SMART: A Single-Cycle Reconfigurable NoC for SoC Applications

Chia-Hsin Owen Chen[†], Sunghyun Park[‡], Tushar Krishna[†], Suvinay Subramanian[†],
Anantha P. Chandrakasan[$], Li-Shiuan Peh[†]
Dept. of Electrical Engineering and Computer Science, Massachusettes Institute of Technology, Cambridge, MA 02139
[†]{owenhsin, tushar, suvinay, peh}@csail.mit.edu, [‡]{pshking}@mit.edu, [$]{anantha}@mtl.mit.edu

***Abstract*—As technology scales, SoCs are increasing in core counts, leading to the need for scalable NoCs to interconnect the multiple cores on the chip. Given aggressive SoC design targets, NoCs have to deliver low latency, high bandwidth, at low power and area overheads. In this paper, we propose Single-cycle Multi-hop Asynchronous Repeated Traversal (SMART) NoC, a NoC that reconfigures and tailors a generic mesh topology for SoC applications at runtime. The heart of our SMART NoC is a novel low-swing *clockless* repeated link circuit embedded within the router crossbars, that allows packets to potentially bypass all the way from source to destination core within a single clock cycle, without being latched at any intermediate router. Our clockless repeater link has been proven in silicon in 45nm SOI. Results show that at 2GHz, we can traverse 8 mm within a single cycle, i.e. 8 hops with 1 mm cores. We implement the SMART NoC to layout and show that SMART NoC gives 60% latency savings, and 2.2X power savings compared to a baseline mesh NoC.**

I. Introduction

Systems-on-Chip (SoCs) have started adding more and more general-purpose/application-specific IP cores with the emergence of diverse compute intensive applications over the past few years [1], [2], and this has intensified with the proliferation of smart phones [3]. Networks-on-chip (NoCs) are used to connect these cores together, and routers are used at crosspoints of shared links to perform multiplexing of different messages flows on the links.

To reduce on-chip latency, one approach has been to tailor the NoC topology to match application communication patterns at *design time*. Examples include Fat Tree [4], Star-Ring [5], Octegon [2], high-radix crossbar [6], and so on. If coupled with sophisticated link designs such as [7]–[10], these NoCs can realize a single cycle transmission between distant cores. However, this requires knowledge of all applications and their communication graphs at design time to be able to pin these dedicated express links to specific pairs of dedicated cores, and assumes sufficient wiring density to support dedicated links between all communicating cores.

The alternate approach has been to use a scalable topology at design time, such as a 2D Mesh connecting a collection of generic IPs (such as ARM processors), then reconfigure it at *run time* to match application traffic. Since router delays can vary depending on congestion [1], [11], some prior research [12]–[16] has proposed pre-reservation of (parts of) the route to provide predictable and bounded delays. These works perform an *offline* computation of contention free routes, allowing flits[1] to bypass queues and arbiters at routers where there is no conflict between the routes of different flows. This paper pushes this idea to the extreme: we enable flits to potentially incur a single-cycle delay all the way from the source to the destination, thus providing a virtually tailored topology within a shared mesh. We call this approach SMART, Single-cycle Multi-hop[2] Asynchronous Repeated Traversal. We present a novel low-swing link circuit that uses clockless repeaters to allow propogation of signals across multiple-mm within a cycle, at low energy. We replace conventional links in the network by these SMART links at design time. We also present a tool flow to perform *online* reconfiguration of network routers at runtime, to enable different applications to run on tailored topologies. Figure 1 shows an overview of our design, where a network reconfigures into 3 different topologies for 3 different applications.

In this work, we make the following contributions:

- First, we present a chip to show the benefit of a novel low-swing clockless repeated link design for fast multi-mm propagation. Simulation results show that 8 mm can be traversed in a cycle at 2 GHz.
- Second, we present a reconfigurable NoC architecture, SMART, integrated with the proposed link design that allows single-cycle traversal between distant cores.
- Lastly, we implement a 4x4 SMART mesh and evaluate the impact on multiple SoC applications and show that we are only 1.5 cycles off in performance from a *dedicated* topology for that application. Compared to a state-of-the-art 3-cycle mesh router, we observe 60% saving in packet latency and 2.2X reduction in power consumption.

