# THE ALPHA 21364 NETWORK ARCHITECTURE

THE ALPHA 21364 PROCESSOR PROVIDES A HIGH-PERFORMANCE, SCALABLE, AND RELIABLE NETWORK ARCHITECTURE WITH A ROUTER THAT RUNS AT 1.2 GHZ AND HAS A PEAK BANDWIDTH OF 22.4 GBYTES/S. SUPPORTING CONFIGURATIONS OF UP TO 128 PROCESSORS, THIS NETWORK ARCHITECTURE IS WELL SUITED FOR COMMUNICATION-INTENSIVE SERVER APPLICATIONS.

Shubhendu S. Mukherjee
Peter Bannon
Steven Lang
Aaron Spink
David Webb
Compaq Computer Corp.

Advances in semiconductor technology have let microprocessors integrate more than a 100 million transistors on a single chip. The Alpha 21364 microprocessor[1,2] uses 152 million transistors to integrate an Alpha 21264 processor core, a 1.75-Mbyte second-level cache, cache coherence hardware, two memory controllers, and a multiprocessor router on a single die, as Figure 1a shows. In the 0.18-micron bulk CMOS process, the 21364 will run at 1.2 GHz and provide 12.8 Gbytes/s of local memory bandwidth and 22.4 Gbytes/s of router bandwidth, as Figure 2 shows.

The Alpha 21364's tightly coupled multiprocessor network connects up to 128 such processors in a 2D torus network; Figure 1b shows a 12-processor configuration. A fully configured, 128-processor, shared-memory system can support up to 4 terabytes of Rambus memory and hundreds of terabytes of disk storage. We could also easily redesign the 21364 to support a much larger configuration.

This multiprocessor configuration supports the massive computation and communication requirements of various application domains, such as high-performance technical computing, database servers, Web servers, and telecommunications. We designed the Alpha 21364 network architecture to meet the communication demands of these memory- and I/O-intensive applications.

The novelty of the Alpha 21364's router architecture lies in its extremely low latency, enormous bandwidth, and support for directory-based cache coherence. The router offers extremely low latency because it operates at 1.2 GHz, the same clock speed as the processor core. The pin-to-pin latency within the router is 13 cycles or 10.8 ns. In comparison, the ASIC-based SGI Spider router runs at 100 MHz and offers a 40-ns pin-to-pin latency.[3]

Similarly, the Alpha 21364 offers an enormous amount of peak and sustained bandwidth. The 21364 router can sustain between 70 and 90 percent of its 22.4-Gbytes/s peak bandwidth. The 21364's router can offer such enormous bandwidth because of aggressive routing algorithms, carefully crafted distributed arbitration schemes, large amounts of on-chip buffering, and a fully pipelined router implementation.

Finally, the network and router architectures have explicit support for directory-based cache coherence, such as separate virtual channels for different coherence protocol packet classes. This helps avoid deadlocks and improves the performance of the 21364's coherence protocol.



0272-1732/02/$17.00 © 2002 IEEE

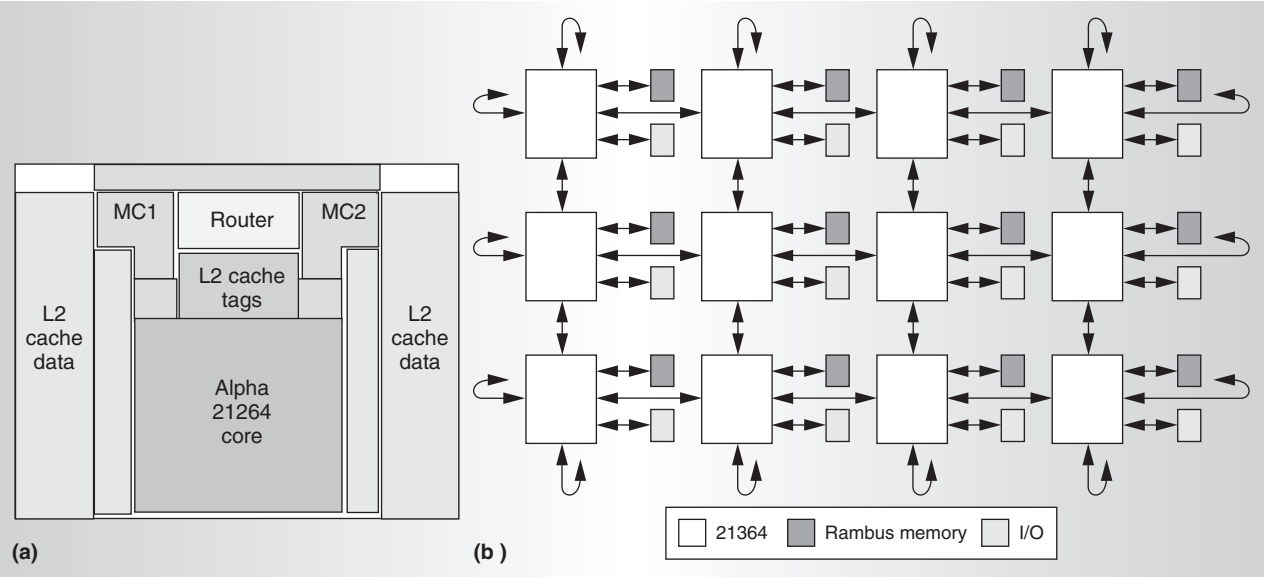


Figure 1. Alpha 21364 floor plan (a) and a 12-processor configuration of Alpha 21364s (b).

