In *Proceedings of VLSI '93*, IFIP, 1993, pp. 311–320.

The Chaos Router Chip: Design and Implementation of an Adaptive Router

Kevin Bolding, Sen-Ching Cheung, Sung-Eun Choi, Carl Ebeling, Soha Hassoun, Ton Anh Ngo and Robert Wille

Department of Computer Science and Engineering, University of Washington, Seattle, Washington, USA, 98195

Chaotic routers are randomizing, non-minimal adaptive packet routers designed for use in the communication networks of parallel computers. Although adaptive routing, and, specifically, chaotic routing, has been shown to be superior to oblivious routing in most cases, the practical application of adaptive routing to multi-computer networks has been difficult to achieve due to the complex nature of adaptive routers. A prototype two-dimensional (mesh and torus) chaotic router chip has been designed and is being fabricated in a 1.2μm CMOS process. The chip exhibits high bandwidth, limited only by the speed of the off-chip drivers, and low input-to-input latency. To achieve this, much attention is given to reducing the critical path complexity of the router. The resulting chip is shown to be as good or better than state-of-the-art oblivious routers in almost all cases.

Keyword Codes: B.4.3; B.7.0; C.2.1
Keywords: Interconnections; Integrated Circuits; Network Architecture and Design

1. INTRODUCTION

Chaotic routing is a randomizing, adaptive, packet routing technique that has previously been shown (in simulations) to be effective for the binary n-cube (hypercube) [1] and two-dimensional torus topologies [2]. The technique is *nonminimal*, so packets do not necessarily take minimal-length paths to their destinations, and randomization plays a critical role in preventing packets from continually circulating in the network without being delivered (*livelock*) [3]. Though the principles apply as well to batched message communication, chaotic routing assumes a continuous workload where messages are presented at the nodes for injection into the network at random real (or fine-grain discrete) times. Routing decisions are made locally in the routing nodes based on the destination address stored in the headers of the packets and the availability of outgoing channels. Packets can "cut-through" nodes if an outgoing channel is immediately available, but they may also be stored in the node if all outgoing channels are blocked, motivating short, e.g. 40 byte, packets.

Although simulation studies have shown chaotic routing to have equal low-load performance to oblivious routing and superior high-load performance, the practicality of building such a router has been questioned due to the router's complexity. Specifically, state-of-the-art routers are of a simple, oblivious design [4]. The simplicity of oblivious routers allows for a high bandwidth, low latency design. On the other hand, adaptive

