SWIFT: A SWing-reduced Interconnect For a Token-based Network-on-Chip in 90nm CMOS

Tushar Krishna[†], Jacob Postman[⋆], Christopher Edmonds[⋆], Li-Shiuan Peh[†], and Patrick Chiang[⋆]

[†]*Department of EECS*
Massachusetts Institute of Technology
{tushar, peh}@csail.mit.edu

[⋆]*School of EECS*
Oregon State University
{postmaja, edmondsc, pchiang}@eecs.oregonstate.edu

***Abstract*— With the advent of chip multi-processors (CMPs), on-chip networks are critical for providing low-power communications that scale to high core counts. With this motivation, we present a 64-bit, 8x8 mesh Network-on-Chip in 90nm CMOS that: a) bypasses flit buffering in routers using Token Flow Control, thereby reducing buffer power along the control path, and b) uses low-voltage-swing crossbars and links to reduce interconnect energy in the data path. These approaches enable 38% power savings and 39% latency reduction, when compared with an equivalent baseline network. An experimental 2x2 core prototype, operating at 400 MHz, validates our design.**

I. Introduction

Due to the diminishing returns and power inefficiencies of complex uniprocessor designs, computer architects now implement multi-core processor chips to provide improved performance with continued transistor scaling. The scalability and performance of these multi-core designs depend heavily on the on-chip communication fabric connecting the cores. Because simple buses are not scalable beyond a few cores, current commercial chips such as the 8-core IBM Cell [1] use a ring to connect their cores together. For core counts of several tens to even hundreds, Network-on-Chip (NoC) research prototypes like MIT's RAW [2], UT Austin's TRIPS [3], and Intel's TeraFLOPS [4] have adopted packet-switched mesh topologies because they are scalable and provide a tile-based, homogeneous network architecture that simplifies the interconnect design and verification. Unfortunately, network power becomes a major concern in these packet-switched, router-based NoCs. For instance, 39% of the 2.3W power budget of a tile is consumed by the network in Intel's TeraFLOPS. If not addressed, rising communications power between on-chip cores will be a major stumbling block for further scaling of processor counts.

The primary contributers to NoC power are the buffers (31% in RAW [2], 35% in TRIPS [3], 22% in TeraFLOPS [4]), which are typically built out of SRAM or register files; the crossbar switches (30% in RAW [2], 33% in TRIPS [3], 15% in TeraFLOPS [4]); and the core-core links (39% in RAW [2], 31% in TRIPS [3], 17% in TeraFLOPS [4]). Buffers are required to prevent collisions between packets that share the same links, and are managed by the *control path*. The crossbar switches and links form the *datapath* through which actual data transmission occurs. Since both components are necessary for realizing a high-throughput design, we make two key observations.

First, a router control path that tries to utilize the available link bandwidth efficiently, by adaptive routing and intelligent flow-control, can improve throughput, and reduce collisions between flits[1] at router input ports. This in turn can potentially allow flits to bypass buffering, even at high traffic rates, thereby reducing both latency and buffer read/write power (through obviating the need for buffer reads/writes and lowering the number of buffers needed for sustaining required bandwidth). While there have been several proposals from academia with this goal [5], [6], none have been demonstrated on an actual chip implementation, thereby enabling the evaluation of practical feasibility and overheads.

Second, when buffering is bypassed, the energy consumed by a flit moving through the network reduces to just the datapath traversal. Hence, implementing low-power crossbar and link circuits becomes even more critical towards reducing total network power. There have been several recent papers that demonstrate low-swing global links [7], [8] in isolation, but very few [9] have attempted to integrate these semi-global, ground-shielded, 1-2mm low-swing links between cores in an integrated SoC application. In addition, while there have been previously reported crossbar designs that utilize partial activation [9] and segmentation [10] to reduce crossbar capacitance, low-swing links have not been utilized directly within the crossbar (to the best of our knowledge).

Consequently, in this paper we present the SWing-reduced Interconnect For a Token-based Network-on-Chip (SWIFT NoC), an architecture that reduces overall NoC power by addressing both of these preceding observations. To reduce the power of the *control path*, we leverage Token Flow Control (TFC) [6], a NoC optimization that allows flits to dynamically bypass buffering at intermediate routers and chooses routes adaptively in response to contention. This provides buffer-less transmission from the source to the destination core. Along the *datapath*, we designed and integrated

This work was supported by NSF Grants CCF-0811375 and CCF-0811820, the MARCO Interconnect Focus Center, and the Gigascale Systems Research Center. We would like to thank the TAPO service for providing fabrication services for our 90nm testchip, and ARM for providing standard cell libraries and memory generators. We would also like to thank Amit Kumar from Princeton University for helping us in designing the SWIFT pipeline, and Kangmin Hu from Oregon State University for assisting us in post-layout verification.

[1]Packets are typically first broken down into smaller fixed-sized units called flits. This allows for varying packet sizes. Each packet has one head flit, some number of body flits and one tail flit.

custom reduced-swing crossbars and core-core links that substantially lower the data transmission power.

The router microarchitecture of SWIFT is designed in RTL and implemented using a conventional 90nm-CMOS standard cell library provided by the foundry. The synthesized router is manually integrated with the custom crossbar and links. The targeted NoC is an 8x8 2-D mesh, while the fabricated test chip is comprised of a 2x2 mesh subsection of four routers that verifies the design practicality and validates the simulated results.

