Efficient and Programmable Ethernet Switching with a NoC-Enhanced FPGA

Andrew Bitar, Jeffrey Cassidy, Natalie Enright Jerger, Vaughn Betz
Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto, Toronto, Ontario, Canada
{andrew.bitar,jeffrey.cassidy}@mail.utoronto.ca,{enright,vaughn}@ece.utoronto.ca

ABSTRACT

Communications systems make heavy use of FPGAs; their programmability allows system designers to keep up with emerging protocols and their high-speed transceivers enable high bandwidth designs. While FPGAs are extensively used for packet parsing, inspection and classification, they have seen less use as the switch fabric between network ports. However, recent work has proposed embedding a network-on-chip (NoC) as a new "hard" resource on FPGAs and we show that by properly leveraging such a NoC one can create a very efficient yet still highly programmable network switch.

We compare a NoC-based 16×16 network switch for 10-Gigabit Ethernet traffic to a recent innovative FPGA-based switch fabric design. The NoC-based switch not only consumes 5.8× less logic area, but also reduces latency by 8.1×. We also show that using the FPGA's programmable interconnect to adjust the packet injection points into the NoC leads to significant performance improvements. A routing algorithm tailored to this application is shown to further improve switch performance and scalability. Overall, we show that an FPGA with a low-cost hard 64-node mesh NoC with 64-bit links can support a 16×16 switch with up to 948 Gbps in aggregate bandwidth, roughly matching the transceiver bandwidth on the latest FPGAs.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*packet-switching networks*; C.2.6 [**Computer-Communication Networks**]: Internetworking—*routers*

General Terms

Design, Performance, Algorithms

Keywords

Switch architecture; Network-on-chip; FPGA

ANCS'14, October 20–21, 2014, Los Angeles, CA, USA.

http://dx.doi.org/10.1145/2658260.2658272.

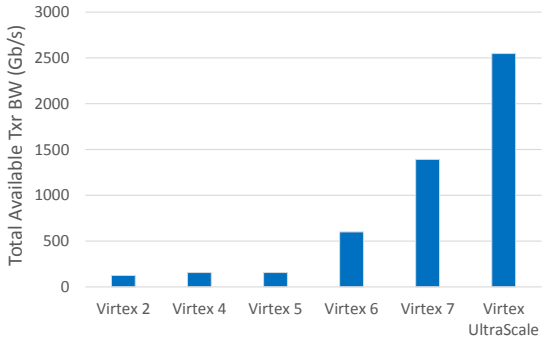


Figure 1: Growth of transceiver bandwidth in the Xilinx Virtex family from Xilinx datasheets [28]

1. INTRODUCTION

Field-programmable gate arrays (FPGAs) have seen widespread adoption in many industries thanks to their reduced engineering costs and faster time to market compared to application-specific integrated circuits (ASICs). The trade-offs that must be made when building designs on an FPGA rather than an ASIC were studied extensively in Kuon and Rose's work, where they showed a programmability overhead of 18×-35× in area and 3×-4× in critical path delay [20]. Despite this, the FPGA market has grown to approximately $5B / year in revenue, with the communications segment accounting for approximately 45% of that revenue.[1] While FPGA vendors continue to push the total transceiver bandwidth on their devices (Figure 1), the question remains whether designers can, in fact, saturate the available bandwidth. The FPGA's soft reconfigurable fabric runs considerably slower than ASICs, forcing designers to handle high bandwidth I/O with wide, slow buses. Not only does this consume significant programmable interconnect area, it also poses design challenges for timing closure.

Despite these challenges, FPGA's have the key advantage of reconfigurability. This characteristic lends itself well to the recent advent of software-defined networking (SDN) [22]. Increasingly, computer networks need the ability to efficiently adapt to new routing and forwarding protocols as they become available. Some recent work has proposed augmenting ASIC designs with some programmability through the use of match tables [9]; the resulting chip has moderately higher hardware cost than a conventional ASIC but enables flexible packet header processing and lookup table reconfig-

[1]Based on revenue reported by the major FPGA vendors.

uration. Should FPGA switch designs succeed in efficiently supporting modern bandwidth demands, then they could provide even greater programmability suitable for SDN.

Recent work by Dai and Zhu presented a new 16×16 FPGA-based switch fabric design that can support an aggregate bandwidth of 160 Gbps [11]. It did so by leveraging the FPGA's hardened memory resources. Using this hard resource, rather than the general-purpose programmable logic fabric, to perform much of the switching is key to successfully handling high bandwidth I/O. Prior to Dai and Zhu's work, there was little published material that showed how a high-radix FPGA-based switch could be built that saturates the available transceiver bandwidth.

Still, Dai and Zhu's switch relies on the FPGA's soft (programmable) interconnect both to bring data from the transceivers to the memory modules and to connect the memory modules. These connections are wide, requiring both a large amount of programmable routing to make all the connections, and many pipeline registers to close timing. Such designs generally require many time-consuming compilations and iterative re-pipelining to close timing. This added design effort is a growing problem in FPGA design, especially with the ever-increasing I/O bandwidth being brought on the chip [1]. As this trend continues, FPGA architecture needs a new interconnect that raises the level of abstraction and makes it simpler to achieve timing closure.

Thanks to the work done by Dally and Towles [13], many chip multiprocessors (CMPs) have adopted Networks-on-Chip (NoCs) to cope with high on-chip bandwidth. Recent work by Abdelfattah and Betz has proposed augmenting FPGA architecture with an embedded NoC to cope with the growing FPGA design challenges [1]. Their work makes a strong case for a NoC that is hardened in the chip's silicon, as it achieves significantly better bandwidth per area than "soft" programmable NoCs while consuming a small fraction of the FPGA area. They show that such a NoC more than pays for itself even if it only handles the distribution of data from a single high-speed DDR3 interface throughout an FPGA design [1]. In this work, we seek to determine if additional gains can be realized in a complete application. Demonstrating such applications would encourage FPGA vendors to augment their chips with this new form of interconnect.

We present a new implementation of a switch fabric crossbar using a NoC-enhanced FPGA. Rather than using the hardened memory resources to perform the switching, the design instead uses the hardened NoC proposed by Abdelfattah and Betz. The NoC not only manages to replace the switching logic, but also most of the soft interconnect needed to bring data from the transceiver to the switch points. This results in significant area and latency savings compared to Dai and Zhu's memory-based switch. The design also provides a high degree of programmability suitable for SDN. Thus, we show that switch fabric design is an important FPGA application that can benefit from an embedded NoC. In doing so, we make the following contributions:

- Describe four different possible NoC-crossbar configurations that can efficiently support a 16×16 160 Gbps switch fabric;
- Develop a custom routing algorithm tailored specifically to this application and show it outperforms traditional routing algorithms;
- Compute the NoC-crossbar's hardware cost and compare it to crossbar designs by Dai and Zhu [11] and Goossens *et al.* [16];
- Perform detailed throughput and latency analysis and simulation to quantify the NoC-crossbar's performance; and
- Demonstrate the switch's potential to scale to higher bandwidths.

N	16
Data Width	256 bits
Core Frequency	160 MHz
Latency	250 ns
Registers	36945 (12%)
LUTs	49537 (32%)
BRAMs	224 (27%)

Table 1: Memory-Based Switch Design on Virtex6-240T

In Section 2, we give a brief overview of Dai and Zhu's memory-based switch, Abdelfattah and Betz's work on a NoC-enhanced FPGA and related work on ASIC NoC-based switch fabrics. Our NoC-based crossbar design is described in Section 3, along with various possible design configurations and routing algorithms. We prove analytically that the switch can support sixteen 10 GbE ports. The hardware cost of our design compared to Dai and Zhu's, as well as another FPGA-based NoC-switch, is presented in Section 4. Section 5 presents our performance evaluation of the design, and Section 6 describes how the design can be scaled to higher bandwidth traffic.

