Design Tradeoffs for Hard and Soft FPGA-based Networks-on-Chip

Mohamed S. Abdelfattah and Vaughn Betz
Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, Canada
{mohamed,vaughn}@eecg.utoronto.ca

***Abstract*—Incorporating Networks-on-Chip (NoC) within FPGAs has the potential not only to improve the efficiency of the interconnect, but also to increase designer productivity and reduce compile time by raising the abstraction level of communication. By comparing NoC components on FPGAs and ASICs we quantify the efficiency gap between the two platforms and use the results to understand the design tradeoffs in that space. The crossbar has the largest FPGA vs. ASIC gaps: 85× area and 4.4× delay, while the input buffers have the smallest: 17× area and 2.9× delay. For a soft NoC router, these results indicate that wide datapaths, deep buffers and a small number of ports and virtual channels (VC) are favorable for FPGA implementation. If one hardens a complete state-of-the-art VC router it is on average 30× more area efficient and can achieve 3.6× the maximum frequency of a soft implementation. We show that this hard router can be integrated with the soft FPGA interconnect, and still achieve an area improvement of 22×. A 64-node NoC of hard routers with soft interconnect utilizes area equivalent to 1.6% of the logic modules in the latest FPGAs, compared to 33% for a soft NoC.**

I. Introduction

The programmable interconnect is in many ways the heart of an FPGA, but recently it has faced many challenges. First, increasing wire resistance and decreasing pass transistor performance in advanced process nodes lead to poor interconnect delay scaling [1]. Second, the high interconnect delay increases the time required to complete a design because it is difficult to predict critical paths from a functional (e.g. RTL) description. This necessitates time-consuming compile/analyze/re-pipeline iterations. Third, modern FPGAs incorporate high-speed I/O interfaces, such as DDR and PCIe. Distributing these data streams requires wide and fast datapaths within the FPGA fabric, consuming large amounts of interconnect and presenting a further timing closure challenge. Fourth, the need to program many individual switches to make each link leads to a large CAD problem and long compilation times. The low level of interconnect abstraction is also a barrier to dividing a design into modules for independent optimization and compilation, or partial reconfiguration.

One method to attack these problems is to use a higher-level protocol for some of the communication within an FPGA, such as that provided by networks-on-chip (NoC). With an NoC, fixed wiring between communicating modules is replaced by a network that routes packets to and from those modules. This not only improves wire utilization but also raises the level of abstraction and so facilitates modular design styles [2]. The network links in an FPGA design must be coded in a latency-tolerant manner, as the exact latency of a packet is usually not fixed. While this forces a change in design style, it simplifies timing closure because interconnect delay no longer affects the cycle time; instead it affects the number of cycles a packet takes to reach its destination. Another attractive FPGA application for NoCs is partial reconfiguration. When swapping modules, the newly configured module will only have to connect to an NoC interface to communicate to any part of the FPGA as opposed to having to route each of its nets in an already-functioning FPGA.

There are additional advantages to using a hard NoC on the FPGA. A hard NoC will not require configuration onto the soft fabric, making compilation simpler and faster. In addition, the modules communicating via an NoC are disjoint which allows their independent synthesis, placement, routing, and timing closure. Communication bandwidth requirements can then be used to determine the optimum position in the network for each module. Hard interfaces on the FPGA such as DDR, PCIe and gigabit Ethernet operate at high clock frequencies and require low latency, high bandwidth communication to various parts of the chip. A high-performance hard NoC is a good match to these interfaces as it can distribute data throughout the chip at similarly high rates without an excessive number of wires. Of course, a hard NoC also has area and delay advantages, which we explore in detail in this work.

There is prior work both in FPGA-ASIC comparison and on FPGA-based NoCs. A comparison of FPGAs and ASICs by Kuon and Rose is based on a set of benchmarks with different logic/memory/multiplier ratios [3]. We perform a narrower but more detailed comparison based on a high performance NoC router. Similar work by Wong et al. compares microprocessors on FPGAs vs. custom CMOS [4]. Lee and Shannon explore how topology parameters impact the frequency of a soft NoC [5]. LiPar [6], NoCem [7] and CONNECT [8] are three virtual channel (VC) NoCs implemented efficiently in soft logic on FPGAs. While architectural decisions were based on FPGA utilization in prior work, we give recommendations based on FPGA silicon area. There has also been interest in a hard NoC on an FPGA. Francis and Moore suggest that a circuit-switched network with time-division multiplexed links should be hardened on the FPGA [9]. Goosens et al. propose

978-1-4673-2845-6/12/$31.00 © 2012 IEEE

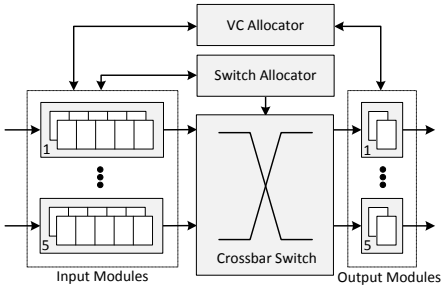

Fig. 1: A virtual channel router with 5 ports, 2 VCs and a 5-word input buffer per VC.

use of a hardwired NoC for both functional communication and FPGA configuration programming [10]. Chung et al. present a programming model that abstracts the distribution of data from external memory throughout the FPGA and mention that their application could benefit from a hard NoC [11]. The literature has shown both the implementation of soft NoCs and the desire for hard NoCs on FPGAs but there has not been a detailed comparison of soft (FPGA) and hard (ASIC) NoCs to date. We make that comparison on the router component level and use this data to give design recommendations for soft networks, and quantify the gains of hardening. Our contributions include:

- Quantify the area and performance gap of NoC components between FPGAs and ASICs.
- Investigate how these area and performance results impact router microarchitecture design decisions.
- Show how to integrate a hard NoC into the FPGA and quantify the overall gains.

