# The Design and Implementation of a Low-Latency On-Chip Network

Robert Mullins, Andrew West and Simon Moore
Computer Laboratory, University of Cambridge
*Robert.Mullins@cl.cam.ac.uk*

*Abstract*— **Many of the issues that will be faced by the designers of multi-billion transistor chips may be alleviated by the presence of a flexible global communication infrastructure. In the short term, such a network will provide scalable chip-wide communication and ease the complexity of handling multi-cycle communications. In the long term, the network will become a primary tool for optimising power and data transfers and for scheduling computations. This paper details the design and implementation of a low-latency on-chip network. The network's speculative routers are in the best case able to route flits in a single clock cycle, helping to minimise on-chip communication latencies and maximise the effectiveness of buffering resources. Results from our 180nm test chip demonstrate an inter-router data transfer rate in excess of 16Gbit/s for each link. In the best case each router hop adds just 1 clock cycle to the final communication latency.**

## I. Introduction

Transistor switching speeds are continually improved through scaling. Unfortunately, the impact of scaling on long wires is a negative one. This forces an increase in communication latencies and the energy required to communicate each bit of information. The growing disparity between communication and switching times will soon make the provision of a chip-wide communication infrastructure a central problem in achieving performance and power dissipation goals. The resulting shift in design trade-offs will lead to an era of "communication-centric" system design.

While constant length global wires fail to scale well, the distance reachable in a single clock cycle in multiples of $\lambda$ does remain essentially constant [7]. This allows the performance of designs of fixed complexity to scale when ported to the next technology node. This observation leads to the concept of a scalable architecture composed of a number of tiles or modules of fixed complexity. The performance of each tile scales as expected and additional performance is possible by adding tiles as scaling permits. Inter-tile communication is handled by an on-chip network which consumes only a few percent of the total chip area. The way in which such a system could scale is illustrated in Table I. In this example the die size is assumed to be fixed at 256mm$^2$. Each tile contains around 11M transistors and the clock period is set to 16 FO4 delays. The table shows how the frequency, size (width) and number of tiles scale. The interconnect delay along one edge of a tile remains constant at around one clock cycle.

The calculation of cycle time in Table I assumes one FO4 delay may be calculated as $500ps * L_{gate}$ (where $L_{gate}$ is the physical gate length as specified in [13]). The channel delays were estimated using results from [1], [7]. In all cases it should be possible to traverse the channel between two routers in less than one clock cycle. Of course, if a longer clock period is employed a smaller number of larger tiles may be used.

| Technology Node | No. of Tiles | Width of Tile | Tile Frequency |
|---|---|---|---|
| 90nm | 32 | 2.8mm | 3.4GHz |
| 65nm | 64 | 2mm | 5.0GHz |
| 45nm | 128 | 1.4mm | 7.0GHz |
| 32nm | 256 | 1mm | 13.0GHz |

TABLE I
PREDICTED SCALING OF A GENERIC TILE-BASED SYSTEM

Such tile-based systems may implement arrays of homogeneous processor/cache tiles [9], [10], finer-grain computing fabrics [14] or networks of heterogeneous IP blocks. Such approaches provide highly reconfigurable platforms for a wide range of performance hungry applications. The provision of an efficient chip-wide dynamic on-chip network is fundamental in achieving performance goals, flexibility and mitigating complexity in such systems.

Packet-switched networks employing Virtual Channel (VC) flow control have recently been proposed as one approach to implementing a chip-wide interconnection network [3]. Figure 1 illustrates the major components of a generic virtual-channel router. Packets gain access to a physical channel by first obtaining a virtual-channel (*VC allocation*). Each of these virtual-channels has its own private input FIFO at the destination router allowing flits[1] from different packets to be sent in an interleaved manner. Access to a physical channel is now allocated on a cycle-by-cycle basis (*switch allocation*) amongst waiting flits from any of the buffered packets which have been assigned a VC. This scheme improves both throughput and latency when compared to a simple wormhole routed network by allowing blocked packets to be bypassed. Particular classes of traffic may be restricted to a subset of the available virtual-channels in order to provide QoS enhancements or circumvent message-dependent deadlocks.