2. BACKGROUND

2.1 Memory-Based Switch

Dai and Zhu have proposed a new 16×16 switch fabric design on an FPGA that is able to handle 10-Gigabit Ethernet traffic (160 Gbps aggregate) [11]. The design is described as a "memory-based switch" (MBS), as it uses the FPGA's hardened SRAM to perform some of the switching function, with the remainder being handled by multiplexers built from the FPGA logic fabric to cascade the memory-based switches into a larger crossbar. Since the SRAMs are hard resources, they can run at a much higher clock frequency than the soft FPGA logic. In their design built on a Xilinx Virtex6-240T, the memory modules are set to run at four times the clock frequency of the soft fabric.

Table 1 summarizes the properties of the MBS design. In comparison, a straightforward logic-based crossbar implementation would have consumed significantly more FPGA area, likely being unable to fit on the Virtex6-240T device. The majority of the logic utilization in the MBS comes from the pipeline registers needed for the wide, long connections that bring data to and from the transceivers, as well as the multiplexers and clock crossing logic. The hardware cost and latency of the MBS are compared with the NoC-based switch in Sections 4 and 5.

Hard/Mixed/Soft NoC	Hard (Hard routers and links)
Number of Routers	64
Channel Width	64 bits
Core Frequency	925.9 MHz
Topology	Mesh

Table 2: Properties of the embedded NoC used in our switch design

2.2 NoC-Enhanced FPGA

Abdelfattah and Betz [1] propose the inclusion of a hardened NoC as a new FPGA resource. Their baseline architecture, as depicted in Figure 2, provides a mesh topology with routers equally spaced across the chip. Since the routers do not carry the overhead of programmability, the network can run at a higher clock frequency, approximately 1 GHz in a 65 nm process, while using significantly less area than a comparable all-soft NoC such as that of Papamichael and Hoe [25]. The NoC's routers interface with the FPGA's programmable logic through a fabric port that absorbs the clock crossing logic needed to bring data from the slow FPGA fabric to the faster NoC. The fabric port is capable of interfacing with FPGA designs of any frequency.

The links between routers may be either dedicated ("hard NoC") or use the FPGA's programmable interconnect fabric ("mixed NoC"). The hard NoC provides a fixed mesh topology and uses no programmable interconnect, thus easing the design and routing of the circuit logic and providing the fastest and most area- and power-efficient design [4]. The mixed NoC has the advantage of flexible, programmable topologies. Due to additional capacitive loading from the interconnection switches, the mixed NoC must run at a lower clock rate and/or insert additional pipeline registers.

In building our NoC-based switch, we use the "hard NoC" proposed by Abdelfattah and Betz, as it runs at a higher clock frequency, thus supporting higher throughputs. Table 2 summarizes the properties of the NoC used by our design. Details of the NoC's router architecture are described in Section 3.

2.3 NoC-Based Switch Fabric Design

NoCs have been used to design high-radix ASIC-based switches targetting supercomputer networks [6, 26]. Ahn *et al.* [6] improve the efficiency of their NoC-based switch design by exploiting properties of the traffic pattern. They focus on topological optimization to reduce area based on how global traffic patterns manifest themselves as particular local traffic patterns within a single switch. Underwood *et al.* [26] also focus on topology modifications to realize a 4 TB/s switch based on a NoC for HPC applications.

Several recent proposals show that a NoC can produce a more efficient switch design than a conventional crossbar [16, 19, 23]. There are several benefits to this approach: faster clock speed due to short wires and simple, distributed routers; improved load balancing and path diversity; and improved scalability. Goossens *et al.* [16] propose a switch design that features a mesh network, load balanced routing and a unidirectional flow from inputs on one side of the mesh to outputs on the other side. Follow-on work [23] improves upon their architecture by placing I/O on all four sides of the chip, reducing the size of the required network. They

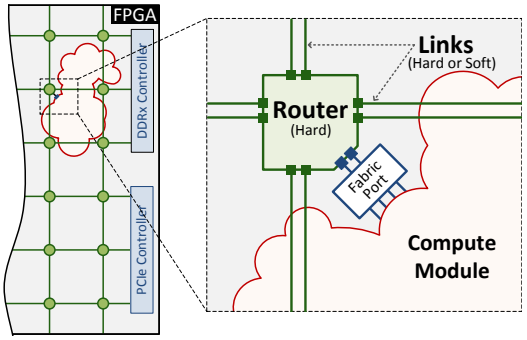


Figure 2: Depiction of the proposed hard NoC-FPGA fabric interface from Abdelfattah and Betz [1]

demonstrated a soft FPGA-based implementation of both of these designs [19].

In this work, we show that a flexible hard NoC coupled with the soft FPGA fabric greatly outperforms a logic-based NoC-switch implemented in a traditional FPGA. We also address the reality of FPGA transceivers being arranged in columns, generally on the east and west side of the chip, and the necessary overhead of bringing data from the transceivers to the NoC-crossbar. Despite using a hard NoC, we show that our design is flexible enough to implement different NoC-crossbar configurations and routing algorithms that can improve performance.

3. PROPOSED DESIGN

The relative economy of Dai and Zhu's memory-based switch was due to their careful optimization of the design to make maximal use of FPGA hard resources to avoid the programmability overhead. We propose to extend that concept by exploring use of the novel hard resource (NoC routers and links) that was proposed by Abdelfattah and Betz. Instead of "memory is the switch", we extend the concept of "the NoC is the switch" [16]. Using the 64-node embedded NoC, we design a crossbar that can support switching between 16 Ethernet ports each running at 10 Gbps. We call this design a "hard-NoC switch" (HNS).

The NoC's architecture is based on that presented by Abdelfattah and Betz [3], who used a parametrized open-source state-of-the-art virtual channel router [8]. The router has five ports, two virtual channels (VCs) per port, and an input buffer depth of ten flits per VC. The router has three pipeline stages and supports speculation which reduces the pipeline depth to two under low-load conditions. The router interfaces with the FPGA's soft logic through a programmable fabric port. Routers are connected with hard (non-programmable) links that are 64 bits wide. The NoC runs at a fixed clock frequency of 926 MHz [1]; thus, each link can support up to 59.3 Gbps.

Data arriving at the FPGA must go through some processing before being brought to the NoC to be switched to the appropriate output (Figure 3). The FPGA's transceiver performs clock recovery and determines the incoming serial data. The data is then converted from serial to parallel, with the exact parallel width chosen by the designer. For example, for 10G Ethernet, a conversion to 64-bit wide data at 160 MHz is reasonable. Once the data is brought onto the

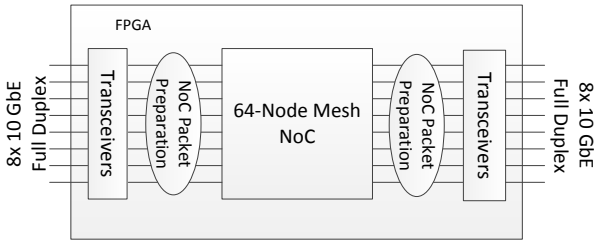


Figure 3: Conceptual overview: the NoC is the switch

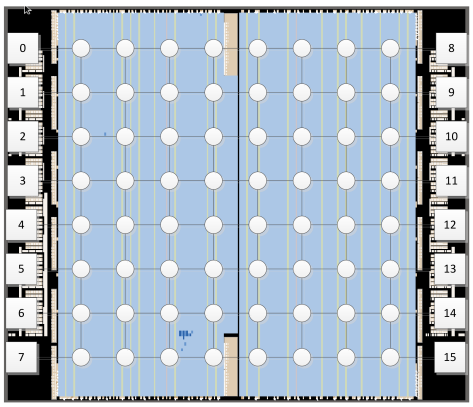


Figure 4: Stratix V floorplan showing transceiver columns at east and west sides with overlaid hard links and routers for NoC

FPGA fabric, the designer has the flexibility to do various amounts of processing before handing the data to the NoC. For example, packet header inspection and buffering can occur in the soft FPGA logic at 160 MHz. This logic examines the destination port of the Ethernet packet and inserts a NoC packet header indicating the appropriate destination router. The data is inserted into the NoC via the fabric port of the on-chip routers. Hardened clock conversion logic within the fabric port up-converts the data rate to the NoC clock rate of 926 MHz [1]; the data is then injected into the NoC. The NoC steers the data across the chip through multiple routers until it reaches the appropriate destination router, whose fabric port down-converts the data back to 160 MHz. More programmable logic can then buffer up flits until the entire Ethernet packet is ready to be sent out to the output transceiver. Note that by simply modifying the soft logic, the radix and communication protocol (e.g. 40G Ethernet or SONET) of the switch can be changed; this reprogrammability is key to the success of FPGAs, making it especially suitable for SDN. In the evaluation of our design, we focus on the crossbar fabric functionality, as other aspects of a full-featured switch, such as packet processing and error checking, are included at the discretion of the designer.

