FastTrack: Leveraging Heterogeneous FPGA Wires to Design Low-cost High-performance Soft NoCs

Nachiket Kapre
University of Waterloo
Ontario, Canada
nachiket@uwaterloo.ca

Tushar Krishna
Georgia Institute of Technology
Atlanta, GA, USA
tushar@ece.gatech.edu

***Abstract*—**
Networks-on-Chip (NoCs) implemented on FPGAs have to be designed differently from ASICs to fully exploit the unique architectural features and properties of the FPGA fabric. The FPGA-friendly bufferless, deflection routed Hoplite NoC is almost an order of magnitude smaller and runs at a faster operating frequency than competing classic buffered FPGA NoCs. It is able achieve this by sacrificing NoC link utilization that suffers due to the cost of packet deflections and associated high latency traversals. In this paper, we address these shortcomings by developing FastTrack, which is an FPGA-optimized, high-radix NoC that exploits the segmented interconnect structure of modern FPGAs. We adapt the NoC organization to use express bypass links in the NoC to skip multiple router stages in a single cycle. Our FastTrack design can be tuned to use different express link lengths for performance, and supports depopulation strategies for controlling the balance between FPGA LUT and wiring cost. For the Xilinx Virtex-7 485T FPGA, an 8×8 FastTrack NoC is 1.7–2.5× larger than a base Hoplite NoC, but operates at almost the same clock frequency. FastTrack delivers throughput and latency improvements across a range of statistical workloads (2.5×), and traces extracted from FPGA accelerator case studies such as Sparse Matrix-Vector Multiplication (2.5×), Graph Analytics (2.8×), Token LU Factorization Dataflow (1.4×) and Multi-processor overlay applications (2×). FastTrack also shows energy efficiency improvements by factors of up to 2.2× over baseline Hoplite due to higher sustained rates and high speed operation of express links made possible by fast FPGA interconnect.

***Keywords*-Network-on-Chip, FPGA, Overlay networks**

I. Introduction

FPGAs are fast becoming ubiquitous computation and acceleration platforms, complementing CPUs and GPUs across systems today. In addition to their traditional role for fast prototyping of designs, modern FPGAs find wide use as DNN accelerators [1–3], Graph accelerators [4], switch fabrics inside HPC systems [5], network function virtualization [6], and cloud service accelerators [7, 8]. To service these various platforms, FPGA boards today house hard IPs such as ARM cores, DSPs, and I/O controllers (PCIe, DRAM), in addition to the reconfigurable substrate. Applications or designs that are mapped over these FPGAs comprise of multiple soft IPs, interacting with each other and the hard IPs in the system. Examples of such soft IPs include processors [9, 10], domain-specific processing elements [1–3], system-level endpoints, and others. Just like in MPSoCs and multi-core CPUs, it is imperative to architect the right interconnection between these various IPs, to achieve high-performance. In fact, the latency and bandwidth of these FPGA interconnects is highly critical to overall performance since the IPs mapped on FPGAs cannot implement expensive latency-hiding techniques such as multi-threading and out-of-order processing like modern CPU cores do.

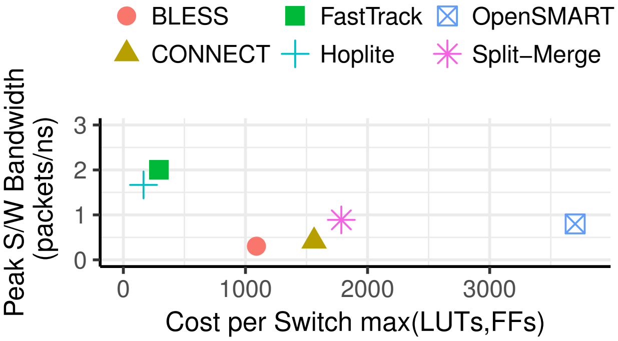


Figure 1: Area-Bandwidth Tradeoffs in implementing NoCs on FPGAs.

Most FPGA designs use point-to-point connections over the configurable fabric. However, these do not scale beyond a certain IP count as they start adding heavy routing pressure on the FPGA wires. It has also been demonstrated [11] that the communication bandwidth of system-level interfaces such as PCIe, DRAM, and Ethernet is too high to support using configured, point-to-point, interconnect alone. Moreover, analyzing all the connectivity requirements upfront when designing the application can become challenging or even infeasible for a FPGA developer. For all these reasons, switched networks-on-chip (NoCs) have started to emerge as a connectivity paradigm for FPGAs [12–14]. These NoCs are *soft* NoCs and can be generated as a soft IP to connect the various application modules on the FPGA fabric, using classic NoC topologies and routing mechanisms. Packet-switching allows the system to dynamically determine the connection requirements of the application at *runtime* while spending a user-determined amount of resources on the NoC

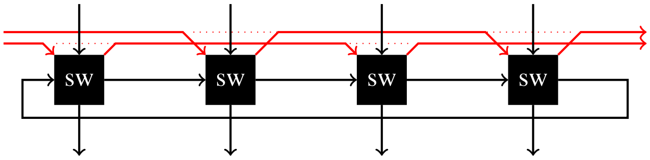


Figure 2: A row of the NoC (4×1 slice) showing express physical channels (in red) in conjunction with regular NoC links (in black). The express link allows packets to skip a router. Link length determined by matching FPGA technology parameters to application requirements.

infrastructure. Vendors like Altera have in fact started to provide tools such as Qys [15] to generate application-specific NoCs built around industry-standard interfaces such as AXI simplifying system composition.

Engineering soft NoCs for FPGAs is a different design problem from designing NoCs for multi-core CPUs, or ASICs, since the trade-offs are quite different. Figure 1 summarizes the state-of-the-art FPGA implementation cost versus bandwidth trends observed in contemporary NoC routers. Buffered ASIC NoCs (such as OpenSMART [16]) can provide high throughput (packets/cycle) but come at a prohibitively high area cost and lower operating frequency when mapped on a FPGA, thereby reducing switch bandwidth (packets/ns). Prior studies [12] have shown that mapping an ASIC-optimized NoC RTL directly on a FPGA is quite sub-optimal, primarily because FPGAs are wire-rich but LUT area-limited. There has been a steady stream of research on trying to strip down ASIC NoCs to make them more FPGA-friendly. Split-Merge [13] and CONNECT [12] were early efforts in this direction, and reduced LUT costs by 40-50% over state-of-the-art ASIC NoCs. However, these designs still require in excess of a thousand LUTs per router, which not only consumes precious FPGA area and power, but also adds high delay penalties at each hop. Bufferless deflection-based NoCs like Hoplite [14] require only 1-2 LUT per bit at every router, thereby reducing area, power, and per-hop latency by orders of magnitude. However, they result in poor wiring utilization due to wasteful deflections, and extremely high worst-case routing times. Deflections cause packets to traverse NoC rings multiple times instead of waiting patiently in buffers. This challenge become worse as the size of the NoC scales, especially for FPGA accelerators [4, 17] and Multi-processor overlays that often employ hundreds of processing elements (PEs) [9, 10].

In this paper, we introduce **FastTrack**, a NoC architecture that is optimized for both area and wire-efficiency. FastTrack provides both the flexibility of packet-switching, and the low-latency enabled by dedicated connections. Our key insight is that not all wires on the FPGA are equal - FPGA chips have long shipped with segmented interconnect carrying wires of varying lengths (speeds) to accommodate circuit connections with diverse requirements [18]. FastTrack makes strategic use of these interconnects via *express channels* in the topology to bypass multiple FPGA routers and provide a fast path for packets on the NoC. In Figure 2 we show an example of a 4×1 Hoplite NoC with a unidirectional torus topology using FastTrack express links of length 2. Each FastTrack switch now has an extra input and output port per dimension and extra wires (red) to allow jumping over multiple router hops. Express channels based NoCs have traditionally lead to a quadratic increase in area and power [16, 19, 20]. In FastTrack, we leverage the FPGA CLB architecture, and provide depopulation strategies to accommodate the logic that drives the express paths in an efficient manner.

The key contributions of this paper are as follows:

- Wire speed characterization of the Xilinx Virtex-7 485T to understand the furthest a packet can traverse on the chip in a single clock (Section III).
- Design of the FastTrack router microarchitecture, topology, and routing policies for efficient FPGA realization (Section IV).
- A design methodology to tune the LUT and wire cost of FastTrack in a fine-grained manner to best use FPGA resources.
- Evaluation of the FastTrack NoC across a range of statistical and FPGA accelerator workloads from Sparse Matrix-Vector Multiplication, Graph Analytics, Token LU Factorization Dataflow, along with Multi-Processor overlays (Section VI).

