface contains a special resource register accessible by the user. This register responds to broadcast increment transactions, which can be used to accelerate processor synchronization primitives. A join or barrier primitive, for example, can be implemented with high efficiency, as processors can communicate their progress through a critical section with high speed address bus transactions which avoid expensive cache block transfers. [4]

## 2.4 Tolerating high bus loads

In a large system, even the fastest memory bus approaches saturation under heavy loads. In these situations, it is important that the bus handle saturation gracefully. Performance should degrade in a linear fashion as opposed to exponentially, while forward progress must be guaranteed and starvation avoided.

Independent address and data busses, multiple outstanding variable latency reads, and a high-bandwidth memory system allow the POWERpath-2 bus to maintain its peak data transfer rate. There are also a number of special features designed to encourage graceful performance under heavy loading.

- The POWERpath-2 memory system supports interleaving on cache line addresses. Parallel access to SIMMs across a 576 bit wide DRAM bus makes it possible for a single, 2-way interleaved memory board to supply the full 1.2 GByte/sec bus bandwidth. This will only occur, however, when read addresses alternately target even and odd cache blocks. By adding additional memory boards, memory interleaving can be increased up to 8-way. Combining high interleaving with protocol tolerance for out of order read responses provides full bandwidth for almost any memory reference pattern. In addition to increasing memory interleaving, up to 16 gigabytes of memory can be added, and will perform at full speed even in the presence of single bit errors due to in line ECC correction.
- In order to prevent starvation and guarantee forward progress for all processes, each node is equipped with a programmable urgent timer. When a particular node is unable to issue a request for a specified amount of time, its bus arbiter automatically raises its priority to urgent. Once urgent, the node will have high priority access to bus resources, such as memory and I/O.
- When several nodes simultaneously request a wide range of data which resides in the cache of a single processor, it is possible for that processor to become swamped by external memory references. When this occurs, the processor is forced to NACK coherent bus requests until its input FIFOs drain sufficiently. To help relieve this degenerate case, any processor which is swamped will assert a "backoff" bus signal. When this signal is asserted, processors will refrain from issuing new coherent requests until either the signal is de-asserted or until their urgent timer expires. This feature relieves bus pressure when data hotspotting occurs in a single processor's cache.

## 3 I/O System Architecture

The Challenge POWERchannel-2 I/O system is designed for large, high-bandwidth I/O configurations, while at the same time supporting very primitive low performance devices at a low base cost. In order to support the base configuration, a narrower bus in the obvious solution. At 320 MBytes/sec, a single HIO bus can match one quarter of the system bus bandwidth to support a few high performance devices, while its narrower interface is cheap enough to connect low performance I/O as well at reasonable cost. Additional HIO busses can be added to the system to increase the I/O capacity.

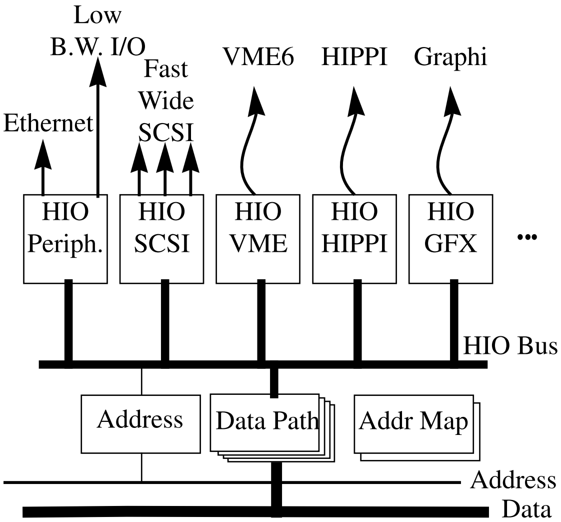



**Figure 6 I/O system architecture**

The HIO bus is a 64-bit multiplexed address/data bus running off the same clock as the system bus. It supports split read transactions, allowing up to four outstanding reads per device. All devices connect to the HIO bus through a personality interface ASIC. Up to seven personality interface ASICs can be attached to a single HIO bus.

