Low-Latency Virtual-Channel Routers for On-Chip Networks

Robert Mullins, Andrew West and Simon Moore
Computer Laboratory, University of Cambridge
William Gates Building, JJ Thomson Avenue, Cambridge CB3 0FD, UK
Robert.Mullins@cl.cam.ac.uk

Abstract

The on-chip communication requirements of many systems are best served through the deployment of a regular chip-wide network. This paper presents the design of a low-latency on-chip network router for such applications. We remove control overheads (routing and arbitration logic) from the critical path in order to minimise cycle-time and latency. Simulations illustrate that dramatic cycle time improvements are possible without compromising router efficiency. Furthermore, these reductions permit flits to be routed in a single cycle, maximising the effectiveness of the router's limited buffering resources.

1. Introduction

The ability to fully exploit modern fabrication technologies is tempered by both physical and logical design complexity. The cost of this complexity suggests the reuse of design and verification effort wherever possible. This is often achieved by composing systems from commodity IP or by reusing custom blocks repeatedly in the same design. The relatively poor scaling of global interconnects and the need to achieve architectural performance gains in an energy-efficient manner, provide pressure to decentralise computation. Together these trends suggest a move towards an increasingly communication-centric view of processor and system architecture [16, 21, 15, 14].

One proposed solution to the problem of chip-wide communication is a network of top-level point-to-point communication channels [1, 8, 12] (See Figure 1). This highly regular wiring strategy aims to reuse a small number of highly optimised wiring layout and driver designs. As channel layouts are reused to create the network, effort in characterising delay, power and verifying signal integrity is minimised. The simple behaviour of the network also aids in predicting performance and ensuring correctness. In contrast, large bus based communication networks present a complex verification task at every level. In addition, the limited ability to scale interconnect delays makes the presence of long global wires and buses increasingly undesirable.

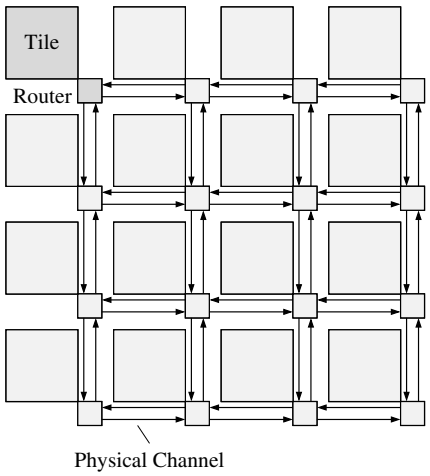

Figure 1. On-Chip Network. Each tile may contain identical logic, as in the case of a multiprocessor or tiled system, or simply represent a partitioning of a SoC design.

Similar observations have already been made in the case of inter-chip and wider-area communication. While much of this work is applicable, some important differences exist [8]. In particular, on-chip designs exploit a far greater number of pins and wires, while inter-chip designs are often pin limited. In addition, while inter-chip router designs may exploit a large number of buffers, on-chip designs must aim to minimise buffer count in order to maximise the silicon real-estate available for computation. Area pressures, together with the need to minimise on-chip communication latencies, suggest the implementation of relatively simple on-chip routers.

This paper describes how router latency may be significantly reduced by hiding control overheads. The

Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)
1063-6897/04 $ 20.00 © 2004 IEEE

creation of a single-cycle architecture also reduces latency and maximises the impact of limited buffering resources. Simulation results illustrate that while these techniques offer dramatic cycle time reductions, they do not compromise router efficiency. Initial circuit-level simulations suggest a router cycle time of 12-FO4 delays[1] plus clock overhead is possible. Previously published delay models have suggested similar router designs require three pipeline stages and a clock cycle-time of 20-FO4 delays [19].

We provide an overview of a generic virtual-channel router implementation in Section 2. Section 3 introduces the techniques we use to optimise the router's control. The critical path of the optimised router is analysed in Section 4. Simulation results comparing a number of router control implementations are presented in Section 5. Section 6 and 7 discuss related work and conclude the paper.

2. Background

A network may be characterised by its topology, routing strategy and method of flow-control [5]. For simplicity we assume a mesh network (with bidirectional links) together with dimension-ordered (XY) routing[2].

The choice of flow control technique is guided by the need to minimise buffer requirements and latency in our on-chip network. Schemes that reserve buffer space or apply flow-control at the packet level, such as store-and-forward [20] or virtual-cut through [13], are unsuitable for these reasons. A wormhole-router provides the necessary fine-grained flow control, while the addition of virtual-channels [4, 6] aids in boosting performance and circumventing message-dependent deadlock. Furthermore, Quality-of-Service (QoS) enhancements are possible by prioritising the allocation of virtual-channels and switch bandwidth.

The remainder of this section provides an overview of the architecture of a generic virtual-channel router.

2.1. Overview of a Virtual-Channel Router

Figure 2 illustrates the major components of a generic virtual-channel router. The router has `P` input ports and `P` output ports, supporting `V` virtual-channels (VCs) per port.

Virtual-channel flow control exploits an array of buffers at each input port. By allocating different packets to each of these buffers, flits[3] from multiple packets may be sent in an interleaved manner over a single physical channel. This improves both throughput and latency by allowing blocked packets to be bypassed.