The NoC has the additional flexibility of being able to implement various routing algorithms and flow control mechanisms. We describe four different possible routing algorithms in Section 3.2, and evaluate their performance in Section 5. A virtual channel (flit-buffered), credit-based flow control mechanism is used in our NoC, as it is supported by the router architecture [8] and takes advantage of the available VCs. In contrast, Goossens *et al.*'s NoC-based switch design uses store-and-forward (packet-buffered) flow control in its NoC in order to be amenable to mathematical analysis [16]. They also focus on 53B ATM cells as the unit of data transfer in their network; our proposed design can support Ethernet frames up to 1518B; supporting large packets with store and forward flow control would lead to infeasibly large buffer sizes. Virtual channel flow control outperforms store-and-forward in NoCs, as it better utilizes buffer space [14] and reduces latency [12].

3.1 NoC Injection Point Placement

As shown in Figure 4, the Altera Stratix V FPGA (28nm) has transceivers in the east and west columns of the chip, as is typical in commercial FPGAs for layout reasons. By assigning one router node per transceiver, we can connect any receiver to any transmitter and implement crossbar functionality. The FPGA's programmable interconnect allows designers to configure these connections to best suit the switch's application. Selecting the optimal transceiver injection points in the NoC is analogous to the memory controller placement problem studied by Abts *et al.* [5].

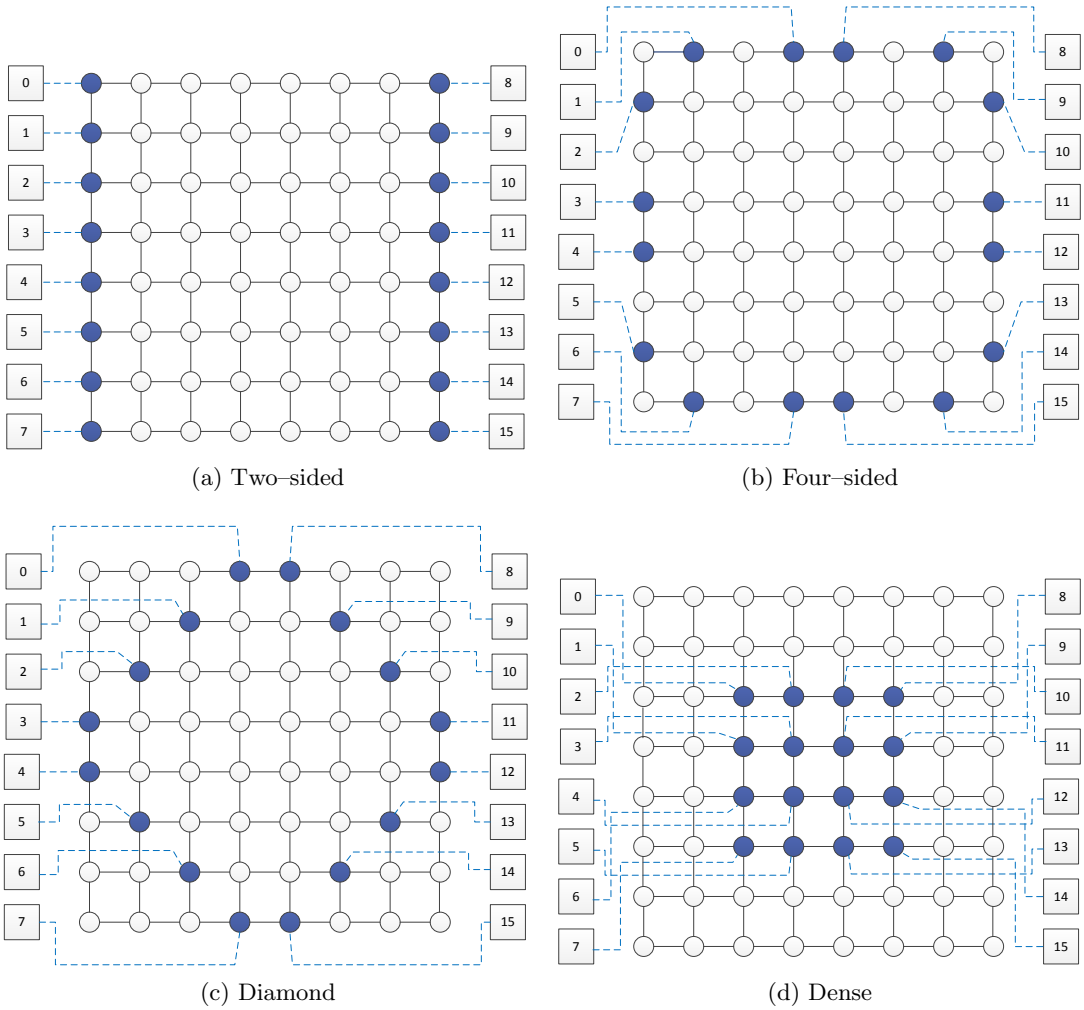
The simplest placement of injection points connects each transceiver to its nearest router, attaching only the 8 east- and west-most routers to a transceiver (Figure 5(a)). Routers in the middle of the chip do not inject or eject traffic, merely serving to connect the outer columns. We shall refer to this switch layout as "two-sided". Although the short connections have the benefit of no additional pipeline registers being consumed, the layout can result in poor utilization of the network – under certain traffic conditions, some links may be very heavily used while others carry no traffic at all, a situation explored in Section 5.

To better spread incoming traffic throughout the network, injection points can be distributed around the perimeter of the mesh, as in Figure 5(b), which we call the "four-sided" switch layout. Pipeline registers will be needed for the longer soft connections in order to close timing. One could also use the soft interconnect to set injection points in a "diamond" configuration, as in Figure 5(c). This injection point placement was inspired by the "diamond" configuration that proved to be efficient for memory controller placement [5]; however, the traffic patterns in the memory controller design and the switch design are quite distinct requiring further analysis and exploration.

Lastly, further extending the soft connections allows the 16 injection points to be configured in a conventional 4×4 mesh topology at the center of the 64-node network (Figure 5(d)). Such a switch configuration minimizes the average hop count between source-destination pairs, and thus has the most potential for latency savings. However, this comes at the cost of additional pipeline registers needed for the soft connections and reduced bisection bandwidth compared to the other configurations. We consider each of these four possible switch configurations in our evaluation.

3.2 Routing Algorithms

The NoC's routing algorithm determines the path taken by a packet from its source to its destination. An oblivious routing algorithm selects a path without considering the network's current state, such as contention along the chosen path. Algorithms that do consider the network's cur-



(a) Two–sided (b) Four–sided

(c) Diamond (d) Dense

Figure 5: Switch configurations – dashed blue lines represent soft connections that use the FPGA's programmable interconnect. Solid black lines represent the hard (non-programmable) NoC inter-router links.

rent state, known as adaptive routing algorithms, attempt to steer away from paths with high contention.