The paper is organized as follows: we first describe the related work in Section II. Then we explain the proposed link design in Section III, and present the architecture of SMART NoC in Section IV. Section V shows the implementation details. Section VI demonstrates some case studies on a 4x4 SMART NoC, and Section VII concludes.

II. Related Work

Reconfigurable topologies. Prior works on reconfigurable NoCs motivated the need for application-specific topology

[1]A flit is a sub-unit of a packet, and is sized to be equal to the link width.

[2]We define hop to be the distance between two IP blocks in the physical layout. We assume 1-hop = 1mm in this paper from place-and-route of a Freescale PowerPC e200z7 core in 45nm.

978-3-9815370-0-0/DATE13/©2013 EDAA

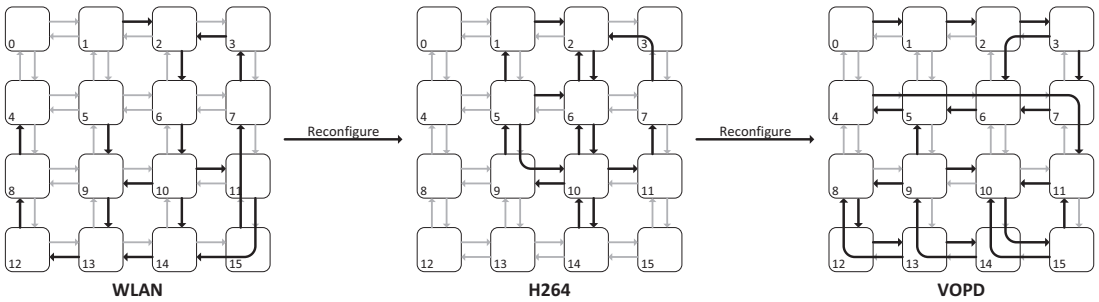


Fig. 1: Mesh reconfiguration for three applications. All links in bold take one-cycle.

reconfiguration and proposed various NoC architectures that support reconfiguration. Application-Aware Reconfigurable NoC [12] adds extra switches next to each router (a second crossbar in principle), and presets static routes based on application traffic. VIP [13] supports reconfiguration virtually, by prioritizing a virtual channel (VC) in the network to always get access to the crossbars, enabling single-cycle-per-hop for flits on this VC. ReNoC [14], [15] adds an extra topology switch (a set of muxes) at the output ports for each router and presets them to enable static routes in the network before the application is run. Skip-links [16] dynamically reconfigures the topology based on the traffic at each router when application is run, and sets up the crossbars to allow flits to bypass buffering and arbitration stages at intermediate routers.

All these prior works reconfigure the topology by enabling some way of bypassing buffering and arbitration at routers. ReNoC is the closest to our work in that it also avoids latching flits at each router. However, none focused on pushing latency down further to traversing multiple hops in a cycle at high frequency. As high-performance SoCs emerge, we believe that applications will be increasingly sensitive to communication latency. SMART is the first work to demonstrate a novel clockless repeater circuit that enables single-cycle traversal over multiple hops at GHz frequencies, leverage these for reconfiguring a NoC to support single-cycle communications for applications, and implement the NoC to layout.

Low-swing signaling. Signaling at low voltage swing is a well-known design technique to efficiently drive a highly-capacitive load in both off-chip and on-chip interface circuits. In general, the low-swing technique can lower energy consumption and propagation delay at the cost of a reduced noise margin [17]. Most existing low-swing on-chip interconnects (lower supply voltage drivers [17], [18], cut-off drivers [18]–[20] and charge sharing techniques [21]–[23]), however, are optimized for low-power signaling to maximize energy efficiency at the link level, leading to propagation delay slacks caused by reduced driving current. While pre-emphasis techniques such as equalization [7]–[9] can generate energy-efficient low-swing signaling along with the inherent channel loss of global links without the delay slack, their application to a mesh NoC that offers path diversity only through short router-to-router links is limited due to huge area overheads of the equalized drivers, poor bandwidth density of differential wiring and lack of point-to-point global wiring space.

To the best of our knowledge, there have been no previous works on low-swing link circuit design optimized for our SMART NoC design goal at the system level: *fast propagation delay through multiple routers in a mesh NoC with reconfigurability*. This design goal prompted our voltage-locked repeater circuit that allows signals to be asynchronously repeated at every router with reduced delay.