The rest of the paper is organized as follows: section II presents background on challenges unique to FPGA NoCs and reviews related work. section III performs an analysis of wire delays on the FPGA. section IV presents the FastTrack microarchitecture and hardware implementation is presented in section V. section VI presents evaluation results. section VII highlights FPGA architecture impact in light of FastTrack. Finally, we conclude in section VIII.

II. Background and Motivation

FPGA overlay NoCs allow FPGA IPs to communicate with each other using a packetized interface. There is a growing demand for the use of FPGAs in data-centers for acceleration [7, 8] and fast switching [5, 6] in highly interconnected environments with fast network ports, PCIe links, DRAM channels, inter-FPGA links and cache-coherent connections to host CPUs. Overlay NoCs allow existing FPGAs to provide a seamless framework for composing complex digital designs connected to external and internal interfaces. In this section, we detail some challenges in designing FPGA overlay NoCs, and the promise that fast FPGA wires can offer to address these challenges.

A. Network-on-Chip (NoC)

A NoC is a multi-ported switch that can be used to connect multiple IPs together using shared resources. Designing a NoC that is low-latency, high-throughput, low-area and low-power is quite challenging, since it often becomes hard to optimize on all fronts. We discuss three classes of routers, and the trade-offs they offer in terms of all of these metrics.

1) Trade-off with Router Microarchitectures:

Buffered Low-Radix Routers.: We typically use four-five ported routers to build the popular mesh and torus topologies. These designs provide connectivity between neighboring routers, and buffer any packets in case of contention. The latency of packets going through these routers is linear with the number of routers. State-of-the-art routers today take 1–2 ns on ASICs [21, 22]. Low-radix routers offer a balance between latency, throughput, area, and cost and are the most popular solutions in ASICs today.

Latency is a challenge in these designs, and there have been proposals to reduce latency by enabling messages to setup fast paths and *bypass* routers. SMART [22] is one such technique, that adds an intelligent mux at every hop. This mux acts as a repeater, forwarding the packet to the next router without latching it if there is no contending packet at the router wishing to use the output link. If there is contention, the mux sends the incoming packet to the router's clocked buffers instead like a traditional design. In SMART NoCs, long-range bypass paths are **virtual**: packets get the illusion of having dedicated repeated wires despite actually going over a series of shared link segments over a conventional low-radix topology.

Low-radix Mesh, Torus, and SMART routers are still quite expensive for FPGAs, especially due to the buffers. We generated low-radix routers using OpenSMART [16] and CONNECT [12] and report their FPGA cost in Table I.

Buffered High-Radix Routers. High-Radix routers offer extremely low-latency and high-throughput, due to *physical* express channels between distant routers. Flattened Butterfly [20] and Multi-drop Express Channels (MECS) [23] have been popular techniques in this domain. However, these designs come at the cost of high-radix (i.e., multi-ported) routers that add delay, area, and power overheads due to larger buffers, crossbars, and arbiters. Moreover, scaling the number of ports can lead to quadratic increase in cost. This is partly why such high-radix topologies have not made it to NoCs in commercial multicore chips, and are only used in HPC switches [19]. Applying this technique directly to an FPGA is even more prohibitive than low-radix routers; for instance, doubling the ports from low-radix (4-port) to 8-port router increases LUT and FF utilization by 3.25× [16].

Bufferless Routers. Bufferless routers have been proposed in the past to address area and power costs of routers. Examples include CHIPPER [24], BLESS [25] and SCEPTER [26]. These designs rely on deflections to handle contention. The key trade-off with these designs is the high-latency for deflections, and a solution to handle livelocks. Bufferless routers have not made it to mainstream ASICs as the power consumption of NoCs at nominal utilization is observed to be quite low [27], while the area costs of routers are acceptable in front of large SRAM and cores. However, these metrics change quite drastically on a FPGA.

Table I: FPGA implementations of 32b NoC routers.

Router	FPGA Device	LUTs	FFs	Clk[2]
OpenSMART 4VC, 1-deep [16]	Virtex-7 VX690T	3700	1700	5
BLESS (no buffers) [25]	Virtex-2 Pro	1090	335	13.2
CONNECT 2VCs 16-deep [12]	Virtex-6 LX240T	1562	635	9.6
Split-Merge DOR [13, 29]	Virtex-6 LX240T	1785	541	4.5
Altera Qsys [15][1]	Stratix IV C2	1673	-	3.1
Hoplite [14]	Virtex-7 485T	78	165	1.2
FastTrack (This Work)	Virtex-7 485T	191–290	290	2

[1] Derived from Table 1 of [15] for fully-connected 16-node system. [2] Period in ns

2) FPGA Overlay NoCs: Unlike ASICs, where logic and wiring resources can be provisioned as required, the FPGA substrate has a fixed balance between these components as decided by the FPGA vendors. Most modern FPGA chips are wire-rich but LUT and FFs are a prime resource. Table I compares the LUT and FF costs across a suite of NoC proposals. ASIC NoCs such as OpenSMART [16] consume thousands of LUTs and FFs. Even bufferless NoCs like BLESS are area expensive in terms of LUTs as they are not optimized for FPGAs. FPGA NoCs such as CMU CONNECT [12] and Penn Split-Merge [13] reduce resource costs but are still quite expensive in terms of LUTs and FFs, particularly against Hoplite [28].

Hoplite NoC. Hoplite [14] is one of the most efficient designs today. It is more than an order of magnitude smaller in LUT cost and 1.5–10× faster in operating frequency than the best known alternatives today. It uses bufferless switches which are heavily optimized for the FPGA microarchitecture, utilizing just 1–2 LUTs per bit of the packet payload on a Xilinx FPGA. Figure 9a shows an example of a base Hoplite switch, taken from Kapre et al. [28, 30], which uses just two 3:1 muxes, both of which can be independently selected by the inputs. The NoC exit is shared with the S port to further reduce LUT costs. Livelock avoidance is implemented using static turn prioritization discussed in [30]. The routing decode logic is based on a simple Dimension-Ordered Routing (DOR) function that routes packets in the X dimension before allowing packets to turn in the Y dimension (W→S always has highest priority). Hoplite uses a 2D unidirectional torus which is chosen specifically to lower the cost of switching crossbar implemented in LUTs.

Optimizing for Performance and Cost. Though Hoplite is extremely efficient in terms of area and power, it uses wiring resources poorly due to repeated traversals of the NoC links by deflected packets. It thus suffers from both high latencies, especially in the worst case, and low-throughput because of the unidirectional torus topology.

In this work, we seek to merge two extreme design points for NoC routers: bufferless and high-radix, and design a FPGA-optimized low-cost version of such a NoC. Our design, FastTrack, seen in the last row of Table I, adds only

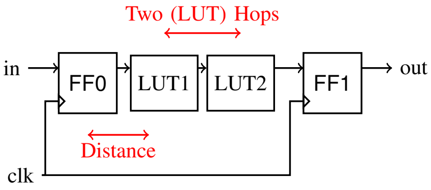


Figure 3: Wire delay characterization of FPGA chips by adjusting length of a registered wire and inserting programmable number of logic stages along the wire.

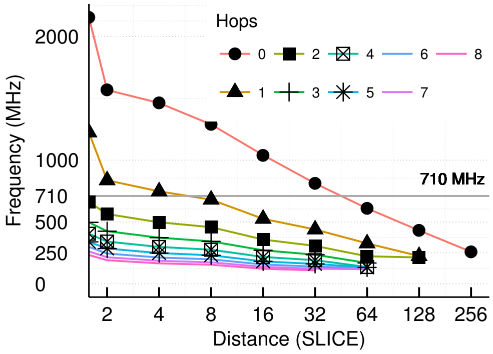


Figure 4: Characterizing the speed of FPGA interconnect for the Xilinx Virtex-7 485T FPGA.

marginal cost over Hoplite, but provides extremely high-performance gains as we show later in our evaluations. The key insight in FastTrack is to leverage the heterogeneity offered by modern FPGA interconnect.

III. FPGA Wire Characterization

We characterize FPGA wiring infrastructure via two experiments, to study the impact of virtual and physical express links on FPGAs.

1) Virtual Express Links: We perform a simple experiment where we create a Verilog circuit with manually instantiated FDRE (Xilinx Flip-Flop) and LUT6 (Xilinx Lookup Table) components as shown in Figure 3. We also supply physical location constraints for these components along an FPGA column with XDC (Xilinx Design Constraints) layout constraints by varying (1) **Distance** (*i.e.* SLICEs) between consecutive components, and (2) number of **Hops** (*i.e.* number of equidistant LUT stages between the two registers). By placing only two registers in two locations, and having no LUT hops, our experiment measures the raw speed of the FPGA wiring fabric in absence of congestion. By introducing the LUT hops in a programmable fashion, we model the effect of the wire exiting the interconnect and entering a NoC router(s). This effect models the behavior of the SMART NoC [22] discussed in Section II-A1 where a packet is able to tunnel through multiple NoC routers in a single clock cycle on an ASIC. Unlike ASICs, FPGAs must pay the penalty of entering and exiting the fabric.