This work makes the following contributions to the field of NoC design:

- This is the first fabricated NoC that incorporates unconventional micro-architectural optimizations, such as buffer-less bypassing and adaptive routing, and realizes them in a single-cycle router pipeline. This implementation results in 39% lower latency and 49% lower buffer power while sustaining the same throughput as a baseline[2].
- This is the first fabricated NoC that uses reduced-swing interconnects in both the crossbar and the links. Data path transmission power is lowered by 62%, when compared with a baseline[2] design.
- Our detailed discussions on the design choices we made and corresponding impact on critical path delay, area, power and overall network performance may present a useful case study for NoC researchers and designers.

The total power savings of the entire network was measured to be 38%.

The rest of the paper is organized as follows. Section II provides relevant background. Section III describes the design of the SWIFT router microarchitecture, while Section IV details the reduced-swing crossbar and link designs. Section V highlights the features added to the chip that facilitate testing and measurements. Section VI presents and analyzes our measured results. Section VII discusses related work and Section VIII concludes.

II. Background

This section gives a brief overview of a baseline router microarchitecture, and Token Flow Control [6].

A. Baseline Router Microarchitecture

High throughput NoC designs are usually packet switched [11], with routers at the intersection of links to manage the flow-control (buffering and arbitration for the crossbar switch) and routing of the flits. There are 3-pipeline stages (buffering, switch allocation and switch traversal) in our baseline router and are described in Section III-A. These are modeled similar to previous NoC prototypes like the TRIPS memory network [3] and Intel's TeraFLOPS [4].

B. Token Flow Control

The ideal energy-delay of a NoC traversal is that of just the on-chip links from the source to the destination. However, in order for multiple packets to share a link, buffers are necessary for temporary storage while the link is occupied. This introduces latency, and also consumes power due to the dynamic and leakage power of the register files used to implement buffers.

[2]The baseline router used for all comparisons was modeled similar to [4], [3] and is described in Section II-A.

No-load bypassing [11] allows incoming flits to skip being buffered and move straight to the output port if there are no other buffered flits at that input port. However, this technique only works when there is very little traffic. Token Flow Control (TFC) [6] goes a step further towards buffer-less traversal by pre-allocating buffers and links in the network using *tokens*, such that the majority of flits can bypass buffering even when there are other contending flits.

Each TFC router contains 2 pipelines, a *non-bypass pipeline* which is 3-stages long and the same as the baseline described in Section II-A, as well as a *bypass pipeline*, which is one-cycle and consists of just the crossbar traversal, followed by the link traversal. A high-level overview of TFC is shown in Figure 1(a).

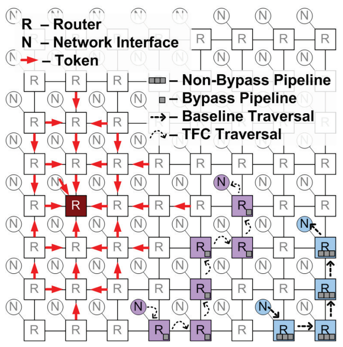
Each router receives hints about the availability of buffers in neighboring nodes in the form of *tokens*, which are passed between routers up to *d*-hops away from each other. The routing algorithm directs flits through paths with more tokens, implying less congestion, while the flow control mechanism allows the flit to try and bypass intermediate router pipeline stages by sending *lookahead* signals one cycle in advance. These lookaheads pre-allocate the crossbar at the next router such that the corresponding flit can use the *bypass pipeline*; and thereby traverse the crossbar and links (datapath) directly, instead of getting buffered. A lookahead is prioritized over locally-buffered flits so that a local switch allocation is killed if it conflicts with a lookahead. If two or more lookaheads from different input ports demand the same output port, a switch priority vector is used to decide the winner and the other lookaheads are killed. The flits corresponding to the killed lookaheads will then have to be buffered and go through the *non-bypass pipeline*.

TFC has three major advantages:

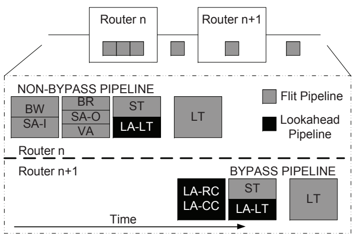
- **Lower latency**: Bypassing obviates the buffer write, read, and arbitration cycles.
- **Fewer buffers**: The ability of flits to bypass at all loads keeps the links better utilized while minimizing buffer usage, and reducing buffer turnaround times. Thus, the same throughput can be realized with fewer buffers.
- **Lower power**: Requiring fewer buffers leads to savings in buffer power (dynamic and leakage) and area, while bypassing further saves dynamic switching energy due to a reduction in the number of buffer writes and reads.

III. Design of the SWIFT Router

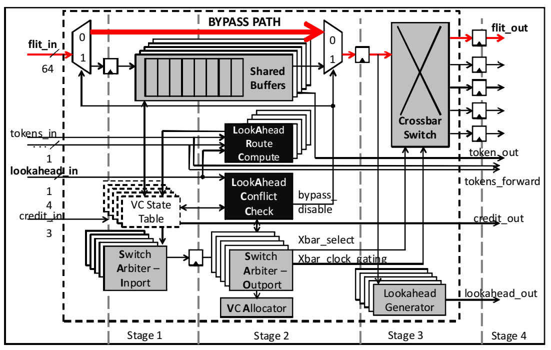
In this section, we describe the microarchitecture of the SWIFT router in detail. SWIFT is based on TFC, with modifications to the architecture proposed in [6] to make it implementable and meet timing. Each router supports two pipelines: a non-bypass pipeline, and a bypass pipeline, as described in Section II-B. Each flit/link is 64-bit wide. All flits try to use the bypass pipeline at all routers, by sending lookahead signals a cycle in advance. If the lookahead gets killed, they use the non-bypass pipeline. These pipeline



(a) NoC traversal with TFC



(b) Non-Bypass and Bypass Pipelines. BW, BR: Buffer Write/Read; SA-I, SA-O: Switch Alloc-Inport/Outport; VA: VC Allocation; ST: Switch Traversal; LT: Link Traversal; LA-RC: Lookahead Route Compute; LA-CC: Lookahead Conflict Check; LA-LT: Lookahead Link Traversal



(c) Router Microarchitecture

Fig. 1. Components of the SWIFT Network-on-Chip

interactions are highlighted in Figure 1(b), while the router microarchitecture is shown in Figure 1(c).