The simplest routing algorithm for a mesh topology is dimension-order routing (DOR). Packets are routed to their destination along one dimension, then along the second dimension. Such an algorithm is both oblivious and deterministic; it does not consider path contention nor does it take advantage of path diversity. However, it has the benefit of being very simple, requiring very little overhead in the packets and in the routers. It also does not need VCs to break deadlock, as the algorithm is inherently deadlock free. For each of the four switch configurations described above, a YX DOR algorithm is tested, where all packets are sent to their destination first along the Y dimension (North/South), then along the X dimension (East/West).[2]

An oblivious routing algorithm such as DOR fails to adapt to stressful traffic patterns that result in high contention. Thus, a minimal adaptive routing algorithm is also tested. Using local queue length as a metric for link contention, the minimal adaptive algorithm chooses the minimum route with the least contention at every hop. Choosing only among minimum routes ensures that packets travel from source to destination using the least amount of hops. However, this algorithm is not inherently deadlock free; two VCs are needed to break deadlock. One VC supports minimal adaptive routing while the other acts as an "escape" VC routing using DOR in case deadlock should occur [15].

As described by Dally and Towles [14], minimal adaptive routing only uses link contention data local to a router, so it can effectively balance *local* link loads, but not *global* link loads. We do not consider more complex globally adaptive algorithms because the nature of our traffic patterns can lead to much simpler traffic spreading algorithms. Since the location of the injection points in the network is known, a custom oblivious routing algorithm can be designed that spreads traffic globally across the network. A 64-node mesh with only 16 injection points can greatly reduce contention by keeping paths between source-destination pairs as disjoint

[2]YX routing was chosen over XY because it has a lower maximum channel load when injection points are arranged in columns.

```
int row, col; // row and column of
              // intermediate node
row = src_row;

if (src and dest are on same side) {

  if (src and dest are <4 hops apart) {
    col = src_col;
  }
  else {
    col = rand{0,1} away from src_col;
  }

}
else {
  col = rand{1,2,3,4,5,6};
}
```

Figure 6: Algorithm for selecting intermediate node in the Column-Select routing algorithm

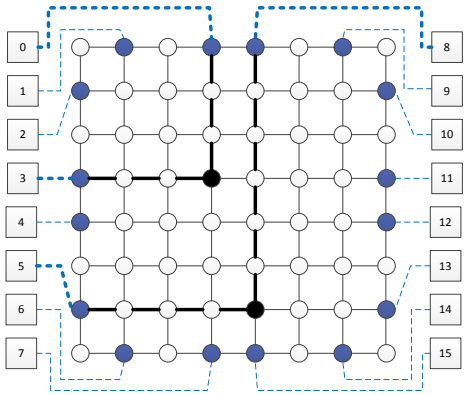


Figure 7: Illustration of the Smart DOR algorithm when routing node 5 to node 8, and node 3 to node 0. The paths between these two routes, which would have overlapped under YX routing, are separated by the algorithm. The black router represents the selected intermediate router.

as possible. We describe here two custom routing algorithms developed for the two-sided and the four-sided switch configurations. As both algorithms use two-phase routing, two VCs are sufficient to break deadlock [24]. On the other hand, a basic DOR algorithm already does a good job in keeping distinct source-destination paths separated in the diamond and dense configurations. We do not present a custom algorithm for those two configurations, since it was found that a custom oblivious routing algorithm does not improve latency much over DOR.

Two-Sided Custom Routing Algorithm: "Column-Select"

The two-sided switch layout with a DOR algorithm will result in all traffic using only the east- and west-most columns for routing in the Y dimension. Our custom routing algorithm focuses on better utilizing the middle six columns in the mesh. Similar to Valiant's algorithm [27], this is done by selecting an intermediate router in the mesh, routing all packets first to this intermediate node before routing to the destination. Routes to and from the intermediate node still follow YX routing. The algorithm for selecting the intermediate node is described in Figure 6. It aims to minimize the contention that occurs in the east- and west-most columns of the mesh, while still keeping most packets in minimal routes. To reduce contention caused by routing between nodes on the same side of the mesh, some packets are permitted to be routed one column away from the minimal route. Allowing these packets to deviate any further from the minimal route negates the latency savings from the reduction in contention. We call this algorithm "Column-Select", as it spreads traffic by selecting different columns for different routes. Unlike the other routing algorithms that we assess, Column-Select can theoretically cause packets to arrive out-of-order. Our simulations reveal that the frequency of out-of-order arrival is <0.2% for stressful permutation traffic.

Although Column-Select's non-determinism makes it more difficult to guarantee a low maximum channel load (as the same column may get picked in several consecutive instances), it does allow source-destination pairs to take advantage of path diversity. We show in Section 5 that the maximum channel load still remains low with both uniform random and permutations of stressful traffic.

Four-Sided Custom Routing Algorithm: "Smart DOR"

As with the two-sided custom algorithm, an intermediate router is analytically selected before sending the packet from the source. Since injection points are spread evenly across the perimeter of the mesh, the intermediate node selected is either the one on a XY or YX route. Of these two possible intermediate nodes, if there is one that is not on the perimeter of the mesh, then that is the one that is chosen. Otherwise, basic YX routing is used. We refer to this algorithm as "Smart DOR", as it is a deterministic algorithm that minimizes overlap of routes of distinct source-destination pairs by analytically selecting between an XY or YX route. Figure 7 illustrates how the algorithm prevents path overlap. Note that Smart DOR guarantees packets will arrive in order.

3.3 Crossbar Throughput Guarantees

To provide a crossbar functionality for the switch, the NoC must support the injection rate of the switch inputs. In other words, the NoC's maximum channel load at worst-case traffic must not exceed its link capacity. *Maximum channel load* is defined as the throughput of the channel carrying the highest throughput of all channels in the network for a given traffic pattern. Should the maximum channel load exceed the link capacity, the NoC would no longer be able to handle the injected data, resulting in packets being dropped. Of the four switch configurations described above, the two-sided configuration with YX routing is the least effective at spreading traffic across the network. Thus, by showing here that this configuration can support 16×16 switching of 10 Gbps traffic, then it can be concluded that the rest of the switch configurations and routing algorithms can support it as well. This is verified by simulation in Section 5.

A switch fabric crossbar must be able to switch traffic at the injection rate as long as every destination is only being driven by a single source at a given time. Since an output port can only sink traffic at up to the injection rate, a switch's ability to handle multiple sources sending traffic to a single destination is enabled not by its crossbar, but by its buffering and allocation mechanism. Efficient switch

Switch Implementation		LUT count	Register count	BRAM count	Tot. Equiv. LAB Area
Memory-Based Switch [11]		49537 (11.7%)	36945 (5.8%)	224 (9.7%)	**5850**
NoC-Based Switch	MDN [19]	75604 (17.8%)	52131 (8.1%)	0 (0%)	**7561**
	HNS Two-Sided	8960 (2.1%)	0 (0%)	-	**896**
	HNS Four-Sided	8960 (2.1%)	1024 (0.2%)	-	**999**
	HNS Diamond	8960 (2.1%)	1024 (0.2%)	-	**999**
	HNS Dense	8960 (2.1%)	2048 (0.3%)	-	**1101**

Table 3: Hardware cost of switch implementations. LUT count for HNS designs refers to equivalent LUT area of the hard NoC. Percentages refer to Stratix V-5SGTC5 resource budget (28 nm).

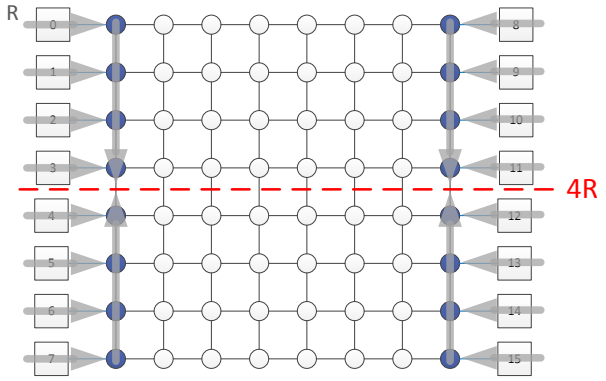


Figure 8: Worst-case maximum channel load for the two-sided crossbar configuration with YX DOR is 4R (R: injection rate)

buffering has been extensively studied in previous work, including Dai and Zhu's Combined Input and Grouped Crosspoint Queued architecture (GCQ) [11]. This paper focuses on the efficacy of using the NoC as the switch fabric's crossbar. Any necessary input or output buffering can be added to the crossbar by using the FPGA's memory resources.