## Network packet classes

Network packets and flits are the basic units of data transfer in the 21364's network. A packet is a message transported across the network from one router to another; it consists of one or more flits. A flit is a portion of a packet transported in parallel on a single clock edge. A flit is 39 bits—32 bits for payload, 7 bits for per-flit error correction code (ECC). Thus, each of the incoming and outgoing interprocessor ports shown in Figure 1b is 39 bits wide. The 21364 network supports packet sizes of one, two, three, 18, and 19 flits. A packet's first one to three flits contain the packet header. Additionally, the 18- or 19-flit packets typically contain 64-byte (or 16-flit) cache blocks or up to 64 bytes of I/O data.

The 21364 network supports seven packet classes:

- *Request (three flits).* A processor or I/O device uses a request packet to obtain data in and/or ownership of a cache block or up to 64 bytes of I/O data.
- *Forward (three flits).* A memory controller (MC1 or MC2 in Figure 1a) uses a forward packet to forward a request packet to a cache block's current owner or sharer (a processor or I/O device).
- *Block response (18 or 19 flits).* A processor or I/O device uses a block response packet to return the data requested by a request class packet or to send a modified cache block back to memory.
- *Nonblock response (two or three flits).* A processor, memory controller, or I/O device uses a nonblock response packet to acknowledge coherence protocol actions, such as a request for a cache block.
- *Write I/O (19 flits).* A processor or I/O device generates a write I/O packet when

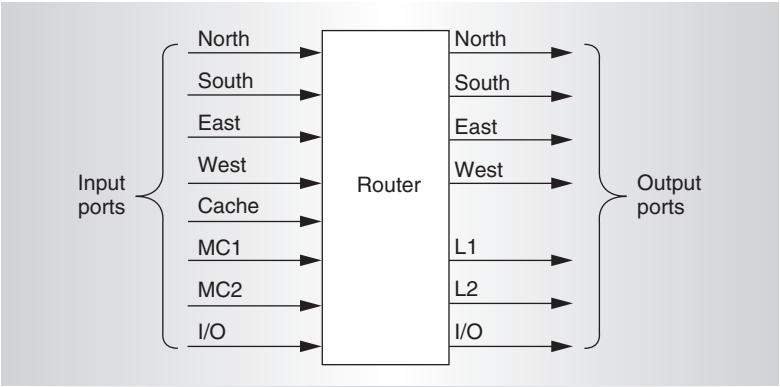


Figure 2. Router ports. This figure shows the eight input ports and the seven output ports of the 21364's router. The north, south, east, and west interprocessor ports correspond to off-chip connections to the 2D torus network. MC1 and MC2 are the two on-chip memory controllers, shown in Figure 1a. The cache input port corresponds to the on-chip second-level cache. The L1 output port connects to the first level cache as well as MC1. Similarly, the L2 output port connects to the first level cache and MC2. Finally, the I/O ports connect to the I/O chip external to the 21364 processor. The total aggregate bandwidth of the seven output ports is 22.4 Gbytes/s.

it stores data to I/O space.
- *Read I/O (three flits).* A processor generates read I/O packets to load data from I/O space.
- *Special (one or three flits).* The network and the coherence protocol use these packets. The special class includes no-op packets, which can carry buffer deallocation information between routers.

The packet header (one to three flits) identifies the packet's class and function. The header also contains routing information for the packet and (optionally) the physical address of the cache block or data block in I/O space that corresponds to this packet. It can also carry flow control information between neighboring routers. Besides the header, block response and write I/O packets also contain 16 flits or 64 bytes of data.

The 21364's coherence protocol and I/O devices use these packet classes to communicate between processors, memory, and I/O devices. Bannon et al. describes the 21364's directory-based coherence protocol.[4]

## Network architecture

The 21364 network is a 2D torus, but it can also support limited configurations of imperfect tori, which lets the network take faulty routers out of its topology. To implement the 2D torus, we carefully designed the routing mechanisms and the corresponding deadlock avoidance techniques.

### Virtual cut-through routing

The 21364 uses virtual cut-through routing in which flits of a packet proceed through multiple routers until a router blocks the header flit. Then, the blocking router buffers all the packet's flits until the congestion clears. Subsequently, the blocking router schedules the packet for delivery to the next router, and the same pattern repeats. To support virtual cut-through routing, the 21364's router provides buffer space for 316 packets.