A. Non-Bypass Pipeline and Microarchitecture

The non-bypass router pipeline, which is essentially similar to the baseline router pipeline[3], is 3 stages long, followed by 1 cycle in the link. The various stages are described next, along with the blocks that are active during that stage.

Stage 1-Buffer Write (BW). The incoming flit is written into buffers at each input port, which were implemented with register files generated from the foundry memory generators. These input buffers are organized as a shared pool among multiple Virtual Channels (VCs)[4]. The addresses of the buffers are connected as a linked list. An incoming flit that requires a free buffer obtains the address from the head of the linked list, and every buffer that is freed up appends its address to the tail of the linked list. One buffer is reserved per VC in order to avoid deadlock. Compared to private buffers per VC which can be implemented as a FIFO, our shared buffer design incurs an overhead of storing the read addresses of all flits in the VC state table, but has the advantage of reducing the numbers of buffers required at each port to satisfy buffer turnaround time[5].

Stage 1-Switch Allocation-Inport (SA-I). An input VC is selected from each input port to place a request for the switch. This is implemented using V:1 round robin arbiters at each input port, where V is the number of VCs per port. Round robin arbiters are simple to implement [11] and ensure that every VC gets a chance to send a flit.

Stage 2-Switch Allocation-Outport (SA-O). The winners of SA-I at each input port place requests for their corresponding output ports. As no u-turns are allowed, there can be a maximum of 4 input ports requesting the same output port. These conflicts are resolved using 4:1 matrix arbiters, one for each output port. Matrix arbiters are used for fair allocation of the crossbar output port to all input ports [11]. Separating switch allocation into two phases of simpler arbitration, SA-I and SA-O, is a common approach to satisfy minimum cycle time constraints [11]. Note that a flit may spend multiple cycles in switch allocation due to contention.

Stage 2-VC Allocation (VA). At the end of SA-O, winning head flits are assigned an input VC for their next hop. (Body and Tail flits follow on the same VC). VC allocation in our design is a simple VC selection scheme, based on [13]. Each output port maintains a queue of free VCs at the input port of the next router. A switch request is allowed to be placed for an output port only if the router connected to that output port has at least one free input VC. The winning head flit of a switch output port, at the end of SA-O, picks up the free VC at the head of the queue and leaves. Thus there is no arbitration required, simplifying the VC allocation process. If the router receives a signal indicating a free VC from the next router, the corresponding VC id is enqueued at the tail of the queue. VA does not add any extra delay to the critical path since the updating of the queue and the computation of the next free VC id take place in parallel to SA-O.

Stage 2-Buffer Read (BR). Flits that won SA-I start a pre-emptive read of the buffers, in parallel to SA-O. This is because the register files require all input signals to be ready before the clock edge. If we wait until SA-O declares the winner of the switch output port, BR would have to be pushed to the next cycle, adding latency. The drawback of this is that there are wasted reads from the buffer which would consume power. We solve this by biasing SA-I to declare the same input VC as the winner until it succeeds to use the crossbar. This ensures that the same address is read out of BR to avoid additional switching power.

Stage 3-Switch Traversal (ST). The flits that won the switch ports traverse the crossbar switch.

Stage 4-Link Traversal (LT). The flits coming out of the crossbar traverse the links to the next routers.

B. Bypass Pipeline and Microarchitecture

The crux of our design is the 1-stage bypass router pipeline (ST), which allows flits to zoom from the source Network Interface (NIC) to the destination NIC, traversing just the crossbars and links, without getting buffered. We next describe two primary control logic components that facilitate this bypass action, namely *lookaheads* and *tokens* which were introduced in Section II-B.

Lookaheads. Each incoming flit is preceded by a lookahead signal which is 14 bits wide in our design. This is

[3]The primary difference is that the route computation in the baseline is done in parallel to the buffer write, while it is done by the lookaheads in SWIFT, as explained in Section III-B.

[4]VCs are like turning lanes on a highway, preventing flits with congested outports from blocking flits with free outports, to enhance throughput [12].

[5]Minimum number of cycles before which same buffer can be reused.

13-9	8	7-5	4	3-1	0
Outport	VCid	Y_hops	Y_direction	X_hops	X_direction

(a)

63-3	2-0
Data	Flit_type

(b)

Fig. 2. (a) Lookahead Payload (b) Flit Payload

also true for flits coming from the NIC and going into the NIC. The lookahead and flit payloads are shown in Figure 2. Lookaheads carry information that would normally be carried by the header fields of each flit, namely the destination coordinates, the input VC id, and the output port the corresponding flit wants to move out from.

Only the head flit's lookahead holds the destination information in the destination field; lookaheads of the body flits have zeros, while that of the tail flit arrives with its LSB 'high' in order to signal the arrival of a tail flit.

The lookaheads perform actions in parallel to the regular non-bypass pipeline. The lookahead pipeline stages are highlighted in black in Figures 1(b) and 1(c).

Lookahead Route Compute (LA-RC). The lookahead of each head flit performs a route compute (LA-RC) to determine the output port at the *next* router [14]. This is an important component of bypassing because it ensures that all incoming flits at a router already know which output port to request, and whether to potentially proceed straight to ST. We used West-first routing, an adaptive-routing algorithm that is deadlock free [11]. The adaptive-routing unit is a combinational logic block that computes the output port based on the availability of the tokens from 3-hop neighboring routers, which is explained later, rather than use local congestion metrics as indication of traffic.