### 3.1 I/O Bus Protocol

The HIO bus is narrower than the POWERpath-2, but still supports transfer sizes up to those supported by the system bus. Rather than require that every transaction handle a full cache line of data, the HIO bus supports several different transaction lengths. Separate arbitration lines and implicit flow control in the protocol eliminate most of the overhead of the multiplexed address data bus. The worst case sustainable situation is bus saturation by read traffic only. In this case the protocol overhead is four cycles out of twenty, delivering 320 MB/s bandwidth out of a possible 400 MB/s.

The POWERpath-2 system bus uses a distributed arbitration scheme to minimize the latency, whereas the I/O bus uses a centralized scheme. In the case of I/O, the centralized scheme was chosen because rather than being a peer bus, the I/O bus to system bus interface is a centralized location for decision making. Also, system latency is less of a problem in I/O as long as throughput can be maintained. Arbitration requests and grants are pipelined to allow full bus utilization so the centralized scheme will not impact throughput.

HIO interface chips can request the bus for a coherent DMA read or write to system memory using a 40-bit system address, make a request for address translation using the mapping resource in the system bus interface, or issue an interrupt or respond to a PIO read. The system bus interface returns DMA read responses and mapping responses and sends out PIOs

Coherency in the system goes as far as the HIO bus. When a DMA read makes it through the system bus interface it becomes a POWERpath-2 read just like one that a

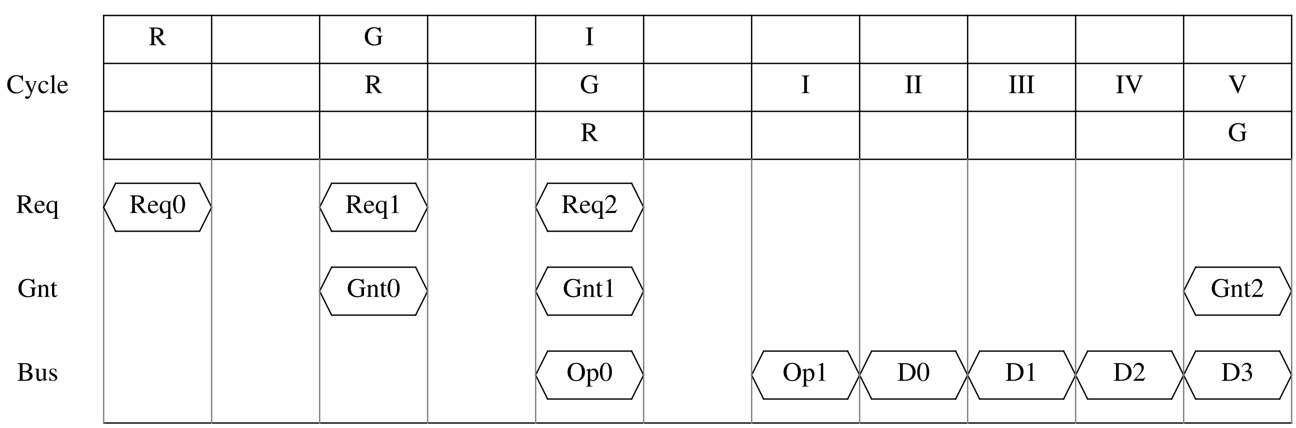



**Figure 7 generic HIO bus transaction (2 full transactions and a third req/gnt)**

processor would issue. When a DMA write goes out on the system bus it becomes a special block write transaction that invalidates copies in all cpu caches. Partial block DMA transfers are defined for the head and tail of transfers and for primitive low bandwidth devices. Partial writes must be merged coherently into main memory.