III. SMART Link

As discussed in Section II, most prior works explore single-cycle-per-hop, which is essentially a link connecting the source and destination router with several clocked repeaters inserted in the middle. However, wire delay is much shorter than a typical router cycle time (500 ps for a 2 GHz clock frequency), which means that it is possible to traverse multiple hops in a single cycle. For example, a full-swing repeated wire delay is only around 100 ps/mm[3].

We present a novel asynchronous low-swing repeater circuit, voltage lock repeater (VLR), for single-cycle multi-hop link traversal which forms the basis of our SMART NoC. Our proposed low-swing link stretches the maximum distance that a full-swing repeated link can span in a cycle at lower energy.

Figure 2 shows the schematic of VLR. We choose a single-ended design over double-ended design because of lower wire capacitance per bit and higher data density. The circuit locks the node X voltage to swing near the threshold voltage of INV1x without the decrease in driving current, enabling lower delay of the next symbol propagation delay. The delay cell in the feedback path generates transient overshoots at node X, resulting in lower repeater propagation delay and larger noise margin without significant energy overhead. The low-swing voltage level is determined by transistor sizes and link wire impedance[4]. Careful transistor sizing and extracted simulations are required to prevent oscillation and static current through the RxP-RxN path in all possible process corners.

While the proposed low-swing repeater does not require clocking power and differential signaling, it has static current paths between two consecutive repeaters, TxP-wire-RxN for logic High and TxN-wire-RxP for logic Low. It should be

[3]Based on the measurements of our chip with min DRC pitch assumed.

[4]V_{high} is given by link wire resistance, TxP's on-state resistance and RxN's on-state resistance while V_{low} is determined by link wire resistance, TxN's on-state resistance and RxP's on-state resistance.

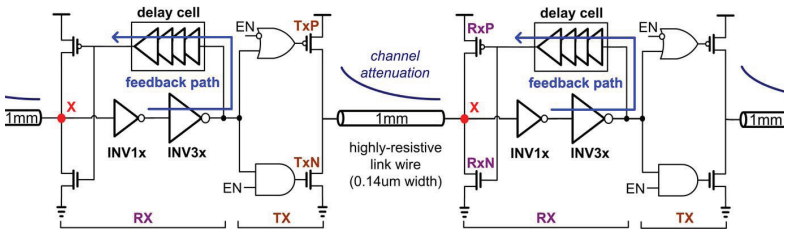


Fig. 2: Proposed clockless low-swing voltage-locked repeater (VLR) for single-cycle multi-hop link traversal.

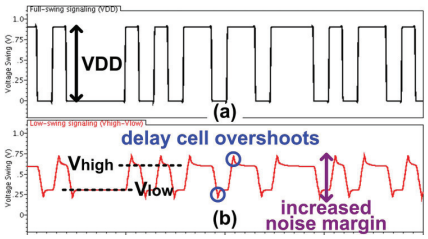


Fig. 3: Simulated waveforms at 6.8 Gb/s: (a) full-swing and (b) low-swing.

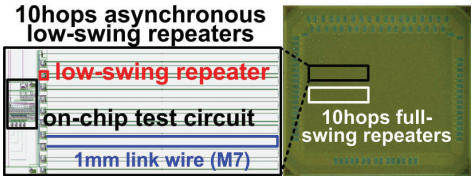


Fig. 4: Test chip die photograph in 45nm SOI CMOS.

TABLE I: Simulation results of max number of hops per cycle

Data Rate	1 Gb/s	2 Gb/s	3 Gb/s
Full-swing*	13 (103 fJ/b/mm)	6 (95 fJ/b/mm)	4 (84 fJ/b/mm)
Low-swing*	16 (128 fJ/b/mm)	8 (104 fJ/b/mm)	6 (87 fJ/b/mm)
Data Rate	4 Gb/s	5 Gb/s	5.5 Gb/s
Full-swing**	4 (98 fJ/b/mm)	3 (89 fJ/b/mm)	3 (85 fJ/b/mm)
Low-swing**	7 (132 fJ/b/mm)	6 (107 fJ/b/mm)	5 (96 fJ/b/mm)

*is resized and optimized for low-frequency (2 GHz) and wider wire spacing.
**is the same circuit as in the fabricated chip with wider wire spacing.

noted, however, that the static energy is much less than a conventional continuous-time comparator since the static current paths include a highly-resistive link wire. Also, switching off the enable signal (EN) when the link is not used help eliminate unnecessary static power.