### Adaptive routing

The 21364's network uses adaptive routing to maximize the sustained bandwidth. However, the adaptive-routing algorithm is very simple, enabling a simpler implementation of the arbitration scheme compared to that of more elaborate, fully adaptive-routing algorithms. In the 21364 scheme, packets adaptively route within the minimum rectangle. That is, given two points in a torus (in this case, the current router and the destination processor), you can draw four rectangles that contain these two points as their diagonally opposite vertices, as Figure 3a shows. The minimum rectangle is the one with the minimum diagonal distance between the current router and destination processor.

The adaptive-routing algorithm picks one output port among a maximum of two output ports that a packet can route in at any router. Thus, each packet at the current router's input port and destined for a network output port has only two choices: It can either continue in the same dimension (such as north input to south output) or turn (say from north input to east output). This is because with every hop a packet will reduce its Manhattan distance to its destination. This shrinks the size of the minimum rectangle in which the packet is routed.

If the adaptive algorithm has a choice between both the available network output ports (that is, neither output port is congested), then it gives preference to the route that continues in the same dimension. This lets a source-destination pair of processors maximize the bandwidth between them by letting multiple packets travel on separate routes. Figures 3b and 3c show the implementation of these preferences.

### Deadlock avoidance rules

Both coherence and adaptive-routing protocols can introduce deadlocks in a network because of cyclic dependences created by these protocols.

*Avoiding deadlocks in the coherence protocol.* The coherence protocol can introduce deadlocks because of cyclic dependence between different packet classes. For example, request packets can fill up a network and prevent block response packets from ever reaching their destinations. The 21364 breaks this cyclic dependence by creating virtual channels[5] for each class of coherence packets and assigning (by design) an ordering constraint among these classes. By creating separate virtual channels for each class of packet, the 21364's router guarantees that it can route

each class independent of other classes. Thus, a request packet can never block a block response packet. The constraint orders the packet types as follows, least dependent classes first: read I/O, write I/O, request, forward, special, nonblock response, and block response. Thus, a request can generate a block response, but a block response cannot generate a request.

Additionally, the 21364 takes three measures for I/O traffic. First, the router uses only deadlock-free virtual channels to force I/O requests of the same class to follow the same route and thereby arrive in order. Second, the router has separate virtual channels for I/O writes and reads (write I/O and read I/O), which makes the implementation simpler because I/O write and I/O read requests have different packet sizes. Finally, I/O reads at a router flush all prior outstanding I/O writes to preserve the I/O ordering rules.

*Avoiding deadlocks in adaptive routing.* Adaptive routing can generate two types of deadlocks: intra- and interdimension. Figures 4a and 4b show examples of these two types of deadlocks. The 21364 breaks these two deadlocks using Jose Duato's theory, which states that adaptive routing will not deadlock a network as long as packets can drain via a deadlock-free path.[6]

In these figures, each arc represents a packet; the labels indicate the packet's source and destination processor. Figure 4a shows a potential deadlock within a dimension (that is, processors 0 to 3 are in the same dimension) of the 2D torus network. The network is deadlocked because each packet is waiting for a buffer in the forward path to free up. Figure 4b shows a potential deadlock across dimensions (with processors 0 and 1, 1 and 2, 2 and 3, and 3 and 0 in different dimensions). This deadlock arises because each packet is waiting for buffers in the next dimension to free up.

Figure 4c shows how the 21364 breaks the intradimension deadlock by dividing up the buffers into virtual channels VC0 and VC1. Dividing the buffers into VC0 and VC1 breaks the cyclic dependence between the packets. Figure 4d shows how the 21364 breaks the interdimension deadlock. In the VC0 and VC1 channels, each packet first routes along the primary axis (horizontally) and then routes along the secondary axis (vertically). When packets

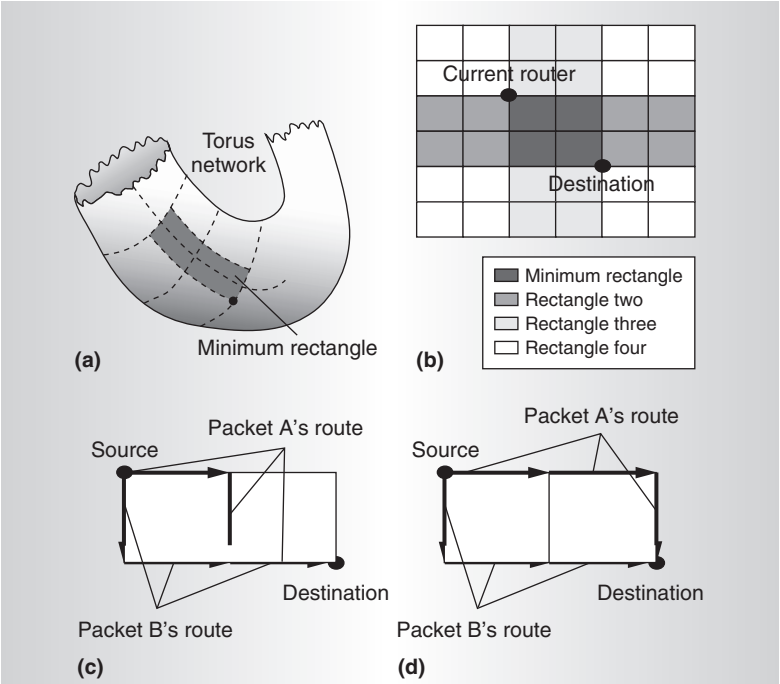


Figure 3. Adaptive routing in a torus network (a) must find the minimum rectangle (b). A preference to turn (c)—for packet A's route—makes the last hop a bottleneck link. A preference to continue straight in the same dimension (b) maximizes bandwidth between a source and destination. Heavy lines are potential routes; lightweight lines are network links.