Pipelining the requests and grants with active transactions helps hide the minimum two clock latency from request to grant and the two clock latency from grant to transaction. Short transactions, which can be common in the case of DMA read, can lead to an inefficiency in bus utilization. If the central arbiter must wait for the transaction op (first cycle of the transaction) before knowing how long a transaction will hold the bus, then 1 cycle transaction will lead to 2 wasted bus cycles. If the bus were supplying its rated bandwidth consisting entirely of DMA reads and responses, then a 2 cycle overhead would amount to 10% performance loss. To solve this problem, bus requests encode information about the length of the requested transaction so the arbiter can grant the bus again in the minimum time.

## 3.2 Flow Control

The only transactions on the bus that need explicit flow control are the PIO transactions from the system bus interface to the HIO interface chips. All other transactions have implicit flow control because an HIO interface will not make a request if it doesn't have a buffer available for the response, and the centralized arbiter will not grant the bus if the system bus interface does not have room to accept the transaction. In other words, only PIOs can arrive unsolicited.

The flow control solution chosen is to make PIOs be solicited. After reset an HIO interface chip signals its available PIO buffer space by making a special request type, the IncPIO. The system bus interface maintains this information in one counter for each HIO device. Every time a PIO is sent, the count is decremented. When a PIO has been retired, the HIO device issues another IncPIO to increment the count. This helps reduce the latency of PIOs since they wait in the system bus interface for a minimum amount of time

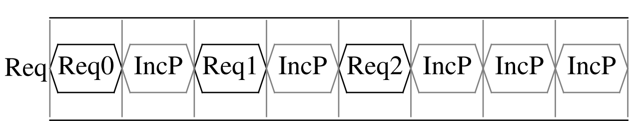



**Figure 8 IncPIOs can happen any time the REQ lines are not being used**

## 3.3 I/O Cache

A very simple fully associative, four line cache handles the system bus coherency of partial DMA writes from I/O which can have from 0 to 32 bytes of data. The cache is used only to keep cache lines within the system coherent space while the partial data is being merged. This increases system bus efficiency by reducing the number of times the system bus is used for less than a full block of data.

To keep the design and verification simple, a two state protocol was chosen. When a CPU reads a line in the I/O cache, the cache provides data and transitions to invalid. If a DMA read from the HIO bus hits in the cache, the data is written back to memory then read in the usual way. While

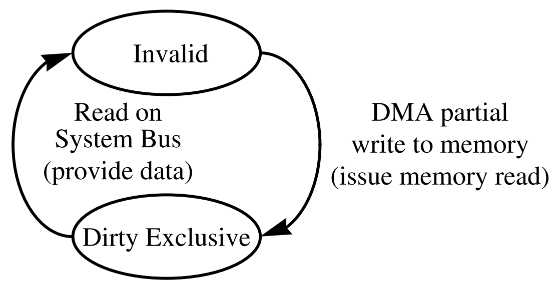



**Figure 9 I/O cache states**

reads could have used the cache to reduce the impact on system bus bandwidth, this case was not optimized since partial DMA operations are infrequent.

## 3.4 Address Translation

The map ram provides address general purpose address translation for I/O devices. This is used to map small address spaces such as VME24 or VME32 into the 40-bit system bus address space. In doing so it provides the capability to scatter/gather I/O virtual addresses into system physical addresses. Two types of mapping are implemented: one level and two level mapping. One level mappings simply return one of the 8k entries in the mapping ram. By convention, each map ram entry maps two megabytes of physical memory. In the two level scheme the map entry is a pointer to page tables in main memory. Each 4kB page has its own map entry so virtual pages can be arbitrarily mapped to physical pages. A single map entry controls 2 MB address space for two level maps as well. Note that PIOs face a similar mapping problem when sent to a VME bus. The VME interface chip handles this translation itself.

# 4 Design Verification

A primary goal any verification project is to define the conditions for completion. The nature of design verification is that verification will never quite be finished, but will asymptotically approach completion. To that end it is useful to develop some metrics to measure progress toward full coverage. Additionally, these metrics also provide insight and ideas into additional areas to be tested.