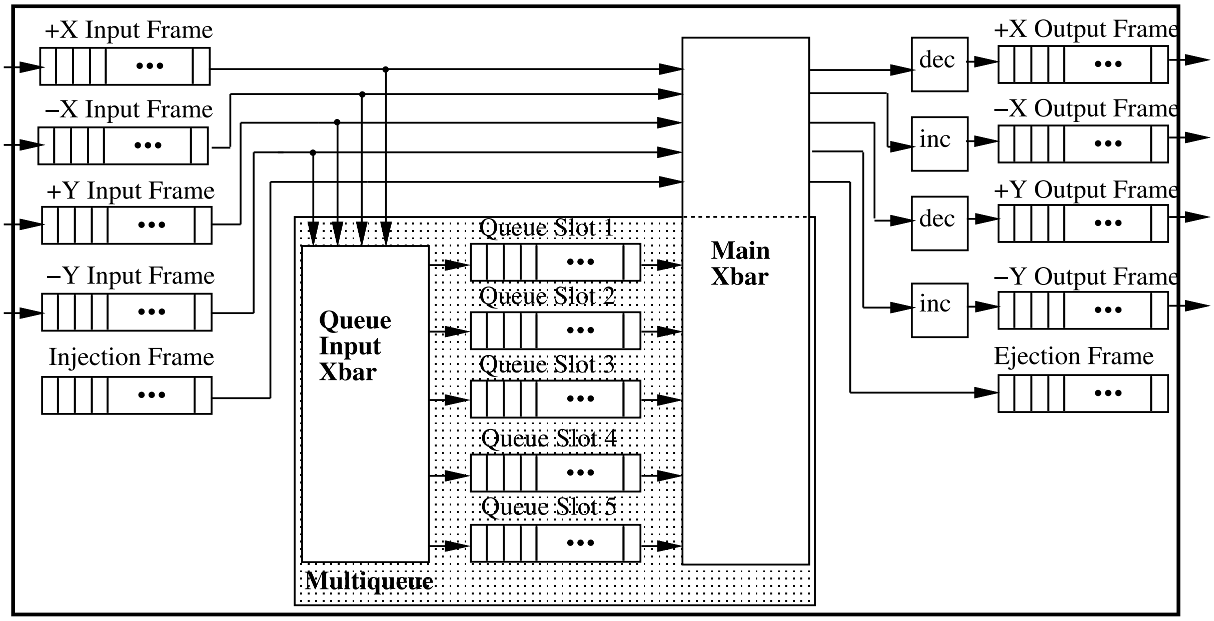

Figure 1. Two-dimensional Chaos router datapath diagram.

routers must make more complex decisions and may be slower and bigger. However, using pipelining techniques and careful design, it should be possible to build a competitive chaos router. To answer this question, we have built a prototype chaos router chip designed to go "head-to-head" with state-of-the-art oblivious routers using similar technology.

2. CHAOTIC ROUTING

The chaotic router implemented in this work is a two-dimensional router which is suitable for mesh or torus-connected networks [2]. The chaos router has three main components: input/output *frames*, a crossbar, and the *multiqueue* (see Figure 1). Each frame and multiqueue slot is capable of holding one fixed-size (20-flit) packet. The basic operation of the chaos router is similar to a typical oblivious cut-through packet router [5]: packets enter the router into an input frame, are connected through the crossbar to an output frame and have their header updated on the way out to reflect their progress. Virtual cut-through allows packets to proceed through the router as soon as their header is received and decoded; it is not necessary for the entire packet to be buffered before moving to the next router.

Two critical enhancements to the oblivious router design distinguish the chaos router. The first is to allow the routing decision to be adaptive so that it no longer specifies a single channel to traverse next, but instead specifies a set of equally profitable channels. The first available profitable channel will be chosen for routing. The second enhancement is the addition of a small (five packet) buffer, the *multiqueue*, which holds packets for which no profitable channels are immediately available. By moving packets out of input frames to the multiqueue, critical channel resources are freed up so other packets can use them. Packets in the multiqueue are stored until a profitable output frame becomes free.

Packets in chaos routers are moved from input buffers into the multiqueue on two occasions: packet exchanges for deadlock prevention and "stalling" [2]. The packet-exchange deadlock prevention protocol mandates that if routers on either side of shared link have packets to send to each other, both packets must be sent, rather than only one side sending a packet. In order to guarantee this, the chaos router implements the following protocol: if a packet is sent to the output frame for channel i and the input frame for channel i has an incoming packet, the packet in input frame i is moved to the queue[1]. The second occasion on which packets are moved into the queue is when they have stalled for a "reasonably long" time at an input frame while waiting for a profitable output frame. For performance reasons, stalled packets are moved to the queue to free up the input channel for other packets. The chaos router determines that a packet is stalled when the entire packet is stored in a single input frame, *i.e.* it can no longer cut-through. Packets in the queue have priority over packets in input frames when competing for outgoing channels.

When an output frame desired by one or more packets in the multiqueue becomes available, a packet is routed out of the multiqueue to that output frame. When more than one such packet exists, FIFO priority is used; also, all packets in the multiqueue have precedence over packets in input frames. If the router is in a situation where a new packet must be read into the multiqueue, but the multiqueue is full, a packet is randomly selected from the multiqueue to be *derouted* to the next available output frame, making room for the new packet. Derouting of packets is the only mechanism for routing packets on non-minimal paths in the network.

3. DESIGN GOALS

Two primary design goals govern network router design: low latency and high throughput. For chaotic routing to be viable, it must perform better than competing routers. Presently, most multi-computer networks use simple and fast oblivious routers, so we have designed the chaos router to meet or surpass their performance level.

3.1. Uncongested Network Performance

When a network is uncongested there are few conflicts for resources, so throughput should be equal to the applied load in all cases. Thus, performance is limited by the minimum latency for packets going through the network. Since the chance of conflicts in routing is small, packets will be routed on minimal-length paths in both chaos and oblivious routers.