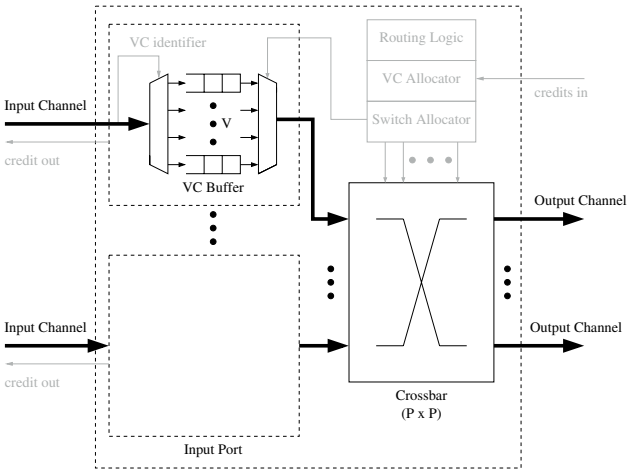

Figure 2. Virtual-Channel Router

The basic steps undertaken by a virtual-channel router are enumerated below:

1. **Routing**. The first flit of a new packet arrives at the router. The routing field is examined and a set of valid output virtual-channels upon which the packet can be routed is produced. The number of output VCs produced by the routing logic will depend on the routing function. Possibilities range from a single output VC to a number of different VCs potentially at different physical channels (*i.e.* adaptive routing).

   The selection of an output VC can also be influenced by the class of the packet to be routed. Packets from particular classes will often be restricted to travelling on a subset of virtual-channels to avoid message-dependent deadlock. A common practise is to provide separate *request* and *reply* virtual-networks.

2. **Virtual-Channel Allocation**. An attempt is made to allocate an unused VC to the new packet. A request is made for one of the virtual-channels returned by the routing function. Allocation involves arbitrating between all those packets requesting the same output VC.

3. **Switch Allocation**. Each packet maintains state indicating the availability of buffer space at their assigned output VC. When flits are waiting to be sent, and buffer space is available, an input VC will request

1 A FO4 (fan-out-of-4) delay is the delay of one inverter driving four copies of itself [7].

2 In the case of our 2D-mesh network, dimension-ordered routing simply means packets are first routed in the X and then the Y direction. The implementation of the routing function in such a scheme is trivial. The fact that some turns are never made within a router may also reduces the complexity of the router's crossbar.

3 Each packet is composed of a number of flits (flow control digits). A flit is the smallest unit of flow control.

Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)
1063-6897/04 $ 20.00 © 2004 IEEE

access to the necessary output channel via the router's crossbar. On each cycle the switch allocation logic matches these requests to output ports, generating the required crossbar control signals.

4. **Crossbar Traversal**. Flits that have been granted passage on the crossbar are passed to the appropriate output channel.

The following sections describe in more detail each of the router's components.

2.2. Input Buffer and Bypass

Each new incoming flit is stored in the VC buffer designated by its VC identifier. This identifier is appended to every flit in the previous router stage. If the VC buffer is empty and the flit is able to access the crossbar immediately, a bypass path is required to expedite its journey.

2.3. Routing Logic

In order for virtual-channel and switch allocation to take place the routing function must first be evaluated to determine which virtual-channel(s) at which output port(s) the packet may request. To ensure that this computation does not lie on the router's critical path, the computation may be performed in the previous router in preparation for use in the next. The idea that the route may be calculated one step ahead of where it is required was first employed by the SGI routing chip [10] and is known as *look-ahead routing*.

2.4. Virtual-Channel Allocation

Peh and Dally detail the complexity of both virtual-channel (VC) allocation and switch-allocation logic in [19]. The following two sections provide a brief overview of these schemes.

The complexity of VC allocation is dependent on the range of the routing function. In the simplest case, where the routing function returns a single VC, the allocation process simply consists of a single arbiter for each output VC. As any of the input VCs may request any output VC, each arbiter must support $P \times V$ inputs.

If the router function returns multiple output VCs restricted to a single physical channel, an additional arbitration stage is required to reduce the number of requests from each input VC to one. The winning request at each virtual channel buffer then proceeds to the second stage as described above. The complexity of such a scheme is illustrated in Figure 3. The routing function determines the output port and VCs that may be requested prior to VC allocation. A VC which is free to be allocated is then selected by the first stage of arbitration. The result of this first stage of arbitration is a request for a single VC at a particular output port. This request is subsequently sent to the appropriate second stage arbiter. While this scheme does not guarantee to allocate all free output VCs to potential waiting input VCs in a single cycle, there is no performance penalty as only one flit may be sent per cycle on an output channel.

In the most general case where the routing channel may return any of $P \times V$ VCs, the number of inputs to the first stage of arbiters must now be increased from V to $P \times V$. In this case some performance degradation may be expected as the scheme makes little effort to perform a good matching of requests to free output VCs.