To explore the high-frequency performance and energy efficiency of the proposed low-swing repeater, a test chip in 45nm SOI CMOS was fabricated and measured. A VLR was embedded at every mm along a 10mm interconnect. Figure 4 shows a die photo of our chip that also includes equivalent full-swing repeaters and an on-chip test circuit.

In terms of bandwidth, the proposed VLR repeaters achieve the maximum data rate of 6.8 Gb/s with 4.14 mW power consumption (i.e. 608 fJ/b energy efficiency) for 10-hop (10 mm) link traversal, maintaining bit error rate (BER) below 10^{-9}. On the other hand, the equivalent full-swing repeaters can transmit 5.5 Gb/s data at most, with BER which is less that 10^{-9}, consuming 4.21 mW (i.e. 765 fJ/b), whereas VLR consumes 3.78 mW (i.e. 687 fJ/b) at the same data rate. Latency wise, experiment results show that the delay of a link with VLRs is around 60 ps/mm, whereas the delay of a link with full-swing repeaters is around 100 ps/mm.

In a SoC, the maximum clock frequency is usually limited by the core and router critical path rather than the link. We thus re-optimize the transistor sizes and wire spacing of our circuits for a lower clock frequency of 2 GHz to meet our system-level design goal of *single-cycle multiple-hop link traversal* without unnecessary energy consumption and the simulation results are shown in Table I[5]. At 2 GHz, 8-hop (8 mm) link can be traversed in a cycle at 104 fJ/b/mm.

IV. Architecture of the SMART NoC

Here, we present the architecture of the SMART NoC that can be tailored at runtime to enable single-cycle communication between any pair of cores for different applications.

SMART Crossbar. The SMART crossbar is the primary building block in a SMART NoC that enables straight and turning paths within the network. Figure 5 shows the architecture of such a crossbar integrated with the voltage lock repeater describe in Section III. The idea is to insert a crossbar between the Rx and Tx components of each repeater. The data sent on the link will first be converted to full-swing (Rx), traverse the full-swing crossbar, then converted back to low-swing again (Tx) and forwarded to the next hop.

Router Microarchitecture. We integrate the SMART crossbar with a conventional router (which comprises of buffers and arbiters). As shown in Figure 6, in addition to the input buffers of the router, the crossbar is also fed by the incoming links to support single-cycle bypass paths. For each direction, an extra multiplexer is added to multiplex the crossbar input port between the input buffer and the incoming link. If the multiplexer is preset to connect the incoming link to the crossbar[6], a bypass path is enabled: incoming flits move directly to the crossbar, traverse it to the outgoing link, and do not get buffered/latched in the router. On the other hand, if the multiplexer is set to connect the input port buffer, the bypass path is disabled, which happens when the output link is shared across communication flows from different input ports. In this case, an incoming flit enters the router, places requests for the output port determined by its preset route, and moves to the crossbar upon successful arbitration.

We design a 3-stage router. In stage 1, the incoming flit gets buffered and generates an output port request based on the preset route in its header. In stage 2, all buffered flits arbitrate for access to the crossbar. In stage 3, flits traverse the crossbar and output link upon successful arbitration.

Routing. Given an application communication graph, one can use NoC synthesis algorithms like NMAP [24] (see Section VI) to map tasks to physical cores and communication flows to static routes on a mesh. Figure 7 shows an example

[5]Smaller transistor sizes and 2X wider wire spacing than fabricated design.
[6]The crossbar signals also need to be preset to connect this input port to another output port

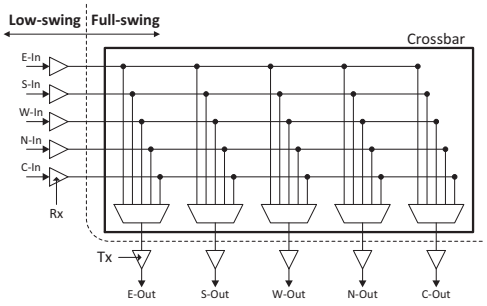


Fig. 5: One-bit SMART Crossbar.