## II. Speculative Router Architectures

The description of virtual-channel flow control in Section I implies that VC allocation and switch allocation are performed sequentially. Peh and Dally [12] describe how this dependency may be relaxed if we speculate that a waiting packet will be successful in acquiring a VC. In this way both VC and switch allocation may be performed in parallel. In order to avoid a negative impact on performance, the

[1]A packet is composed of a number of flits (flow-control digits)

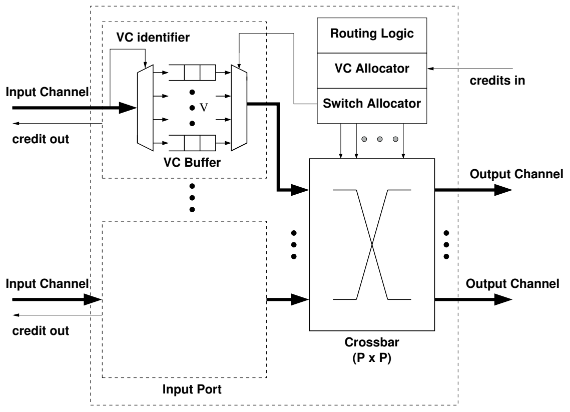

Fig. 1. A Virtual-Channel Router

switch allocator must prioritise non-speculative requests over speculative ones. This is achieved by implementing two switch allocators: one handling non-speculative requests from packets which have been allocated a VC and one for requests from packets awaiting VC allocation. We will refer to these as the *high* and *low priority switch allocators* respectively from this point onwards. Speculative requests are only granted for a particular output when no regular requests are present. In the case that a speculative request is granted we must ensure that the VC has in fact been allocated and buffer space exists downstream. Fortunately, such checks may be performed in parallel with crossbar traversal.

## III. Single-Cycle Routers

The introduction of further speculative optimisations to reduce the router pipeline depth to a single pipeline stage was proposed in [11]. These optimisations almost completely remove any control overhead from the critical path. Both VC and switch allocation are now performed concurrently with the transport of flits across the datapath and physical channel. The ability to make such optimisations is based on the following observations: if we assume that the network is heavily loaded it should be possible to make scheduling decisions accurately one clock cycle in advance. This is because all the information necessary to make such a decision is present when many packets are buffered. At the other extreme, when the network is very lightly loaded, we may assume that contention for a VC or physical channel is low. In this case it is also possible to schedule one cycle in advance by speculating that any new request for a VC or physical channel may be granted immediately. Simulation results predicted that for all intermediate throughputs the router sacrifices only a few percent of performance over a perfect single-cycle sequential scheme [11].

Figure 2 provides an outline of the single-cycle router architecture. In this scheme VC and switch allocation is effectively performed one cycle in advance and concurrently with the transport of flits. Each allocator's output is a set of grant-enable signals which are registered and used on the succeeding clock cycle to generate VC and switch allocation grant signals. The presence of buffered flits allow resources to be scheduled one cycle in advance, in this case the asserted grant-enable signals correspond to the subset of requests to be granted on the next clock cycle. If it is not possible to schedule a particular VC or output in advance, a prediction is made that there will be only one subsequent request for the resource. In this case multiple grant-enable signals are set. This allows any request on the next clock cycle, for the resource in question, to be successful. Cases where multiple requests are made on the following clock cycle are detected by the abort logic described in Section III-A.

Datapath control signals are produced early in the clock cycle by the *"fast"* logic blocks, simply by combining the output port requests from each buffered (or newly arrived) flit and the registered grant-enable signals. The output port required by each flit is known without the need to first evaluate a routing function by performing this task in the previous router (look-ahead routing [6]).

### A. Abort Detection

One issue which must be considered carefully is the case when our prediction that requests on the next clock cycle will not contend is subsequently proven false. Fortunately, to detect these abort cases we only need consider newly arrived flits. If flits were buffered on the previous clock cycle, speculation would not have been necessary.

The abort logic associated with both VC and switch allocation consists mainly of a comparison between each of the output ports required by each new flit[2]. If we assume there are $P$-input ports this requires $P(P-1)/2$ comparisons. The abort logic detects cases where we are speculating and two or more flits requiring the same output port resource (physical link or VC) have arrived simultaneously. In these cases the allocation of the resource is blocked incurring a one clock cycle penalty. The correct scheduling of the resource takes place during this time and non-speculative grant-enable signals are generated for use on the following clock cycle.

In order to use the simple abort logic described above some additional logic is required to account for one remaining corner case. This is the scenario when a tail flit leaves an input buffer and exposes a new packet (buffered head flit). As this packet can now request any output port it is possible it will contend with another such packet or newly arrived flit. The solution adopted for such cases is to always stall such head flits for one cycle to ensure they are handled properly by the switch and VC allocation logic and need no further special treatment. This has a negligible impact on performance.