## 4.1 Completion Metrics

There were four different metrics that were used to gauge the progress of verification. The first is that every functional block be tested with all combination of timing relationships between inputs. This is confirmed with design reviews of both the ASIC code and the diagnostic coverage. The second is that all state to state transitions be covered in all state machines. Special tools were written to make these measurements during regression runs. Third, pseudo-random tests were run through many hours of system simulation. Finally, graphs of total bugs found vs. time should show a knee and plateau as verification completes.

The functional block coverage is a primary goal for any verification effort. The most successful tests were those that run for several iterations, putting the machine into a known state and varying a single parameter such as the timing between two opposing transactions. By sweeping the relative timing through the entire window of interaction you can have some confidence that you have covered all variations of that particular case.

Testing that all finite state machines have been exercised is an easy metric to gather, and has good correlation to diagnostic coverage. There are three levels of detail which can be implemented: 1) test that all states are visited; 2) test that all valid state to state transitions are made; 3) test that all valid state to state transitions are made and that all arcs have been covered with all input combinations. Challenge verification implemented the second strategy. While it would have been preferable to also include inputs in the test so that the exact arc taken could be tested, this required more integration work with our HDL and design tools than would be justified by its probable benefits.

Another orthogonal technique which is explained in greater detail in a later section is pseudo random tests with random interleaving. These tests were treated these as an insurance policy rather than a cure all. Directed tests were written to cover the known design space, while pseudo-random tests tried to find new areas not previously covered. After a new bug was found by pseudo-random testing, a whole set of directed tests could be defined.

Careful tracking of bugs can provide data showing asymptotic completion of verification. The end of this sections contains the final plots of bugs versus time. Several factors are used to interpret the plots. First, one would expect to find more bugs earlier in the project, and zero or virtually no bugs at the end. Secondly, there will occasionally be plateaus in the plot, suggesting that it is time to reevaluate the test strategy and possibly define new testing areas. This can give a stair step appearance to the plot. Finally, verification effort may not be constant vs. time, which might produce false lulls in the bug rate. If no new tests are written after bugs are no longer found, it is guaranteed that no more bugs will be found.

## 4.2 Pseudo-random vector generation

Simulating random transaction sequences is a useful method of generating unanticipated states, but it is often difficult to determine whether the results of a random transaction stream are correct. One way to solve this problem is to generate a series of pseudo-random transactions which expect a particular memory result upon completion. [5] By allowing several pseudo-random transaction streams to execute simultaneously and interact, unanticipated states and conflicts can be generated. This method is used to help verify coherence mechanisms as well as processor and memory interfaces. During simulation, a number of pseudo-random transaction generators inject system stimulus at processor interface points. Each pseudo-random stream or group of streams is allocated interleaved portions of the address space to avoid data interference. The address space interleaving is done in eight byte, cache line, and page boundaries for different regions to encourage false sharing as well as simulate traffic interaction patterns between typical unrelated processes. Upon completion, each transaction generator checks the addresses it wrote to ensure coherency was not lost. A passive bus watcher also observes traffic and watches for illegal bus states or transaction combinations, as well as gathering bus performance statistics.

Pseudo-random transaction generators are dispatched in groups of 1 or more, depending on how many processors the particular transaction generator expects. When a transaction generator completes execution on a single processor or group of processors, those processors are free to be issued additional transaction generators. A resource scheduling algorithm allows several groups of transaction generators execute simultaneously, and as individual processors are issued new transaction generators while other processors continue working with old ones, a large variety of system states and interactions are generated. Finally, a transaction frequency biasing matrix can be used to focus a particular simulation on certain types of transaction interactions.

## 4.3 Simulating OS loads via light weight threads

The pseudo-random transaction generation method is an efficient verification tool because it relies on a light weight CPU emulator, which is much faster and smaller than a full CPU model. This is also a pitfall of this technique, for although the CPU emulator will generate correct transactions, it will never exactly match the behavior of a full CPU model. To fill this gap in the design verification, simulations are also run with a full RTL model of the first target CPU, the MIPS R4400. By using a full CPU model for some of the simulations, special behaviors and corner cases associated with the processor implementation are uncovered and tested.