II. Router Microarchitecture

We use a parametrized open-source state-of-the-art virtual channel router [12]. We focus on this full-featured router for two reasons. First as FPGAs increase in capacity and hence contain larger applications, it will often be necessary to traverse many network nodes to communicate across the chip, and consequently we expect routers with low latency, such as the chosen router, to be important. As we show in Section V, this router can also achieve high clock frequencies, helping it keep up with the throughput demands of the high bandwidth I/O interfaces on modern FPGAs. Second, since our focus is quantitatively examining the speed and area of hard and soft implementations of a wide variety of NoC components, use of a more full-featured router with more components yields a more thorough study; simpler routers would use a subset of the components we study.

The router operates in a 3 stage pipeline that can be reduced to 2 stages if speculation succeeds [12]. Ingress flits are stored in the input buffers and immediately bid for VC allocation; this is followed by switch allocation and switch traversal. Lookahead routing is done in parallel to switch allocation and is appended to the head flit immediately before traversing the crossbar switch. Finally, flits are registered at the output modules and then traverse inter-router links.

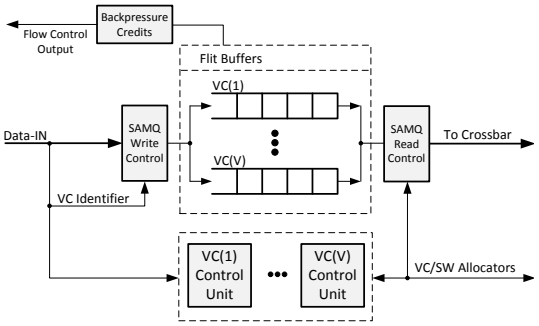

Fig. 2: Input module for one router port and "V" virtual channels. Not all connections are shown.

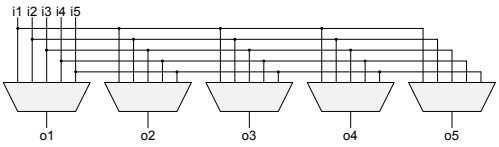

Fig. 3: A 5-port multiplexer-based crossbar switch.

Fig. 1 shows a block diagram of the router. For each of the presented router components there are different implementation variants. We restrict our discussion to architectures that satisfy current NoC cost limitations and bandwidth demands; that is, current implementation norms. The following references provide an in-depth discussion of implementation variations for each component [13–16].

A. Input Module

The main function of the input module is to buffer incoming flits until routing and resource allocation are complete. Depending on the VC identifier of the packet, it is stored in a different part of the input buffer. Routing information, already computed by the preceding router hop, is decoded and forwarded to the VC and switch allocators to bid for VCs and switching resources. The flit remains in the buffer until both a VC is allocated and the switch is free for traversal, at which point route lookahead information is attached to the head flit and it is ejected from the input module onto the switch.

Since route computation is done one hop earlier and in parallel to switch allocation, it does not impede router latency and effectively removes this step from the router pipeline [14]. Moreover, the low-overhead route computation logic is replicated for each VC to compute the route for all input ports simultaneously and support the queuing of multiple packets per VC [17]. A two-phase routing algorithm, known as Valiant's, routes first to a random intermediate node then to the destination node to improve load balancing [18]. The algorithm used to route each of the two phases is dimension-ordered routing which routes in each dimension sequentially and deterministically [13].

Dual-ported memory implements the input buffer. Internally, it is organized as a statically allocated multi-queue (SAMQ) buffer which divides the memory into equal portions for each VC [16]. Memory width is always the same as flit width to allow reading and writing flits in one cycle [17]. In this implementation, the memory buffer has one write port and

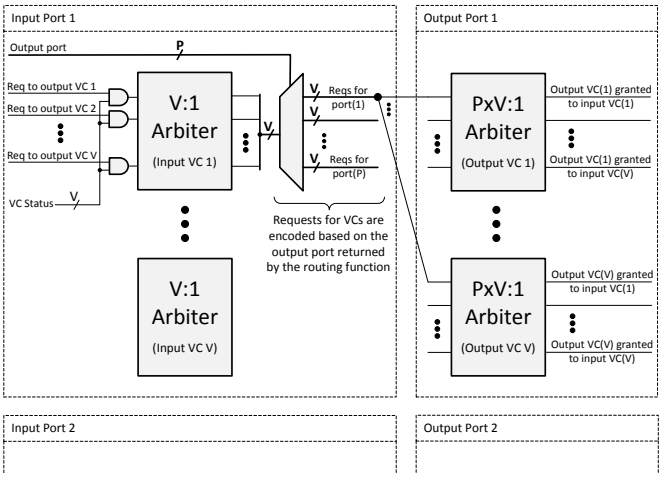

Fig. 4: Separable input-first VC allocator with "V" virtual channels and "P" input/output ports.

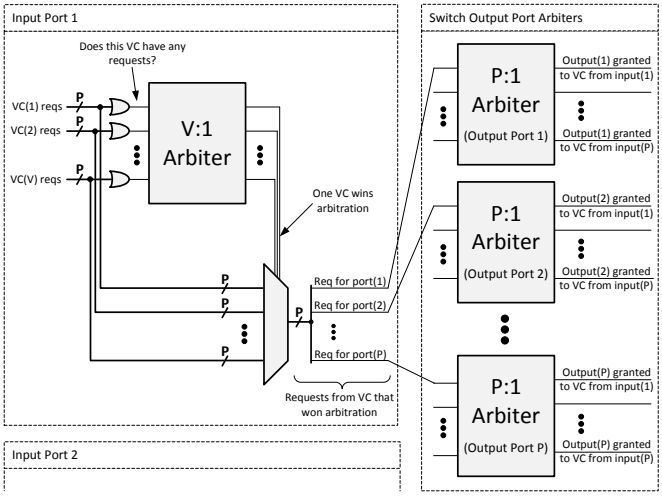

Fig. 5: Separable input-first switch allocator with "V" virtual channels and "P" input/output ports.

two read ports to allow the queuing of more than one packet in a VC buffer. This is required for simultaneous loading of a packet's tail flit and the next packet's head flit to update the destination port for the new route. Without this optimization, a pipeline stall is introduced between packets.