### B. Calculating the next set of requests

To enable the VC and switch allocators to produce accurate grant-enable signals for the next clock cycle they may be fed a set of requests that we know will be present on the *next* clock cycle. These may be calculated by considering available

[2]At most one new flit may be received at each input port per clock cycle

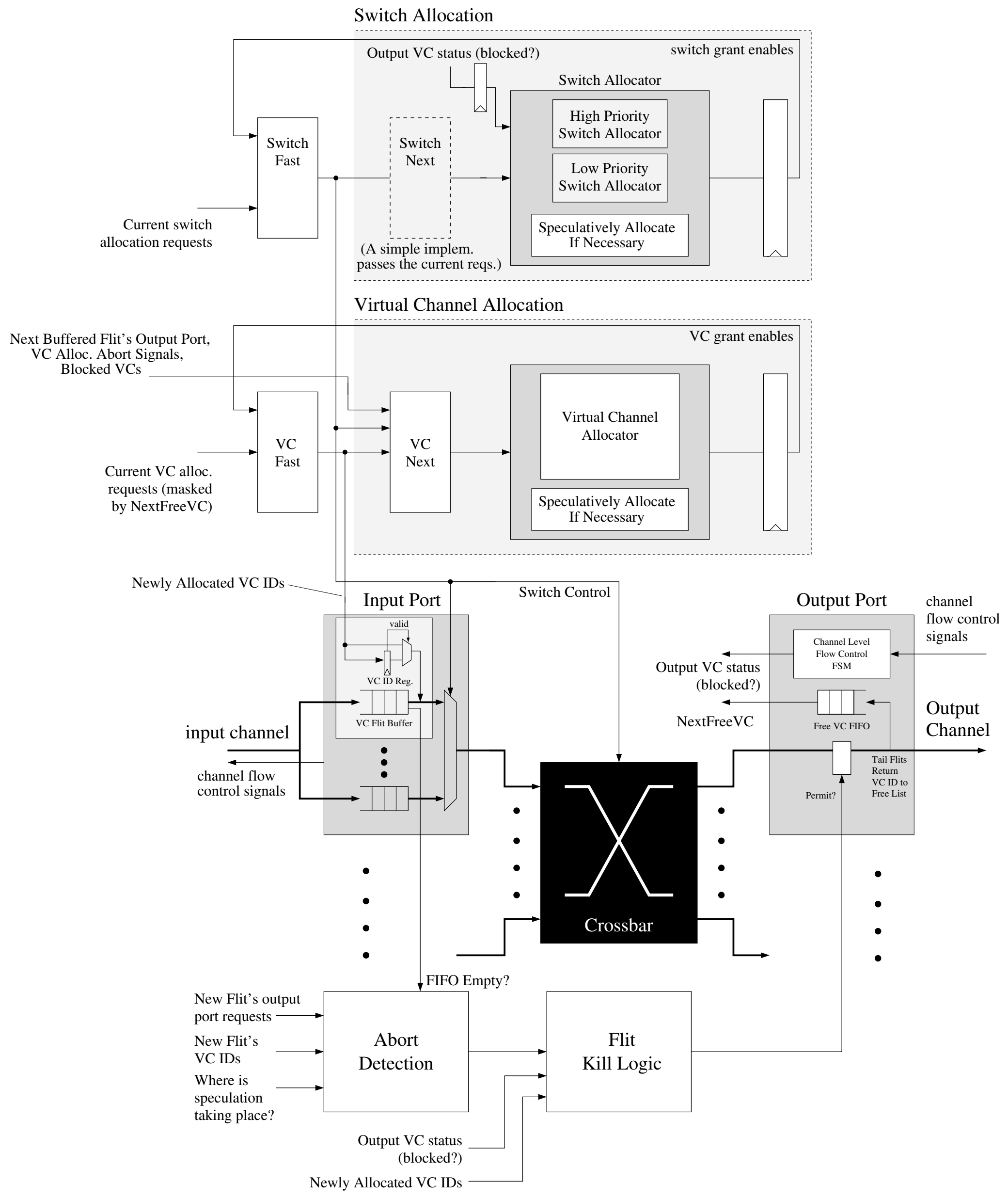
Switch Allocation
Output VC status (blocked?)
switch grant enables
Switch Allocator
High Priority Switch Allocator
Low Priority Switch Allocator
Speculatively Allocate If Necessary
Switch Fast
Switch Next
(A simple implem. passes the current reqs.)
Current switch allocation requests
Virtual Channel Allocation
VC grant enables
Next Buffered Flit's Output Port, VC Alloc. Abort Signals, Blocked VCs
VC Fast
VC Next
Virtual Channel Allocator
Speculatively Allocate If Necessary
Current VC alloc. requests (masked by NextFreeVC)
Newly Allocated VC IDs
Input Port
Switch Control
valid
VC ID Reg.
VC Flit Buffer
input channel
channel flow control signals
Output Port
channel flow control signals
Channel Level Flow Control FSM
Output VC status (blocked?)
NextFreeVC
Free VC FIFO
Output Channel
Tail Flits Return VC ID to Free List
Permit?
Crossbar
FIFO Empty?
New Flit's output port requests
New Flit's VC IDs
Where is speculation taking place?
Abort Detection
Flit Kill Logic
Output VC status (blocked?)
Newly Allocated VC IDs