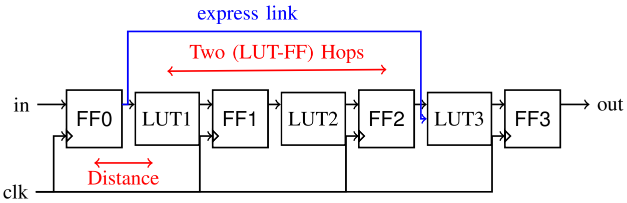


Figure 5: Idea of Express physical channels bypassing multiple LUT-FF stages to exploit fast FPGA wiring.

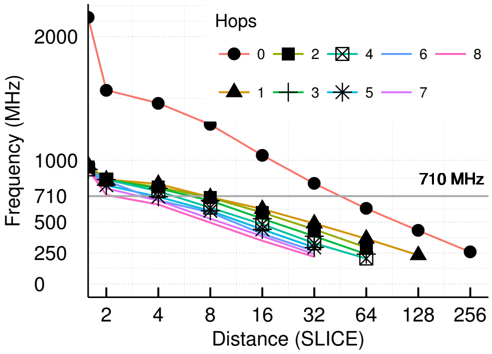
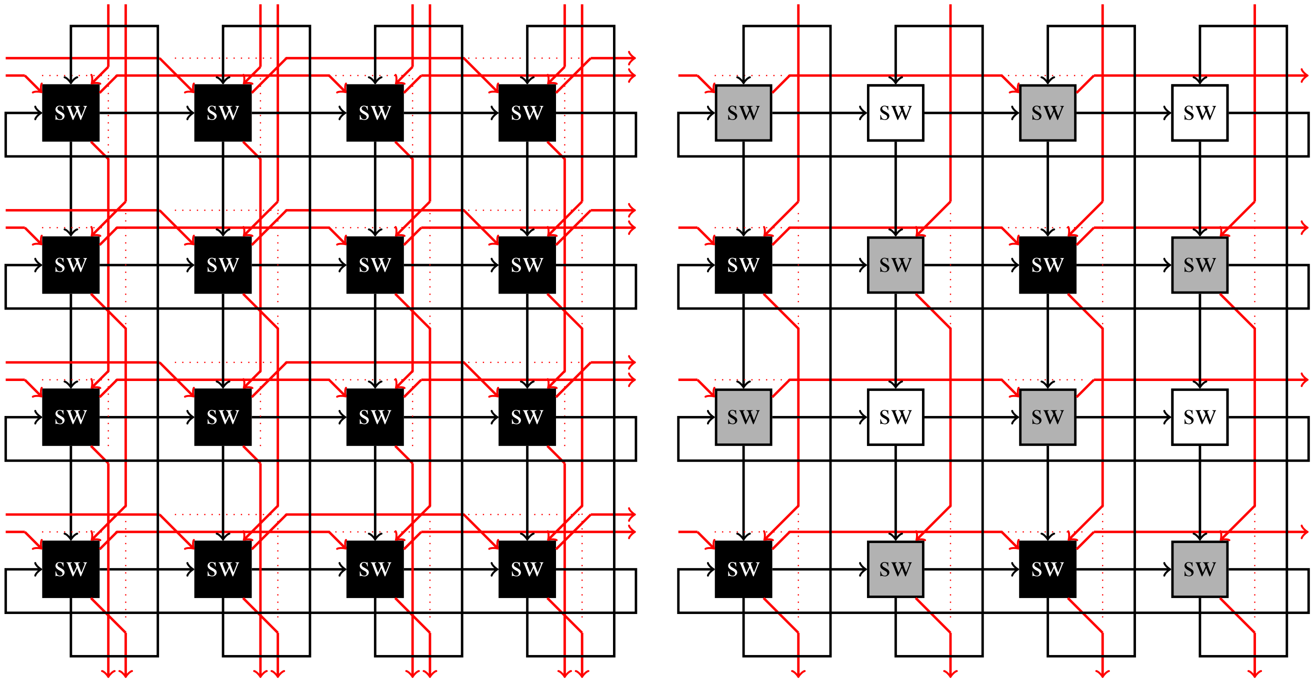


Figure 6: Quantifying the impact of express physical bypass channels on frequency of the resulting design.

We plot the results in Figure 4. There is a ceiling of ≈710 MHz, the peak frequency of the clock distribution network, so results above this frequency are purely theoretical. As expected, the fastest frequency is achieved with Hops=0 and Distance=1. As we increase the distance between the registers the frequency drops to 250 MHz at a distance of 256 SLICEs (close to chip dimensions). Even with a single LUT hop (Hop=1), we observe a noticeable drop in frequency to 450 MHz at a maximum distance of 128 SLICEs. With more LUT hops, the frequency degrades to ≈200 MHz at almost all distance counts. Our experiment shows that the FPGA can support close-to-full-chip traversal at 250 MHz in absence of congestion. Long wires use the faster FPGA routing tracks and are able to travel long distances quickly. We confirm the well-known observation that the cost of CLB-to-CLB routing, and getting onto and off the interconnect fabric, is large.

2) Physical Express Links: Next, we conduct a feasibility experiment to understand the potential for providing express bypass links in the NoC to enhance packet routing latencies. We do this by adapting our wire delay characterization experiment from Section III to provide configurable bypass to a fully-pipelined sequence of tightly-coupled LUT-FF pairs. The LUT-FF pair is implemented in the same primitive to guarantee fast performance and model the effect of a packet traversing one NoC router per pair. As before, we vary the Distance between the LUTs where FFs are directly attached to adjacent LUTs. We also provide a new *express link* wire that bypasses a programmable number of LUT Hops. Based on the desired operating frequency of the



(a) Fully Populated Topology FT(16, 2, 1)

(b) Depopulated Lower Cost Topology FT(16, 2, 2)

Figure 7: FastTrack Topology for a 4×4 Torus NoC with D=2. The notation is FT (N^2, D, R). The red wires represent express links that exploit faster FPGA routing resources to bypass multiple NoC routers. Wraparound links are shown with dotted lines. The shades correspond to FastTrack router complexity, with darker switches being more expensive.

circuit, we can choose `Distance` and `Hop` combination as required by the design. Alternatively, given a design with required `Distance` and operating frequency constraint, we can choose `Hop` that meets the constraints. We illustrate the experiment circuit in Figure 5 and show a single express link bypassing two LUT-FF stages.

In Figure 6, we show the effect of implementing physical express links on the Xilinx Virtex-7 485T FPGA. Unlike the previous experiment (Figure 4), when using the express links design the frequency degrades a lot more gracefully even when bypassing several LUT-FF stages. Instead of bottoming out at ≈200 MHz at short `Distance` values, the frequency degradation is linear with increasing `Distance`. It eventually drops to 250 MHz but allows us to travel 32–64 SLICE hops. This experiment provides evidence that express links are able to bypass several LUT-FF stages using fast FPGA routing tracks to retain high-speed operation in absence of congestion. It is important to note that on an ASIC, the virtual express channels would still provide scalable performance unlike FPGAs, where the cost of exiting and re-entering the FPGA interconnect is much higher.

3) Leveraging FPGA Wires for Overlay NoCs: Figure 6 demonstrates that FPGA wires are quite fast, and the interconnect alone can provide multi-hop traversal within a cycle at 250MHz. However, exit from the interconnect fabric is expensive in terms of delay, and traversal through a series of LUTs within a cycle can limit frequency tremendously. While these observations are not surprising for FPGA practitioners, this has not been exploited to design faster overlay NoCs to the best of our knowledge. We leverage these observations to motivate the design of FastTrack NoC.

IV. FastTrack

In this section, we introduce the key idea behind FastTrack, and discuss the topology, microarchitecture and routing policies that are possible in our design. These three NoC design aspects influence both the cost (area and power), and performance of the resulting implementation.

A. FastTrack Topology

FastTrack introduces additional physical links within the NoC topology to provide dense connectivity while minimizing area overhead over the baseline NoC.

Each FastTrack topology is parameterized formally as FT(N^2, D, R) over the following variables:

- System size $N \times N$
- Length D ($1 \leq D \leq \frac{N}{2}$) of each express link in hops.
- Depopulation factor R ($1 \leq R \leq D$) that controls the cost of the topology by instantiating nominal Hoplite routers in between FastTrack routers. (R-1) is the number of nominal Hoplite switches between any pair of FastTrack switches D hops away.