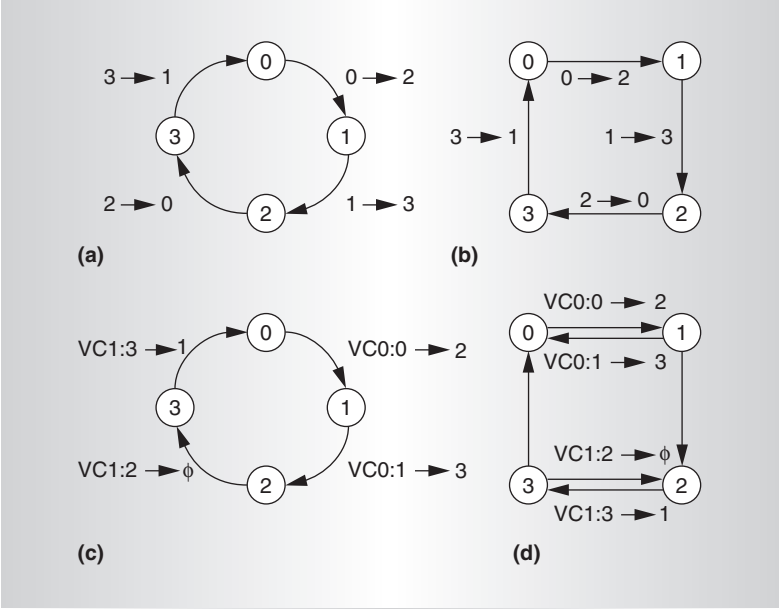


Figure 4. Potential intradimension (a) and interdimension (b) deadlocks. The 21364 breaks the intradimension deadlock by dividing up the buffers into virtual channels VC0 and VC1 (c). It goes through a similar process to break the interdimension deadlock (d).

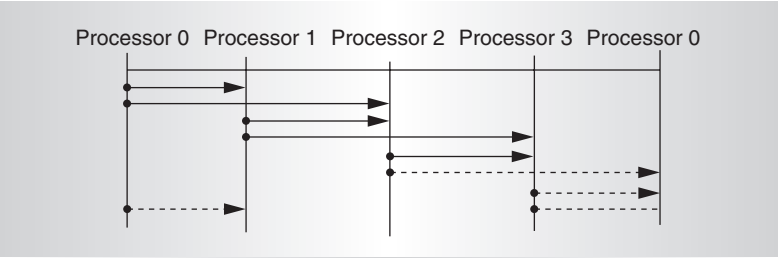


Figure 5. VC0 and VC1 assignments within a dimension containing four processors. The solid lines represent VC0s and dashed lines represent VC1s. Because the 21364 routes only within the minimum rectangle, an arrow does not span more than two hops in a dimension with four processors. We base the assignments in this figure on Dally's scheme. For the consecutive physical links, the VC0 to VC1 ratios are 2:1, 3:0, 2:1, and 0:3.

change dimensions, they recompute their virtual channels in the new dimension. Thus, packets in VC0 and VC1 along the primary axis depend on packets in the secondary axis, but packets in the secondary axis do not depend on packets in the primary axis. This scheme avoids the cyclic dependence across dimensions and removes the interdimension deadlock.

The 21364 creates logically distinct adaptive and deadlock-free networks using virtual channels. Each of the virtual channels corresponding to a particular packet class—except the one corresponding to the special class—is further subdivided into three sets of virtual channels: adaptive, VC0, and VC1. Thus, the 21364 has a total of 19 virtual channels (three each for the six nonspecial classes and one for the special class).

The adaptive virtual channels form the adaptive network and have the bulk of a router's buffers associated with them. The VC0 and VC1 combination creates a deadlock-free network, which provides a guaranteed deadlock-free path from any source to any destination within the network. Thus, packets blocked in the adaptive channel can drain via VC0 and VC1.

The VC0 and VC1 virtual channels must carefully map onto the physical links to create the deadlock-free network. The 21364 has separate rules to break deadlocks within a dimension and across dimensions. Within a dimension, the 21364 maps the VC0s and VC1s so there is at least one processor not crossed by dependence chains formed by VC0s. The same applies to VC1 mappings. This ensures that there is no cyclic dependence in a virtual channel within a dimension.