Lookahead Conflict Check (LA-CC). The lookahead places a request for the output port in this stage. This happens in parallel to the SA-O stage of the non-bypass pipeline. As explained in Section II-B, a lookahead is given preference over the winners of SA-O in TFC, and conflicts between multiple lookaheads are resolved using a switch priority vector (which prioritizes each input port every 20 cycles in a round-robin manner). Muxes connect the input ports of the winning lookaheads directly to the crossbar ports. The corresponding flits that arrive in the next cycle *bypass* the buffers, as shown in Figure 1(c). The flits corresponding to the killed lookaheads, meanwhile, use the non-bypass pipeline.

Lookahead Link Traversal (LA-LT). While the flit performs its crossbar traversal, its lookahead is generated and sent to the next router.

Tokens. Tokens indicate non-congested paths, as described in Section II-B and Figure 1(a). We implemented the tokens as 1-bit wires going from a router to its three-hop neighborhood[6], via registers at each intermediate router. The wire being high/low determines whether the token is on/off. The token is set by the number of free buffers at each input port. If the number of free buffers goes lower than a threshold (which is three in order to account for flits already in flight), the token is turned off. We propagate tokens up to the network interfaces at each router, such that even source and destination routers can be bypassed. Each router receives

[6]Three was chosen based on experiments in [6].

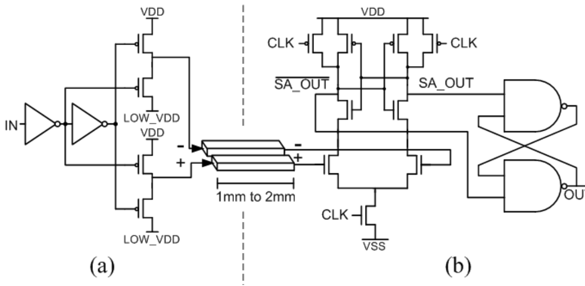


Fig. 3. (a) Reduced Swing Driver (RSD) (b) Sense Amplifier and SR Latch Receiver

a total 23-bits of tokens[7] and these act as inputs to the combinational block that performs the route computation, which was explained earlier.

IV. Low-Voltage Swing On-Chip Wires

SWIFT's bypass pipeline is comprised of two stages, switch and link traversal, corresponding to the datapaths through the crossbar switch and the core-to-core interconnect respectively. Both are critical NoC components and major contributors to power and area budgets, amounting to 32% of the network power and nearly 10% of total chip area in [4].

Unlike locally connected logic cells that communicate through short, locally-routed wires, crossbar and link interconnect exhibit long distances and close inter-wire coupling, such that dynamic power consumption is dominated by large wire capacitances rather than gate input capacitances. To reduce the power required to drive these large capacitive wire loads, reduced-voltage swing signaling was implemented using dual voltage supply differential reduced-swing drivers (RSD), followed by a simple sense-amplifier receiver as shown in Figure 3. The lower supply voltage is generated off-chip, allowing for signal swings to be adjusted easily during testing as the difference between the two supplies. In practice, voltage supplies 0.2V to 0.4V below the core logic supply are often already available on-chip for SRAM caches or other components that operate at a reduced supply voltage. Occupying 7.8um^2 and 15.2um^2 respectively, the same driver and receiver are used in both the crossbar and link designs.

While differential signaling approximately doubles the wire capacitance of each bit by introducing a second wire, it removes the necessity of multiple buffer stages and enables the use of reduced voltage swings, resulting in quadratic power savings in the datapath. For example, if the power required to drive a wire is given by Eq. (1), then the power required to drive the differential wires at 200mV is approximately given by Eq. (2).

$$P_{(swing=1.2V)} = \frac{1}{2} C_{wire} V^2 f \tag{1}$$

$$P_{(swing=200mV)} = \frac{1}{2}(2C_{wire})\frac{1}{36}V^2 f = \frac{1}{18}P_{(swing=1.2V)} \tag{2}$$

Hence, reducing the voltage swing from 1.2V to 200mV results in greater than 94% reduction in the power required to drive the interconnect wire. At 200mV swing, more

[7]There are a total of 36 tokens in a 3-hop neighborhood and one is received from the local port, as can be inferred from Figure 1(a). However, the West-first algorithm in our design allowed us to remove tokens from the west neighborhood since a packet has to go west irrespective of token availability.

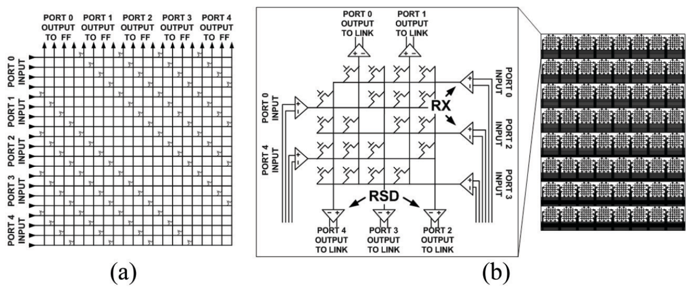


Fig. 4. (a) Simple Grid Crossbar (b) This Design: Crossbar with distributed low-swing bit cells

than 98% of the crossbar and link power is consumed by the clock distribution, switch selection signals, driver input capacitance, and the sense amplifiers used to resolve the low-voltage swing signals. Further reductions in voltage swing require either larger transistors or offset correction in the receiver to overcome input offset, diminishing the return on additional voltage swing reduction in the datapath. Therefore, area-efficient sense amplifiers comprised of near-minimum sized transistors and with approximately 100mV simulated 3σ offset are used rather than larger or more complex devices that achieve better input sensitivity.