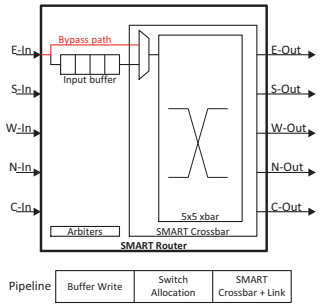


Fig. 6: SMART Router Microarchitecture and Pipeline.

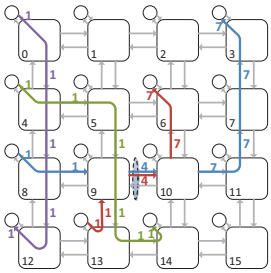


Fig. 7: SMART NoC in action with four flows. (The number next to each arrow indicates the traversal time of that flow.)

SMART NoC with preset routes for four arbitrary flows. In this example, the green and purple flows do not overlap with any other flow, and thus traverse through a series of SMART crossbars and links, incurring just a single-cycle delay from the source NIC to the destination NIC, without entering any of the intermediate routers. The red and blue flows, on the other hand, overlap over the link between routers 9 and 10, and thus need to be stopped at the routers before and after this link to arbitrate for the shared crossbar ports[7]. The rest of the traversal takes a single-cycle. It should be noted that before the application is run, all the crossbar select lines are preset such that they either always receive a flit from one of the incoming links, or from a router buffer.

Since the routes are static, we adopt source routing and encode the route in 2 bits for each router. At the source router, the 2-bit corresponds to East, South, West and North output ports, while at all other routers, the bits correspond to Left, Right, Straight and Core. The direction Left, Right and Straight are relative to the input port of the flit. In this work, we avoid network deadlocks by enforcing a deadlock-free turn model across the routes for all flows.[8]

Flow Control. In a conventional hop-by-hop traversal model, a flit gets buffered at each hop. Thus, a router only needs to keep track of the free VCs/buffers at its neighbors before sending a flit out. Without loss of generality, we adopt the virtual cut-through flow control to simplify the design. A queue is maintained at each output port to track the available free VCs at the downstream router connected to that output port. A free VC is dequeued from this queue before a head flit is sent out of the corresponding output port. Once a VC becomes free at the downstream router, the router sends a credit signal (VCid) back to the upstream router which enqueues this VCid into the queue.

In the SMART NoC, a flit could traverse multiple hops and get buffered, bringing up challenging flow control issues. A router needs to keep track of free VCs at the endpoint of an arbitrary SMART route, though it does not know the SMART route till runtime. We solve this problem by using a reverse credit mesh network, similar to the forward data mesh network that delivers flits. The only overhead of the credit mesh network is a [log(# VCs) + 1 (valid)]-bit SMART crossbar added at each router. For example, if the number of VCs is 2, the overhead of the credit network is 2-bit wide crossbars. If a forward route is preset, the reverse credit route is preset as well. A credit that traverses multiple hops does not enter the intermediate routers and goes directly to the SMART crossbar which redirects it along the correct direction.

For example, in Figure 7, for the blue flow, credits from NIC3 are fowarded by preset credit crossbars at routers 3, 7 and 11 to router 10's East output port in a single-cycle without going into intermediate routers; credits from router 10's West input port are sent to router 9's East output port and credits from router 9's West input port are sent to NIC8.

The beauty of this design is that the router does not need to be aware of the reconfiguration and compute whether to buffer/forward credits. Since the credits crossbars act as a wrapper around the router, and are preset before the application starts, the credits automatically get sent to the correct routers/NICs. Thus, if a router receives a credit, it simply enqueues the VCid into its free VC queue. This free VC queue might actually be tracking the VCs at an input port of a router multiple hops away, and not the neighbor, as explained above.

V. Implementation Tool Flow

To demonstrate the feasibility of the SMART NoC architecture, we present a tool to build SMART NoCs. The tool takes network configurations as input (e.g., the dimension of the mesh, flit width, number of VCs and buffers), and generates the RTL description as well as the layout of the SMART NoC integrated with the proposed SMART link. We next describe how each component is generated.