Fig. 2. A single-cycle speculative virtual-channel router architecture. When necessary the router is able to speculate that flits arriving on the next clock cycle may be routed without contention. During switch allocation the router is also able to speculate on the successful acquisition of VCs by new packets and on the availability of buffer space at the flit's destination.

information such as the current requests and those granted on the current cycle. Information about the next buffered flit in each VC buffer may also be exploited.

To ensure that the abort logic is the only place where we need to handle mispredictions, it is important that the set of requests output by the *next request logic* contains at least the requests from those flits already buffered. Presenting additional requests, e.g. those granted on the current cycle, may reduce performance but will not cause the router to malfunction. If requests that are to be made by buffered flits are not considered, grant-enable signals may be set speculatively enabling multiple buffered flits to gain access to the same output (or VC). As only newly arrived flits are considered by the abort logic, this problem would go unchecked.

In the final router implementation we chose to accurately calculate VC next requests using all the information available. In the case of the switch scheduler we simply used the current set of requests to schedule the switch for the next cycle. This provided a significant improvement in cycle time with a small architectural performance penalty (see comparison between *spec-fast* and *spec-accurate* in Section V). The simplification is aided by the fact that those switch requests recently granted have a low arbitration priority.

### C. *Pipelining the use of VC state*

The use of VC status information provides an example of how internal control paths may be pipelined with only minor changes to the architecture. In order to reduce cycle time it was advantageous to pipeline the VC status data used by the switch allocation logic. By adding a pipelining register, the information provided to the switch allocator about which VC is blocked becomes more out-of-date. In order to ensure the quality of the switch schedule does not suffer significantly the availability of both high- and low-priority switch allocators is exploited. If a request is associated with a VC that appears to be blocked it is steered to the low-priority allocator. Actual VC blocked status is checked when the flit is selected for transport (in parallel with its journey to its output port).

This sort of modification is simplified by the way in which the architecture decouples scheduling from the datapath. The allocator's task is simply to provide the best schedule it can for the next clock cycle with the information available. Final checks on the validity of the schedule are delayed until the schedule is applied. If advantageous, further pipelining of the control logic internally could be exploited without compromising the best case single cycle routing latency. This could involve further pipelining of the allocators themselves.

## IV. Implementation

The Lochside test chip consists of 16 traffic generating tiles interconnected by a 4x4 mesh network. The chip is implemented in UMC's L180 logic process (1.8V core, 0.18$\mu$m) with all aluminium interconnect. Tiles and routers are interconnected as shown in Figure 3. Each router is connected to its neighbour using two unidirectional 80-bit channels (64-bits of data and 16-bits of control information). Each of the router's input ports support 4 virtual-channels and may buffer 4 flits on each virtual-channel.

The implementation is fully testable via traditional scan chain techniques. An on-chip PLL may be used to provide a clock source and is distributed to each tile using a simple hand-crafted H-tree. Alternatively, a Distributed Clock Generator (DCG) [5] may be selected as the global clock source. In both cases, tile level clock distribution was achieved by running a standard-cell clock tree synthesis tool.

The vast majority of the design is implemented in a standard cell style. Exceptions include the DCG nodes and latch-based virtual-channel buffers which benefited from a full-custom implementation. The final router design was generated from a highly parameterised network router model that allows a range of router designs to be synthesized.