Each NoC channel in a FastTrack NoC topology consists of two kinds of links: short (Sh) links between adjacent routers, and a variable number of express (Ex) links between routers that are D-hops apart. The express links are *braided* through the rings to provide a distance D for each connection. This increases the number of wires required by the NoC by a factor of $\frac{D}{R}+1$. In the most expensive extreme configuration, where R=1, we would need $D\times$ more wires in each channel. For average FPGA applications, it has even been argued that FPGA wires are "free" [12] as the FPGA vendors allocate more interconnect than needed to handle worst-case customer requirements. However, to be fair, we consider a resource equivalent design would use multiple physical Hoplite channels without FastTrack adaptation. We explore this iso-resources case in Section VI, and even in this case demonstrate that FastTrack makes better use of available wiring resources and outperforms the multi-channel alternative. In the cheapest configuration that still retains express links R=D, we would have at most a doubling of the wiring requirements. In Figure 7, we show two FastTrack topologies: FT(16, 2, 1) and FT(16, 2, 2). In addition to wiring costs, the NoC switch must also be modified to support an extra express input and express output, as we describe in subsection IV-B. Thus the choice of D and R opens up a space of possible topologies where we can tradeoff area (logic cost of switches, and wires) in exchange for performance. This offers us a better knob to control cost unlike the expensive high-radix router designs in [16].

In Figure 8, we show an example of packet traversal path from (0,3) → (3,0) exploiting two express links and two short links. The packet starts off in the slow links, and upgrades to faster links later as the express path does not directly connect to the packet destination wholly within the express network.

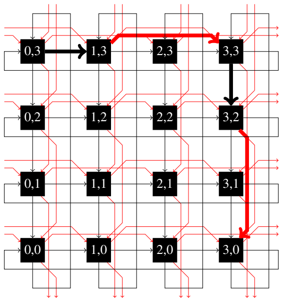


Figure 8: Packet traversal from (0,3)→(3,0).

B. Router Microarchitecture

The FastTrack NoC router is different from the baseline Hoplite router (Figure 9a) as it needs to support switching along the express links in addition to the original short links. We have a few choices in determining the cost of the switch by choosing which connections are possible. In general, the switch needs to support two extra inputs and outputs (one set per dimension) for the express links. This increases the number of inputs to the switching multiplexers and requires a redesign of the routing function and offers us several design choices and associated routing policies when handing packets.

FT (Full) Router. Figure 9b shows the microarchitecture of a fully-loaded FastTrack router. These were the Black routers in Figure 7. We permit packets to hop on to an express link from any input port, and from any incoming link (short or express). Packets may start on an express link, or start on a short link if the express one is occupied and then upgrade to it later when its free. This freedom comes at a cost of more inputs to each of the output multiplexers permitting this lane change. However, packets cannot transfer from express to short links except when turning to avoid livelock ($W_{Ex} \rightarrow S_{Sh}$, and $N_{Ex} \rightarrow E_{Sh}$, See subsection IV-D). Compared to a baseline Hoplite router (Figure 9a), which requires 2× 3:1 multiplexers, the FT router requires a 5:1 mux, and 4× 4:1 multiplexers.

FT$_{lite}$ Routers (Topology). FastTrack also supports lighter variants of routers, which are lower area, at the cost of reduced routing flexibility. One class of FT$_{lite}$ routers exist because of the depopulation factor R discussed earlier. For instance, the Grey routers in Figure 7 have only one express link coming in and out, thus requiring one less set of output multiplexers and one less input to the remaining express output. The White routers in Figure 7 are just the original Hoplite routers (Figure 9a) with no express links connected to them.

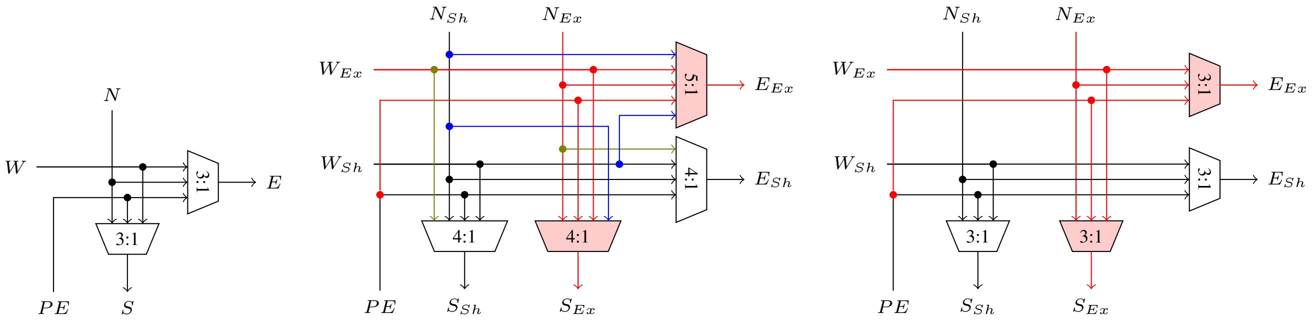
FT$_{lite}$ Routers (Switch Complexity). We also add a second class of FT$_{lite}$ routers that have express links, but restricted policies for hopping on to these links. Figure 9c shows the microarchitecture of a FT$_{lite}(Inject)$ router that permits packets to hop on express links only at the client/PE injection port, not from other ports. A packet on an express link permanently stays on the express links while those on short links are destined to remain on the short links until delivery. This implementation is low cost and requires a set of 3:1 multiplexers for the S_{Ex} and E_{Ex} outputs. The Black routers in Figure 7 can also use this design variant.

C. Routing Policies

The routing policy sets the select lines of the various muxes inside the FastTrack router.

Dimension-Ordered Routing. By default, the FastTrack NoC uses XY routing, i.e., all packets use the E links before using the S links. Once the direction is chosen, packets try to use express links if $\Delta X \geq D$ or $\Delta Y \geq D$, else short links.

Congestion. The various multiplexer configurations described in subsection IV-B and the DOR function prune the set of possible connections that may be possible during packet routing on the NoC. The absence of a particular input to a multiplexer restricts packets along certain paths in the NoC. The use of DOR routing forces packets to traverse in the X ring first before turning along Y. However, conflicts are possible when multiple packets want the same resource (outgoing link). This is resolved via the following routing policies:



(a) Base Hoplite (b) FT (Full): Ex→Sh (Olive links) + Sh→Ex (Blue links) (c) FT$_{lite}$ (Inject): No lane crossing (Red colored resources extra over base Hoplite)

Figure 9: Microarchitecture of different variants of FastTrack routers exploring cost-performance tradeoffs.

- Client/PE input port gets the lowest priority when injecting traffic into the NoC. This is necessary as all outgoing ports may be needed to service in-flight packets in the NoC in a given cycle. With deflection routing there are no slots to buffer packets on contention and blocking PE injection is needed.
- The DOR routing function will determine if the address ∆s of the packet necessitate the use of a express link. If the express link is already occupied by an in-flight NoC packet, or if the DOR function determines the packet address ∆s are too small, the packet is sent along the short link. In case of contention for the same short link, packets are deflected.
- Packets within the X ring of the network have higher priority over packets in the Y ring. This is designed in avoid livelocks in the deflection routed torus, explained in the following subsection.

D. Livelock Avoidance

Since FastTrack is a deflection based NoC, it can suffer from livelocks. Livelocks are the problem where packets continue to get deflected forever and never reach their destination. In FastTrack, there can be two scenarios that could lead to livelock:

- A turning packet may keep getting deflected within a ring and is never able to turn creating a livelock.
- A packet on a express link that needs to get on a slow link to reach its destination may not be able to do so due to repeated contention on the slow links.

Traditional approaches to livelock avoidance in bufferless routers [25, 26] involves adding counters in the packet to limit the number of deflections and have the router intelligently prioritize those packets that have been deflected a certain threshold number of times. For FPGA NoCs, all of these would contribute to precious LUT area. Instead, we develop a lightweight solution for FastTrack.

To avoid the first kind of livelock, we modify the DOR routing scheme to prioritize turning traffic over column traffic as demonstrated in [30]. Thus $W \rightarrow S$ turn has higher priority and can cause N packet to get deflected E, a turn that is not normally possible. The routing function is designed to ensure a packet is deflected *exactly* once per ring and makes progress towards the destination by dropping down the Y ring one switch at a time.