Fig. 2 shows a block diagram of the input module used in this study. The VC control units include the route-computation logic and state registers to keep track of the input VC status, the destination output port and the assigned output VC [17]. The SAMQ controller governs the read and write ports of the flit buffer. It selects the read and write addresses depending on the input VC and the granted output VC from switch allocation respectively. A backpressure control unit tracks buffer space availability per VC and transmits credits to upstream router ports on dedicated flow control links.

B. Crossbar

We use a multiplexer-based crossbar, as depicted in Fig. 3, rather than a tri-state buffer crossbar. Moderns FPGAs can only implement crossbars with multiplexers, and modern ASIC crossbars are also usually implemented this way. In the best-case scenario, five flits may traverse the crossbar simultaneously if each of them is destined for a different output port. However this rarely occurs, thus requiring the queuing of flits in the input module buffers until the output port is free and the flit can proceed. We found the ASIC crossbar area to be gate limited and not wire limited for the range of parameters considered here; other recent work has shown that crossbars as large as 128 ports are also gate limited [19].

C. Virtual Channel Allocator

VC allocation is performed at packet granularity. Like route computation, the body flits inherit the decision made for the head flit. The route computation unit selects an output port for a packet, and any available VC on that output port is a candidate VC for the packet.

Fig. 4 shows the separable input-first VC allocator used in this study. The first stage filters requests for output VCs by selecting a single output VC request for each input VC. This ensures that each input VC will be allocated a maximum of one output VC, which is necessary for correctness. The second stage ensures that each output VC is not over-allocated. For each output VC, where P is the number of ports and V is the number of VCs per port, the second stage takes $P \times V$ requests from all input VCs and grants one request. A virtual connection is established from the granted input VC to the output VC for the duration of one packet.

Arbiters grant requests in a round robin fashion which ensures that the most-recently granted requester has the lowest priority in the next round of arbitration [13]. For a large number of inputs, arbiters have long combinational delays. To circumvent this limitation, arbiters are organized hierarchically as tree arbiters, trading area for delay [15].

D. Switch Allocator

A switch allocator matches requests from $P \times V$ input VCs to P crossbar ports. Unlike routing and VC allocation, this is done on the flit granularity in a wormhole-switched router [13]. Fig. 5 shows a separable input-first switch allocator. In the first stage of switch allocation, VCs in a single input port compete among themselves for a crossbar input port. The granted VC forwards its requests to the second stage of arbitration which selects between all input ports bidding for each of the output ports. This ensures that only one VC can attempt to access the crossbar from each input port, and that only one crossbar input can connect to a crossbar output at a given time.

Speculative switch allocation can reduce the router latency from 3 cycles to 2 by performing VC and switch allocation together, provided that a VC is allocated in this cycle [20]. Architecturally, this duplicates the switch allocator; one instance handles non-speculative requests and the other handles speculative ones. Priority is given to non-speculative requests using reduction logic and selection circuitry [15].

E. Output Module

The crossbar output can be connected to the outgoing wires and the downstream routers directly. However, to improve

TABLE I: Estimated FPGA Resource Usage Area [4]

Resource	Relative Area (LAB)	Tile Area (mm^2)
LAB/MLAB	1	0.0221
ALM	0.10	0.0022
ALUT (half-ALM)	0.05	0.0011
BRAM - 9 kbit	2.87	0.0635
BRAM - 144 kbit	26.7	0.5897
DSP Block	11.9	0.2623

clock frequency a pipeline stage is placed at the crossbar outputs [17]. Furthermore, the output registers are replicated per VC to buffer an additional flit before proceeding downstream allowing one extra flit to traverse the crossbar before receiving credits from the downstream router.

III. Methodology

The router is implemented both on the largest Stratix III FPGA (EP3SL340) and TSMC's 65 nm ASIC process technology. This allows a direct FPGA vs. ASIC comparison since Stratix III devices are manufactured in the same 65 nm TSMC process technology [21]. Moreover, the area for Stratix III resources is publicly available [4]. Table I shows the area, including interconnect, of various FPGA blocks.

A. FPGA CAD Flow

Altera Quartus II v11.1 software is used with the highest optimization options to implement the router components. This excludes physical synthesis which reduced critical path delay by 5% and increased area by 15% on average. We set an impossible timing constraint of 1 GHz to force the tools to optimize for timing aggressively and report the maximum achievable frequency. While 1 GHz is clearly not a realizable constraint, we found it achieved timing and area optimization results comparable to those obtained with a timing constraint that is difficult, but just barely achievable. Clock jitter and on-die variation are modeled using the "derive_clock_uncertainty" command which applies clock uncertainty constraints based on knowledge of the clock tree [22].

All the circuit I/Os, except the clock, are tied to lookup tables (LUT) using the "virtual pin" option. This mimics the actual placement of an NoC router, and avoids any placement, routing or timing analysis bias that could result from using actual FPGA I/O pins. For example, the placement of the NoC component could be highly distorted by a large number of connections to I/Os located around the device periphery, when in a real system the input and outputs of the component would be within the fabric. Additionally, the use of virtual pins allows the compilation of designs with more I/Os than physically available on the FPGA.

Resource utilization is used to calculate the occupied FPGA silicon area by multiplying the used resource count by its physical area in Table I. Simply counting the used logic array blocks (LABs) or adaptive logic modules (ALMs) can overestimate the area required, as many LABs and ALMs are only partially occupied, and could accept more logic in a very full design. Instead, we use the post-routing area utilization from the resource section in the fitter report which accounts for the porosity in packing, thereby giving a realistic area estimate for a highly utilized FPGA. Note that the LUTs used for "virtual pins" are subtracted out.

The fastest FPGA speed grade corresponds to typical transistors, whereas the slowest FPGA speed grade matches the worst-case transistors of a process. For the best comparison, we use the fastest FPGA speed grade and the typical transistor model for the ASIC tools. For purely combinational modules, such as the crossbar, registers are placed on the inputs and outputs to force timing analysis. Maximum delay is extracted from the timing reports using the most pessimistic (slow, 85 ^{o}C) timing model, assuming a 1.1 V power supply.