Worst-case traffic for the two-sided crossbar can be derived as follows. Under YX routing, which does not allow 180° turns, the traffic across any horizontal cut (i.e. cutting vertical links) may only come from the east- and west-most columns of the mesh. Taking a horizontal cut across the middle of the mesh, the worst-case traffic occurs when all eight nodes from the bottom-half of the mesh are sending to the eight nodes at the top-half, and/or vice-versa (Figure 8). With YX routing, this results in traffic equivalent to four times the injection rate being sent across each of the east- and west-most links that are cut. Under this worst-case scenario, the maximum channel load is therefore four times the injection rate. For 10-Gigabit Ethernet traffic, the NoC's links must support 40 Gbps in order to maintain this throughput. The proposed 64-node, 64-bit wide NoC runs at 925.9 MHz [1], giving it a link capacity of 59.26 Gbps. We conclude that the 16×16 hard-NoC switch can guarantee 10 GbE throughput.

4. HARDWARE COST

Abdelfattah and Betz synthesize routers and hard links in a TSMC 65nm process [3]. Based on data provided by their web-based tool [2], the 64-node, 64-bit wide NoC consumes an equivalent area of 896 Logic Array Blocks (LABs) for both its routers and links. This accounts for the area consumed by the NoC crossbar, but not for the soft links between the NoC and the transceivers. Knowing that the NoC's routers are equally spaced across the FPGA, three to four adjacent routers cover a distance of approximately 1/3 to 1/2 of the chip. Since soft links run at 160 MHz (Section 3), those that bypass three to four routers will therefore require one stage of pipeline registers to close timing. Since the soft links are bi-directional (64 bits in each direction), every soft link requiring pipelining will need 128 pipeline registers. The four-sided and diamond configurations have 8 soft links needing pipelining, while the dense configuration has 16, resulting in 1024 and 2048 pipeline registers consumed, respectively.

Dai and Zhu provided a working Verilog design targeting a Xilinx Virtex-6 FPGA (40nm), and list synthesis results by FPGA resource type (shown in Table 3). Goossens *et al.*'s NoC-based switch, originally designed for ASICs [16], was also implemented as a soft design in an FPGA by Karadeniz *et al.* [19]. Their design, called the Multidirectional NoC (MDN), was synthesized on a Xilinx Virtex-5 FPGA (65 nm). The resource consumption results from their synthesis include the resources consumed by their design's Network Interface (NI) blocks. One NI is placed at each of their switch's I/O ports to perform buffering and partitioning of packets prior to sending them into the NoC-based crossbar. As we wish to only compare the hardware cost of the crossbar, the cost of the NI is removed in the MDN results shown in Table 3.

The most advanced process for an FPGA in production is 28 nm, which is offered by both Altera (Stratix V) and Xilinx (Virtex 7). We therefore convert the memory-based and NoC-based switch designs to a 28 nm equivalent for comparison purposes. Assuming the resource consumption of each design remains approximately the same when synthesized on Altera's 28 nm device, we compute the hardware cost as a percentage of a Stratix V-5SGTC5 device (Table 3).[3] In order to compare the area cost of each switch configuration, the resource count of each resource type (look-up table (LUT), register, BRAM) is converted to a total equivalent LAB count. Block RAM (BRAM) area cost is equivalent to four times the LAB area cost on Stratix V devices [21]. To be conservative, we assume that registers used in Dai and Zhu's and Goossens *et al.*'s designs do not consume any additional resources by using the registers in the Logic Elements (LEs)

[3]Xilinx and Altera devices have very similar LUT and register architectures. Virtex devices use 18Kb BRAMs, which are nearly equivalent in size to Stratix's 20Kb BRAMs.

Flit size	64 bits
Packet Size	64-1504 bytes (mean: 580)
Injection Rate	10 Gbps
Injection Process	Bernoulli
Router Delay	3 cycles (2 w/ speculation)
Router Buffer Depth	10 flits
Flow Control	Virtual Channel
Num. VCs	2
Sim Warmup Period	30,000 cycles
Sim Data Collection Period	100,000 cycles

Table 4: Simulation Settings

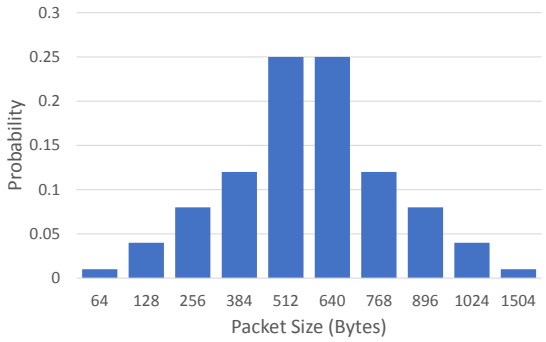


Figure 9: Distribution of packet sizes

that have already been consumed by the design's LUTs. On the other hand, we assume the registers used in the HNS designs consume their own LEs. Since 10 6-LUTs fit into Stratix LABs, we also assume that there are always exactly 10 LUTs per LAB.[4]

Our HNS switch configurations consume only 2.1% of the LUT area available, and little to no registers. When compared to the FPGA implementation of the MDN switch [19], the HNS consumes 6.9×-8.4× less area. The area savings is in large part due to the hardening of the NoC in the FPGA's silicon, as Abdelfattah and Betz showed that a hard NoC is 26× more area-efficient than an equivalent all-soft NoC [1].

Comparing to the memory-based crossbar, the HNS design consumes 5.5× less LUT area and 18× less registers. Moreover, the HNS does not use any BRAMs in the crossbar, unlike the MBS which relies on memory to perform much of the switching. Although use of the hardened SRAM resources allows the MBS to achieve better area-efficiency compared to the MDN switch, it still suffers from high amounts of pipelining for its soft connections. Additionally, while the MBS implements clock crossing logic in the FPGA's logic fabric, Abdelfattah and Betz's NoC includes a fabric port that subsumes this logic [1], leading to additional area savings. Overall, the HNS crossbar is 5.3×-6.5× more area-efficient than the MBS crossbar.

5. PERFORMANCE EVALUATION

Two key performance metrics must be evaluated for a switch: throughput and latency. The throughput supported by the NoC is dependent on its maximum channel load. This can be analytically determined based on the injection points placement and the routing algorithm used. Latency, on the other hand, is highly dependent on the applied traffic.

5.1 Evaluation Setup

In order to measure the average latency of the different proposed configurations of the HNS, the design was simulated in Booksim, a comprehensive interconnection network simulator [18]. Along with providing standard traffic generators and network topologies, the simulator also provides the ability to create custom traffic patterns and routing algorithms, which was necessary in the evaluation of the HNS. Booksim was also used by Dai and Zhu to evaluate the performance of the memory-based switch, enabling a fair comparison for the two designs.

[4]Note that it is possible to fit more than 10 LUTs in a single LAB if (n<6)-LUTs are used.

Table 4 summarizes the simulation settings used in the evaluation of latency. The flit size was set to match the channel width of the NoC. Dai and Zhu tested their memory-based switch only at two possible packet sizes, 32 bytes and 512 bytes. Since Ethernet frames range in size from 64 to 1,518 bytes [17], we instead test our switch with a distribution of packet sizes spanning this range, as this is both a more realistic and more difficult test scenario. Every time a source wished to inject a packet into the network, it randomly selected between ten different packet sizes based on the probability distribution shown in Figure 9. This distribution was not based on a specific application; although different applications that use Ethernet traffic have characteristic packet size distributions, we use a normal-like distribution to test our design in the most general scenario.