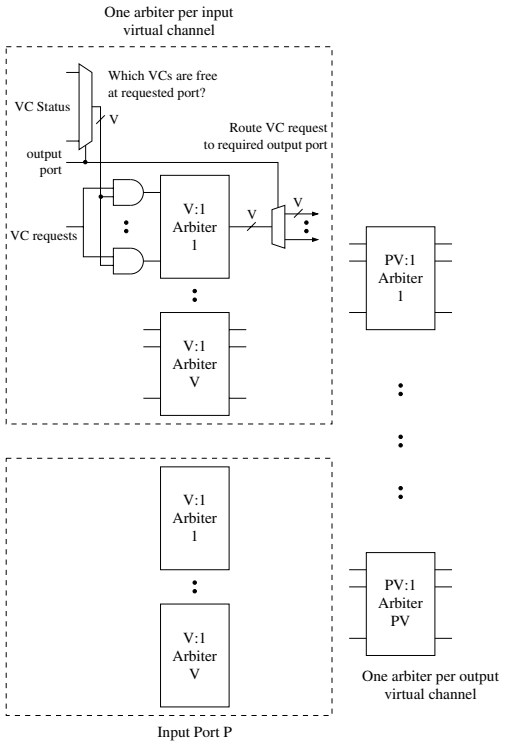

Figure 3. Arbitration complexity of a virtual channel allocator. In this example, the routing function returns a single output port and one or more VC requests.

2.5. Switch Allocation

Individual flits arbitrate for access to physical channels via the crossbar on each cycle. Arbitration may be performed in two stages [19]. The first reflects the sharing of a single crossbar port by V input virtual-channels, this requires a V-input arbiter for each input port. The second

Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)
1063-6897/04 $ 20.00 © 2004 IEEE

stage must arbitrate between winning requests from each input port (P inputs) for each output channel. The scheme is illustrated in Figure 4. The request for a particular output port is routed from the VC which wins the first stage of arbitration.

In order to improve fairness, the state of the V-input arbiter is only updated if the request is also successful in the second stage of arbitration. We assume this organisation wherever multiple stages of arbitration are present.

This switch allocator organisation may reduce the number of requests for different output ports in the first stage of arbitration, resulting in some wasted switch bandwidth.

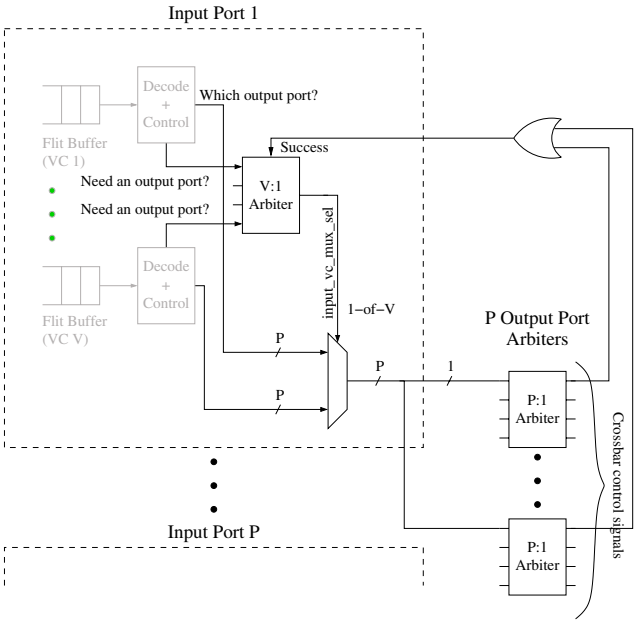

Figure 4. Switch Allocation Logic

2.6. Speculative Switch Arbitration

Virtual-channel flow control as discussed performs VC allocation and switch allocation sequentially. This guarantees that only packets that have successfully obtained an output VC from the VC allocator can make requests for their desired output channel.

Peh and Dally [19] describe how this dependency may be relaxed if we speculate that a waiting packet will successfully be allocated an output VC. In this way both VC and switch allocation can be performed in parallel. To avoid a negative impact on performance the switch allocator in the speculative design must prioritise non-speculative requests over speculative ones. This is achieved by implementing two switch allocators, one handling speculative requests (from packets that are also requesting a VC be to allocated) and another for non-speculative requests (from packets which have already been allocated a VC). Only when no non-speculative requests are granted for a particular output port are successful speculative requests granted.

In the case that a speculative request is granted we must ensure that the VC has been allocated and it is capable of receiving a new flit (has free buffer space) before the flit is actually sent. Fortunately, such checks may be performed in parallel with crossbar traversal.

2.7. Crossbar

In the architecture illustrated in Figure 2 each input port is forced to share a single crossbar port even when multiple flits could be sent from different virtual-channel buffers. This restriction allows the crossbar size to be kept small and independent of the number of virtual-channels. Dally [6] and Chien [2] suggest that providing a single crossbar input for each physical input port will have little impact on performance as the data rate out of each input port is limited by its input bandwidth. While simulation results indicate some advantage in providing larger crossbars (see Figure 8) this is often unrealistic as crossbar implementations scale very poorly. A more effective use of area may simply be to increase the size or number of VC buffers.

3. Low-Latency Router Control

The following section details how we may further optimise the design of both the switch and the virtual-channel allocation logic. While these ideas apply equally to pipelined implementations, our aim is to create a low-latency single cycle implementation.

3.1. The Free Virtual Channel Queue

The first stage of arbitration in the virtual-channel allocator ensures each VC makes a single request for a output VC. The requests are generated as a product of the routing function and a VC status mask, indicating the availability of free VCs at a particular output port. An alternative is to simply queue free VC identifiers and provide a mask with a single bit set (indicating the free VC at the head of the queue), thus avoiding the need to arbitrate between multiple free VCs. A separate queue is provided for each output port and for each virtual-network (traffic-class), *e.g.* two queues per output port to provide request and reply networks. The scheme effectively removes the need for arbitration by predetermining the order of grants.

Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)
1063-6897/04 $ 20.00 © 2004 IEEE

3.2. Tree Arbiters

A well understood method for creating arbiters with a large number of inputs is to organise them as a tree of smaller arbiters. In this scheme each arbiter propagates requests eagerly up the tree prior to determining which input they will actually grant. The root arbiter's grant is subsequently propagated back down the tree, granting a single input request.

The large P×V-input arbiters used in the second stage of VC allocation may be simplified by adopting this approach. Here the single P×V-input arbiter is replaced by a single P-input arbiter and P groups of V-input arbiters. The V-input arbiters arbitrate between requests from the same input port (different VCs) and the P-input arbiter selects the winning port. A VC allocator exploiting free VC queues implemented using this approach is illustrated in Figure 5.

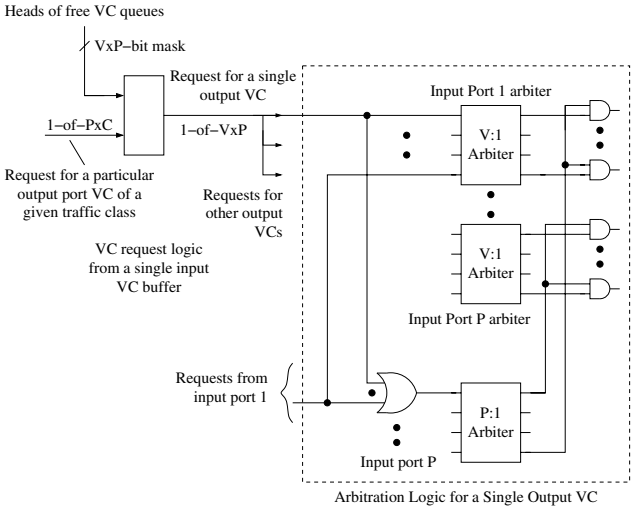

Figure 5. Virtual Channel Allocator logic. A free VC queue and tree arbiter organisation is used to simplify the implementation.

3.3. Precomputing Arbitration Decisions

An arbiter may provide a *least recently served* priority scheme by maintaining a queue recording the order in which requests have been granted. When arbitration takes place the request with the highest priority, indicated by its position in this queue, is granted. Immediately after an input is granted it is reduced to the lowest priority by placing it at the end of the queue. The matrix-arbiter [19] is an efficient circuit-level implementation of such a scheme.

Figure 6(a) illustrates how a single grant output is generated in such a design. Here the state indicating that one request has priority over another is stored in the upper triangle of a matrix of flip-flops. The arbiter ensures a grant is generated only if an input request with a higher priority is not asserted. Such a design requires each of the arbitration requests to be used in the generation of each output (fanout of R). Furthermore, each grant signal is generated by a NOR gate with a fan-in of R. The flip-flop matrix is updated after each clock cycle to reflect the new request priorities.

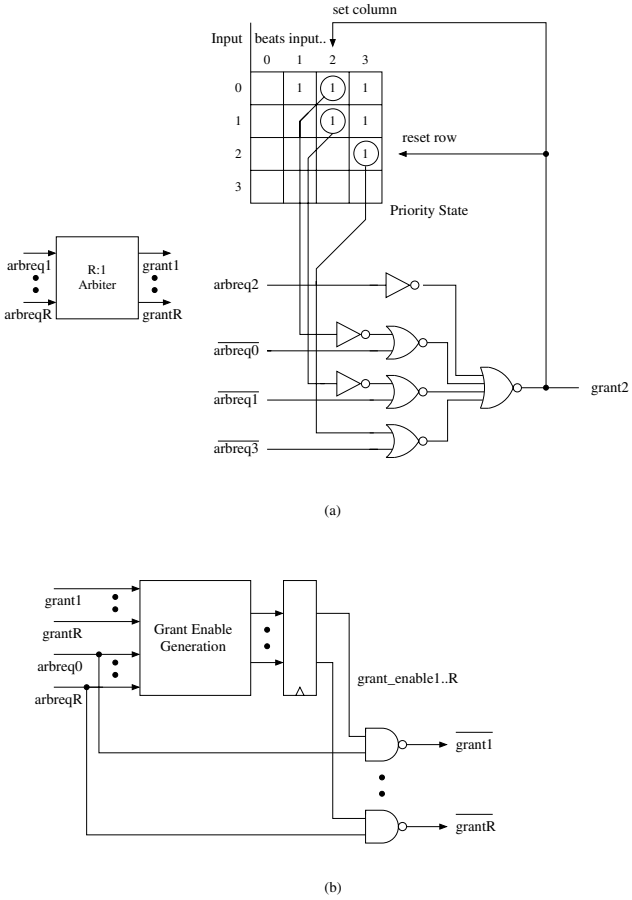

Figure 6. R-input Arbiters. (a) A matrix arbiter. The schematic shows how a single grant output is produced for a 4-input arbiter. Each arbreq signal has a fan-out of R. (b) An arbiter which precomputes its grant enable signals.