We are concerned with the minimum network latency for a given packet – the minimum time possible from the injection of a packet into the network until it is entirely received at its destination. Two factors which affect the minimum latency are the network cycle time, c, and the delay the message incurs at each node (in cycles), d. The minimum latency for a message of length L flits traveling D hops is then $c[(D+1)d+L-1)]$, since the header of the packet is delayed d cycles for each of the D hops (plus one additional "hop" for delivery) and then the remaining $L-1$ flits follow in cut-through fashion. Since the number of hops the packet takes and the packet length will be the same with either chaotic or oblivious routing when loading is light, the only factors which may vary are c

[1]Packets from the injection frame do not enter the queue due to deadlock prevention constraints.

and d. We compare these for oblivious and chaotic routers.

Cycle Time

The network cycle time is a critical factor in performance, as it is a multiplicative factor affecting the time to transfer each flit of the packet. A fundamental limit of the network cycle time is imposed by the chip-to-chip interface in the network: data cannot get on or off the chip at a rate faster than the I/O pads and wires connecting them can be cycled. Since both oblivious and chaotic routers must communicate off-chip using the same technology, they both face the same cycle-time limit imposed by the off-chip drivers.

In oblivious routers, pipelining of the simple routing decision, as in the Caltech MRC, allows the router to run as fast or faster than the pad drivers[2] [4]. Therefore, this limit is clearly achievable for the given technology. For chaotic routing, the design presented in Section 4 achieves this using a three stage synchronous pipeline. Thus, for both oblivious and chaotic routers, the network cycle time, c, is the same.

Routing Latency

The second factor in comparing the minimum latency is d, the routing latency per node. This factor represents the time the header of a packet spends entering a node and waiting to be routed to an outgoing link, when there is no contention. Essentially, d reflects the complexity of the routing decision, and we should expect it to be higher for the chaos router than for the oblivious router.

For an oblivious router with minimal buffering, the time allowed must be at least one cycle for the header to arrive on the inputs, one cycle for the routing decision to be made, and one cycle for the header to be routed to the outputs and updated. This gives a minimum of three cycles, so $d \geq 3$ cycles. The performance exhibited by the Caltech "Elko" MRC is consistent with our observation, having an asynchronous "cycle time" of 14.5 ns and minimum input-to-output latency of about 44.5 ns ($c = 14.5$ ns and $d = 3$ cycles)[4].

With chaotic routing, the first two cycles of a pipeline are used to make the routing decision, followed by a third cycle for the packet to cross the crossbar and enter the output frame. When congestion is minimal or absent, incoming packets will (after a cycle delay to cross the incoming link) immediately enter the routing pipeline and have no extra delay. Thus, for the chaos router, $d = 4$ cycles.

Therefore, the input-to-input latency for an oblivious router with buffering is three or more cycles and is four cycles for a chaos router. Since this delay is added only to the routing of the header flit, the minimum latency will be only one cycle per hop greater with chaotic routing than with oblivious routing.

3.2. Congested Network Performance

As network congestion increases, fewer and fewer packets experience the minimum possible network latency. As conflicts increase, the importance of making "smart" routing decisions increases as well. Since chaotic and oblivious routers have equal raw throughput capabilities, we expect chaotic routing to give better results due to its adaptive decisions which route packets around congestion. Simulation results illustrate chaotic routing's

[2] The MRC is a self-timed chip, so there are no explicit clocks.

superiority, with networks using chaotic routing saturating at higher throughputs and with lower latencies in torus and hypercube networks [2, 6].

4. CHAOS ROUTER DESIGN

As outlined in Section 3, to compete with a state-of-the-art oblivious router, the chaos router must have comparable throughput capability at all loads and match the minimum latency experienced during light loads. The goal of equal throughput resulted in a design in which the channels can be driven at the maximum rate allowed by the technology with no lengthening of the cycle time due to the router design. This is achieved by straightforward pipelining techniques at the expense of adding additional latency.

The chaos router can be viewed as two communicating components: the "core" router and the multiqueue. Since latency-critical operation (during low loads) will utilize only the core router, its operation has been streamlined the most. When packets enter the multiqueue, the delays due to resource contention are high enough to allow a slower, but simpler, hardware design.