10-Gigabit Ethernet traffic can support data rates up to 10 Gbps. This implies that the average Ethernet data rate is below 10 Gbps. However, in order to demonstrate that the HNS can fully support 10 GbE, a Bernoulli injection process was used with an average injection rate of 10 Gbps. Although Ethernet traffic typically has bursty behaviour, the data rates of the bursts can never exceed the capacity of the Ethernet channel. Our latency measurements represent a more aggressive scenario as the injection rate may exceed 10 Gbps at certain intervals of the simulation.

The simulated router architecture is similar to the one proposed by Abdelfattah and Betz for their embedded NoC. The router operates with a 3-stage pipeline, with the possibility of being reduced to two stages given successful speculation. With 2 VCs and a buffer depth of 10 flits, the router's buffer size is far below the average size of Ethernet packets being routed. Although it is usually recommended that router buffer size be equal to a packet size for on-chip networks, that is unrealistic given the large packets in this application. The most critical factor in buffer sizing is being able to cover credit turnaround time, and our ten-flit buffers are more than sufficient for this purpose. Goossens *et al.* proposed splitting large packets into several smaller portions that are routed independently by the NoC and reassembled at the output [16]. However, since an Ethernet frame must be sent with no intra-packet gaps, the latency of a packet becomes at least the latency of the portion that takes the longest to reach the output.

The switch's general use case was modelled with uniform random traffic, a traffic pattern that models every transceiver being equally likely to send packets to any of the others. The same traffic pattern was used by Dai and

Zhu to test the memory-based switch, therefore providing a fair comparison point. The random nature of this traffic pattern permits cases where multiple sources are sending packets to a single destination at the same time – a case that does not need to be supported by a switch fabric's crossbar. Since our NoC-based crossbar has built-in buffering at the crosspoints (i.e. routers), it is able to support such traffic scenarios, unlike a conventional fully-connected crossbar. However, more strenuous traffic patterns with higher frequencies of multiple-sources-to-single-destination (MSSD) would need input and/or crosspoint/output buffering [11]. As mentioned previously, this can be added using the FPGA's memory resources.

Permutation traffic is typically used to stress a network [14]. To stress our crossbar, we run permutations of distinct source-destination pairs, where a single source drives a single destination for the entirety of the simulation. We use a script that searches for a traffic permutation that leads to the worst-case latency for each crossbar configuration under a single-source-to-single-destination (SSSD) traffic pattern. This traffic pattern also verifies that the HNS can support the rated aggregate bandwidth of the switch.

5.2 Throughput Results

The maximum channel load achieved by each routing algorithm for each HNS configuration can be seen in Table 5. The worst-case maximum channel load was analytically determined (Section 3.3), then verified using stressful SSSD traffic permutations in simulation. Maximum channel load under uniform random traffic (UR) was also measured in order to measure the throughput supported by the switch in the general case.

Having the injection points lined up in only two of the possible eight columns in the two-sided configuration leads to the worst maximum channel load. As shown in Section 3.3, a maximum channel load of four times the injection rate still allows the embedded 64-node NoC to handle 10-Gigabit Ethernet traffic, as the maximum supported link bandwidth is approximately six times the injection rate (∼59 Gbps). By using our custom two-sided routing algorithm, Column-Select, the maximum channel load under stressful permutation traffic is reduced to twice the injection rate. The two-sided switch configuration can thus support the same throughput as the other configurations, and at the least area cost since it does not need additional pipeline registers for its soft transceiver-to-NoC connections. The Smart DOR custom four-sided routing algorithm achieves the same maximum channel load reduction.

If a designer wishes to keep the routing algorithm simple, then the results show that changing to a Diamond or Dense configuration can also reduce the maximum channel load to twice the injection rate, while keeping the routing algorithm as a simple YX DOR scheme. This flexibility highlights one of the benefits of using a FPGA; the switch designer can choose whether to use an improved routing algorithm or change the switch injection points in the NoC in order to achieve their desired throughput.

Overall, through routing algorithm and/or configuration changes, the worst-case maximum channel load of the NoC-based crossbar can be brought down to as low as twice the injection rate (Table 5). Since the maximum link capacity that can be supported by the 64-node, 64-bit wide NoC is 59.3 Gbps, the maximum injection rate that can be fully supported is 29.6 Gbps. Thus, the HNS crossbar can fully support a 16×16 switch with an aggregate bandwidth of 474 Gbps (29.6 Gbps per port). In the general case of uniform random traffic, maximum channel load can be reduced to the injection rate. Under such traffic, the crossbar can support an aggregate injection bandwidth of up to 948 Gbps (59.25 Gbps per port).

Switch Configuration	Routing Algorithm	Max. Channel Load (R: injection rate)	
		UR	Permutation
Two-sided	YX	2R	4R
	Min Adapt	2R	4R
	Column-Select	1R	2R
Four-sided	YX	1R	4R
	Min Adapt	1R	4R
	Smart DOR	1R	2R
Diamond	YX	1R	2R
	Min Adapt	1R	2R
Dense	YX	1R	2R
	Min Adapt	1R	2R

Table 5: Maximum channel load of different switch configurations and routing algorithms

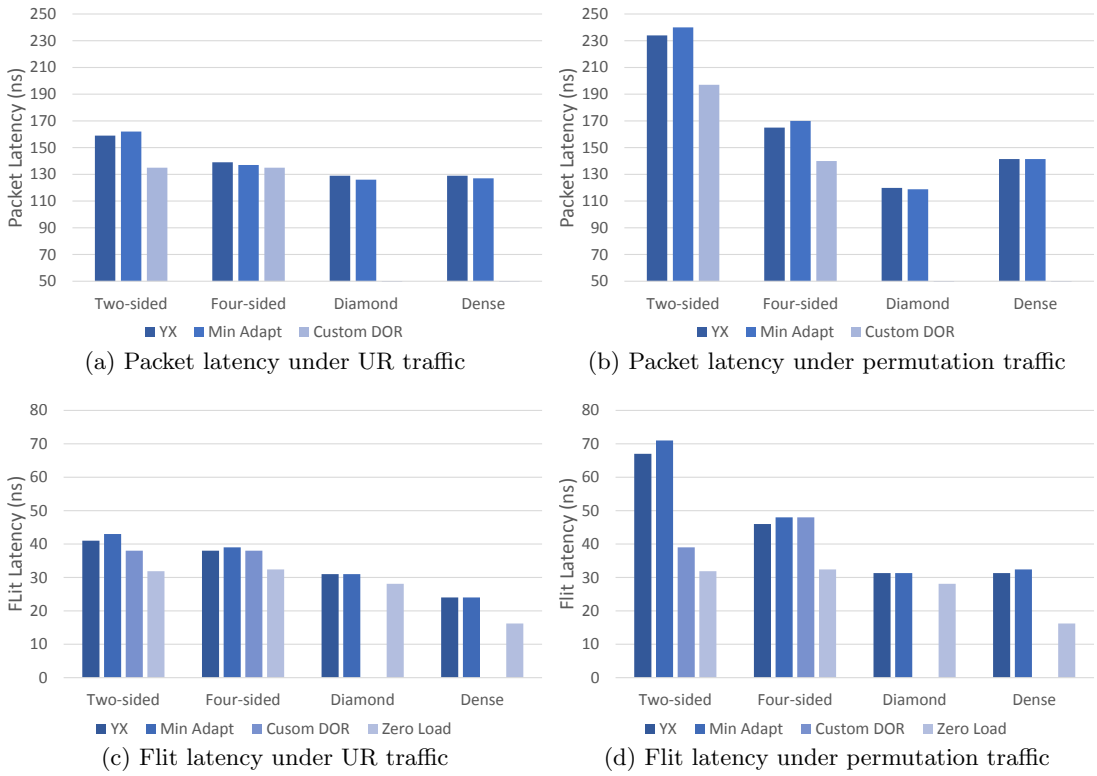
5.3 Latency Results

Packet and flit latency results for each HNS configuration are shown in Figure 10. The flit latencies are compared with the zero-load latency of each configuration. The zero-load latency is defined as the average flit latency over all source-destination pairs when there is no routing contention. This acts as the lower bound flit latency for each crossbar configuration.