An alternative design is illustrated in Figure 6(b). Here arbitration outcomes are determined one cycle before they are required (the logic to do this is very similar to that used in the matrix arbiter). On the cycle prior to a grant being asserted grant enable signals are generated and latched. The grant signals are subsequently generated as the product of the precomputed grant enable signals and incoming arbitration requests. The scheme works if at least one request is present on the preceding clock cycle,

Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)
1063-6897/04 $ 20.00 © 2004 IEEE

if no requests are present we must be more creative in the generation of grant-enable signals. A number of options are available to us depending on the environment in which the arbiter is deployed:

- **Safe Environment** The environment guarantees that only one new request may be added per clock cycle. In this case it is safe to assert *all* the grant enable signals when no arbitration requests were present (or one request was present that was granted) in the previous cycle (or on reset). The next request is then granted instantly regardless of its origin.
- **Unsafe Environment** In this case the environment may assert multiple new requests in a single clock cycle. Our options are:
  (a) To set a single grant-enable signal speculatively, perhaps by examining the pattern of previous requests. While this is safe, it does not guarantee that a new request will be granted in a single cycle.
  (b) Again set a single grant-enable signal but with the aid of a hint (again perhaps speculatively).
  (c) Set all grant-enable signals. In this case we are forced to detect the case where multiple requests are granted and abort *all* the operations they enabled.

The following sections describe for each instance of an arbiter in the router design how it may be replaced with a design that precomputes the arbitration outcome. We assume a speculative switch allocation scheme as described in Section 2.6.

3.3.1. Switch Allocation Logic (V:1) Arbiters In the case of switch allocation we must consider the speculative and non-speculative requests separately.

Speculative Switch Arbitration Requests. Speculative requests are made by header flits which are awaiting VC allocation. The arbitration outcome is precomputed by considering the requests that remain after VC allocation. If no requests remain it is safe to assert all grant-enable signals as at most one new header flit may be received in the following cycle. This corresponds to the description of a *safe* environment in Section 3.3.

Non-speculative Switch Arbitration Requests are produced by each input VC meeting the following criteria:

1. Currently holding a flit
2. Current packet has already been allocated a VC
3. Output VC in question has free buffer space

Grant enable signals can be generated after VC allocation has taken place (criteria 2) and credits indicating free buffer space have been received and processed (criteria 3). If no request is asserted at the end of the cycle it is guaranteed that no request will be made on the following cycle (remember new flits make speculative requests). In this case we simply assert no grant-enable signals.

3.3.2. Switch Allocation Logic (P:1) Arbiters The second stage of switch allocation arbitrates between winning requests from participating input ports. The result of precomputing the first stage of requests (detailed above) may be used to determine which input ports will make requests on the next cycle. This allows the arbitration outcome for the second stage to be precomputed. Again we must consider the special case where no requests are present when generating grant enables in both the speculative and non-speculative cases.

Non-speculative Switch Arbitration Requests

If at the end of a clock cycle no flit is present that has been allocated an output VC, no non-speculative switch arbitration requests will be made on the following cycle. Any new flits arriving at the router will make speculative requests to the switch allocator.

Speculative Switch Arbitration Requests

In some cases it is difficult to precompute grant-enable signals. Consider the case where no flits are buffered in the router and two flits arrive (at different input ports) destined for the same output port. Here arbitration needs to take place on the cycle the flits are received, the outcome cannot easily be predetermined. This situation may arise in the case of the second stage speculative switch arbiters.

In Section 3.3 we outlined possible solutions to this problem. The first was to predict where the next request would originate from. While possible, this is likely to incur a significant latency penalty for packets whose arrival could not be predicted. A more accurate prediction could be assisted by the neighbouring routers indicating the possibility that they may make a request in the following clock cycle. While this is a possibility, we choose to simply assert all grant-enable signals in the case where no requests are present in the current cycle (option C in Section 3.3). If multiple requests are received on the following cycle by the same arbiter all operations enabled by the grant signals must be aborted.

In many scenarios this will have little impact on performance. In the case of a lightly loaded network, the probability that multiple requests will be made to the same output in this way is small. Even if this is the case, and a one cycle penalty is necessary, the latency will be increased regardless as both packets must be sent on the same output channel. In the case of a more heavily loaded network, flit buffer occupancy is likely to be higher making the case that no requests remain at the end of a cycle less likely.

3.3.3. VC Allocation Logic (V:1) Arbiters Requests made for the same output VC from the same input port are

Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)
1063-6897/04 $ 20.00 © 2004 IEEE

arbitrated by P groups of V-input arbiters at each output port. Grant-enable signals are precomputed regardless of the state of the VC (whether it is free or not). This is safe as each arbiter is dedicated to a particular output VC and requests will only be made if the VC is free. In the case where no requests are made, all the grant-enable signals for the arbiter may be asserted. This environment is safe since at most one new flit may be received per cycle at one input port.

3.3.4. VC Allocation Logic (P:1) Arbiters These arbiters face the same problem as the second stage P-input arbiters in the speculative switch allocator. If no request is present on the preceding clock cycle it cannot easily be determined from which input port the next flit will be received. Again we may proceed by asserting all grant-enable signals and aborting granted operations in the case that two or more requests are subsequently received.

Note that the reorganisation of the monolithic $P \times V$-input arbiters as a tree arbiter simplifies the precomputation of grant signals.