A. *Reduced-Swing Link*

A significant body of work exists exploring high-speed, energy-efficient, on-chip interconnects that attempt to achieve the lowest mW/Gbps on narrow interconnect wires across distances up to 10mm. However a typical 2D-mesh NoC is likely to require wide, parallel links spanning a distance of just 1-2mm [4], limiting the practicality of previously proposed on-chip interconnects. For instance, while [7] and [8] achieve 3Gbps and 4Gbps on a 10mm link, the transceivers occupy 887um^2 and 1760um^2 respectively. By contrast, in this design, the combined driver and receiver area is only 23um^2, allowing the many transceivers required for a wide data bus to be realized in an acceptable area footprint.

A major concern for the proposed interconnect is susceptibility to crosstalk interference from digital logic signals on lower metal layers, as reduced-swing signals are particularly sensitive to coupled noise from nearby full-swing wires. To address this, shielding wires were routed on M6 between and in parallel to the differential pairs on M7. This differential-mode shielding approach adds less capacitance to the signal wires than routing ground shielding under the entire link, while still minimizing crosstalk from signals that would couple asymmetrically on to one of the differential wires. Worst case simulated differential mode crosstalk from a 1mm full swing aggressor signal is reduced from 128mV without shielding to 29mV with shielding, providing sufficient voltage margin to preserve signal integrity at 200mV signal swing and 100mV receiver input offset.

B. *Reduced-Swing Crossbar*

An obvious approach to the crossbar layout is to route a grid of vertical and horizontal wires with pass-gates or tri-state buffers at their intersection points, as shown in Figure 4(a). While simple, this approach suffers from a number of major disadvantages including poor transistor density, low bandwidth and a n^2 bit-to-area relationship. Higher crossbar speeds and improved density can be achieved using mux-based switches that place buffered muxes throughout the area of the crossbar or by implementing crossbar pipelining to improve speed by allowing sections of the wire load to be driven in separate clock cycles. While simple to implement in digital design flows, both approaches introduce additional loading in the form of increased fanout buffering and clock distribution that results in increased power consumption.

Based on these observations, the crossbar implemented in this design improves both performance and energy efficiency by replacing crossbar wires with low-swing interconnect. This approach seeks to drive as much of the large wire capacitances of the crossbar as possible with a reduced voltage swing, without introducing additional clocked elements or buffers. Implemented as a bit-sliced crossbar, each of the 64-bits in each of the five input buses is connected to a one-bit wide, 5-input to 5-output crossbar, along with the corresponding bits from each of the other four ports. An 8x8 grid is then patterned out of 64 of these bit-cell crossbars in order to construct a 64-bit wide, 5x5 crossbar.

Each crossbar bit-slice consists of 5 sense amplifiers, 20 pass-gates and 5 RSDs as shown in Figure 4(b). Each of the five reduced-swing differential inputs is driven at the periphery of the crossbar by an RSD connected to the output of the router logic. At the positive clock edge, each of the five low-swing differential inputs is converted to full-swing logic by the sense amplifier and drives a short 6um wire through a pass-gate transistor, and then into the interconnect RSD at one of the four possible output ports (U-turns are not allowed). The receiver, which consists of a near-minimum sized, 9-transistor sense-amplifier followed by a minimum-sized NAND-SR latch, acts as a DFF with low-swing differential inputs, replacing the flip-flop that would otherwise reside at the output of the crossbar-traversal pipeline stage. Like mux-based crossbars, this crossbar topology results in irregular datapaths across the 64b parallel interconnect, requiring that the maximum crossbar speed be bounded by the longest datapath delay through the crossbar. Full-swing select signals are routed on M1 and M2, differential data signals are routed on minimum width wires on M3-M5, and a separate clock wire is routed on M7 for each port. The clock is custom-routed to closely match the worst case RC delay of the datapath in order to minimize clock skew.

V. Design features for Testability

In this section, we describe the various features added to the design to facilitate testing and measurement.

Network Interfaces (NIC). Each router is connected to a Local Network Interface (L-NIC) which houses a *traffic injector* and a *traffic receiver*.

The *traffic injectors* generate uniform random traffic (traffic to destinations that are randomly generated using a PRBS generator), at an injection rate specified via a scan chain. They also generate lookaheads based on tokens. To avoid the need for buffering, the traffic injectors generate packets one at a time. Each traffic injector uses a 32-bit counter that signifies the current time stamp, synchronized via the global

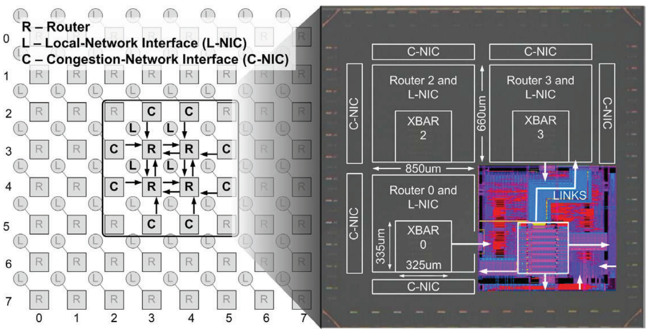
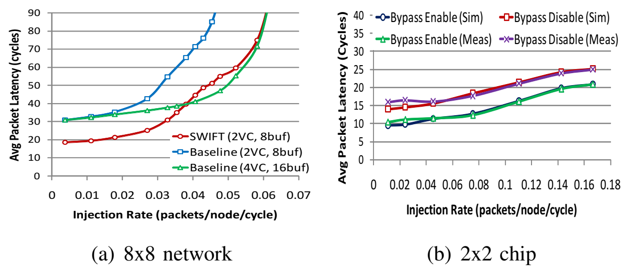


Fig. 5. SWIFT NoC overview and die photo with node 1 layout overlay
Specs: Freq=400MHz, Voltage=1.2V, Area=4mm^2, Transistors=688K



(a) 8x8 network (b) 2x2 chip

Fig. 6. Performance Results

reset signal. Data flits carry their generation time in the data field to aid in the calculation of flit latency.