The 21364 can choose among a variety of such virtual-channel mappings because virtual-channel assignments are programmable at boot time. Perhaps the simplest scheme that satisfies the mapping requirement just described comes from Dally;[5] this scheme calls for incrementally numbering all processors in a dimension. Then, for all source and destination processors, we can make the following virtual-channel assignments: If the source's number is less than the destination's, assign that source-destination pair VC0. If the source's number is greater than the destination's, assign that pair VC1.

Unfortunately, in this scheme, the assignments of virtual channels to physical links are not well balanced, as Figure 5 shows. Such a situation can cause underutilization of network link bandwidth under heavy load. In the 21364, we search for an optimal virtual-channel-to-physical-link assignment using a hill-climbing algorithm. This scheme does not incur any overhead, because we run the algorithm offline and only once for a dimension with a specific size (ranging from two to 16 processors).

Figure 4d shows how the 21364 breaks interdimension deadlocks by designating one of the two directions as the primary axis and the other as the secondary axis. A packet can always proceed along the primary axis in VC0, VC1, or adaptive channels. However, along the secondary axis, a packet can proceed only via the adaptive channel, unless the secondary axis row or column (in which the packet is routing) also contains the destination 21364. If the secondary axis contains the destination 21364, then the packet can route in VC0, VC1, or adaptive channels.

The 21364's deadlock-avoidance rules, however, do not prevent a packet in VC0 or VC1 from returning to the adaptive channel. Thus, a packet blocked in the adaptive channel can drop down to VC0 or VC1. However, in subsequent routers along the packet's path, the packet can return to the adaptive channel, if the adaptive channel is not congested. This works for the 21364 because virtual cut-through routing stores entire packets at a router, even though packets that are not blocked can span multiple routers at the same time. The 21364's ability to buffer an entire packet at a router removes dependence between consecutive routers, which lets packets move from deadlock-free VC0 and

VC1 channels to the adaptive channel. Additionally, a 21364's choice of direction and virtual channel are independent of a packet's prior route in the network, which helps remove cyclic dependences among routers.

## Router architecture

The 21364's router has nine pipeline types, based on the input and output ports. There are three types of input and output ports: local (cache and memory controller), interprocessor (off-chip network), and I/O. Any type of input port can route packets to any type of output port, leading to nine types of pipelines. Figure 6 shows two such pipeline types.

In addition to the pipeline latency, there is a total of six cycles of delay, which includes synchronization delay; pad receiver and driver delay; and transport delay from the pins to the router and from the router back to the pins. Thus, the on-chip pin-to-pin latency from a network input to a network output is 13 cycles. At 1.2 GHz, this leads to a pin-to-pin latency of 10.8 ns.

The network links that connect the different 21364 chips run at 0.8 GHz, 33 percent slower than the internal router clock. The 21364 chip runs synchronously with the outgoing links, but asynchronously with the incoming links. The 21364 sends its clock with the packet along the outgoing links. Such clock forwarding provides rapid transport of bits between connected 21364 chips and minimizes synchronization time between them.

### Router table lookup and decode stages

The 21364's router table consists of three parts:

- a 128-entry configuration table with one entry for each destination processor in the network;
- a virtual channel table consisting of two 16-bit vectors, which contain the deadlock-free virtual channel assignments; and
- an additional table to support broadcasting invalidations to clusters of processors (as required by 21364's coherence protocol[4]).

As in the SGI Spider switch, software programs the 21364 router table at boot time.[3]

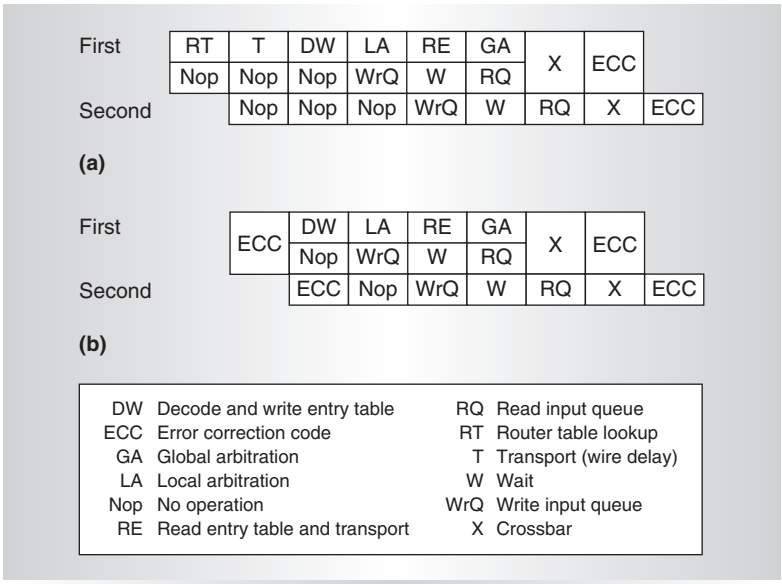


Figure 6. Two of the nine logical router pipelines in the 21364: the router pipeline for a local input port (cache or memory controller) to an interprocessor output port (a) and the router pipeline from an interprocessor (north, south, east, or west) input port to an interprocessor output port (b). The first flit goes through two pipelines, one for scheduling (upper pipeline) and another for data (lower pipeline). Second and subsequent flits follow the data pipeline.