To avoid the second kind of livelock, we have to make two changes. First ensure that packets enter the express links only if the packet destinations are directly reachable entirely within the express network. Second, we ensure that Express to Short transitions are only possible at a turn from $W_{Ex} \rightarrow S_{Sh}$ or $N_{Ex} \rightarrow E_{Sh}$ ports. This assigns the highest priority to the W_{Ex} or N_{Ex} ports, and with suitably design routing function, we can ensure that deflected packets can eventually arrive on this port after at most two deflections at this router. W_{Sh} packets that are deflected by $W_{Ex} \rightarrow S_{Sh}$ turn may use E_{Ex} port and return as a higher priority W_{Ex} packet after exactly *one* traversal around the ring. A N_{Ex} packet that want to go S_{Ex} can be deflected to E_{Ex} and will return as W_{Ex} packets with high priority. A N_{Sh} packet that wanted to proceed to S_{Sh} will need to deflect E_{Sh} and return as a W_{Sh} packet after *one* deflection, and suffer at most *second* deflection as a W_{Sh} packet. To avoid livelocks at exits, we must allow N packets to take either E ports. This bounds the number of possible deflections in the router for all ports, and guarantees forward progress.

V. Hardware Implementation

In this section, we discuss the results of our hardware implementation experiments. We use Vivado 2017.2 and Xilinx Virtex-7 485T (-2) for our mapping. We use parameterized RTL Verilog code for the different designs and generate the different configurations used in this paper through appropriate assignment of values to the Verilog design parameters. The routers are locked to rectangular regions of the chip in a manner the uniformly divides the FPGA resources into rectangular tiles. For the unidirectional torus ring topology, we adopt a folded layout to balance wire lengths. We use a lightweight dummy client to ensure the design does not get optimized away by the CAD tools. We

Table II: Resource Usage and Frequency of an 8×8 NoC (256b) on a Virtex-7 485T -2 speed grade.

Config.	LUTs	FFs	MHz	Power (W)
Hoplite	34K	83K	344	9.8
FT (64,2,1)	104K (2.6×)	150K (1.8×)	320 (0.93×)	25.1 (2.5×)
FT (64,2,2)	69K (1.7×)	117K (1.4×)	323 (0.93×)	19.9 (2×)

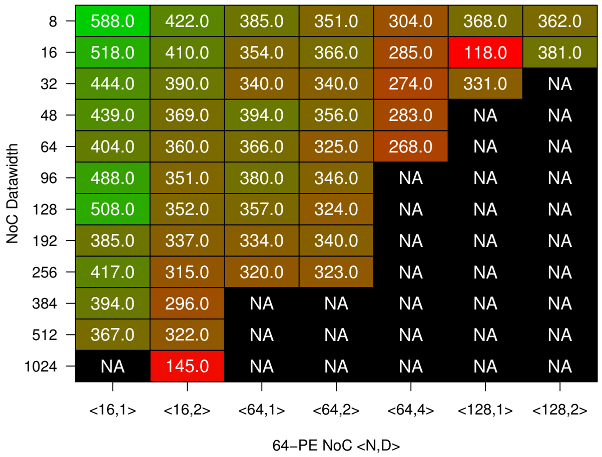
measure LUT and FF costs (area), wires per ring (area) as well as Frequency (performance) of the design.

A. FPGA Mapping Results

We present the implementation results of mapping a 256b-wide 8×8 NoC on the complete FPGA chip in Table II as reported by Vivado 2017.2 after placement and routing. We supply a timing constraint of 500 MHz to drive the CAD tool. When compared to baseline Hoplite design, the FastTrack topologies are 1.7–2.6× larger, about the same speed faster, and 2.5× more power hungry. As expected FastTrack NoCs are larger due to more expensive switches, but the frequency is still close to the baseline Hoplite frequency due to the long wires on the FPGA fabric. In this implementation, each Hoplite router is pipelined with registers at inputs and outputs. We can also insert a configurable number of additional registers along the NoC links if an even faster frequency if desired. When showing cost-aware results later, we also compare performance against replicated Hoplite designs that reallocate the LUT and wiring resource to simply create parallel Hoplite channels. In this scenario we do not modify the client interfaces and only permit a single packet injection and delivery to ensure fair comparison. When considering power consumption, FastTrack is 2–2.5× more expensive than Hoplite due to the 2× increase in registers and the longer wire lengths on the express links being driven by those registers.

B. Routability Analysis

While FPGAs are wire-rich architectures, their capacity to support multiple parallel fast tracks will be ultimately constrained by device limits. For various NoC system sizes, we characterize the NoC datawidths that the FPGA can support before saturating available resources. In Figure 10, we show the peak feasible datawidth in the baseline Hoplite NoCs and Fast-Track NoCs on the Virtex-7 485T FPGA. For 4×4 NoC, with D=2, we are able to support 512b datawidths. These wide payloads allows the deflection routed NoC to send an entire x86 cacheline directly as a single packet. For larger NoC sizes, the wiring capacity is reduced by the corresponding factor and a cacheline transfer must be serialized. Furthermore, we also consider the effect of NoC on user design logic and note 20–30 MHz reduction in frequency due to congestion. This is common to both Hoplite and FastTrack designs.



NoC Datawidth	<16,1>	<16,2>	<64,1>	<64,2>	<64,4>	<128,1>	<128,2>
8	588.0	422.0	385.0	351.0	304.0	368.0	362.0
16	518.0	410.0	354.0	366.0	285.0	118.0	381.0
32	444.0	390.0	340.0	340.0	274.0	331.0	NA
48	439.0	369.0	394.0	356.0	283.0	NA	NA
64	404.0	360.0	366.0	325.0	268.0	NA	NA
96	488.0	351.0	380.0	346.0	NA	NA	NA
128	508.0	352.0	357.0	324.0	NA	NA	NA
192	385.0	337.0	334.0	340.0	NA	NA	NA
256	417.0	315.0	320.0	323.0	NA	NA	NA
384	394.0	296.0	NA	NA	NA	NA	NA
512	367.0	322.0	NA	NA	NA	NA	NA
1024	NA	145.0	NA	NA	NA	NA	NA

Figure 10: Peak Frequency (MHz) of FastTrack NoCs of varying datawidths mapped to the Xilinx Virtex-7 485T FPGA. Black cells with NA did not fit the device.

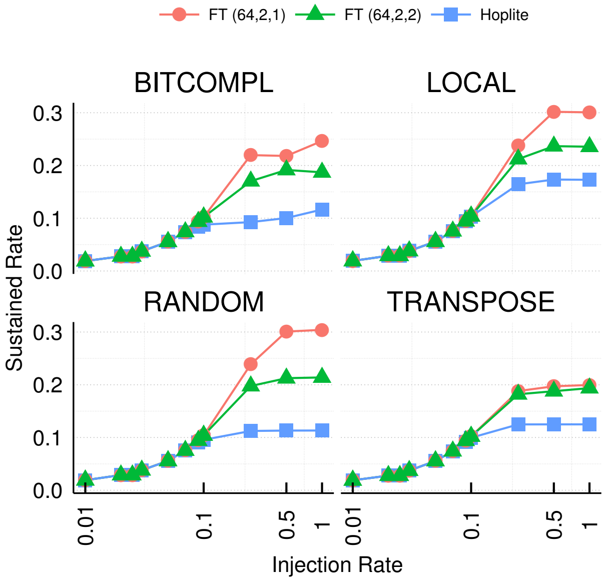


Figure 11: Effect of various 64 PE FastTrack NoCs (D,R combinations) on sustained rate for synthetic traffic.

VI. Evaluation

In this section, we evaluate different FastTrack and Hoplite NoC configurations and measure throughput, latency (average and worst-case), and power metrics. We consider various system sizes from 16–256 PEs and consider various combinations of D and R that configure FastTrack cost. We use traffic from synthetic sources (RANDOM, LOCAL, BITCOMPL, and TRANSPOSE) with varying injection rates for identical workloads with 1K packets/PE to offer a self-contained comparison against Hoplite. We also consider

realistic workloads extracted from FPGA accelerator case studies such as Sparse Matrix-Vector Multiplication, Graph Analytics, Token LU Factorization Dataflow, and Multi-Processor FPGA overlays. For most of the plots we consider 8×8 NoC configuration with `RANDOM` traffic pattern unless indicated.

How does FastTrack throughput compare against original Hoplite NoC for synthetic workloads? The key question we must address is whether FastTrack is able to make good use of the additional bandwidth made available by the faster links. For an 8×8 NoC, FastTrack outperforms Hoplite by up to 2.5× for `RANDOM`, 2× for `BITCOMPL` and 1.5× for `LOCAL` workloads as shown in Figure 11. The performance wins are non-existent for `TRANSPOSE` traffic and at injection rates below 10% across all traffic patterns. This is expected as these traffic patterns and conditions do not send (enough) packets to destinations that would require the use of the faster links As expected, larger values of R, with greater depopulation, result in reduced performance when compared to the fully populated FastTrack NoC. It is worth noting that the depopulated FastTrack NoCs still beat baseline Hoplite.