4. Analysis of Dependencies/ Critical Path

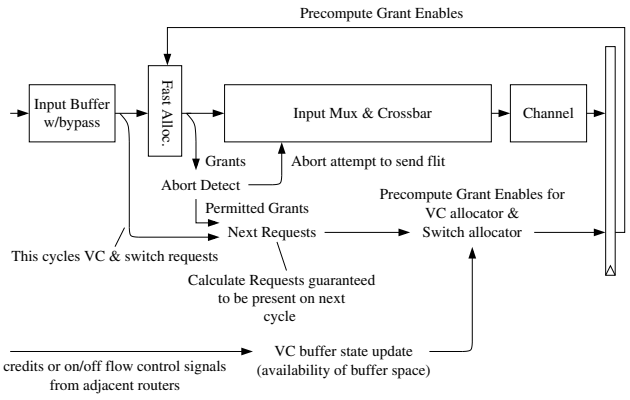

Figure 7. Data dependencies within our single-cycle optimised router architecture

Figure 7 illustrates the dependencies within our optimised router design. Virtual-channel flit FIFOs are assumed to be able to receive a flit in one clock cycle ready for use in the next. The case where the flit is needed on the same cycle is handled by a bypass.

The *fast allocator* is used to generate VC and switch grant signals from the precomputed grant enables. The presence of precomputed grant-enables at the start of the clock cycle means that the logic required to generate the crossbar and crossbar input multiplexer control signals becomes trivial. Cases where the fast allocator produces invalid control signals are quickly detected and the associated operations aborted (in these cases valid control signals are guaranteed to be generated on the next clock cycle). The permitted grants and existing requests are then used to calculate the request signals guaranteed to be present on the next cycle (of course new requests may also be made as new flits arrive). The permitted grants are also used to update the state of the matrix arbiters. Once the requests present on the next cycle have been computed and updated VC buffer state information is available, grant enables for the next cycle may be computed.

One concern is the need to update VC buffer state information prior to precomputing the grant-enable signals for the P:1 non-speculative switch arbiters. One possibility is to precompute grant-enable signals using the older state before it is updated. Unfortunately, the buffer state of multiple output VCs assigned to VCs at a single input port may be updated in a single cycle. This prevents us from setting all grant-enable signals safely. Although this could be done if we are able to abort grants in the case that two or more requests are subsequently received. In the simulations that follow we assume that this dependency is not on the router's critical path and may be tolerated. In our implementation we have adopted a simple on/off channel flow control mechanism which simplifies the logic needed to maintain the buffer state. Such a scheme would be less desirable if the router did not operate in a single cycle.

Initial results from preliminary extracted layout (180nm technology) suggest that the design will operate at our target cycle time of 12 FO4 delays plus clock overhead. This is approximately twice the tile frequency in our planned system. In our test network each flit carries 64-bits of data and routers are placed 1mm apart. All signal transitions (in each output channel and the crossbar) are in the same direction during evaluation avoiding worse-case crosstalk. Typical case communication delays between routers are within 2 FO4 delays. Inter-wire capacitance values for communication channels were calculated using QuickCap [11]. The precomputation of grant-enable signals is essential in meeting our cycle time. Our best 5-input matrix-arbiter designs have a typical latency of approximately 3 FO4 delays. The complete control logic takes the majority of the clock cycle in the optimised design, although almost none of this is now on the critical path.

5. Simulation Results

A parameterised network model was constructed using HASE (Hierarchical Architectural Simulation Environment) [3]. The underlying simulation system is multi-threaded and event-driven. Each tile or node generates packets with random destinations. Packets are

Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)
1063-6897/04 $ 20.00 © 2004 IEEE

generated at a constant rate and queued until they are able to enter the network. The interval between the creation of individual packets is random (geometric distribution) to prevent packets being injected into the network synchronously. Network latency is measured from the time the first flit is created to the time the last flit in the packet is received at its destination, including any time spent buffered at the source node. Each node injects 1000 packets into the network and performance statistics are gathered after an initial warm-up period of 100 packets/node. The network is an 8x8 mesh, each router has 5 input and 5 output ports. Packets are 5 flits in length. In all simulations we assume a single cycle router implementation.

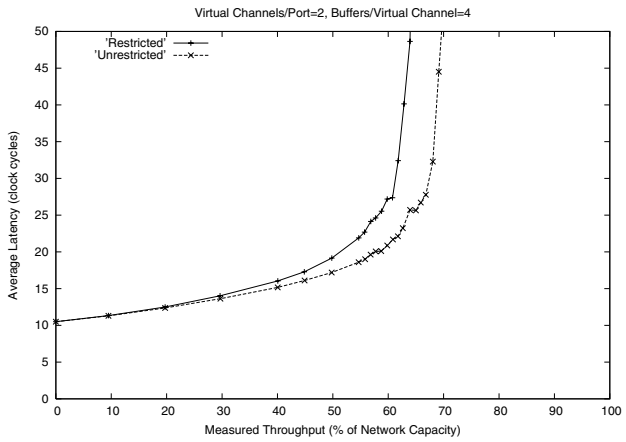

Figure 8. Latency versus throughput for P and PV-input (restricted/unrestricted) crossbar implementations. 2 virtual channels per input port and 4 flit buffers per virtual channel.