The *traffic receivers* at the L-NICs receive flits and send back the free VC and On/Off token signals. In addition, they compute the total received packets and total packet latency.

For the 2x2 mesh test chip, in addition to the L-NICs, each router includes Congestion Network Interfaces (C-NICs) connected to its two *unconnected* ports at the edges of the mesh. These are added to enable simulations of the router with varying activity at *all* ports. The C-NICs also incorporate a traffic injector and a traffic receiver. They also send out tokens to emulate a larger mesh. Overall we have 12 traffic injectors on the chip, as shown in Figure 5.

In our test chip, the traffic injectors work in two modes: *non-congestion* and *congestion*. In the *non-congestion* mode, only the 4 traffic injectors in the L-NICs inject packets to be sent to the other three L-NICs in the 2x2 network. In the *congestion* mode, we try to mimic the traffic within the 2x2 network as if it were at the center of an 8x8 mesh. The 4 L-NICs and the 8 C-NICs inject traffic meant for the other 63 nodes in the hypothetical 8x8 network. Injection rate at the C-NICs is set at double the L-NIC injection rate to correctly model the channel load of one slice of an 8x8 mesh [11].

Error Detection. Specific error bits are set at each of the four routers and the four L-NICs if flits are received out-of-order or at incorrect destinations. All of these error bits are scanned out at the end of the simulation to determine the frequency at which the chip fails.

Scan Chains. A 1640-bit scan chain connects through the 12 NICs in the 2x2 chip to set the various parameters (PRBS seeds, the injection rates, and the modes of operation) at the beginning of a simulation. *Bypassing*, *clock-gating* and *congestion* modes can each be enabled or disabled. At the end of the simulation, the scan chain reads out the packet statistics (total packets received, total latency, simulation cycles) and the error bits.

VI. Results

In this section, we report the simulated/measured results of the SWIFT NoC, and compare it to a baseline NoC.

A. The SWIFT Network-on-Chip

SWIFT NoC parameters are shown in Table I. We chose 2 VCs and 8 buffers based on simulations. This is also the minimum number of buffers required per port with one buffer reserved per VC for deadlock avoidance, and six being the buffer turnaround time with on-off signaling between neighboring routers [11]. We used standard-cell libraries from ARM for synthesis. The 'Place and Route' of the router RTL met timing at 600MHz. The use of standard-cells, instead of custom layouts, limits our router design from running at GHz speeds like [4]. The custom crossbar and links are operational at 2GHz with 250mV swing, based on extracted simulations. In an actual Chip-Multi Processor, a tile consists of both the processing core, and a router, with the router accounting for less than a quarter of the tile area [4]. Since we do not integrate processing cores in this design, we place the routers next to each other for area reasons. This results in assymetric link lengths in our chip, but we size each driver and sense amplifier for 1*mm* links. A photo of our prototype test chip overlaid with the layout of node 1 is shown in Figure 5.

Due to the 4*mm*^{2} network size, we use a synchronous clock with tunable delays to each core rather than a globally-asynchronous, locally-synchronous approach as in [4], which was outside the scope of this work. The test chip operates at 400MHz at low load, however is limited to 225MHz at high-injection due to voltage supply droop caused by excessive resistance in the power distribution.

B. Evaluations

For a fair comparison of performance and power, we designed and laid out a baseline VC router and crossbar in the same 90nm technology, implementing the state-of-the-art pipeline described in Section II-A.

Performance. We evaluate the latency-throughput characteristics of SWIFT with uniform random traffic. Figure 6(a) shows the average simulated packet latencies of the networks as a function of injected load with the SWIFT 8x8 network displaying a 39% latency reduction at low load and achieving the same saturation throughput[8] as that of the baseline router, but with half as many buffers (eight versus sixteen). Accordingly, the baseline parameters were fixed as shown in Table I for a fair comparison.

The measured latency versus injection curves for our 2x2 prototype chip match those from RTL simulation, as shown in Figure 6(b), confirming the functionality of the chip. The 2x2 network delivers a peak throughput of 113 bits/cycle.

We do not provide an extensive performance evaluation of the TFC microarchitecture, as that was done in [6] under various synthetic and real traffic conditions.

Power. The power distributions for the SWIFT and baseline routers are shown in Figure 7. Because the L-NIC shares a supply with the router, and the crossbar shares a supply

[8]Saturation throughput is defined as the point where the latency becomes 3 times the no-load latency.

TABLE I
Comparison of NoC designs

Characteristic	Intel TeraFLOPS [4]	UT TRIPS [3]	Baseline*	SWIFT
Process parameters				
Technology	65nm	130nm	90nm	90nm
Chip Frequency	5GHz	366 MHz	NA	400 MHz
Router Area	$0.34mm^2$	$1.10mm^2$	$0.48^{\dagger}mm^2$	$0.48mm^2$
Network parameters				
Topology	8x10 mesh	4x10 mesh	8x8 mesh	8x8 mesh‡
Flit size	39b	138b	64b	64b
Packet Length	2 or higher flits	1-5 flits	5 flits	5 flits
Routing	Source	Y-X dimension order	X-Y dimension order	Adaptive (West-first)
Flow Control	Wormhole with VCs	Wormhole with VCs	Wormhole with VCs	TFC [6]
Buffer Management	On/Off	Credit-based	On/Off	TFC [6]
Router parameters				
Ports	5	6	5	5
VCs per port	2	4	4	2
Buffers per port	32	8	16	8
Crossbar	5x5	6x6	5x5	5x5

* Not fabricated, only laid out for comparison purposes.
† Baseline tile was given same area as SWIFT for place-and-route.
‡ 2x2 mesh for test chip.