The performance of the design is limited by our current PGA package (due to both thermal and bond-wire IR drop issues). This limits the performance when running all traffic generators to around 250MHz. If only two random packet sources are enabled the maximum clock rate may be increased to 300MHz.

Once a flit is received at a router's input port it may be allocated a virtual-channel and access to an output port, traverse the crossbar and arrive at the destination router in a single clock cycle (best case latency is simply one cycle per hop). At 250MHz each router is able to transfer data at a maximum rate of 16Gbits/s on each input and output link.

## V. Results

Each tile's traffic generator is able to produce a wide range of traffic patterns. Traffic destinations may be selected randomly or deterministically with control over packet length. Error detecting code and flit ordering checks are also performed at each tile. Each tile maintains statistics on the number of packets sent and received, together with the timing information necessary to calculate average latency and throughput. Each tile is able to inject traffic at a controlled rate into a tile output queue. Packets injected into this queue when it is full may be counted. The configuration system also provides the necessary control logic in order to synchronise the execution of commands at each tile, e.g. in order to start and stop all tiles simultaneously.

Figure 5 shows the recorded average packet latency versus measured throughput for our 4x4 mesh network. For each experiment 16K packets were sent to uniformly distributed destinations from each tile. Curves are plotted for a range of fixed packet lengths. Experiments which resulted in the tile output queue becoming full and generated packets being dropped are not plotted.

### A. *Performance*

The router was synthesized to operate at 200MHz under worst case PVT operating conditions (around 35 FO4 including clocking overhead). If the speculative scheduling optimisations are removed but VC and switch allocation is still performed in parallel, the clock period is extended by a factor of 1.65. It may be noted that the optimised router does

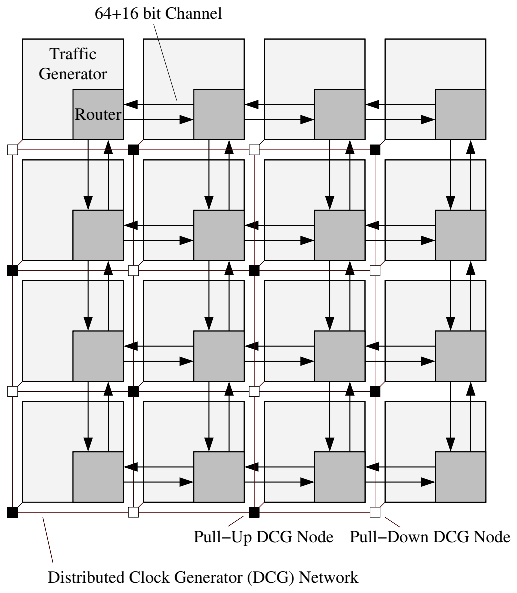

Fig. 3. Block Diagram of the Lochside Chip

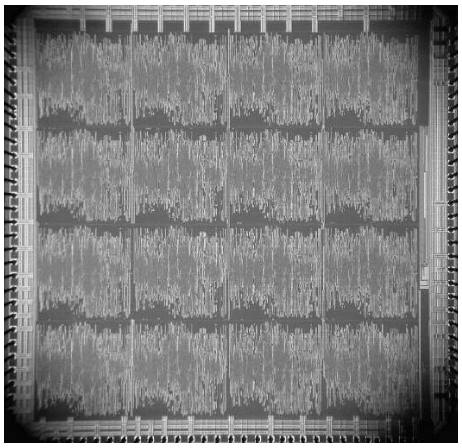

Fig. 4. Lochside Die Micrograph. Die size is 5mm x 5mm. The chip contains approximately 5 million transistors.

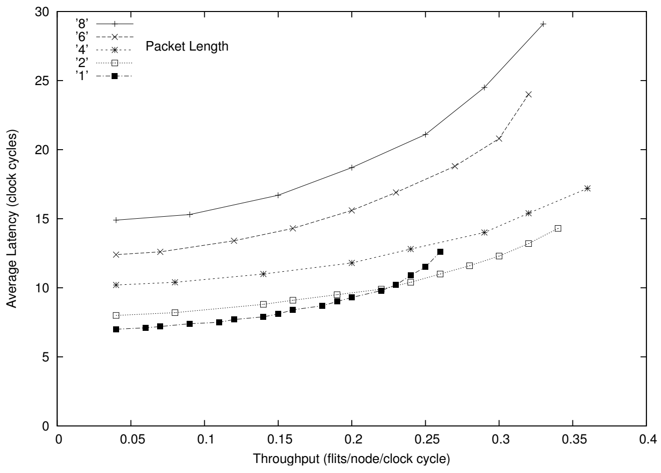

Fig. 5. Latency versus throughput measured from test-chip for a range of packet lengths.