We explore the impact on network performance of a number of different switch and virtual-channel allocator implementations:

- **Sequential** Virtual-channel allocation is performed prior to switch allocation. This removes the need for the switch allocation logic to speculate on the ability of new packets to acquire VCs.
- **Parallel-NoSpec** Virtual-channel allocation is performed in parallel with switch allocation but no speculative requests are made to the switch allocator. Only when a packet is allocated a VC may it proceed to switch allocation (a new packet will require at least one cycle to obtain a VC before it can arbitrate for access to the crossbar).
- **Parallel-Spec** Virtual-channel and switch allocation are performed in parallel. Packets that are awaiting VC allocation are permitted to make speculative requests for switch allocation. This enables flits to be received and routed on an output in a single cycle.
- **Parallel-Spec-PreComp** Extends the **Parallel-Spec** implementation to model the precomputation of grant enable signals. In cases where the arbiters operates in an unsafe environment, an arbiter which sets all grant enable signals is modelled. In the case where two or more requests are received on the subsequent clock cycle, no input is granted (option c. in Section 3.3).

Results are shown in Figure 9 for a range of buffer sizes and virtual-channel configurations. An initial inspection of the results shows that all but the **Parallel-NoSpec** model have very similar performance characteristics. At closer inspection and perhaps surprisingly the **Sequential** scheme does not necessary outperform the parallel schemes. This behaviour is the result of two effects. Firstly, the speculative switch allocator prioritises packets during switch allocation that have held a VC for at least one cycle. This can be modelled in the sequential case, slightly improving performance. Secondly, in the case of the speculative allocator two requests from each input port may be considered after the first stage of arbitration. This potentially increases the chance of finding a more complete matching of waiting flits and ready output ports. Performance could be potentially improved further in the parallel schemes by ensuring speculative requests are only made if at least one free VC is available at the required output port.

Figure 8 illustrates the performance impact of restricting the number of inputs to the crossbar, as discussed in Section 2.7. The unrestricted case models the provision of a crossbar input for every input virtual-channel. The restricted case models the case where virtual-channels at each input port share a single crossbar port.

6. Related Work

The techniques that have been described allow router and arbitration latency to be hidden. This is achieved by predetermining the outcome of routing and arbitration decisions one cycle before they are required. Routing and arbitration latency may also be eliminated by statically scheduling buffer and channel resources [21]. For many applications this is prohibitively expensive due to the size of the routing memory required. Statically scheduled schemes also suffer from the inability to handle dynamic traffic. Flit-reservation flow-control [18] exploits the ability to preschedule resources but achieves greater flexibility by deferring scheduling decisions until run-time. Channel and buffer usage is scheduled by sending control flits ahead of data flits on an independent, less congested, network. Scheduling decisions are recorded in reservation tables

Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)
1063-6897/04 $ 20.00 © 2004 IEEE