What is the effect of the use of FastTrack links on average latency behavior of the synthetic workloads? Latency reduction for NoC packets is anticipated to be the primary outcome of the use of FastTrack express links. In Figure 12, we show the average latency trends for the various NoCs are observe significant improvement in the throughput where the NoC is saturated (congested) and latencies start to climb significantly. At 100 cycles average latency we see as much as 5× higher saturation throughput when using FastTrack R=1 (full population) NoC for `RANDOM` and `BITCOMPL` traffic. The saturation throughput wins are a more modest 2× for `LOCAL` and `TRANSPOSE` traffic pattern. Again, we see the actual use of the faster links depends on the spatial distribution of packet traversals and influences the presence and quantum of latency improvements.

Does FastTrack make efficient use of FPGA interconnect resources over a multi-channel Hoplite design? As discussed previously in Section IV-A, an iso-wiring resource evaluation will fairly judge whether FastTrack makes better use of FPGA interconnect than replicating the original Hoplite NoC. In Figure 13, we see that FastTrack stays competitive and delivers better throughput and average latency for `RANDOM` traffic pattern when compared to the multi-channel Hoplite design requiring identical wiring resources. For N=64, and D=2 and R=1,2 designs, the Hoplite-3x design with three independent physical channels will use the same number of wiring resources as FastTrack. FastTrack beats multi-channel Hoplite by 1.2–1.4× for sustained rate, and by ≈0.7–5× for average latency. Furthermore, to exacerbate the gap, the multi-channel NoC might need identical wiring resources but costs the designer 1.5× more LUTs than FastTrack.

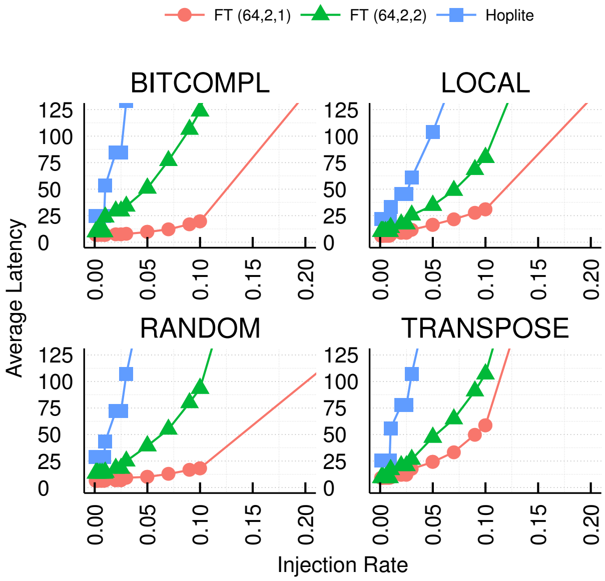
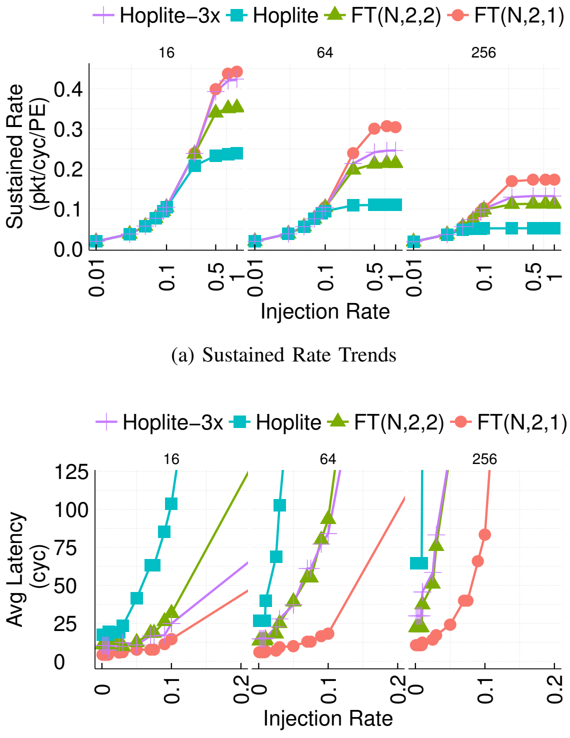


Figure 12: Effect of various 64 PE FastTrack NoCs (D, R combinations) on avg. latency of packets for synthetic traffic.

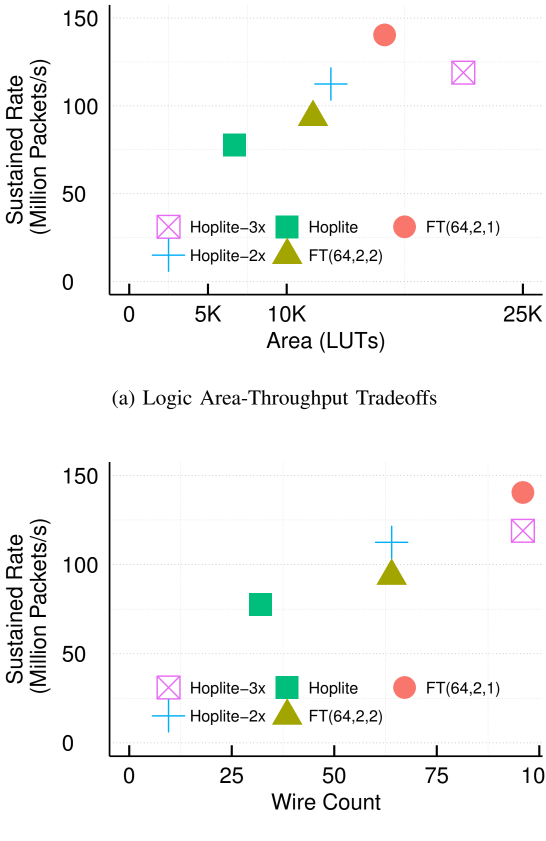


(a) Sustained Rate Trends

(b) Average Latency Trends

Figure 13: Measuring the effect of multiple physical channels on Hoplite vs. FastTrack for N=16, 64, and 256.

When considering FPGA implementation costs, is FastTrack a competitive alternative to baseline Hoplite NoC? Next, we evaluate the LUT cost and operating fre-



(a) Logic Area-Throughput Tradeoffs

(b) Wire Count-Throughput Tradeoffs

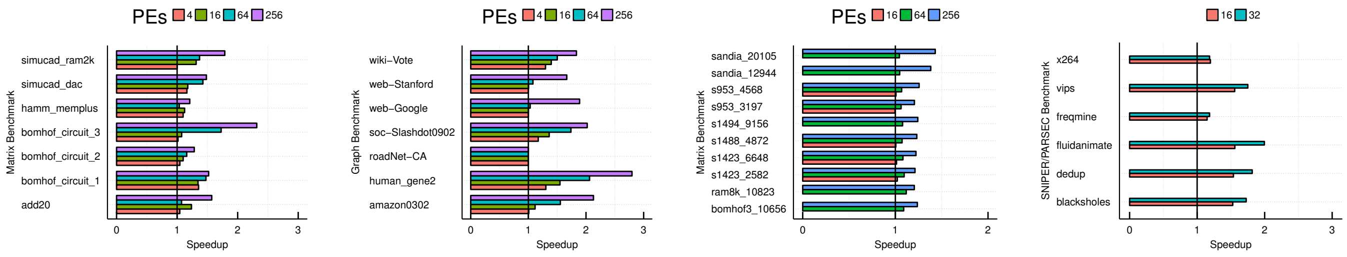
Figure 14: Considering the effect of FPGA cost (Logic and wiring) when comparing performance of 8×8 FastTrack NoC using `RANDOM` traffic pattern at 100% injection rates.

quency of the FastTrack and Hoplite NoC designs e this expense through a cost-aware comparison. In Figure 14a, we show the throughput of the NoC as a function of FPGA LUT area and operating frequency of the NoC. Here we see, that the R=1 and 2 designs offer almost 2.5–3× higher throughput over baseline Hoplite. When comparing against Hoplite-2x (a duplicated NoC), we see that advantages reduce to ≈1.8× for the best FastTrack configuration. When considering wiring costs, we see in Figure 14b that FastTrack allows better use of wires over Hoplite-3x with a 1.2× throughput advantage for the FT(64,2,1) design at 100% injection rate. When equating FT(64,2,2) design against Hoplite-2x with identical wiring costs, there is an advantage in favor of multi-channel Hoplite. Both the FT designs require fewer LUTs than the multi-channel counterparts thereby delivering better overall results when factoring both logic and wiring cost into the comparison. This shows that reclaiming the area of the FastTrack NoC to simply implement replicated Hoplite NoCs will not match the performance delivered by the use of faster links.