When comparing the four different HNS configurations with basic YX routing, the two-sided configuration has the lowest area consumption, but worst latency. Because it connects each transceiver to its nearest router, it does not need any long connections that require pipeline registers. However, this is also its downfall; the six innermost columns of routers add unnecessary latency to the network. Using YX routing creates severe contention, as multiple source-destination pairs are routed along shared paths. Interestingly, the ability of minimal adaptive routing to adapt to network contention does not reduce latency in the two-sided configuration. This is likely because the majority of contention in the network happens when routing within the east-most or west-most columns of the mesh. Since the path choices available to minimal adaptive routing must be minimum paths, the algorithm cannot escape contention when routing within those two columns. A fully adaptive routing algorithm could better steer away from this contention, but would require a more complex router to support it [14].

In Section 3, we showed that a custom oblivious routing algorithm can be designed to make better use of the network resources, thereby reducing network contention. This is verified by Figure 10, as the Column-Select and Smart DOR algorithms lead to a packet latency reduction of 15.8% and 15.1%, respectively, under stressful permutation traffic.

The four-sided, diamond, and dense crossbar configurations take advantage of better injection point placement to



(a) Packet latency under UR traffic

(b) Packet latency under permutation traffic

(c) Flit latency under UR traffic

(d) Flit latency under permutation traffic

Figure 10: Packet and flit latency under uniform random (UR) and stress permutation traffic. "Custom DOR" refers to Column-Select for the two-sided configuration, and Smart DOR for the four-sided configuration.

reduce network contention. Under uniform random traffic, the dense configuration has the lowest latency results thanks, in part, to its reduced zero-load latency. However, the diamond configuration succeeds the most at spreading traffic throughout the 64-node mesh, thus achieving the best performance under stressful permutation traffic. With simple YX DOR, changing the crossbar configuration from two-sided to a diamond results in a latency reduction of 48.7%. In fact, permutations of SSSD traffic lead to better diamond performance compared to uniform random traffic where MSSD traffic is possible. This is because a diamond's injection point placement keeps most distinct source-destination routes separated, allowing it to handle SSSD traffic with minimal congestion. The fact that the diamond's flit latency approaches the zero-load latency of the configuration verifies that congestion is very low. The diamond configuration therefore performs best for the crossbar functionality of the switch fabric.

6. DISCUSSION

6.1 Area and Performance Wins

In proposing the HNS, we sought to present an important FPGA application that could benefit from an embedded NoC. The HNS yielded advantages over previous efficient FPGA-based switch designs, such as the memory-based switch, confirming that a NoC-enhanced FPGA would be a useful architecture for this application. In Section 4, we showed the hard-NoC switch was 5.3×-6.6× more area efficient than the memory-based switch. In order to compare latency, we consider the fact that the crossbar traversal latency of Dai and Zhu's MBS is 250 ns (Table 1). The average crossbar traversal latency of the HNS is measured by the flit latency in Figure 10. The diamond configuration has an average flit latency of 31 ns under both uniform random and stressful permutation traffic – more than 8× better than the memory-based switch. This performance advantage can be largely attributed to the NoC's higher clock frequency (926 MHz vs. 160 MHz).

Freeing the FPGA's programmable logic in the HNS design leaves potential for other system components to run in parallel. Packet processing is a possible application, as it could perform packet manipulations necessary for a given protocol. Attig and Brebner showed that packet parsing logic that can handle up to 343 Gbps of Ethernet traffic consumes 9.2% of a Virtex-7 device [7], which can easily fit alongside the HNS.

One key feature of the embedded NoC is its ability to easily interface with all of the different FPGA I/Os. This includes the ability to communicate with DDR memory [1]. Should a certain output port on the FPGA be overloaded by a burst of traffic, packets destined for that port could be steered to the FPGA DDR controller through the NoC and temporarily buffered in DDR until the traffic burst subsides. Thus, packets could be preserved, rather than dropped, while network issues are handled elsewhere. Traffic management logic to handle such a scenario could be implemented in the FPGA's programmable logic fabric.

6.2 Preserving the FPGA's Generality

Direct network topologies, such as a mesh or torus, are not designed for high-radix switching applications. Instead, indirect networks, where a node can only act as either a terminal (injects/accepts packets) or a router (routes packets), provide better suited topologies. For example, Ahn *et al.* investigated indirect Clos and butterfly topologies for their ASIC-based NoC-switch [6]. However, the NoC-enhanced FPGA used in our design was not built specifically for a switch application; Abdelfattah and Betz sought to introduce a new communication architecture that would benefit many different FPGA applications [1]. A mesh is a low-cost topology that efficiently services the entire FPGA.

Despite using a mesh topology, the HNS design can still support high throughputs at low latencies, and scale to even higher throughputs in an area-efficient way (Section 6.3). The coupling of the FPGA programmable fabric with the hard NoC makes this possible; customizing the NoC injection points via the programmable interconnect led to 48.7% reduction in latency. Implementing application-specific routing algorithms also led to significant latency improvements. All of this was done without sacrificing the generality of the FPGA; an entirely distinct application could efficiently use the same NoC by setting up its own injection points and routing algorithm. In contrast, Karadeniz *et al.* also used the FPGA to build a NoC-based switch without sacrificing the FPGA's generality [19], but the cost of building the entire design from the FPGA fabric resulted in 6.9×-8.4× worse area consumption.

6.3 Scalability

With high bandwidth applications such as server virtualization and cloud computing becoming increasingly common, switch fabrics must be able to scale efficiently to serve growing bandwidth demands. This includes being able to support the next generation of Ethernet. Chanda of Cisco Systems described 40-Gigabit Ethernet as "the next logical step in the evolution of the data network," predicting that 40 GbE switching will completely replace 10 GbE by 2018 [10]. By using a NoC-enhanced FPGA for our switch fabric, we present a flexible design capable of being reconfigured to support new generations of traffic protocols.

We now propose scaling the HNS design to support 40-Gigabit Ethernet. If the aggregate bandwidth that needs to be supported by the switch remains 160 Gbps, then the radix of the switch reduces from 16×16 to 4×4. In this scenario, the architecture of the NoC-based crossbar does not need to change. The programmable logic bringing the data from the transceivers to the NoC can be reconfigured to split all 40-Gigabit traffic entering the chip into four different injection points into the NoC, which are independently switched to the output, then re-assembled by more programmable logic before being brought to the output transceiver. The NoC-based crossbar effectively remains 16×16, but is now supporting a 4×4 switch of higher port bandwidth through channel bonding. The FPGA's logic would need to be wide and fast enough to handle 40 Gbps, which can be done by running 128-bit wide buses at 312.5 MHz, or 256-bit wide buses at 156.25 MHz. Both of these can be supported by the latest FPGA devices.

It is more likely that the aggregate bandwidth demand on the switch will also increase with future generations of Ethernet. Let us consider a 16×16 switch fabric that must support 40-Gigabit Ethernet at each of its ports – an aggregate bandwidth of 640 Gbps. As is shown in Section 5, the 64-node, 64-bit wide NoC-crossbar can fully support an aggregate bandwidth of 474 Gbps, while offering up to 948 Gbps in the uniform random case. To fully support a 640 Gbps switch, architectural changes to the NoC are needed.