This capability gives software the flexibility to optimize the desired routing for maximal performance and to map out faulty nodes in the network.

The first flit of a packet entering from a local or I/O port accesses the configuration table and sets up most of the 16-bit routing information in a packet's header. These 16 bits include

- two bits for the east-west and north-south directions (positive or negative),
- eight bits for the destination coordinates along the two dimensions,
- one bit (used by incomplete torus networks) to indicate if the packet can be routed in the adaptive channel,
- one bit to indicate if the packet is an I/O packet,
- two bits to encode the virtual channel number (adaptive, VC0, and VC1), and
- two reserved bits.

Each configuration table entry contains 24 bits that include the header's routing information (except the two bits that encode the virtual-channel number), six access control bits, three bits to encode routing information

**Table 1. No. of input buffers in the 21364 router.**

| | Interprocessor | | | | |
|---|---|---|---|---|---|
| **Class** | **Adaptive** | **VC0 or VC1** | **Cache** | **Memory controller** | **I/O** |
| Request | 8 | 1 | 8 | 0 | 8 |
| Forward | 8 | 1 | 0 | 8 | 0 |
| Block response | 3 | 1 | 6 | 4 | 5 |
| Nonblock response | 8 | 1 | 8 | 9 | 9 |
| Write I/O | 1 | 2 | 4 | 0 | 2 |
| Read I/O | 1 | 2 | 4 | 0 | 2 |
| Special | 8 | 0 | 0 | 0 | 0 |

for incomplete tori networks (with mapped-out nodes), and one parity bit.

The decode stage identifies the packet class, determines the virtual channel (by accessing the virtual channel table), computes the output port, and figures out the deadlock-free direction. The decode phase also prepares the packet for subsequent operations in the pipeline.

### Error correction code manipulation

Each 32-bit flit of a 21364 network packet is protected by 7-bit ECC. The router checks ECC for every flit of a packet arriving through an interprocessor or I/O port. The router regenerates ECC for every flit of a packet leaving through an interprocessor or I/O output port, as Figure 6 shows. ECC regeneration is particularly necessary for the packet's first flit because the router pipeline can modify the header before forwarding the packet.

If the router pipeline detects a single-bit error, it corrects the error and reports it back to the operating system via an interrupt; it does not correct double-bit errors. Instead, the 21364 alerts every reachable 21364 to the occurrence of such an error and enters an error recovery mode.

### Input buffering

The 21364 router provides buffering only at each of the input ports. Table 1 shows the distribution of input buffers at each input port. Each input buffer can hold a complete packet of the specific packet class, except for local ports—one cache, two memory controllers, and one I/O—for which packet payloads reside in the 21364's internal buffers. The design subdivides interprocessor ports into adaptive and deadlock-free channels, whereas the local ports have a single monolithic buffer space and do not need virtual channels. The write I/O and read I/O classes use a special adaptive channel for their first hop in a faulty network. Subsequent hops for write I/O and read I/O traffic are always in order and strictly follow the VC0 and VC1 channels. There are also six other special buffers, not shown in the table. The router has a total of 316 packet buffers.

Each input port has an entry table that holds the in-flight status for each packet and input buffers that hold the packets. A packet's first flit writes the corresponding entry table entry during the decode and write stage shown in Figure 6. An entry table entry contains bits for

- validation (this is a single bit);
- the target output ports;
- indicating whether the packet can adapt and/or route in the adaptive channels;
- supporting the antistarvation algorithm, which detects starved packets and drains them via the output ports (in rare cases, the distributed and speculative nature of 21364's arbitration mechanism can cause starvation of packets residing at the input buffers); and
- other miscellaneous information.

Readiness tests use this information in the local arbitration phase and read it in the read-entry-table-and-transport phase to determine the routing path of each packet. The 21364 router writes flits to and reads flits from the packet buffers in the write input queue and read input queue stages after the scheduling pipeline has made the routing decision for the first flit.

Either the previous 21364 router in a packet's path or the cache, memory controller, or I/O chip where the packet originated controls