not quite achieve a speedup equal to the reduction in clock cycle time. This is due to our router producing a slightly inferior routing and VC allocation schedule as a result of the approximations exploited to reduce cycle time. It is not, as may be expected, directly as a consequence of VC and switch aborts. The number of aborts is in fact consistently very low. Overall our speculative scheduling optimisations reduce average communication latency by a factor of 1.3 to 1.6.

Figure 6 plots latency against throughput for three router architectures: (*spec-fast*) full speculation with no switch next request logic, (*spec-accurate*) full speculation with accurate switch next request logic and (*sequential*) concurrent switch and VC allocation followed by switch traversal in the same clock cycle. Each architecture was synthesized to calculate its minimum clock period and the latency figures scaled to account for these differences. The clock period results were, 30, 41 and 50 FO4 delays respectively (including 2 FO4 of clock uncertainty). The packet length in these experiments was 256-bits (4 flits).

In our final design the switch allocation critical path is composed of the following delays: (32%) input register, steering and buffering switch request to allocator, (53%) switch allocation, (15%) selecting speculative or non-speculative switch allocator result and speculatively setting grant-enables if necessary.

The performance achieved by this implementation closely tracks that predicted by earlier router simulation models.

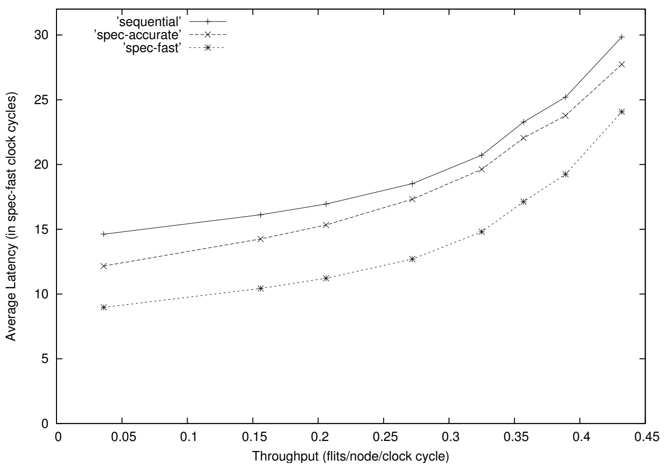

Fig. 6. Latency/Throughput comparison for three router architectures. Latency is scaled to account for differences in each router's cycle time.

### B. Area

A 4x4 mesh network is not really practical for the size of our test chip or its technology. The small tile size is dominated by the area of each router (more than two thirds of a tile's area is taken by the network). However, if we move to the next technology node (130nm) and imagine a larger chip (4x4 array of 3mm x 3mm tiles) synthesis results have shown that the network area overhead is reduced to only $5-6\%$. This overhead would drop even further if the scaling in Table I was adopted.

The area overhead of the speculative single-cycle architecture is small at around $8\%$. This compares the area of two single cycle routers one with our optimisations and one without. If the unoptimised case was pipelined, the difference in area would fall as registers would be required to buffer intermediate results.

## VI. Related Work

Our implementation compares favourably to other on-chip network designs and implementations published to date. Comparable networks which have been implemented include the Philips Æthereal network-on-chip [4] and the RAW processor's dynamic networks [14]. The Philips team report a similar peak link bandwidth of 16Gbit/s while operating at 500MHz in a 130nm process. Control decisions are actually taken at 166MHz. The router differs from ours in its ability to offer guaranteed services by reserving consecutive routing slots in consecutive routers. Virtual-channels are not supported for improving the performance of best-effort traffic. RAW's dynamic networks operate at 225 MHz (worst-case PVT). In this case the whole network is duplicated in preference to exploiting virtual-channels. The RAW processor was implemented using IBM's 180nm 6LM ASIC copper process

Other research projects include Netchip [8] which aims to automatically generate application-specific on-chip networks. The authors emphasise the need to maintain a high switch operating frequency and adopt a deeply pipelined architecture (7-stage router). Unfortunately this significantly increases buffering requirements by extending round-trip time. It also incurs a significant overhead in terms of the additional pipelining registers required. Even if a very short clock period of less than 10 FO4 is possible, best case communication latencies would still be more than double that of our current single cycle design.