B. ASIC CAD Flow

Synopsys Design Compiler vF-2011.09-SP4 is used for synthesis, and area and delay estimation. The general-purpose typical process library is used with standard threshold voltage and 0.9 V supply voltage. Unlike the FPGA CAD flow, timing constraints and optimizations impact the ASIC area dramatically. This is because timing optimizations entail standard cell upsizing and buffer insertion whereas FPGA subcircuits are fixed. For this reason, a two-step compilation procedure, described below, is used to reach a realistic point in the large tradeoff space between area and delay.

We perform compilation using a top-down flow, with "Ultra-effort" optimizations for both area and delay. This turns on all optimization options in the synthesis algorithm and accurately predicts post-layout critical path delay and area using topographical technology [23]. All registers are replaced with their scan-enabled equivalent to allow the necessary post-manufacturing testing for ASICs. In modeling the wire delay, a conservative wire model from TSMC is used. Capacitance and resistance per unit length are used together with a fanout-dependent length model to estimate the wire delay. Additionally, these parameters are automatically adjusted based on the area of the design hierarchy spanned by each net.

In step one, we perform an ultra-effort compilation with an impossible 0 ns timing constraint and extract the negative slack of the critical path from the timing report. Area numbers are bloated when trying to satisfy the impossible timing constraints and are discarded from this compilation. The negative slack from step one is used as the target clock period in step two of the ASIC compilation. This provides a reasonable target for the CAD tools and results in realistic cell upsizing, logic duplication and buffer insertion, and hence realistic area numbers. With the clock period adjusted, the design is recompiled and the implementation area and delay are extracted from the synthesis reports. Note that any positive or negative slack in this step is also added to the critical path delay measurement.

Generally ASICs are not routable if they are 100% filled with cells. To account for whitespace, buffers inserted during placement and routing, and wiring, we assume a 60% rule-of-thumb fill factor and so inflate the area results by 66.7%. Fill factors as low as 10% and as high as 90% have been used in

the literature [3, 19] but we chose 60% to model the typical case after conversations with ASIC design engineers.

C. Methodology Verification

To verify the methodology, we compare the results obtained with our methodology to those of Kuon and Rose on their largest benchmark, raytracer [3]. As shown in Table II, the area and delay ratios are quite close; we expect some difference as our results are from a 65 nm process while theirs are from 90 nm.

TABLE II: Raytracer area and delay ratios.

	Kuon and Rose [3]	This Work
FPGA Device	Stratix II	Stratix III
ASIC Technology	90 nm	65 nm
Area Ratio	26.0	25.6
Delay Ratio	3.5	4.1

IV. Results

Figures 6 and 7 show the FPGA/ASIC area and delay ratios for the router components as they vary with the four main router parameters: flit width, number of ports, number of VCs and input buffer depth. These results are summarized in Table IV and V in which the minimum, maximum and geometric mean is given for each component. Additionally we give the results for a complete VC router built out of those components. We choose a realistic range of router parameters, based on a study of the literature, such that the geometric average of the area or delay ratio is indicative of the FPGA-to-ASIC gap for an NoC that is likely to be constructed.

A single parameter is varied in each experiment to study the effect of this parameter in isolation. The rest of the parameters are set to the default values for a "baseline" router shown in Table III. One exception is that the buffer depth is varied proportionally when the number of virtual channels is swept.

TABLE III: Baseline router parameters.

Width	Num. of Ports	Num. of VCs	Buffer Depth
32	5	2	10 (5/VC)

In our comparison, we investigate the NoC's logic, that is, the network router and we omit the FPGA:ASIC comparison of NoC links. Nevertheless, in Section V we show how to integrate the hard router logic with the FPGA's soft interconnect including the analysis of these links.

A. Input Module

The input module consists of a memory buffer and mixed logic for routing and control. To synthesize an efficient FPGA implementation, the memory buffer is modified to target the three variants of RAM on FPGAs: registers, LUTRAM[1] and block RAM (BRAM). LUTRAM uses FPGA LUTs as small memory buffers and BRAMs are dedicated hard memory blocks that support tens of kilobits. Both LUTRAM and BRAM can implement dual-ported memories (1r1w) and a second read port (2r1w) is added by replicating the RAM module. Registers are more flexible and can construct multiple read ports by replicating the read port itself and not the storage registers. On ASICs, the memory buffer is always implemented using a 2D flip-flop array which is the norm for building small memories. Fig. 8 shows the FPGA area of various buffers when implemented using the three alternatives mentioned. The minimum-area implementation is selected for the comparison against ASICs. In all the data points when varying the buffer depth (equal to #VCs$\times$buffers/VC), the BRAM-based implementation has the lowest area. In fact, the area remains constant since the 9-kbit BRAM can handle up to 256 memory words. LUTRAM is slightly less efficient than BRAM with shallow buffers, but the area increases rapidly when the LUTRAM uses all of its storage bits at steps of 32 words deep. At these points, another LUTRAM module is combined using multiplexers to extend buffer depth. BRAMs and LUTRAMs have width limitations but can be grouped together to implement wider memories. This explains the linear area increase with width shown in Fig. 8.

When the memory buffers are built out of registers, there are no bits wasted. LUTRAM can only be instantiated in quantized steps of the LUTRAM size (640 bits); hence some bits can go unused. This is more true for BRAM which can only be instantiated in steps of 9 kbits for the considered architecture. Nevertheless, the bit density for a register-based memory buffer is 0.77 $kbit/mm^2$ compared to 23 $kbit/mm^2$ for LUTRAM and 142 $kbit/mm^2$ for the 9-kbit BRAM [4]. This means that a 9-kbit BRAM with only 16% of its bits used is just as area-efficient as a fully utilized LUTRAM on a Stratix III FPGA. Although prior work has gravitated towards the use of LUTRAM [8], when looking at it from a silicon perspective, the high density of BRAM makes it more area-efficient for most width$\times$depth combinations. However, in architectures with deeper BRAM, such as Virtex 7, LUTRAM would be the more efficient alternative for shallow buffers.