Does FastTrack work for real workloads such as communication traces from FPGA accelerators? Yes, FastTrack delivers superior results when routing communication traces from real applications. We consider Sparse Matrix-Vector Multiplication (used by many Deep Learning kernels [31]), Graph Analytics, Token LU Factorization Dataflow and Multi-Processor FPGA Overlay applications for this experiment. The SpMV and Graph Analytics workloads are throughput-bound while Dataflow workload is latency sensitive. We use datasets from the Matrix Market suite [32], SNAP graphs [33], LU factorization of SPICE circuits [34] and SNIPER-PARSEC benchmarks [35]. Dataflow workloads are latency sensitive as packets are injected into the NoC along a dependency chain. For these experiments, we measure workload completion time and quantify the speedup from using the most suitable FastTrack topology against a baseline Hoplite topology at identical PE counts (upto 256 PEs for SpMV, Graphs and LU factorization, 32 PEs for PARSEC). Note that this speedup is above the parallel speedups already provided when using a parallel implementation supported by Hoplite. In Figure 15, we observe FastTrack enhanced speedups as high as 2.5× for SpMV, 2.8× for Graph Analytics, 1.4× for LU factorization, and 2× for Multi-Processor workloads. The speedups grow with increasing PE counts. Certain benchmarks like `hamm_memplus`, `bomhof_circuit_2`, `roadNet-CA`, and `freqmine` are characterized by predominantly local traffic and does not need nor benefit from a faster NoC. Performance scaling is best for Graph workload at large PE counts highlighting the benefit of this NoC at larger system sizes. For dataflow workloads, we see most of the speedups at 256 PEs, due to serialization bottlenecks within the PE logic for smaller PE counts. We note that the LU factorization traces are notoriously difficult to parallelize with low ILP, and the observed performance improvements are a direct result of using FastTrack.

To what extent can FastTrack improve worst-case latency of the deflection-routed NoC? Deflection routing typically exacerbates the packet routing latencies due to the possibility of repeated deflections of an unfortunate packet. FastTrack should be able to alleviate this high cost by allowing deflected packets to use the faster links. In Figure 16, we show a packet latency histogram for an 8×8 NoC routing the `RANDOM` traffic pattern. We see the, at $<$ 10% injection rate, the worst case packet latency for the fully populated and depopulated FastTrack NoC which is 7× and 3× smaller than base Hoplite respectively. The reduction in latency is a more direct consequence of the use of fast links that skip switch hops and reduce the number of cycles required for traversals.

If we vary length D, and R of FastTrack NoC express link, how does that influence performance? Now that we have established the performance advantages of using the FastTrack NoC, we now investigate opportunities for tradeoffs when configuring the NoC. FastTrack topology generation offers us the opportunity to vary D and R



(a) Sparse Matrix-Vector Multiplication (b) Graph Analytics (c) Token LU Factorization Dataflow (d) Multi-Processor Overlay

Figure 15: FPGA accelerator communication traces for Sparse Matrix-Vector Multiplication, Graph Analytics, Dataflow, and Multi-Processor Overlays. Speedups reported are for best FastTrack configuration against baseline Hoplite performance.

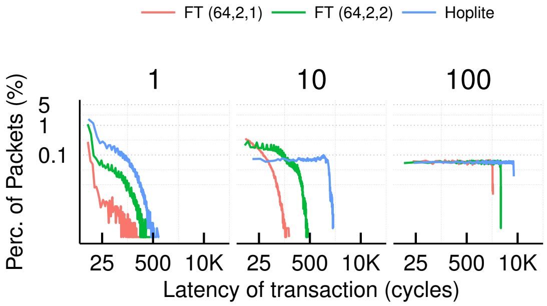


Figure 16: Histogram of packet latencies for a 64 PE FastTrack NoC configurations (D and R combinations) for RANDOM traffic.

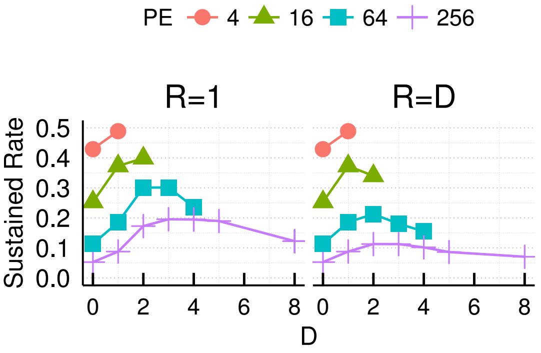
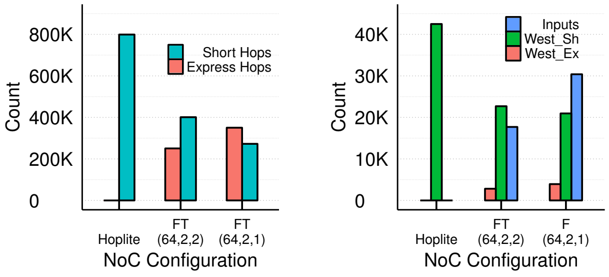


Figure 17: Effect of varying the distance of hop D on sustained rate of the RANDOM workload for various NoC system sizes.

together to determine the best balance between additional cost and delivered performance. The length D of the express influences throughput of a fully populated FastTrack NoC as shown in Figure 17 (R=1 case) for 50% injection rate of RANDOM traffic on an 8×8 NoC. One would expect larger D to result in superior performance, but this is true only upto a limit. For the 8×8 NoC, we see a drop in throughput for D=4, with better performance at D=2 and 3. This can be explained by considering the effect of providing links that are too long; packets are unable to exploit the fast links for traversal distances $\Delta < D$. This eliminates an increasing subset of the workload from being able to use the fast links. We show the effect of extreme depopulation in Figure 17 (R=D case). With depopulation, the quantum of throughput improvements get reduced, but associated LUT implementation cost is lowered. Thus, we need to judiciously choose D and R for a cost-aware design.

Can we quantify the effectiveness of express links in the FastTrack NoC? The use of express links depends on the traffic pattern, and the degree of depopulation of the NoC. For the RANDOM pattern on an 8×8 NoC, we see the correlated increase in the use of express links and drop in the use of short links in Figure 18a. We observe that the use of the express links actually reduces the total number of deflections (short+express) for any FastTrack NoC. As we reduce the extent of depopulation, there are more opportunities for using express links, and as a result we observe a greater number of hops in those links. We track the cumulative number of deflected packets at each input port in Figure 18b for the same experiment configuration.



(a) Link Usage (b) Deflections

Figure 18: Tracking the usage of short and express links, and deflections for a 64 PE evaluation using RANDOM traffic.

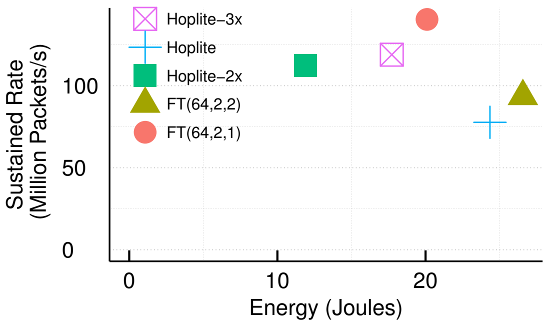


Figure 19: Throughput-Energy tradeoffs for a 64 PE NoC with RANDOM traffic.

Here, input deflections refer to packets that are forced to use a short link when they rather use a faster link. The use of FastTrack reduces the number of deflection on the West input by almost 25%. As we increase depopulation (increase R), we see fewer opportunities for Input to deflect to a short link and fewer deflections in the express links due to limited usage.

How does FastTrack compare to Hoplite in terms of dynamic energy efficiency? FastTrack NoCs contain express links that are longer than the baseline Hoplite links, and toggle 2× more registers to support data transfer on these longer links. However, FastTrack does route the workloads in less time, and also reduces deflections in the NoC resulting in lower activity rates. Here, we use dynamic power data, reported by Vivado Power analysis, from Table II and total time required for routing the workload to compute energy. In Figure 19, we show the sustained throughput of the NoC as a function of the energy spent in the system. FastTrack FT(64,2,1) beats baseline Hoplite throughput by almost 1.8× while requiring ≈20% less energy. Despite the higher power requirement of the longer links, a combination of superior throughput and fast clock rates on the long links made possible by fast FPGA interconnect, FastTrack outperforms Hoplite on energy efficiency. FastTrack FT(64,2,1) is slightly better than multi-channel replicated Hoplite NoC (3 channels) delivering a throughput improvement of 1.2× at a cost of 15% more energy. The two-channel replicated Hoplite NoC offers even lower energy but sacrifices throughput. The replicated Hoplite NoCs requires much lower energy as the link lengths are still short like original Hoplite, but the workload routing time is reduced due to parallelism.