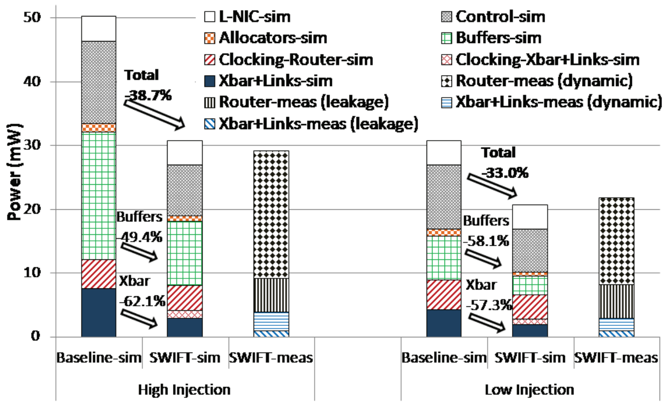


Fig. 7. Power breakdown of the SWIFT router and Baseline router

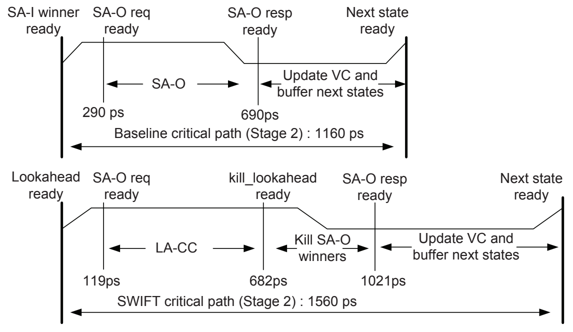


Fig. 8. Critical Paths of the SWIFT router and Baseline router

with the link, it was not possible to measure all of the blocks separately. Instead, post-layout, extracted simulations were performed to obtain an accurate relative breakdown of the power consumption of the different components which were compared and validated with chip measurements.

With the C-NICs, bypassing, and clock-gating all enabled, we performed experiments for both low (0.03 packets/node/cycle) and high (1 packet/node/cycle) injection rates. The low-swing drivers were set to 300mV signal swing, while VDD was set to 1.2V for a 225MHz clock frequency. We observe a power reduction versus the post-layout baseline router simulation of 49.4% in the buffers (control path) and 62.1% in the crossbar and links (datapath), giving a total power reduction of 38.7% at high injection, with the chip consuming a peak power of 116.5 mW.

Chip measurements with bypassing disabled showed that the feature reduces buffer power by 28.5% at high injection and by 47.1% at low injection, with greater power savings at low injection due to most flits being able to bypass buffer reads and writes.

Lookahead signals allow the crossbar allocation to be determined a cycle prior to traversal, making per-port, cycle-to-cycle *clock gating* possible. This was implemented in the crossbar's forwarded clock, reducing the crossbar clock distribution power by 77% and 47% and sense amplifier power by 73% and 43% at low and high injection respectively. Simulations show clock gating the links would save an additional 43% of the total crossbar and link power at low injection and 17% at high injection.

The average energy efficiency of the crossbar and link at the network saturation point was observed to be 64fJ/bit.

C. *Overheads*

Many of the architectural and circuit novelties in SWIFT are features that would enhance any baseline design, since at a high-level, they perform a more efficient allocation of network links and buffers, and enable low-power traversal. However, the comparison of the SWIFT router with the baseline would only be complete if the trade-offs and overheads are analyzed.

Timing. Figure 8 shows the the critical paths of the SWIFT and baseline routers, and it occurs during the SA-O stage in both cases[9]. The baseline is 400ps faster. Disecting the various components of the critical path gives interesting insights. The generation of the SA-O request signals, and the updation of the VC and buffer states is faster in SWIFT due to fewer number of VCs and buffers. The primary bottleneck in SWIFT turns out to be the 339ps incurred in killing the SA-O winners. The SWIFT router performs SA-O and LA-CC independently, in parallel, and then kills all the SA-O assignments which conflict with the lookahead assignments

[9]Note that our choice of the VC selection scheme made the delay of the VC allocation stage trivial, whereas if a separable VC allocator was used, like that in [4], that would have been the critical path.

for the same input or output ports of the crossbar, to maintain higher priority for lookaheads. In hindsight, we could have done this faster by combining LA-CC and SA-O: muxing out requests from SA-I winners (buffered flits) if lookaheads arrived at those input ports, and then arbitrating for output ports using SA-O, biasing it to give preference to lookaheads.

Area. The baseline and SWIFT routers primarily differ in the following hardware components: tokens, lookahead signals with corresponding bypassing logic, buffers, VCs, and crossbar implementation. We estimate the standard-cell area contributions of each of these components, and compare these in Table II. For the custom crossbar, we use the combined area of the RSD, switching devices and sense amplifier circuits as the metric to compare against the matrix crossbar's cell area. The total area of the SWIFT router is 25.7% smaller than the baseline router. This is the essential take-away of the SWIFT design: the 8% extra circuitry for tokens and bypassing, in turn results in a 11.2% reduction in buffer area and 49.3% reduction in control area (due to fewer VCs and corresponding smaller decoders and allocators) required to maintain the same peak bandwidth, thereby reducing both area and power!