4.1. Core Router Design

In order to minimize latency, the depth of the routing pipeline is kept to a minimum. Although several packets may be waiting to be routed through the router at any given time, matching each of the routes at once while arbitrating between conflicting requests is a very complex task [7]. By performing only one route at a time, the decision is simplified and can be achieved in a reasonable amount of time. This is achieved by picking a single "action dimension", specified by the *ActionDim* vector, to which a packet may be routed during a cycle. ActionDim is cycled through all output dimensions having packets waiting to be routed to them from either an input frame or the multiqueue.

The main routing decision of the chaos router is controlled by ActionDim, which operates as follows (refer to Figure 1):

1. ActionDim is incremented to the next *interesting* dimension in a round robin fashion. A dimension i is interesting if either:
   - Output frame i is available (has room for a new packet) *and*
     - A packet in the multiqueue or some input frame desires output channel i *or*
     - The multiqueue is full (this is derouting mode).
   - A packet in input frame i is *stalled*, i.e., it is completely stored in the input frame, and the multiqueue is not full. This is used to detect packets which have stalled and should be read into the multiqueue.
2. For ActionDim = i:
   - If a packet in the multiqueue desires output channel i, and output frame i is available, the multiqueue selects the oldest packet which can use channel i and sends it to its output frame. ActionDim stalls for three cycles waiting for this process to complete. If the multiqueue is full and no packets want

output channel i, a packet is randomly selected to be derouted to channel i and ActionDim stalls five cycles. Details of this operation are presented in Section 4.2.

- Otherwise, if a packet in input frame j desires output channel i and output frame i is available, the packet is routed directly (cuts through). This takes only a single cycle.
- Regardless of whether a packet is routed, if a packet is present in input frame i, it is read into the multiqueue. This operation implements the packet-exchange protocol and also provides the mechanism for reading "stalled" packets into the queue. This is concurrent with the above two operations and takes a single cycle unless the multiqueue is full, in which case ActionDim stalls until the packet can be read into the multiqueue.

The above routing procedure forms the first two stages of the routing pipeline. The next stage consists of transmitting the packet across the crossbars and decrementing or incrementing the header on the way out of the router. Thus, a packet experiencing no routing conflicts and cutting through the node will have a delay of three cycles plus one cycle delay to be transmitted off the chip. Three cycles minimum are added if the packet travels through the multiqueue.

4.2. Multiqueue Design

Logically, the multiqueue is a set of FIFO queues holding packets waiting for each output frame. Since a packet may desire multiple output frames, it may be present in multiple queues simultaneously. However, consistency between the logical queues must be maintained so that a single packet is not routed to multiple output frames. To implement the logical multiqueue, the datapath and control logic are separated.

The multiqueue datapath consists of five queue *slots*, each capable of holding a single packet. The control logic sets up transmissions from input frames to queue slots (via the queue input crossbar) and from queue slots to output frames (via the main crossbar). Once a transmission is set up, it proceeds autonomously in a cut-through fashion until it is finished.

4.2.1. Multiqueue Scoreboarding

The multiqueue scoreboard manages information about messages in the multiqueue. As the header of a packet enters the multiqueue, an entry is made in the scoreboard reflecting the queue slot the packet is in, its age relative to other packets in the multiqueue, and the packet's destination.

The primary functional/storage units in the scoreboard are two matrices: the *destination matrix* and the *storage matrix*. Each is a two-dimensional matrix with five columns, one for each slot in the multiqueue (see Figure 2). The relative age of a matrix entry is determined by its column position, with the oldest packet being in the rightmost position. New entries into the matrices are made on the left side, preserving the right-to-left age ordering. When the header of a packet leaves the multiqueue, its entry is deleted from the scoreboard and the matrices are compressed to the right.