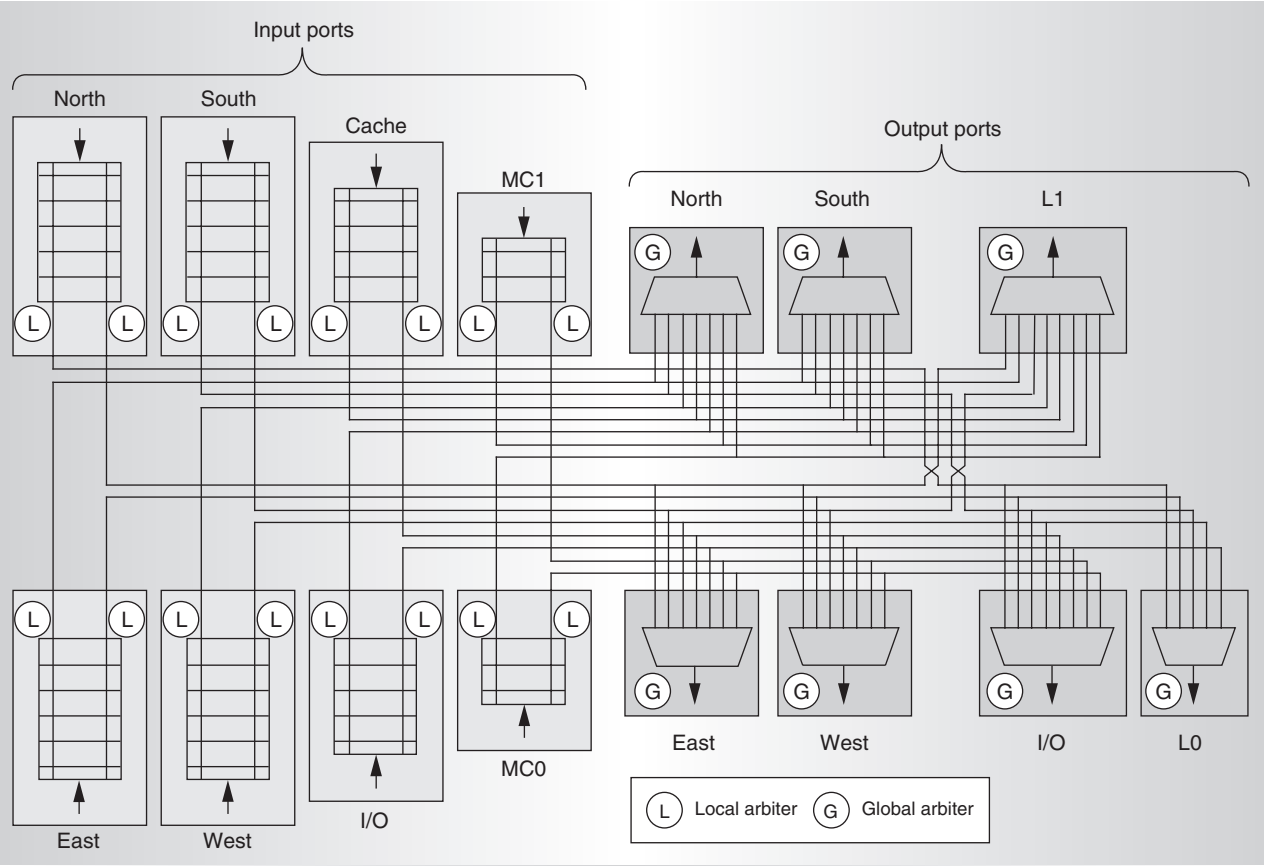


Figure 7. Connections between local and global arbiters.

the allocation of input buffers at a router. Thus, each resource delivering a packet to the router knows the number of occupied packet buffers in the next hop. When a router deallocates a packet buffer, it sends the deallocation information to the previous router or I/O chip via no-op packets or by piggybacking the deallocation information on packets routed for the previous router or I/O port.

## Arbitration

The 21364 router's most challenging component is the arbitration mechanism that schedules the dispatch of packets arriving at its input ports. To avoid making the arbitration mechanism a central bottleneck, the 21364 breaks the arbitration logic into local and global arbitration, as Figure 7 shows. There are 16 local arbiters, two for each input port. There are seven global arbiters, one for each output port. In each cycle, a local arbiter may speculatively schedule a packet for dispatch to an output port. One cycle after the completion of the local arbitration (as Figure 6 shows), each global arbiter selects one out of up to seven packets speculatively scheduled for dispatch through the output port. Once the global arbitration makes such a selection, all flits in the crossbar stage follow the input port to the output port connection.

Local arbiters perform various readiness tests to determine if a packet can be speculatively scheduled for dispatch via the router. These tests ensure that

- the nominated packet is valid at the input buffer and has not yet been dispatched,
- the necessary dispatch path from the input buffer to the output port is free,
- the router dispatches the packet in only one of the routes allowed,
- the target router, I/O chip, or local resource (in the next hop) has a free input buffer in the specific virtual channel,
- the target output port is free,
- the antistarvation mechanism is not

blocking the packet, and
- a read I/O packet does not pass a write I/O packet.

A global arbiter selects packets speculatively scheduled for dispatch through its output port. In subsequent cycles, local arbiters speculatively schedule packets not selected by any global arbiter.

To ensure fairness, the local and global arbiters use a least recently selected scheme to select a packet. Each local arbiter uses the LRS scheme to select both a class (from among the several packet classes) and a virtual channel (VC0, VC1, or adaptive) within the class. Similarly, the global arbiter uses the LRS policy to select an input port (from among the several input ports that each output port can use).