A study of virtual-channel router implementations undertaken by Peh and Dally [12] suggests that an on-chip network typically requires 3 pipeline stages operating at a clock frequency of 20 FO4. While our actual switch and VC allocator implementations offer improvements over the published delay models, clocking and test overheads and internal buffering delays extend our clock period to around 35 FO4 in the final implementation. Improvements to the input port logic to reduce this delay are ongoing. Even at 35 FO4 our network's best case latency would be nearly half that of their reported pipelined design.

### *A. Global Synchronisation*

The speculative techniques at the heart of our router exploit the presence of a global clock. Global synchronisation offers regular snapshots of state and ensures the system proceeds in a deterministic fasion. This simplifies the implementation of the speculative scheduling mechanisms and ensures abort detection and handling mispredictions is relatively simple.

The cost of providing a low-skew high-frequency global clock is in both its complexity and the power it consumes. In many designs this cost may be considered to be too high. This has prompted asynchronous on-chip interconnect techniques to be investigated [2], [15]. While such approaches are promising, it is also possible to make similar trade-offs while retaining a synchronous router implementation. The first approach is to exploit known relationships between router clock signals while relaxing global synchronisation. Examples include source-synchronous communication and the use of clock predictive synchronisers. Global synchronisation may be relaxed further by generating clock pulses locally on demand or in a data-driven manner. This allows each router to operate at a rate dictated by the data it is transporting. This both reduces synchronisation overheads and provides a simple high-level approach to clock gating. Work in this area is ongoing. Techniques such as the DCG [5] may also be employed as previously discussed.

## VII. CONCLUSION

This paper has detailed the design of an on-chip network which can provide an efficient global communications infrastructure for future gigascale ICs. A speculative architecture is able to accurately produce datapath control signals one cycle in advance of their use. This enables both datapath and control logic to operate concurrently providing significant latency improvements over previously published work. A number of trade-offs between cycle time and speculation accuracy have also been introduced and evaluated.

The optimisations proposed are orthogonal to other well known techniques for boosting performance such as adaptive routing and are independent of the network topology selected.

## ACKNOWLEDGEMENTS

This work is supported by EPSRC (grant GR/L86326) and the Cambridge-MIT Institute.

## REFERENCES

[1] V. Agarwal, M.S.Hrishikesh, S. W. Keckler, and D. Burger. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA)*, 2000.

[2] J. Bainbridge and S. B. Furber. Chain: A delay-insensitive chip area interconnect. *IEEE Micro*, 22(5):16–23, 2002.

[3] 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.

[4] J. Dielissen, A. Radulescu, K. Goossens, and E. Rijpkema. Concepts and Implementation of the Philips Network-on-Chip. In *IP-Based SOC Design*, Grenoble, France, Nov 2003.

[5] S. Fairbanks and S. Moore. Self-timed circuitry for global clocking. In *Proceedings of the 11th International Symposium on Asynchronous Circuits and Systems*, 2005.

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

[7] R. Ho. *On-Chip Wires: Scaling and Efficiency*. PhD thesis, Stanford University, 2003.

[8] A. Jalabert, S. Murali, L. Benini, and G. D. Micheli. xpipesCompiler: A tool for instantiating application specific Networks on Chip. In *Design, Automation and Test in Europe (DATE)*, Paris, France, Feb 2004.

[9] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, and K. Asanovic. The Vector-Thread Architecture. In *31st International Symposium on Computer Architecture (ISCA-31)*, Munich, Germany, June 2004.

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

[11] R. D. Mullins, A. F. West, and S. W. Moore. Low-Latency Virtual-Channel Routers for On-Chip Networks. In *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA)*, 2004.

[12] 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.

[13] Semiconductor Industry Association. International technology roadmap for semiconductors (2004 update), 2004.

[14] M. B. Taylor et al. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In *The 31st Annual International Symposium on Computer Architecture (ISCA-31)*, Munich, Germany, June 2004.

[15] T.Felicijan and S.B.Furber. An Asynchronous On-Chip Network Router with Quality-of-Service (QoS) Support. In *Proceedings IEEE International SOC Conference*, pages 274–277, Santa Clara, CA, September 2004.