The SWIFT NoC also has some wiring overheads. The 23-bit *token* signals from the 3-hop neighborhood at each router add 7% extra wires per port compared to the 64-bit datapath. The 14 lookahead bits at each port carry information that is normally included in data flits and so are not strictly an overhead[10]. Additionally, while Table II highlights that the active device area of the reduced swing custom crossbar is less than that of a synthesized design, differential signaling requires routing twice as many wires as well as potentially requiring an additional metal layer if shielding is required for the application.

Power. The west-first adaptive routing combinational logic (using tokens), the lookahead arbitration logic, and the bypass muxes account for less than 1% of the total power consumed in the router, and are thus not a serious overhead. This is expected because the allocators account for only 3% of the total power, consistent with previous NoC prototypes. The control power of the SWIFT NoC is infact observed to be 37.4% lower than the baseline NoC because of fewer buffers and VCs (hence smaller decoders and muxes) required to maintain the same throughput.

The overall control power of SWIFT is about 26% of the router power, as can be seen from Figure 7. The high percentage is primarily due to a large number of flip-flops in the router, many of which were added conservatively to enable the design to meet timing, and could have been avoided or replaced by latches. In addition, the shared buffers require significant state information to track free buffer slots and addresses for each flit that adds more flip-flops.

VII. Related Work

MIT's RAW [2], UT Austin's TRIPS [3], and Intel's TeraFLOPS [4] are previously fabricated mesh NoCs. RAW uses four physical 4x4 mesh networks with no VCs. The TRIPS memory-memory network is a 4x10 mesh with VC routers, and Intel's TeraFLOPS is an 8x10 mesh with VC routers. Table I compares our baseline and SWIFT network designs with the TRIPS and TeraFLOPS network, as they are closest to our baseline VC router. The SWIFT NoC is unique because it is the first fabricated NoC to simultaneously address network latency (buffer-less bypassing, single-cycle router), throughput (adaptive routing, buffer-less bypassing at all traffic through tokens), and power (buffering, low-swing interconnect circuits, clock gating). Furthermore, unlike prior NoC designs that use VCs, the SWIFT design uses shared buffers and VC selection to simplify VC management.

[10]The flit width can then either be shrunk or packets can be sent using fewer flits, both not incorporated in our results, which will further enhance SWIFT's area or performance benefits over the baseline.

TABLE II
Area Comparison (Absolute and Percentage)

Component	SWIFT Area, % of total	Baseline Area, % of total
Tokens	1,235 um^2, 0.82%	0
Bypass	10,682 um^2, 7.10%	0
Buffers	72,118 um^2, 47.94%	81,231 um^2, 40.08%
Crossbar	15,800 um^2, 10.50%	21,584 um^2, 10.64%
Control	50,596 um^2, 33.63%	99,856 um^2, 49.27%
Total	150,431 um^2, 100%	202,671 um^2, 100%

VIII. Conclusions

In this work, we presented the SWIFT NoC (SWing-reduced Interconnect For a Token-based Network-on-Chip) which enables bufferless traversal in the network through a reduced-swing datapath. We experimentally demonstrated a 90nm-CMOS test chip for a 2x2 mesh network that validates the benefits of this router architecture and circuitry.

References

[1] J. A. Kahle *et al.*, "Introduction to the Cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, no. 4/5, 2005.
[2] M. B. Taylor *et al.*, "The raw microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.
[3] P. Gratz *et al.*, "On-chip interconnection networks of the TRIPS chip," *IEEE Micro*, vol. 27, no. 5, pp. 41–50, Sept. 2007.
[4] Y. Hoskote *et al.*, "A 5-GHz mesh interconect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Sept. 2007.
[5] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: Towards the ideal interconnection fabric," in *Proc. Int. Symp. Computer Architecture*, June 2007.
[6] A. Kumar, L.-S. Peh, and N. K. Jha, "Token flow control," in *Proc. 41st International Symposium on Microarchitecture*, November 2008.
[7] J. Bae, J.-Y. Kim, and H.-J. Yoo, "A 0.6pj/b 3gb/s/ch transceiver in 0.18 um cmos for 10mm on-chip interconnects," in *IEEE International Symposium on Circuits and Systems*, May 2008, pp. 2861–2864.
[8] B. Kim and V. Stojanovic, "A 4gb/s/ch 356fj/b 10mm equalized on-chip interconnect with nonlinear charge-injecting transmit filter and transimpedance receiver in 90nm cmos," *IEEE International Solid-State Circuits Conference,*, pp. 66–68, Feb. 2009.
[9] K. Lee, S.-J. Lee, and H.-J. Yoo, "Low-power network-on-chip for high-performance soc design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, pp. 148–160, 2006.
[10] H.-S. Wang, L.-S. Peh, and S. Malik, "Power-driven design of router microarchitectures in on-chip networks," in *Proc. Int. Symp. Microarchitecture*, Nov. 2003, pp. 105–116.
[11] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Pub., 2003.
[12] W. J. Dally, "Virtual-channel flow control," in *Proc. Int. Symp. Computer Architecture*, May 1990, pp. 60–68.
[13] A. Kumar *et al.*, "A 4.6Tbits/s 3.6GHz single-cycle noc router with a novel switch allocator in 65nm CMOS," in *Proc. Int. Conf. Computer Design*, Oct. 2007.
[14] M. Galles, "Scalable pipelined interconnect for distributed endpoint routing: The SGI SPIDER chip." in *Proc. Hot Interconnects 4*, Aug. 1996.