Additionally, the 21364 provides two special modes—rotary rule and coherence dependence priority (CDP) rule—which prioritize packets according to their packet class and the input port they arrive from. The rotary rule gives priority to packets arriving from an interprocessor port; doing so lets older packets residing in the network move sooner than younger packets generated from the local or I/O ports. The CDP rule prioritizes the packets according to their class ordering, discussed earlier. Thus, the CDP rule gives block response packets priority over request packets.

The rotary and CDP rules let the 21364 network avoid network saturation. However, these rules or heuristics are not always successful in the more general case for networks that accept significantly greater number of requests and have significantly greater number of buffers in the routers compared to the 21364 router. Thottethodi, Lebeck, and Mukherjee demonstrated that a network could avoid saturation in a more general way by using the network's global knowledge.[7]

With the advent of systems-on-a-chip in the mainline processor market, we expect 21364-like on-chip networks to become more common. However, the router pipeline will face challenges similar to that faced by a microprocessor pipeline today. Specifically, architects will have to carefully design the data and control paths to avoid wire delays on long global wires. Additionally, as the technology shrinks, architects will need to scale the arbitration, saturation, and routing algorithms to match the technology. MICRO

## Acknowledgments

Many engineers have made the Alpha 21364 network architecture possible. Richard Kessler was one of the most important contributors toward the design of the 21364's network architecture. Jim Burnette provided valuable feedback during the design's initial stages. Zarka Cvetanovic and Simon Steely provided valuable performance simulation and feedback. Joel Emer, Keith Farkas, Geoff Lowney, Paul Rubinfeld, Simon Steely, and David Wood provided helpful feedback on different drafts of this article.

## References

1. P. Bannon, "Alpha 21364: A Scalable Single-Chip SMP," *Eleventh Ann. Microprocessor Forum*, MicroDesign Resources, Sebastopol, Calif., 1998.
2. A. Jain et al., "A 1.2 GHz Alpha Microprocessor with 44.8 GB/sec of Chip Pin Bandwidth," *2001 IEEE Int'l Solid-State Circuits Conf.* (ISSCC 01), IEEE Press, Piscataway, N.J., 2001, p. 240.
3. M. Galles, "Spider: A High-Speed Network Interconnect," *IEEE Micro*, vol. 17, no. 1, Jan.-Feb. 1997, pp. 34-39.
4. P. Bannon et al., "Alpha 21364: A Single-Chip Shared Memory Multiprocessor," *Government Microcircuits Applications Conf. 2001 Digest of Papers* (Gomac), 2001, Defense Technical Information Center, Belvoir, Va., pp 334-337; http://www.dtic.mil.
5. W.J. Dally, "Virtual Channel Flow Control," *17th Ann. Int'l Symp. Computer Architecture* (ISCA 90), IEEE CS Press, Los Alamitos, Calif., 1990, pp. 60-68.
6. J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 12, Dec. 1993, pp. 1320-1331.
7. M. Thottethodi, A. Lebeck, and S.S. Mukherjee, "Self-Tuned Congestion Control for Multiprocessor Networks," *Proc. Seventh Int'l Conf. High-Performance Computer Architecture* (HPCA), IEEE CS Press, Los Alamitos, Calif., 2001, pp. 107-120.

**Shubhendu S. Mukherjee** is a senior hardware engineer in Intel's VSSAD group and partic-

ipated in the Alpha 21364 development in a previous position at Compaq Computer Corp. His research interests include multiprocessor networks, fault-tolerant processors, and performance modeling. Mukherjee has a PhD in computer science and engineering from the University of Wisconsin-Madison. He is a member of the IEEE and ACM.

**Peter Bannon** is a principal member of the technical staff in the Alpha Development Group at Compaq Computer Corp and currently coarchitect of the Alpha 21364 design. His previous experience includes the design and verification of several microprocessor chips, including the Alpha 21164 and Alpha 21164PC. Bannon has a BS in computer system design from the University of Massachusetts.

**Steve Lang** is an IC design engineer at Compaq Computer Corp. His previous experience includes work on the architecture and implementation of microprocessors, core-logic chipsets, and peripheral chips. Lang has an SB in electrical engineering and an SM in electrical engineering and computer science, both from MIT.

**Aaron Spink** is a senior hardware engineer at Intel, currently involved in the definition and design of future system interfaces for the Itanium processor family. He participated in Alpha 21364 development in a previous position at Compaq Computer Corp. His previous experience includes work on StrongARM microprocessors for Digital Equipment Corp. Spink has a BSE in electrical engineering from the University of Michigan.

**David Webb** is a senior member of the technical staff at Compaq Computer Corp. and the architecture lead for the Alpha 21364 router. His previous experience includes work on the VAX 9000 and the 21264 memory system, for which he was the lead architect. Webb has a BS in electrical engineering from the Worcester Polytechnic Institute.

Direct questions and comments to Shubhendu S. Mukherjee, 334 South Street, SHR1-T25, Shrewsbury, MA 01545; shubu.mukherjee@intel.com.