As Tables IV and V show, the input module has the lowest area and delay gaps of the presented components. The area gap varies from 8-36$\times$ with a geometric mean of 17$\times$. The lower gap occurs when a deep buffer is used and the FPGA BRAM is well-utilized. Width is found to have only a small effect on the input module area and delay ratios, but varying the number of VCs presents a more interesting result. The input module consists of both control logic, which is inefficient on FPGAs, and memory buffers implemented as compact hard blocks. As we vary the number of VCs in Fig. 6 the FPGA implementation becomes twice as efficient between 1 and 6 VCs because we are able to pack more buffer space into the same BRAM module. However, as we increase the number of VCs further, the efficiency drops because the control logic for a large number of VCs becomes the dominant area component. The FPGA-to-ASIC delay ratio is 2.9$\times$ and is always limited by the logic component of the input module and not the fast memory modules.

[1] Stratix IV was used for LUTRAM experiments because Stratix III suffers from a bug that halves LUTRAM capacity.

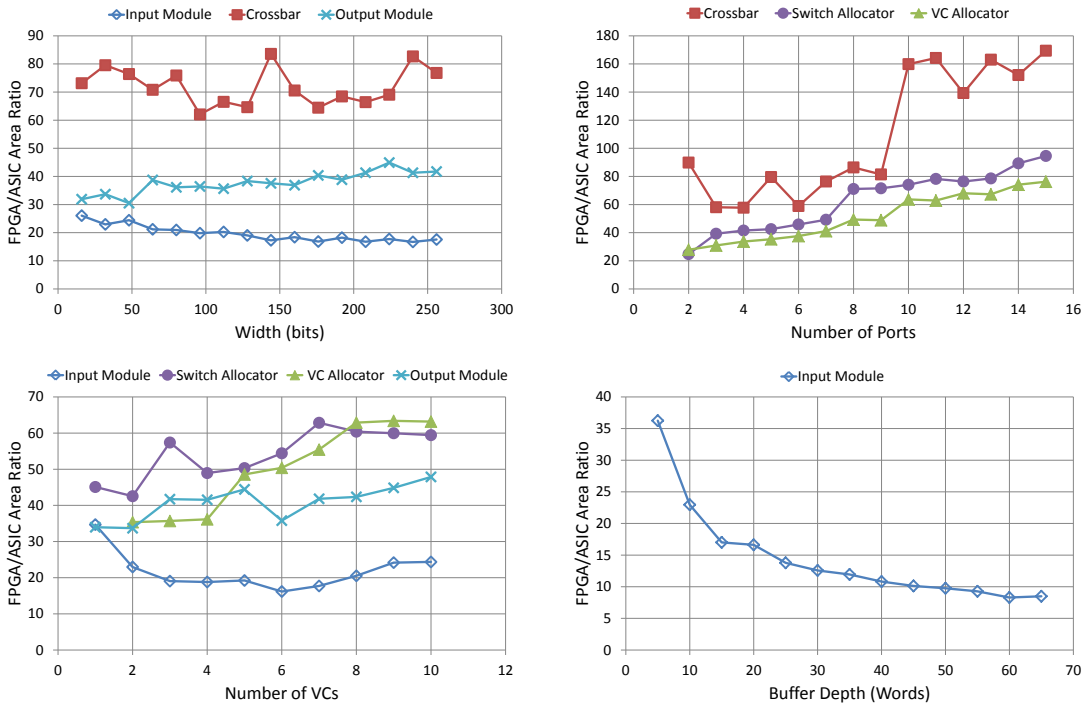

Fig. 6: FPGA/ASIC area ratios.

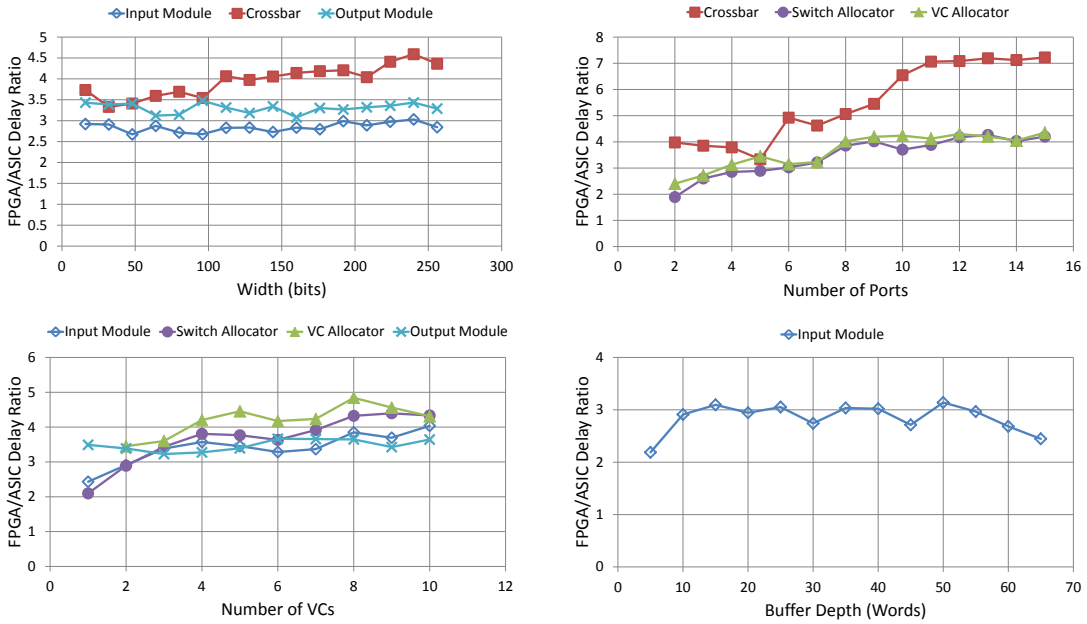

Fig. 7: FPGA/ASIC critical path delay ratios.

TABLE IV: Summary of FPGA/ASIC area ratios.

Module	Min.	Max.	Geometric Mean
Input Module	8	36	17
Crossbar	57	169	85
VC Allocator	27	76	48
Switch Allocator	24	94	56
Output Module	30	47	39
Router	13	64	30

TABLE V: Summary of FPGA/ASIC delay ratios.