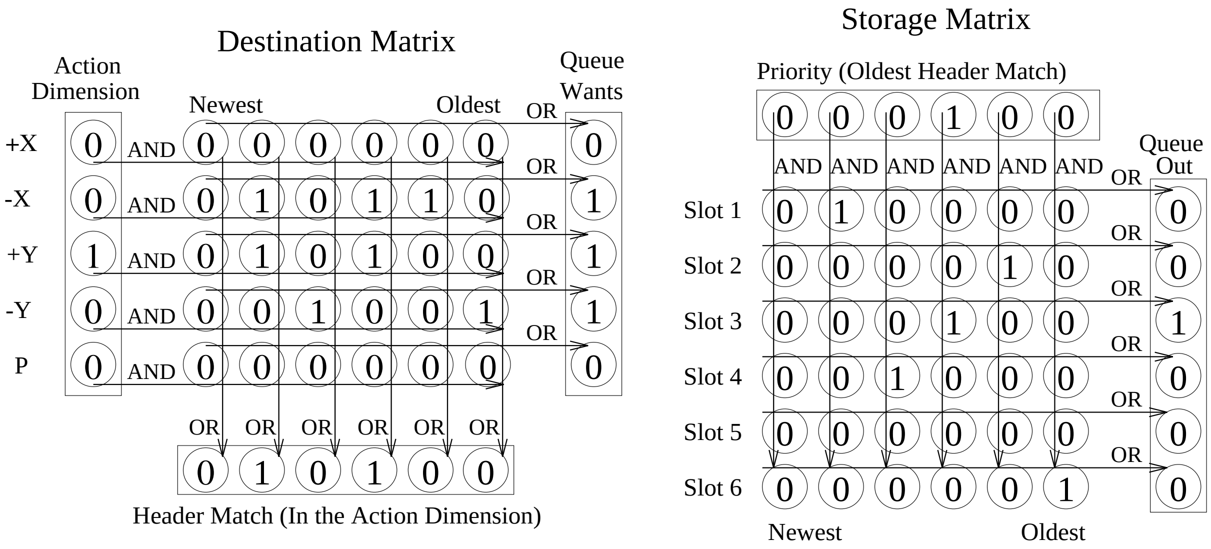

Figure 2. Multiqueue destination and storage matrices example. The matrices reflect a multiqueue with a capacity of six packets, currently containing four packets. The oldest packet is stored in queue slot 6 and waiting for the -Y output frame. ActionDim is the +Y dimension and matches two messages (those in the the third and fifth columns). The oldest of these two is given priority and its slot (queue slot 3) is selected for QueueOut.

Destination Matrix

The destination matrix has two distinct functions: to request specific output frames for routing and to find messages which can profitably use the output channel indicated by ActionDim. Its columns reflect the relative age of a particular message and its rows the dimensions that the message would like to be routed in. Each row is thus a chronological queue for a particular output dimension.

QueueWants is an output vector indicating the output dimensions desired by at least one packet in the multiqueue. If an element of the QueueWants vector is '1', then at least one packet in the multiqueue can profitably use the corresponding output channel. When derouting, the queue "wants" any available channel except the processor channel, so all elements of the QueueWants vector except for the element corresponding to the ejection channel are set high. Figure 2 shows an example of this calculation.

HeaderMatch is a vector indicating the packets in the multiqueue that desire the channel indicated by ActionDim. Thus, if the ith element of HeaderMatch is asserted, then the ith packet in the matrix can profitably use channel[ActionDim] for routing. The corresponding queue slot can be located using the storage matrix (see below). Figure 2 shows an example of this calculation.

Storage Matrix

The storage matrix keeps track of which queue slot each packet is in. As the header of packet enters the multiqueue, a one-hot vector is loaded into the storage matrix indicating the queue slot. The columns of the storage matrix are locked together with the columns of the destination matrix so that the columns correspond to the same packet.

Priority is an output vector indicating the highest priority message selected in the HeaderMatch vector by the destination matrix. This is the oldest matched message, except when derouting, when it is randomly selected. Thus, this vector indicates the column of the matrices corresponding to the selected packet to send out of the multiqueue to the channel indicated by ActionDim. The destination matrix transforms this vector into the QueueOut vector, which indicates the queue slot holding the appropriate packet. An example of this computation is shown in Figure 2.