Voltage Lock Repeater. To integrate the VLRs into the design, we implement a SKILL script to take 1-bit Tx/Rx layout and data width as input and place-and-route them regularly to multi-bit Tx/Rx blocks. Figure 8 shows an example of a 32-bit Tx block. We do not embed the VLRs in the crossbar as in [25] because that leads to high area overhead. Also, we do not use existing commercial place-and-route tools because these tools are often designed for general circuit blocks and cannot leverage the regularity property, adding unnecessary overhead. In addition, the script also generates the timing liberty format

[7]If flits from the red and blue flow arrive at router 9 at *exactly* the same time, they will be sent out serially from the crossbar's East output port.

[8]Deadlock can also be avoided by marking one of the VCs as an escape VC [11] and enforcing a deadlock-free route within that. The exact deadlock-avoidance mechanism is orthogonal to this work.

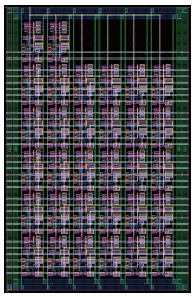


Fig. 8: 32-bit Tx block Layout

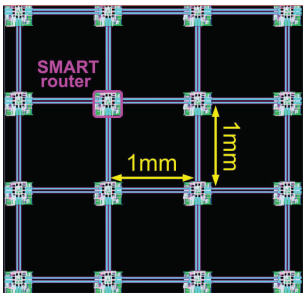


Fig. 9: Generated 4x4 NoC Layout

TABLE II: 4x4 NoC Configuration

Technology	45nm
V_{dd}, Freq	0.9 V, 2 GHz
Topology	4x4 mesh
Channel width	32 bits
Credit width	2 bits
Router ports	5
VCs per port	2, 10-flit deep
Packet size	256 bits
Flit size	32 bits
Header width	20 bits (Head), 4 bits (Body, Tail)

(.lib) and the library exchange format (.lef) files to allow the generated layout to be place-and-routed with the router.

SMART Router and NoC. Given router parameters, the tool generates the RTL description of the router in Verilog using an in-house parameterized library of various router components. The input/output ports are clock-gated to reduce unnecessary dynamic power consumption based on the preset signals, which are set before each application runs. We synthesize the router and place-and-route it along with VLRs. Next, we tile the routers and connect them as a mesh. Due to the limitation of the general routing tool that introduces unnecessary wiring overhead, we use custom TCL scripts to control the tool to generate links between the routers.

Reconfiguration Registers. To support SMART path reconfiguration for different applications, we encode the preset signals for crossbars and input/output ports into a double-word configuration register for each router. These registers are memory mapped such that these can be set by performing a few memory store operations. Before each application runs, these registers need to be set properly to suit the application's traffic characteristic. The network needs to be emptied while setting the registers. The values of the registers are determined based on the mapped flows on the mesh. Application developers need to prepend the application with memory store instructions to set the registers properly and the reconfiguration cost at runtime is just the amount of time to execute these instructions. For example, for a 16-node SMART NoC, there are 16 registers to be set which correspond to 16 instructions. If there is only 1 core that can perform the reconfiguration, a separate network (e.g. ring) is required to set these registers.

VI. Case Study

Configurations. We implemented a 4x4 SMART NoC and evaluated it with a suite of SoC applications. The configuration of the network is shown in Table II and the final layout is shown in Figure 9. It should be noted that the routers are assumed to be 1mm spaced and the black regions shown are reserved for the cores. We refer to this design as **SMART**.

We evaluate **SMART** against two baselines: **Mesh** and **Dedicated**. **Mesh** is a state-of-the-art NoC with no reconfiguration [11], where each hop takes 3 cycles in router and 1 cycle in link. **Dedicated** is a NoC with 1-cycle dedicated links between all communicating cores tailored to each application. While this has area overheads, we use this design as an ideal yardstick for **SMART**. All designs use the SMART links.

We generate synthetic traffic from 8 SoC task graphs, modeling a uniform random injection rate to meet the specified bandwidth for each flow[9]. We feed this traffic through post-layout simulation of the SMART NoC to get average network latency. We also use the VCD files from these simulations to estimate power using Synopsys Prime Power.