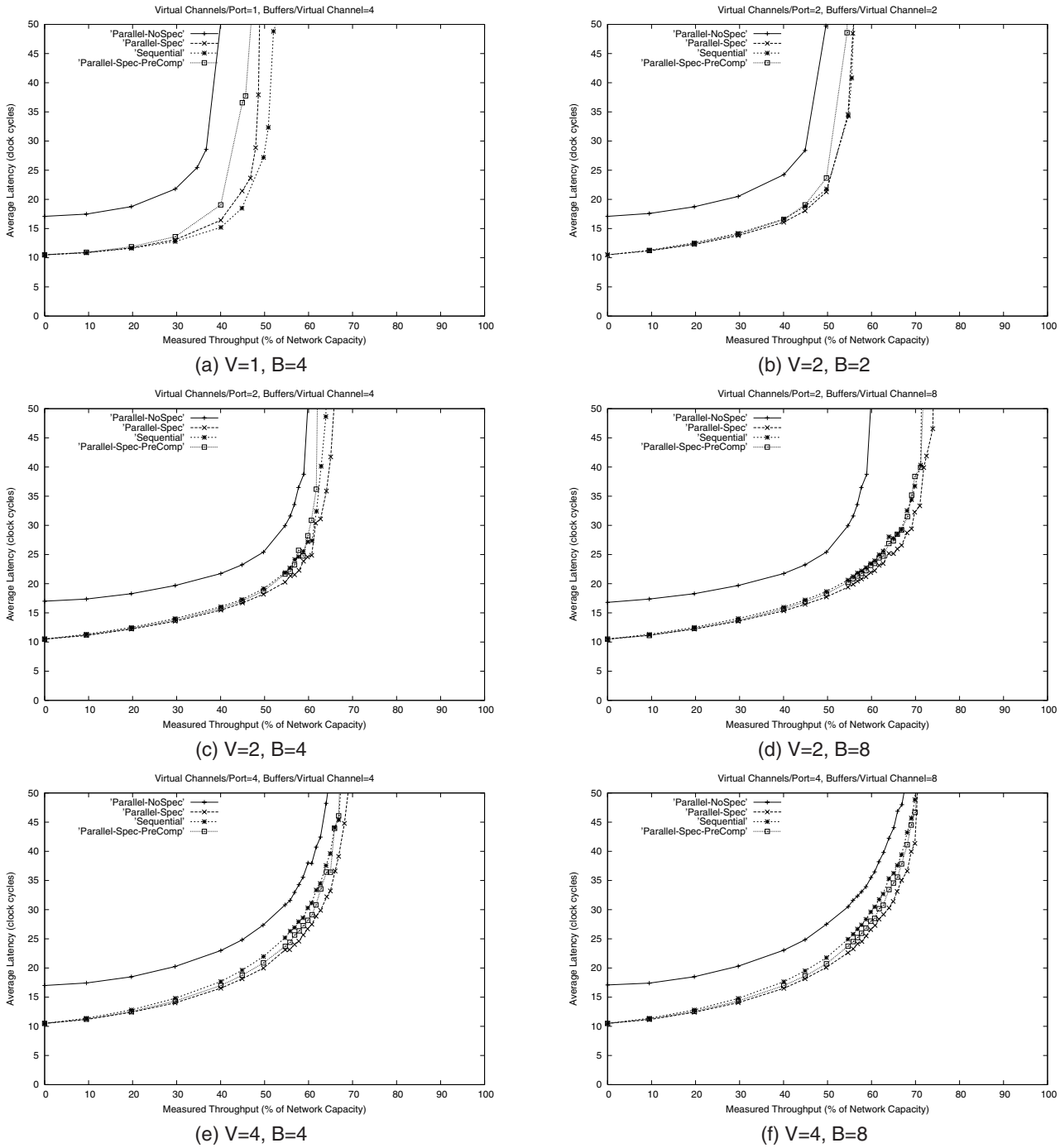

Figure 9. Latency versus throughput for a number of different VC and switch allocation implementations (5-flit packets). For each graph, `V` is the number of virtual-channels per input port and `B` is the number of buffers per virtual-channel.

Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)
1063-6897/04 $ 20.00 © 2004 IEEE

associated with each input and output port. Arbitrated access to output reservation tables together with the relatively complex scheduling operations add significantly to the complexity of the router. A preliminary investigation also suggests that our cycle time would be extended by such an approach. While the ability to schedule resources more than a cycle before their use is a powerful technique, this will always incur a bookkeeping overhead in recording and generating a schedule. A detailed analysis of an implementation of flit-reservation flow-control is presented in [17].

7. Conclusion

This paper has demonstrated a low-cost approach to significantly reducing the cycle-time of on-chip routers. Simulation results have shown that the critical path is reduced significantly without compromising router efficiency. Preliminary layout of the major components of the router has been completed and a 0.18μm VLSI implementation clocked at 1.2GHz is planned. The design is supported by a novel grid-based distributed clocking scheme to ensure minimal skew between adjacent routers [9].

Acknowledgements

The authors would like to thank the SCALE group (MIT) and the anonymous reviewers for their suggestions and comments. We would also like to thank Roland Ibbet and Frederic Mallet for their support in using HASE (University of Edinburgh). This work is supported by EPSRC (grant GR/L86326) and the Cambridge-MIT Institute (CMI).

References

[1] L. Benini and G. D. Micheli. Networks on Chips: A New SOC Paradigm. *IEEE Computer*, 2002.

[2] A. A. Chien. A cost and speed model for k-ary n-cube wormhole routers. In *Proceedings of Hot Interconnects*, 1993.

[3] P. Coe, F. Howell, R. Ibbett, and L. Williams. A Hierarchical Computer Architecture Design and Simulation Environment. *ACM Transactions on Modeling and Computer Simulation*, 8(4), October 1998.

[4] W. J. Dally. Wire-Efficient VLSI Multiprocessor Communication Networks. In P. Losleben, editor, *Proceedings of the Stanford Conference on Advanced Research in VLSI*. MIT Press, March 1987.

[5] W. J. Dally. *VLSI and Parallel Processing*, chapter Network and Processor Architectures for Message-Driven Computing. Morgan Kaufmann, 1989.

[6] W. J. Dally. Virtual-Channel Flow Control. In *Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA)*, 1990.

[7] W. J. Dally and J. W. Poulton. *Digital Systems Engineering*. Cambridge University Press, 1998.

[8] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proceedings of the 38th Design Automation Conference (DAC)*, June 2001.

[9] S. Fairbanks and S. Moore. The Distributed Clock Generator. In *Proceedings of the second ACiD-WG Workshop*, Munich, Germany, January 2002.

[10] M. Galles. Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SGI SPIDER Chip. In *Proceedings of Hot Interconnects Symposium IV*, 1996.

[11] R. B. Iverson and Y. L. L. Coz. *Users Guide for QuickCAP^{TM}*. Random Logic Corporation, 2003.

[12] A. Jantsch and H. Tenhunen, editors. *Networks on Chip*. Kluwer Academic Publishers, 2003.

[13] P. Kermani and L. Kleinrock. Virtual Cut-Through: A New Computer Communication Switching Technique. *Computer Networks*, 3:267–286, January 1979.

[14] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[15] R. Krashinsky, C. Batten, and K. Asanovic. *SCALE-0 Architecture Manual*. MIT Laboratory for Computer Science, 2003.

[16] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. Dally, and M. Horowitz. Smart Memories: A Modular Reconfigurable Architecture. In *The 27th Annual International Symposium on Computer Architecture (ISCA)*, June 2000.

[17] L.-S. Peh. *Flow control and micro-architectural mechanisms for extending performance of interconnection networks*. PhD thesis, Stanford University, 2001.

[18] L.-S. Peh and W. J. Dally. Flit-Reservation Flow Control. In *International Symposium on High-Performance Computer Architecture*, pages 73–84, Jan 2000.

[19] L.-S. Peh and W. J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *International Symposium on High-Performance Computer Architecture*, pages 255–266, Jan 2001.

[20] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[21] M. B. Taylor *et al.* The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. *IEEE Micro*, March/April 2002.

Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)
1063-6897/04 $ 20.00 © 2004 IEEE