4.2.2. Multiqueue Performance

Since the multiqueue is not on the critical path of the router, optimization of its performance is not extremely important. Thus, the multiqueue may take several cycles to complete its operations. There are two primary operations: *read* and *send*. In a read operation, the control logic specifies that a packet should be read into the multiqueue. Since the multiqueue only has to select an empty queue slot to place the packet in, this operation usually takes only a single cycle and does not stall the router. In fact, it can be performed concurrently with the routing of another message across the main crossbar. Only if the multiqueue is full, but must read in another packet, does the router stall until room is available in the multiqueue. On the other hand, a send from the multiqueue is more complicated. Selecting a packet to be sent requires three cycles during normal operation. When derouting, two extra cycles are necessary to randomly pick a packet from the queue and read in its replacement, resulting in a total delay of five cycles.

4.3. Channel Design

Each of the five external channels is implemented using one set of bi-directional wires. The decision to use one set of bi-directional wires instead of two sets of uni-directional wires is based on pin limitations. With the same number of pins per channel, using uni-directional wires requires that the flit size be one half of what it would be using bi-directional wires. When network loading is heavy and the channel is in demand in both directions, all of the wires are being used all of the time in either case, so the bandwidth is equal. However, when the network is lightly loaded and latency is dominated by the length of the packet (in flits), packets using the uni-directional channels will experience much greater latency due to their smaller flit size. Unfortunately, arbitration between the two sides sharing a channel is not trivial and, thus, there is a penalty to sharing the channels. This results in a "dead" cycle, when neither side can transmit data, occurring when changing the direction of the channel.

5. DESIGN CONSEQUENCES

Although the serialization of the routing decision causes the chaos router to spend as many as five cycles to route a single packet, this constraint does not seriously affect the performance. Throughput through the node would be affected if this serialization caused a backlog of packets to queue up. However, because the packets are composed of multiple flits, there is a limit on how frequently, on average, a new packet can arrive on a channel. In order to keep up with newly arriving packets, all that is necessary is that each channel

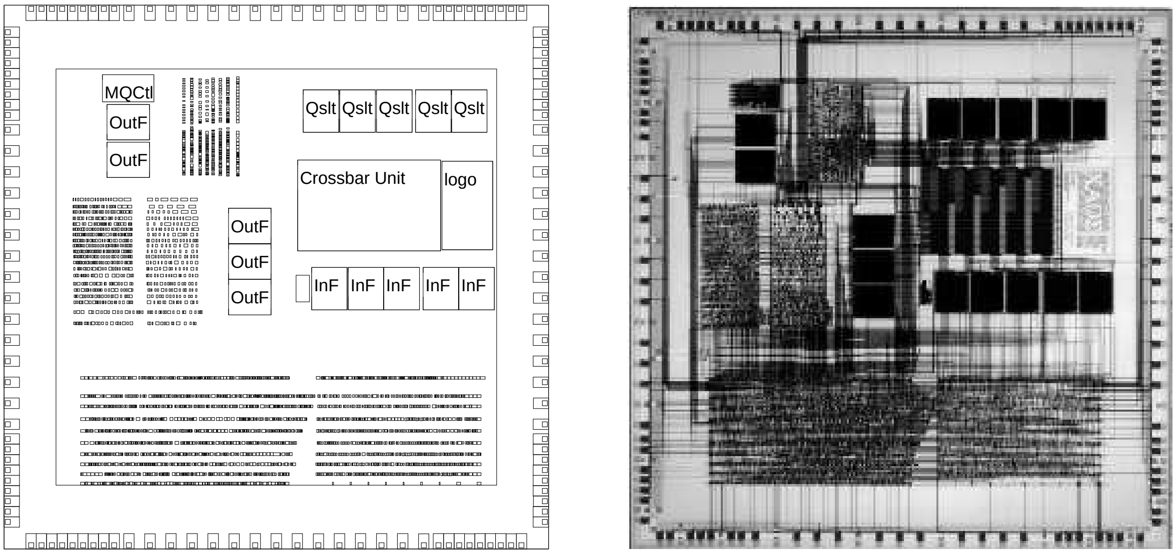

Figure 3. Floorplan (left) and die photo (right) of the prototype chaos router chip.

is visited at least once every L cycles for packets of length L flits. The only time five cycles are spent during a routing decision is during derouting mode. During this worst-case situation, the router may stall for five cycles at each of the $d+1$ channels in a degree d router. However, when in derouting mode, the processor channel is not considered, so d may be used instead of $d+1$ in this analysis, resulting in each channel being visited once every $5d$ cycles. In the 2-D chaos router, $L = 20$ flits and $d = 4$, so throughput will not suffer. Since the worst-case situation is only likely to happen when traffic is heavy, though, the bi-directional channels are most likely to be fully utilized in both directions. This means that new packets can only arrive once every $2L$ (or 40) cycles, allowing plenty of time for the router to route all incoming packets with no backlog.