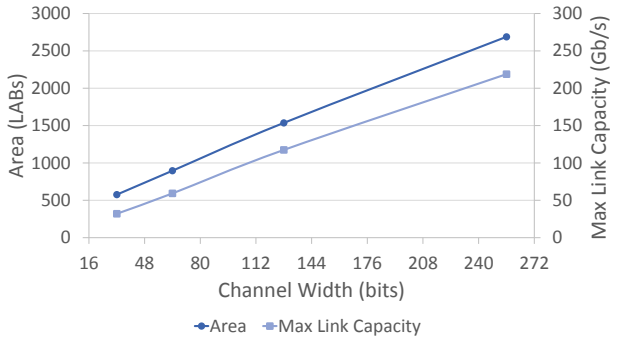


Figure 11: Hardware cost and max link capacity of varying the 64-node NoC's channel width [2]

We propose widening the NoC channel width from 64-bit to 128-bit as an efficient and feasible method of scaling the crossbar. Widening the NoC's channels increases its link capacity from 59.26 to 117.4 Gbps, at an area increase of 1.7× (Figure 11). The total logic area of the 64-node, 128-bit wide NoC still only consumes 3.6% of a Stratix V-5SGTC5 FPGA. Thus, this widened NoC can efficiently implement a 16×16 crossbar capable of fully supporting up to 939 Gbps in aggregate bandwidth (58.7 Gbps at each port).

Future generations of Ethernet will undoubtedly support bandwidths that go beyond 40 Gbps. 100-Gigabit Ethernet is already on the horizon, with talk of 1-Terabit Ethernet soon becoming a possibility. As illustrated in Figure 11, widening channel width beyond 128 bits can continue to provide even higher throughput, at the cost of more transistor consumption. We predict that as the number of transistors on a chip continues to grow in future generations of FPGAs, having an embedded NoC with wider channels will remain a small fraction of the device's area budget. Thus, widening the NoC's channel width provides a means to efficiently scale the HNS design to future generations of Ethernet.

7. CONCLUSION

As FPGAs continue to support increasingly high I/O bandwidth, there is a growing need for a new FPGA interconnect architecture that can better support such data rates. Abdelfattah and Betz propose embedding a network-on-chip in the FPGA silicon, thereby providing designers a new interconnect that raises the level of abstraction and makes it simpler to close timing for high throughput designs.

In this work, we present an important FPGA application that can benefit from an embedded NoC. A network switch fabric designed using the NoC to perform the crossbar function is shown to support 16×16 switching at 10-Gigabit Ethernet rates. Additionally, we show how a designer can use the FPGA's reconfigurable fabric to customize the switch to support different traffic protocols and different NoC injection points. Four different injection point placements are proposed. Arranging the traffic injection points in a "dia-

mond" configuration leads to the best latency results, offering a 48.7% improvement over a "two-sided" configuration.

Customizing the NoC's routing algorithm also proved to be an effective way to improve its performance. By developing a Column-Select algorithm for the two-sided configuration, and a Smart DOR algorithm for the four-sided configuration, we achieve 15.8% and 15.1% latency reduction, respectively. Additionally, the algorithms allow the configurations to improve their maximum supported aggregate bandwidth from 237 Gbps to 474 Gbps, matching the bandwidth's of the "diamond" and "dense" configurations. Each of the configurations also managed to support up to 948 Gbps in aggregate bandwidth given uniform random traffic.

The NoC-based design is compared to another implementation of a switch done by Dai and Zhu using the traditional FPGA resources. Dai and Zhu's switch design takes advantage of the FPGA's hardened SRAM resources to saturate the transceiver bandwidth. The diamond HNS design consumes 5.8× less area and achieves an 8.1× latency reduction.

Finally, we show that widening the channels of the embedded NoC is an effective way to scale the switch to support future generations of Ethernet traffic, such as 40 and 100 GbE. The "hard-NoC switch" can support the high bandwidth of modern switch fabric design while preserving significant flexibility through the use of the FPGA's programmable logic. We conclude that network switch fabric designs clearly benefit from a NoC-enhanced FPGA, providing support for the inclusion of an embedded NoC in future FPGA devices.

8. ACKNOWLEDGEMENTS

The authors would like to thank Mohamed Abdelfattah and Mario Badr for their insightful discussions and opinions, David Lewis and Tim Vanderhoek for providing Stratix V relative block areas, and the anonymous reviewers for their valuable feedback. This work was supported by NSERC, Altera, and a QEII-GSST scholarship.

9. REFERENCES

[1] M. Abdelfattah and V. Betz. The case for embedded networks-on-chip on FPGAs. *IEEE Micro*, pages 80–89, 2013.

[2] M. S. Abdelfattah. NoC Designer. `http://www.eecg.utoronto.ca/~mohamed/noc_designer.html`, 2013. Accesed: 2014-05-07.

[3] M. S. Abdelfattah and V. Betz. Design tradeoffs for hard and soft FPGA-based Networks-on-Chip. In *FPT*, pages 95–103, 2012.

[4] M. S. Abdelfattah and V. Betz. The power of communication: Energy-efficient NOCS for FPGAS. In *FPL*, pages 1–8. IEEE, 2013.

[5] D. Abts, N. D. Enright Jerger, J. Kim, D. Gibson, and M. H. Lipasti. Achieving predictable performance through better memory controller placement in many-core CMPs. In *SIGARCH*, volume 37, pages 451–461. ACM, 2009.

[6] J. H. Ahn, S. Choo, and J. Kim. Network within a network approach to create a scalable high-radix router microarchitecture. In *HPCA*, pages 1–12. IEEE, 2012.

[7] M. Attig and G. Brebner. 400 Gb/s programmable packet parsing on a single FPGA. In *ANCS*, pages 12–23. IEEE, 2011.

[8] D. U. Becker. *Efficient microarchitecture for network-on-chip routers*. PhD thesis, Stanford University, 2012.

[9] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *SIGCOMM*, pages 99–110. ACM, 2013.

[10] G. Chanda. The market need for 40 gigabit ethernet. Technical report, Cisco Systems, 2012.

[11] Z. Dai and J. Zhu. Saturating the transceiver bandwidth : Switch fabric design on FPGAs. In *FPGA*, pages 67–75, 2012.

[12] W. J. Dally. Virtual-channel flow control. *IEEE TPDS*, 3(2):194–205, 1992.

[13] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *DAC*, pages 684–689. IEEE, 2001.

[14] W. J. Dally and B. P. Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[15] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *TPDS*, 4(12):1320–1331, 1993.

[16] K. Goossens, L. Mhamdi, and I. V. Senin. Internet-router buffered crossbars based on networks on chip. In *DSD*, pages 365–374, 2009.

[17] IEEE Standard 802.3 for Ethernet. Technical report, IEEE, 2012.

[18] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. Shaw, J. Kim, and W. Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *ISPASS*, pages 86–96. IEEE, 2013.

[19] T. Karadeniz, L. Mhamdi, K. Goossens, and J. J. Garcia-Luna-Aceves. Hardware design and implementation of a network-on-chip based load balancing switch fabric. In *ReConFig*, pages 1–7, 2012.

[20] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *TCAD*, 26(2):203–215, 2007.

[21] D. Lewis. Altera, personal communication, April 2014.

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM*, 38(2):69–74, 2008.

[23] L. Mhamdi, K. Goossens, and I. V. Senin. Buffered crossbar fabrics based on networks on chip. In *CNSR*, pages 74–79, 2010.

[24] T. Nesson and S. L. Johnsson. ROMM routing on mesh and torus networks. In *SPAA*, pages 275–287. ACM, 1995.

[25] M. K. Papamichael and J. C. Hoe. Connect: Re-examining conventional wisdom for designing NoCs in the context of FPGAs. In *FPGA*, pages 37–46. ACM, 2012.

[26] K. Underwood, E. Borch, J. Sizer, T. Stremcha, and M. Strom. Evaluating on-die interconnects for a 4TB/s router. In *ICS*, pages 203–212, 2013.

[27] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Theory of computing*, pages 263–277. ACM, 1981.

[28] Xilinx Inc. Xilinx Virtex Family Datasheets.