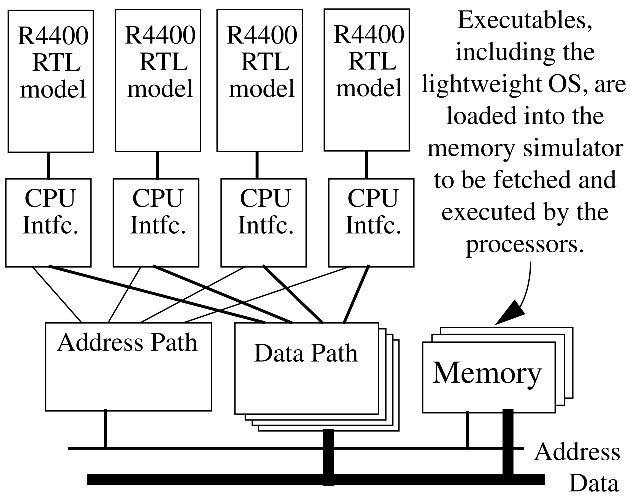
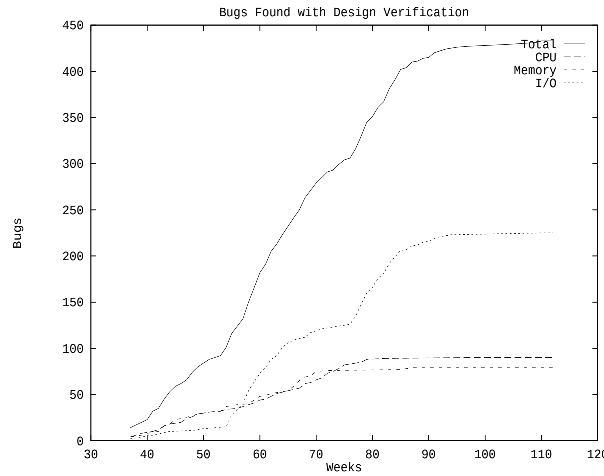



**Figure 10 System simulation using full RTL CPU models for design verification**

A multiprocessor simulation includes several full CPU models, which is a large burden on the simulation environment. With simulation performance hovering around 1 cycle per second, running multiprocessor UNIX is not feasible. To simulate the behavior of an operating system with the characteristics of parallel applications, a lightweight operating system to manage virtual memory and context switching is used. Once the lightweight kernel is running, parallel applications can fork, join, and request shared memory regions via low overhead system calls. This environment closely approximates system loading in a typical runtime environment, and is useful to fill the last holes in coherence design verification.

## 4.4 Effectiveness of design verification

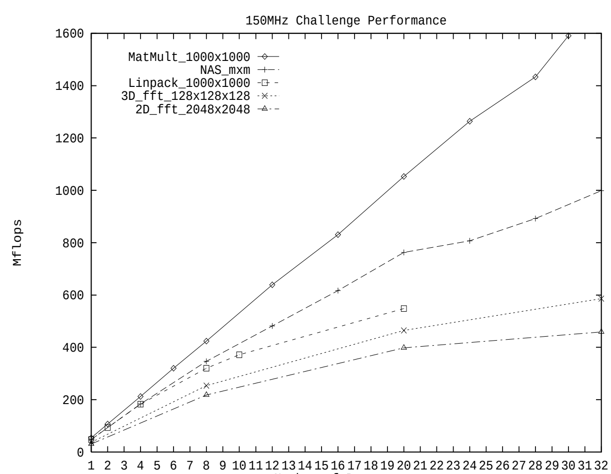
Design verification played a key role in the success of the Challenge project. Because the entire system is implemented in ASICs, each bug found after tape out means a costly chip respin. The high level of integration also poses difficulties in identifying bugs, as lab instruments only examine chip interfaces for short bursts of time. These obstacles, combined with the high complexity involved in implementing the coherence protocols and bus interfaces, require design verification to be as close to perfect as possible within a very tight schedule.