The latency added by the serialization of the routing decision is also trivial compared to latency due to routing conflicts. When the network load is high and there are many packets waiting in the multiqueue, the latency experienced by a single packet will be extended by as many as $5d$ cycles. However, this is very small when compared to the latency experienced due to conflicts at high loads. When loading is light to medium, few packets will enter the multiqueue and, thus, the router will be able to route a new packet nearly every cycle and little delay will be added.

6. IMPLEMENTATION

To study the performance of chaotic routing, a prototype chip has been constructed. We chose a two-dimensional (mesh and torus) topology, requiring five 16-bit bi-directional channels. Our implementation uses 1.2μm CMOS in a 132 pin package. As stated in Section 3, we intend to support a clock speed which is as fast as the output pads can be toggled. For this technology, the pads cannot be run faster than about 66 MHz (15 ns cycles). Thus, we have designed for a 15 ns cycle time, which has been achieved in simulation.

Figure 3 shows the floorplan and layout of the chaos router chip. The cut-through

FIFO's, the crossbar unit, and the multiqueue control logic are implemented in custom logic. The remainder of the chip is implemented using approximately 3800 standard cells, resulting in a chip approximately 10500μm on a side.

7. CONCLUSIONS

We have demonstrated the feasibility of adaptive routing by building a chaos router which is competitive with state-of-the-art oblivious routers. This has been achieved through careful design to minimize the complexity on the critical path of the router and through extensive pipelining to achieve a fast cycle time. Since the cycle time of the chip is limited by the speed of the output drivers, the complexity of the chip does not affect the raw bandwidth when compared with simpler designs. The latency across the chip has been kept to four cycles which is comparable to the three cycle latency across an oblivious router. Thus, we have designed a chip with the advantages of adaptive routing, but with only minimal additional cost compared to oblivious routers.

ACKNOWLEDGMENTS

We wish to thank Smaragda Konstantinidou for the original design of the chaos routing algorithm and Lawrence Snyder for guiding the chaos project. The initial design of the chaos chip was done as a class project by the authors and Shyng Chen, Pai Chou, David Hubbell, Donald Lobo, Miles Ohlrich, Wayne Ohlrich, Ross Ortega and Peter Sullivan. Finally, we wish to thank Larry McMurchie for his support in managing the myriad of design tools necessary for this effort.

REFERENCES

1. Smaragda Konstantinidou and Lawrence Snyder. "Chaos router: Architecture and performance." Proceedings of the 18th International Symposium on Computer Architecture, pages 212–221. IEEE, May 1991.
2. Kevin Bolding and Lawrence Snyder. "Mesh and torus chaotic routing." Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference, pages 333–347, March 1992.
3. Smaragda Konstantinidou and Lawrence Snyder. "The chaos router: A practical application of randomization in network routing." Proceedings of the 2nd Symposium on Parallel Algorithms and Architectures, pages 21–30. ACM, 1990.
4. Charles L. Seitz and Wen-King Su. "A family of routing and communication chips based on the Mosaic." Symp. on Integrated Systems: Proc. of the 1993 Washington Conf., 1993.
5. P. Kermani and L. Kleinrock. "Virtual cut-through: A new computer communication switching technique." Computer Networks, 3:267–286, 1979.
6. Melanie L. Fulgham and Lawrence Snyder. A study of chaotic routing with nonuniform traffic. Technical Report CSE-93-06-01, University of Washington, Seattle, WA, June 1993.
7. Smaragda Konstantinidou. Deterministic and Chaotic Adaptive Routing in Multicomputers. PhD thesis, University of Washington, Seattle, WA, May 1991.