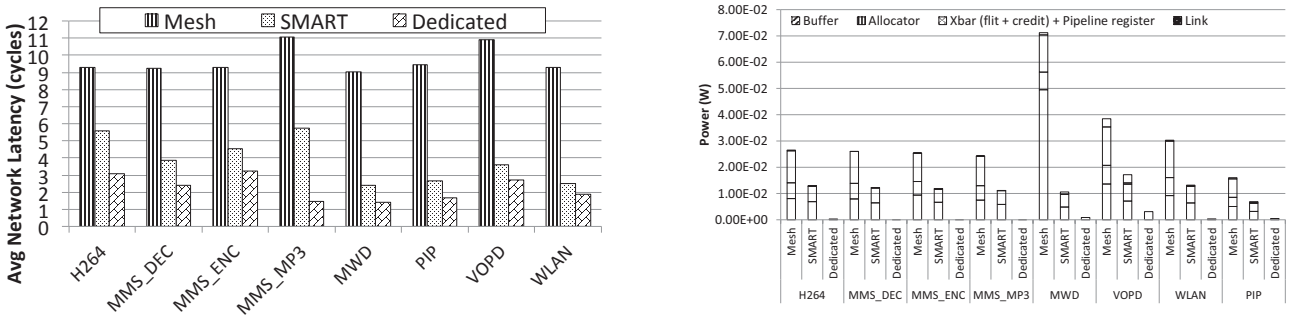
To determine the preset signals for each application, we take a task graph and adopt a modified NMAP [13] algorithm to map the tasks to phyical cores in the mesh. We first map the task with highest communication demand to the core with the most number of neighbors (i.e. middle of the mesh). Then, we pick a task that communicates the most with the mapped tasks and find an unmapped core that minimizes the chance of getting buffered at intermediate cores. This process is iterated to map all tasks to physical cores. As the tasks are mapped to the physical cores, the flows between tasks are also mapped to routes with minimum number of hops between cores. Note that since the reconfiguration process only involves a few memory stores, the overhead of the reconfiguration can be omitted.

Performance Evaluation. Figure 10a shows the average network latency across the applications for the baseline and SMART NoCs. Compared to the Mesh, SMART reduces network latency by 60.1% on average due to the bypassing of the complete router pipelines[10]. On average, SMART reduces the network latency to 3.8 cycles, which is only 1.5 cycles higher than that of the Dedicated 1-cycle topology. For PIP, VOPD and WLAN, the latencies achieved by SMART and Dedicated are almost identical. If there are multiple traffic flows to the same destination, they need to stop at a router at the destination to go up serially into the NIC, both in SMART and Dedicated. However, SMART is limited by the available link bandwidth in a mesh to multiplex all flows, while Dedicated has no bandwidth limitation. This allows Dedicated to have 2-4 cycles lower latency than SMART in H264 and MMS_MP3 where one core acts as a sink for most flows, while another acts as the source for most flows, thus resulting in heavy contention and multiplexing. This can be ameliorated by splitting the 32-bit wide SMART channels into two 16-bit narrower channels (or more)[11], then clocking them at twice or

[9]The bandwidth requirements of the three MMS benchmarks are scaled up 100x to allow reasonable on-chip traffic in our 2 GHz design. All other benchmarks' bandwidth remain unchanged.

[10]In the worst case, if all flows contend, SMART and Mesh will have the same network latency.

[11]Essentially, this increases the radix of the router and the path diversity.



(a) Performance

(b) Power Breakdown

Fig. 10: Evaluation of SMART NoC across SoC Applications.

thrice the rate, leveraging the high frequency of SMART links to mitigate conflicts. SMART can also enable non-minimal routes for higher path diversity without any delay penalty. We leave these as future work.

In an actual SoC, the task to core mapping may not be able to change drastically across applications as cores are often heterogenous, and certain tasks are tied to specific cores. This will result in longer paths, magnifying the benefits of SMART.

Power Analysis. Figure 10b shows the post-layout dynamic power breakdown across the applications for all three designs.

All designs send the same traffic through the network, and hence have similar link power. Compared with Mesh, where flits need to stop at every router, SMART reduces power by 2.2X on average both due to bypassing of buffers, and due to clock gating at routers where there is no traffic. The total power for Dedicated is much lower than SMART because only link power is plotted, which is negligible due to low network activity. A Dedicated topology also has high-radix routers at destinations (if it acts as a sink for multiple flows), pipeline registers and muxes at the source (if multiple flows originate from it), which we ignored in the power estimates, though these will not be negligible.