Module	Min.	Max.	Geometric Mean
Input Module	2.2	4.0	2.9
Crossbar	3.3	6.9	4.4
VC Allocator	2.0	4.8	3.9
Switch Allocator	1.9	4.2	3.3
Output Module	3.1	3.7	3.4
Router	2.4	5.5	3.6

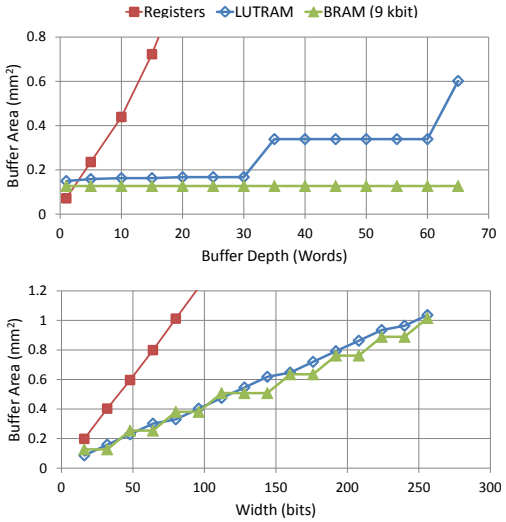

Fig. 8: Variation of physical FPGA area of memory buffers using three implementation alternatives.

B. Crossbar

Crossbars show the largest area gap; a minimum of 57×, a maximum of 169× and a geometric average of 85×. It is worth noting that there is a 2× FPGA efficiency loss for crossbars with 10 or more ports. This is due to two causes. First, the required FPGA LUTs per multiplexer port increases faster than the ASIC gates per port. Second, there is an increased demand for interconnect ports at the logic module inputs causing LUTs in that logic module to be unusable; the ratio of LUTs that are unused for this reason grows from 1% to 10% between 9 and 10 multiplexer ports. When the width is varied however we see very little variation between a 16-bit and a 256-bit wide crossbar.

The crossbar delay gap grows significantly from 3.3-6.9× with increasing port count. This trend is due to the increase in FPGA area, which causes the multiplexers to be fragmented over multiple logic modules thus extending the critical path. Overall, the average delay gap is 4.4× for the crossbar; the largest out of all the components.

The results show that crossbars are inefficient on FPGAs and their scaling behavior is also much worse than ASICs. This is a prime example of a circuit that would bring area and delay advantages if it were hardened on the FPGA. In this scenario, the crossbar can be overprovisioned with a large number of ports so that it can support different NoC organizations. If a small number of router ports are required but a large number of ports are available, the additional ports can be used towards crossbar speedup which relieves crossbar traffic by allowing multiple VCs from the same input port to traverse the switch simultaneously. This also simplifies switch allocation [13].

C. VC and Switch Allocators

Allocators are built out of arbiters which consist of combinational logic and some registers. Ideally the ratio of LUTs to registers should match the FPGA architecture; for Stratix III a 1:1 ratio would use the resources most efficiently. Deviation from this ratio means that some logic blocks will have either registers or LUTs used but not both. The unused part of the logic block is area overhead when compared to ASICs.

Although there are other sources of FPGA inefficiencies, there is a direct correlation between the LUT-to-register ratio and the FPGA-to-ASIC area gap. For the VC allocator the average LUT-to-register ratio is 8:1 and the area gap is 48×, while the speculative switch allocator has an average LUT-to-register ratio of 20:1 and the area gap is higher; approximately 56×. This difference between the two allocators is due to the selection logic which is used in the speculative switch allocator and absent from the VC allocator.

Allocator delay increases with circuit size for both hard and soft implementations; however, the delay rises more rapidly for the soft version. Consequently, the delay ratio of the allocators is proportional to the circuit size, and grows with increasing port or VC count. We suspect this is because the fixed FPGA fabric restricts the placement and routing optimizations that can be performed on large circuits while ASIC flows have more options, such as upsizing cells or wires on critical paths. Overall, the delay gap was around 3.6× for the allocators.

D. Output Module

The output module is the smallest router component and is dominated by the output registers. Indeed, the LUT-to-register ratio is 0.6:1 contributing to its smaller area gap of 39× when compared to the allocators. The average delay ratio of 3.4× is also relatively low because the simple circuitry does not stress the FPGA interconnect.

E. Router Area Composition on FPGA and ASIC

Figures 9 and 10 show the router area composition on FPGAs and ASICs respectively. Moreover, the total router area of select data points is given on the top axes.

The main discrepancy between the FPGA and ASIC router composition is the proportion of the input modules and the crossbar. The input modules are the largest components for most router variants on both the soft and hard implementations. It follows from the area ratios that the input modules are relatively larger on ASICs than on FPGAs; in fact, they occupy 36-83% of the ASIC router area compared to 14-60% on the FPGA. The crossbar is the smallest component of an ASIC VC router. On FPGAs, however, it becomes a critical component with a wide datapath or a large number of ports where it occupies up to 26% of the area.

With an increasing number of VCs, the VC allocator area becomes the dominant one on both FPGAs and ASICs. Increasing the number of ports also increases the VC allocator area but to a lesser extent. This is due to the second stage of VC allocation which occupies most of the area and is constructed out of $P \times V$:1 arbiters. They require an additional P inputs per arbiter when the number of VCs is increased whereas only V additional inputs are required when the number of ports is raised. Since P is larger than V for the baseline router, the VC allocator's area grows more slowly with the number of ports than it does with the number of VCs. The speculative switch allocator also grows with increasing

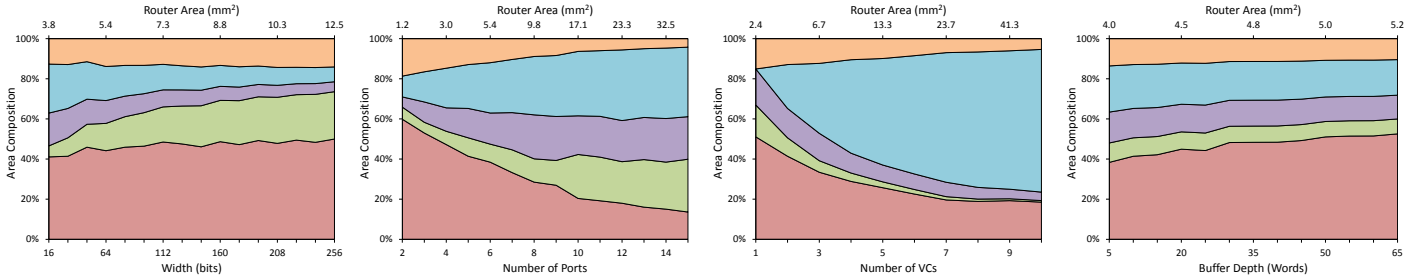

Fig. 9: FPGA router area composition by component and total router area. Starting from the bottom(red): Input module, crossbar, switch allocator, VC allocator and output module.