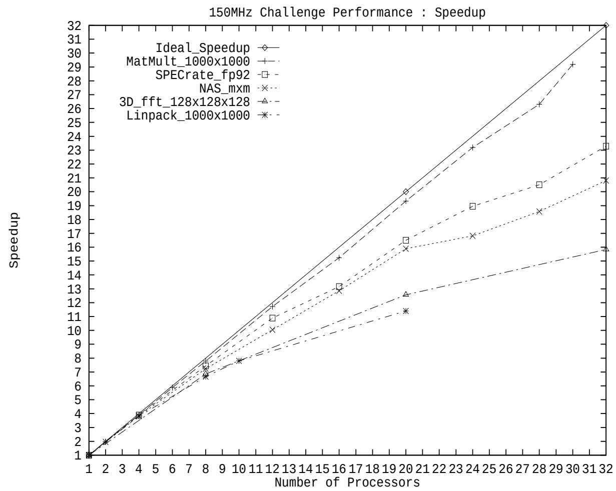

The Challenge systems first shipped with 12 separate ASIC designs averaging 80,000 gates each. A total of nine ASIC spins were required for system functionality. The total time which elapsed between the beginnings of the first design specification to systems being shipped to customers was 26 months. The short product cycle time realized by the Challenge system was largely the result of an aggressive and thorough design verification effort. Before each chip was taped out, the bug graph for that chip and subsystem had to level out in order to minimize the risk of silicon bugs.

## 5 Performance

Performance benchmarks presented in this paper were written in Fortran and parallelized with SGI's PFA parallelizing Fortran compiler. The single, coherent image of




system memory coupled with the symmetric multiprocessing programming paradigm produced excellent performance with a minimal amount of programming effort.




Results shown here were obtained using 150 MHz MIPS R4400s, the first line of processors available in Challenge. The system is designed to support several generations of future processors and I/O devices with straightforward upgrades, including the MIPS streaming superscalar TFP processor.

## 6 Conclusion

The effectiveness and scalability of a multiprocessor relies on its interconnect, while utility and cost effectiveness come from a balanced design. The SGI Challenge systems use a high speed shared bus to provide coherent, scalable connectivity between processors, memory, and I/O. High performance processors, interleaved memory, and a flexible high speed I/O bus are balanced building blocks to create a variety of system configurations. A RISC design philosophy and aggressive verification techniques brought design concept to product in 26 months, resulting in a system which not only benchmarks well, but also solves the needs of scientific, engineering, and other real applications.

## References

[1] Papamarcos, M., and Patel, J. "A Low Overhead Coherent Solution for Multiprocessors with Private Cache Memories". In *Proceedings of the 11th International Symposium on Computer Architecture*. IEEE, NY, 1984, pp. 348-354.

[2] Archibald, J., and Baer, J.L., "Cache Coherence Protocols: Evaluation Using Multiprocessor Simulation Model", *ACM Transactions on Computer Systems*, Vol 4, No. 4, Nov 1986, pp. 273-298.

[3] Goodman, J. R., "Using Cache Memory to Reduce Processor-Memory Traffic". In *Proceedings of the 10th International Symposium on Computer Architecture*. IEEE, NY, 1983, pp. 124-131.

[4] Goodman, J.R., Vernon, M.K., and Woest, P.J., "Efficient Synchronization Primitives for Large Scale Cache Coherent Multiprocessors". In *Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems*. IEEE CS Press, Los Alamitos, CA, 1989, pp. 64-73.

[5] Wood, D., Gobson, G., and Katz, R., "Verifying a Multiprocessor Cache Controller Using Random Test Generation," In *IEEE Design and Test of Computers*, August 1990, pp. 13-25.