VII. Conclusion

In this paper, we proposed SMART NoCs and demonstrated how scalable NoCs such as meshes can realize single-cycle, cross-chip communication while delivering high bandwidth by dynamically reconfiguring its switches to match application traffic. In the past, SoC architectures, compilers and applications have been aggressively optimizing for locality. As we drive towards more and more sophisticated SMART NoCs, we hope that will pave the way towards locality-oblivious SoC design, easing the move towards many-core SoCs.

Acknowledgement

The authers acknowledge the support of DARPA under the Ubiquitous High-Performance Computing (UHPC) program, and Michel Kinsy from MIT for providing H264 task graph.

References

[1] K. Goossens *et al.*, "Aethereal network on chip: Concepts, architectures, and implementations," *IEEE Des. Test*, vol. 22, no. 5, pp. 414–421, 2005.

[2] F. Karim *et al.*, "An interconnect architecture for networking systems on chips," *IEEE Micro*, vol. 22, no. 5, pp. 36–45, Sep 2002.

[3] N.-S. Woo, "High performance SOC for mobile applications," in *ASSCC*, 2010.

[4] A. Adriahantenaina *et al.*, "SPIN: A scalable, packet switched, on-chip micro-network," in *DATE*, 2003.

[5] J.-Y. Kim *et al.*, "A 118.4 gb/s multi-casting network-on-chip with hierarchical star-ring combined topology for real-time object recognition," *JSSC*, vol. 45, no. 7, pp. 1399–1409, 2010.

[6] G. Passas *et al.*, "A 128 x 128 x 24gb/s crossbar interconnecting 128 tiles in a single hop and occupying 6% of their area," in *NOCS*, 2010.

[7] R. Ho *et al.*, "High-speed and low-energy capacitive-driven on-chip wires," *ISSCC*, 2007.

[8] E. Mensink *et al.*, "A 0.28pj/b 2gb/s/ch transceiver in 90nm cmos for 10mm on-chip interconnects," *ISSCC*, 2000.

[9] 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," *ISSCC*, 2009.

[10] T. Krishna *et al.*, "NoC with near-ideal express virtual channels using global-line communication," *HOTI*, 2008.

[11] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2004.

[12] M. Modarressi *et al.*, "Application-aware topology reconfiguration for on-chip networks," *TVLSI*, 2011.

[13] ——, "Virtual point-to-point connections for nocs," *TCAD*, 2010.

[14] M. B. Stensgaard and J. Sparso, "Renoc: A network-on-chip architecture with reconfigurable topology," in *NOCS*, 2008.

[15] M. B. Stuart *et al.*, "Synthesis of topology configurations and deadlock free routing algorithms for renoc-based systems-on-chip," in *CODES+ISSS*, 2009.

[16] C. Jackson and S. J. Hollis, "Skip-links: A dynamically reconfiguring topology for energy-efficient nocs," in *ISSOC*, 2010.

[17] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective, second edition*. Prentice Hall, 2003.

[18] H. Zhang *et al.*, "Low-swing on-chip signaling techniques: Effectiveness and robustness," *VLSI*, vol. 8, pp. 264–272, 2010.

[19] R. Golshan *et al.*, "A novel reduced swing cmos bus interface circuit for high speed low power vlsi systems," in *ISCAS*, 1994.

[20] B.-D. Yang *et al.*, "High-Speed and Low-Swing On-Chip Bus Interface Using Threshold Voltage Swing Driver and Dual Sense Amplifier Receiver," *ESSCIRC*, pp. 144–147, September 2000.

[21] E. Kyriakis-Bitzaros, "Design of low power cmos drivers based on charge recycling," in *ISCAS*, 1997.

[22] M. Hiraki *et al.*, "Data-dependent logic swing internal bus architecture for ultralow-power lsis," *JSSC*, pp. 397–402, April 1995.

[23] H. Yamauchi *et al.*, "An asymptotically zero power charge-recycling bus architecture for battery-operated ultrahigh data rate ulsis," *JSSC*, pp. 423–431, April 1995.

[24] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto noc architectures," in *DATE*, 2004.

[25] C.-H. O. Chen *et al.*, "A low-swing crossbar and link generator for low-power networks-on-chip," in *ICCAD*, 2011.