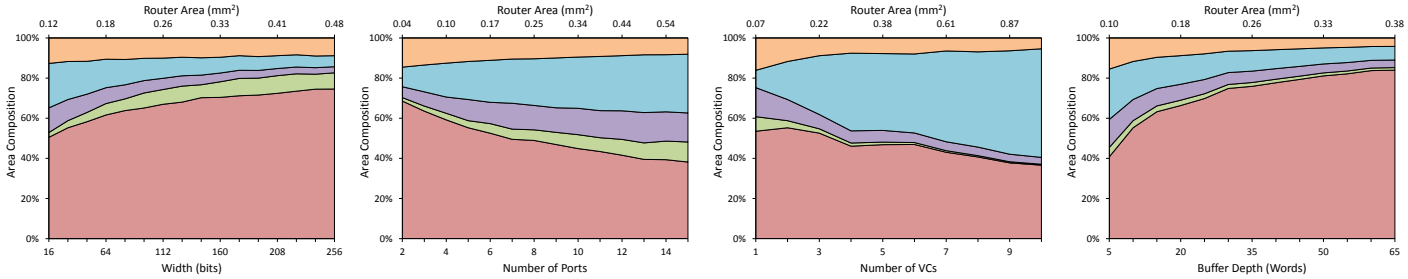

Fig. 10: ASIC router area composition by component and total router area. Starting from the bottom(red): Input module, crossbar, switch allocator, VC allocator and output module.

port and VC counts but is more affected by the number of ports. With 15 router ports, the switch allocator makes up 22% of the FPGA router area.

V. Architectural Implications

The strengths of a soft NoC lie in its reconfigurability. Based on the results in Section IV, we summarize the design recommendations for a soft NoC:

1) BRAM was most efficient for memory buffer implementation even for shallow buffers, so buffer depth is free until the BRAM is full.
2) To increase bandwidth, it is efficient to increase the flit width rather than the number of ports or VCs. As the width is increased the area ratio falls from 32× to 26×.
3) The number of ports and number of VCs scale poorly on FPGAs because of the quadratic increase of allocator and crossbar area.

We now look at the gains of hardening NoC components. One viable option is to harden the crossbar and allocators and leave the input and output modules soft. This solution moves the critical path from the switch allocator to the input module allowing the router to run at 386 MHz compared to 153 MHz for a fully soft implementation. Such a heterogeneous router occupies $2.335\ mm^2$ for the baseline parameters. The 1.8× area improvement over a soft implementation is, however, unconvincing. Furthermore we have not yet accounted for the area of the interconnect ports; that is, the switch and connection blocks that would route wires into, out of and around the hard component.

Alternatively, if the baseline router were to be completely hardened on the FPGA it could run at 807 MHz which would saturate the FPGA's clocking network; limited to 730 MHz for the fastest speed grade on Stratix III FPGAs [24]. At this frequency, 3 short (length=4) vertical wires or 7 short (length=4) horizontal wires could be used to connect the routers using general interconnect without degrading the frequency. This corresponds to approximately 1/9 of the FPGA's vertical dimension and 1/6 of the horizontal dimension suggesting that such spacing of hard routers would allow them to operate at the FPGA's peak frequency even when using the plentiful but slow short wires for connection. The hard router core occupies $0.137\ mm^2$. In the following, we derive a first-order approximation of the area required for the hard router's interconnect ports to find the total area for a hard router.

50% of the LAB area ($0.011\ mm^2$) consists of programmable interconnect and supplies 52 input ports and 40 output ports [25]. The baseline router has 32 data plus 2 backpressure inputs and output per port, for a total of $34\times5=170$ inputs and outputs. This is conservatively 4.25 times that of a LAB making the interconnect ports area of the hard router equal to $0.047\ mm^2$ with the same connectivity flexibility as a LAB. The total hard router area is now $0.137\ mm^2+0.047\ mm^2=0.184\ mm^2$, which is equivalent to 8.3 LABs. We round it up to 9 LABs, a 21× area improvement over the soft router implementation.

Fig. 11, drawn to scale, shows the floorplan of the hard router embedded in the FPGA fabric; on Stratix III FPGAs, the ratio of height to width for a LAB is 2:1 [26]. The hard router is laid out such that it occupies the width and height of 3 LABs and connects its inputs to programmable wiring using two levels of multiplexers and its outputs using one level, similarly to LABs. This ensures equivalent interconnection flexibility thereby keeping the FPGA routable.

Finally we must ensure a sufficient number of interconnect wires intersect the hard router to connect to all of the intercon-

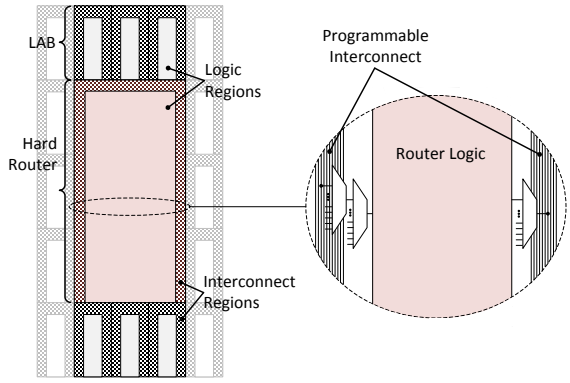

Fig. 11: Floor plan of a hard router embedded in the FPGA fabric.

nect ports. The router occupies an area equivalent to 9 LABs, and its perimeter intersects 8 interconnect switch boxes. This means it could make 8× the number of connections a LAB would make, of which only 4.25× are needed, indicating that the router has lower local interconnect stress than a LAB. We repeat this analysis for all of the data points in this study and find that the geometric mean area gap drops from 30× to 22× after accounting for interconnect ports.