VII. Discussion

We believe that changes to the FPGA architecture can provide even more benefits to a FastTrack-like design.

Hyperflex [36] interconnect in the Intel Stratix 10 FPGA family offers the ability to use configurable pipelining within the wiring infrastructure of the FPGA. The use of these resources will permit FPGA overlay NoCs to run blazing fast while exploiting deeply pipelined interconnect resources. In the context of FastTrack, there are two considerations. First, how does the use of Hyperflex affect the distance traveled by a packet in a clock? We expect the fundamental properties of wire speeds to stay unchanged and thereby still allow express links to significantly reduce end-to-end routing latency in the NoC. While a HyperFlex-pipelined NoC link will run at a high clock frequency it will need to traverse multiple pipeline stages resulting in a high end-to-end latency. Second, an adaptation to Hyperflex by exposing the selection of registering to user logic, may pave the way for the direct application of SMART NoC to FPGA fabrics. We recognize that this feature is currently not supported, but offer a reason to consider its integration.

Hard NoCs [11] can deliver NoC speeds and properties that are superior to FPGA overlay NoCs. While the results from this study were primarily using ASIC NoCs ported to FPGA fabrics for comparison, the key benefits remain valid even against a lean FPGA-friendly NoC such as Hoplite or FastTrack. What FastTrack shows, is the demand for configurability in wiring infrastructure even within a hard NoC solution. It is the position of these authors, that hardening NoC routers is inappropriate due to the sheer diversity of FPGA workloads, but hardening the NoC interconnect links offers a promising compromise. When hardening NoC links, lessons from FastTrack NoC engineering should hold valuable insights on how to make use of express long distance wiring in addition to short links for best outcomes.

VIII. Conclusions

The FastTrack FPGA overlay NoC shows how to exploit fast FPGA interconnect to improve the performance of packet-switched routing on FPGAs through the strategic use of express links. This is based on the observation that modern FPGA interconnect provides wires with variable length (speeds) that makes these faster express links possible. Our NoC implementation flow explores different FastTrack design configurations (length of express link, depopulation factor) and tailors the NoC to best support application traffic requirements. An 8×8 FastTrack NoC is 1.7–2.5× larger than baseline Hoplite NoC while operating at almost the same frequency, and requiring 2.5× more power. For statistical workloads, we observe throughput and latency improvements to the tune of 2.5× for sustained rate and almost 2× for average latency. When considering communication traces extracted from FPGA accelerator case studies such as Sparse Matrix-Vector Multiplication, Graph Analytics, Token LU Factorization Dataflow and Multiprocessor overlay applications, we see similar improvements of up to 2.8× over baseline Hoplite performance. FastTrack also beats Hoplite in terms of energy efficiency by a margin of 2.25× as it retains high frequency operation on the express links and takes advantage of faster throughputs.

Acknowledgments: The authors would like to thank Jan Gray for providing access to the Hoplite source code.

References

[1] C. Zhang *et al.*, “Optimizing fpga-based accelerator design for deep convolutional neural networks,” in *FPGA*, 2015.

[2] M. Alwani *et al.*, “Fused-layer CNN accelerators,” in *MICRO*, 2016.

[3] H. Sharma *et al.*, “From high-level deep neural models to fpgas,” in *MICRO*, 2016.

[4] X. Ma, D. Zhang, and D. Chiou, “FPGA-Accelerated transactional execution of graph workloads,” in *FPGA*, 2017.

[5] “The data vortex network.” http://www.datavortex.com/network.

[6] D. Unnikrishnan *et al.*, “Scalable network virtualization using fpgas,” in *FPGA*, 2010.

[7] A. M. Caulfield *et al.*, “Configurable clouds,” *IEEE Micro*, vol. 37, no. 3, 2017.

[8] Amazon Inc., “Amazon EC2 F1 Instances: Run Customizable FPGAs in the AWS Cloud.” https://aws.amazon.com/ec2/instance-types/f1/, 2017.

[9] J. Gray, “GRVI-Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator,” in *FCCM*, 2016.

[10] H. B. Kumar *et al.*, “120-core microaptiv MIPS overlay for the Terasic DE5-NET FPGA board,” in *FPGA*, 2017.

[11] M. S. Abdelfattah and V. Betz, “Networks-on-Chip for FPGAs: Hard, Soft or Mixed?,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, Sept. 2014.

[12] M. K. Papamichael and J. C. Hoe, “CONNECT: re-examining conventional wisdom for designing NoCs in the context of FPGAs,” in *FPGA*, 2012.

[13] Y. Huan and A. DeHon, “FPGA optimized packet-switched NoC using split and merge primitives,” in *FPT*, Dec. 2012.

[14] N. Kapre and J. Gray, “Hoplite: Building austere overlay NoCs for FPGAs,” in *FPL*, Sept 2015.

[15] Altera, “Applying the benefits of network on a chip architecture to fpga system design.” https://www.altera.com/en_US/pdfs/literature/wp/wp-01149-noc-qsys.pdf, Apr 2011.

[16] H. Kwon and T. Krishna, “OpenSMART: Single-cycle multi-hop NoC generator in BSV and chisel,” in *ISPASS*, 2017.

[17] J. Fowers *et al.*, “A high memory bandwidth FPGA Accelerator for sparse matrix-vector multiplication,” in *FCCM*, May 2014.

[18] V. Betz and J. Rose, “FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density,” in *FPGA*, 1999.

[19] G. Faanes *et al.*, “Cray cascade: a scalable hpc system based on a dragonfly network,” in *SC*, 2012.

[20] J. Kim, J. Balfour, and W. Dally, “Flattened butterfly topology for on-chip networks,” in *MICRO*, 2007.

[21] P. Salihundam *et al.*, “A 2 tb/s 6x4 mesh network for a single-chip cloud computer with dvfs in 45 nm cmos,” *IEEE Journal of Solid-State Circuits*, vol. 46, no. 4, 2011.

[22] T. Krishna *et al.*, “SMART: single-cycle multihop traversals over a shared network on chip,” *IEEE Micro*, vol. 34, no. 3, 2014.

[23] B. Grot *et al.*, “Express cube topologies for on-chip interconnects,” in *HPCA*, 2009.

[24] C. Fallin *et al.*, “Chipper: A low-complexity bufferless deflection router,” in *HPCA*, 2011.

[25] Y. Cai, K. Mai, and O. Mutlu, “Comparative evaluation of fpga and asic implementations of bufferless and buffered routing algorithms for on-chip networks,” in *ISQED*, 2015.

[26] B. K. Daya *et al.*, “Quest for high-performance bufferless nocs with single-cycle express paths and self-learning throttling,” in *DAC*, 2016.

[27] G. Michelogiannakis *et al.*, “Evaluating Bufferless Flow Control for On-Chip Networks,” in *NOCS*, May 2010.

[28] N. Kapre and J. Gray, “Hoplite: A Deflection-Routed Directional Torus NoC for FPGAs,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, Mar. 2017.

[29] N. Kapre *et al.*, “Packet switched vs. time multiplexed FPGA overlay networks,” in *FCCM*, 2006.

[30] S. Wasly *et al.*, “HopliteRT: An Efficient FPGA NoC for Real-Time Applications,” in *FPT*, Dec 2017.

[31] A. Parashar *et al.*, “SCNN: An accelerator for compressed-sparse convolutional neural networks,” in *ISCA*, 2017.

[32] R. F. Boisvert *et al.*, “The Matrix Market: A web resource for test matrix collections,” *Quality of Numerical Software: Assessment and Enhancement*, 1997.

[33] J. Leskovec and A. Krevl, “SNAP Datasets: Stanford large network dataset collection.” http://snap.stanford.edu/data, June 2014.

[34] N. Kapre and A. DeHon, “Parallelizing sparse matrix solve for spice circuit simulation using fpgas,” in *FPT*, Dec 2009.

[35] C. Bienia and K. Li, “Parsec 2.0: A new benchmark suite for chip-multiprocessors,” in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.

[36] D. Lewis *et al.*, “The Stratix$^{\text{TM}}$10 highly pipelined FPGA architecture,” in *FPGA*, 2016.