To see the cost of a complete system, consider hardening a 64-node NoC on the FPGA. This will occupy the area equivalent to 9 LABs × 64 nodes = 576 LABs. We envision such a network to be implemented on large, high performance FPGAs such as Stratix V or Virtex 7. A 64-node hard NoC composed of state-of-the-art VC routers will occupy 1.6% of the LAB area (~0.6% of total chip area) of the largest Stratix V FPGA, compared to 33% of the LABs (~12% of total area) for a soft implementation. Despite its low area, this NoC provides significant communication bandwidth. Each link has a peak bandwidth of 2.9 GB/s in each direction, yielding a total bisection bandwidth up to 46.7 GB/s across any vertical or horizontal line that partitions the chip.

VI. Conclusion

Augmenting future FPGAs with NoCs would facilitate interfacing to high-speed I/Os, simplify compilation and partial reconfiguration via modularity, and ease the timing closure bottleneck. To inform the architecture of such an NoC we have investigated the area and delay gap per NoC component for hard versus soft implementation.

The area ratio of 13-64× is largest when the area is dominated by allocation logic and when the buffers are not well-utilized. The delay gap is smaller and ranges between 2.4-5.5×. Soft NoCs were found to be more efficient when implemented using BRAMs for memory buffers, and scaling the width is a more efficient way of increasing bandwidth than increasing the number of ports or VCs. Finally, we showed how a hard router could be integrated with the programmable interconnect of the FPGA fabric. Including programmable interconnect, it is on average 22× more area efficient than a soft implementation. A 64-node hard NoC has lower area overhead than a 3-node soft NoC.

To fully reap the benefits of a hard NoC, future work includes analysis of dedicated inter-router interconnect. Pertaining to that, clock rate conversion circuitry may be required to take advantage of the speed gains of hardening the NoC and to allow flexibility in selecting the clock frequency of each network node. We will also investigate how NoC design choices will be influenced by important traffic hot spots such as DDR interfaces.

Acknowledgment

This work is funded by NSERC and Altera Corporation. The authors would like to thank Daniel Becker of Stanford for the open-source router implementation and CMC for providing the ASIC CAD tools.

References

[1] R. Ho, K. W. Mai, and M. A. Horowitz, "The Future of Wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, 2001.
[2] W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *DAC*, 2001, pp. 684–689.
[3] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *TCAD*, vol. 26, no. 2, pp. 203–215, 2007.
[4] H. Wong, V. Betz, and J. Rose, "Comparing FPGA vs. Custom CMOS and the Impact on Processor Microarchitecture," in *FPGA*, 2011, pp. 5–14.
[5] J. Lee and L. Shannon, "Predicting the Performance of Application-Specific NoCs Implemented on FPGAs," in *FPGA*, 2010, pp. 23–32.
[6] B. Sethuraman *et al.*, "LiPaR: A Light-Weight Parallel Router for FPGA-based Networks-on-Chip," in *GLSVLSI*, 2005, pp. 452–457.
[7] G. Schelle and D. Grunwald, "Exploring FPGA network on chip implementations across various application and network loads," in *FPL*, 2008, pp. 41–46.
[8] M. K. Papamichael and J. C. Hoe, "CONNECT: Re-Examining Conventional Wisdom for Designing NoCs in the Context of FPGAs," in *FPGA*, 2012, pp. 37–46.
[9] R. Francis and S. Moore, "Exploring Hard and Soft Networks-on-Chip for FPGAs," in *FPT*, 2008, pp. 261–264.
[10] K. Goossens *et al.*, "Hardwired Networks on Chip in FPGAs to Unify Functional and Configuration Interconnects," in *NOCS*, 2008, pp. 45–54.
[11] E. S. Chung, J. C. Hoe, and K. Mai, "CoRAM: An In-Fabric Memory Architecture for FPGA-based Computing," in *FPGA*, 2011, pp. 97–106.
[12] Concurrent VLSI Architecture Group, Stanford University. Open Source Network-on-Chip Router RTL. [Online]. Available: https://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router
[13] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Boston, MA: Morgan Kaufmann Publishers, 2004.
[14] R. Mullins, A. West, and S. Moore, "Low-Latency Virtual-Channel Routers for On-Chip Networks," in *ISCA*, 2004, pp. 188–198.
[15] D. U. Becker and W. J. Dally, "Allocator Implementations for Network-on-Chip Routers," in *SC*, 2009, pp. 1–12.
[16] Y. Tamir and G. L. Frazier, "High-Performance Multi-Queue Buffers for VLSI Communication Switches," in *ISCA*, 1988, pp. 343–354.
[17] J. Balfour and W. J. Dally, "Design Tradeoffs for Tiled CMP On-Chip Networks," in *ICS*, 2006, pp. 187–198.
[18] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *STOC*, 1981, pp. 263–277.
[19] G. Passas, M. Katevenis, and D. Pnevmatikatos, "Crossbar NoCs Are Scalable Beyond 100 Nodes," *TCAD*, vol. 31, no. 4, pp. 573–585, 2012.
[20] L.-S. Peh and W. J. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers," in *HPCA*, 2001, pp. 255–267.
[21] Altera Corp., "Stratix III FPGA: Lowest Power, Highest Performance 65-nm FPGA," Press Release, 2007.
[22] R. Scoville. (2010) TimeQuest User Guide. [Online]. Available: http://www.alterawiki.com/uploads/3/3f/TimeQuest_User_Guide.pdf
[23] Synopsys Inc., *Design Compiler Optimization Reference Manual*, 2010.
[24] Altera Corp., "Stratix III Device Handbook," 2011. [Online]. Available: http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf
[25] D. Lewis, Altera Corp., Toronto, ON, private communication, 2012.
[26] D. Lewis *et al.*, "Architectural Enhancements in Stratix-III and Stratix-IV," in *FPGA*, 2009, pp. 33–41.