The Case for Embedded Networks on Chip on Field-Programmable Gate Arrays

The authors propose augmenting the FPGA architecture with an embedded network on chip to implement the system-level communication infrastructure and mitigate the hardware design challenges faced by current bus-based interconnects. With a flexible interface between the NoC and the FPGA fabric, an embedded NoC maintains configurability and simplifies the distribution of I/O data throughout the chip, and is always more energy-efficient compared to custom buses configured into the fabric.

Mohamed S. Abdelfattah
Vaughn Betz
University of Toronto

Computing applications are increasing in size and complexity, while multicore CPUs and GPUs are constrained by a power and memory wall. Field-programmable gate arrays (FPGAs) offer an alternative computing platform that demonstrates considerable performance-per-watt improvements for some applications.[1-4] These improvements are enabled by the customization of hardware to the exact application computing and memory needs. Such customization can make very efficient use of the silicon but comes at the cost of high design time and effort.

Although CPUs and GPUs have a predefined datapath and memory hierarchy, each FPGA application is designed out of an array of logic and other blocks to form a customized datapath and memory hierarchy. This flexibility is both the blessing and the curse of FPGAs; the high degree of customization allows very fine-grained optimization and a correspondingly high level of parallelism, but it comes at the cost of increased design effort and time. An FPGA designer must design the custom computing kernels and then integrate the system using a form of system-level interconnect (often pipelined buses) that connects these kernels to each other and, more importantly, to external interfaces such as DDRx and PCI Express (PCIe).

On CPUs and GPUs, connecting to external interfaces is as simple as specifying program instructions to access the relevant device registers; the hardware that moves the data already exists. On FPGAs, however, usually no program instructions exist, because the custom datapath implements only the exact application. The designer must architect a custom interconnect that connects the assortment of custom kernels in an application and integrates them with interfaces to external memory and other I/Os. These kernels can have different data widths and

Published by the IEEE Computer Society

operating frequencies and can be located anywhere on the FPGA device. Furthermore, this system interconnect must adhere to the stringent timing constraints of external devices, such as DDRx or PCIe, which is no trivial task.

For example, the task of creating the interconnect to access DDR3 memory is very common in all reconfigurable computing applications but is quite challenging. On the DDR3 side are 64 bits at 1,600 MHz, and, on the FPGA side, a custom bus configured from the FPGA fabric can typically run at 200 MHz. The only way to overcome this speed mismatch is to go parallel; 64 bits are converted into a slow 512-bit-wide bus on the FPGA. This huge bus uses many FPGA resources and hence much power, and its design at 200 MHz is not simple. System integration tools such as Xilinx's XPS and Altera's Qsys have been developed to automate and ease the creation of a system interconnect out of the FPGA fabric. Even with these tools, however, connecting to external memory often requires many compilation and design tweaks and, therefore, high design time and effort. We propose a change to the hardware itself, to embed networks on chip[5] (NoCs) on the FPGA device to abstract the movement of latency-tolerant data, such as DDRx transfers, thereby simplifying the integration of systems on FPGAs. As we will show, this change also improves area and power efficiency.

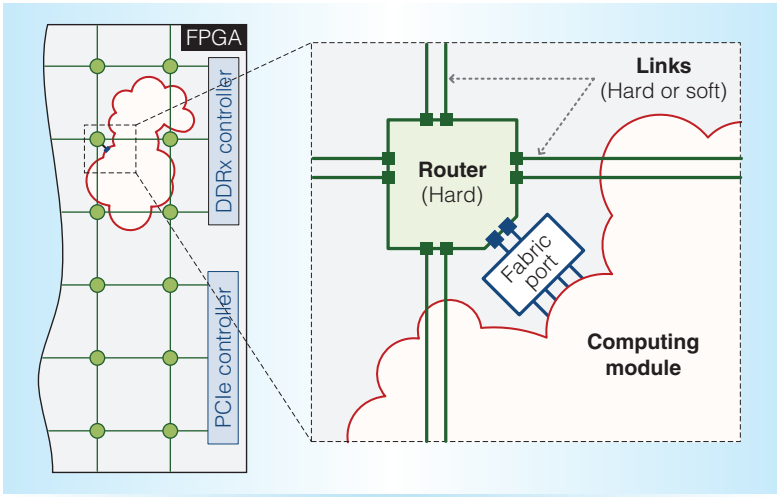

Figure 1. An embedded mesh network on chip (NoC) implemented on a field-programmable gate array (FPGA). The NoC consists of routers, links, and fabric ports.

Network architecture

NoCs are forecast to become the main interconnection substrate for large systems on chip (SoCs); they have been described by Benini and De Micheli as "…the only path to mastering the complexity of SoC designs in the years to come."[6] As FPGAs become ever larger reconfigurable computing platforms, we believe that they too will benefit from NoCs in integrating their computing kernels, memory, and I/O. In this section, we present our NoC architecture alternatives.

Currently, hierarchical buses are configured out of the FPGA fabric to interconnect large systems. This approach lets us adapt the bus structure to each specific application but requires the buses to be built from very low-level components (logic blocks and programmable interconnect). NoCs instead communicate via packets transmitted through routers that provide both buffering and switching. We embed application-agnostic NoC components in the FPGA silicon to achieve high efficiency, while the dynamic switching of the NoC still lets us move data between modules configured anywhere in the FPGA. We must also be able to connect any module on the FPGA to the NoC; we achieve this high level of configurability with a new "fabric port" between the embedded NoC and the FPGA fabric.

Figure 1 details the envisioned NoC, which consists of routers, links, and fabric ports. Crucially, NoC routers are connected to external interfaces, such as DDRx, PCIe, and Gigabit Ethernet, allowing these interfaces to access the entire FPGA fabric through the embedded NoC with timing closure guarantees. We use high-performance, full-featured, packet-switched routers,[7] because we believe that the capabilities offered by these routers—such as virtual channels—are useful for large FPGA systems. A clear use-case for virtual channels is to assign a higher priority to latency-sensitive packets, such as DDR3 data destined for a control processor, and lower priority to more latency-tolerant transfers.

The fabric port allows access to the NoC by the diverse modules that can be

Table 1. Summary of mixed and hard FPGA NoCs at 65 nm.

Feature		Mixed NoCs	Hard NoCs
Description		Hard routers, soft links	Hard routers, hard links
Special feature		Configurable topology	Low-voltage (low-V) mode
Comparison to soft NoCs	Area	20× smaller	23× smaller
	Speed	5× faster	6× faster
	Power	9× less power	11× less power (15× low-V)
Frequency		730 MHz	910 MHz
Critical path		Soft interconnect	Switch allocator in router

configured into an FPGA by adapting the width, speed, and operating voltage of data between a module on the FPGA fabric and the embedded NoC. Unlike with NoCs for multiprocessors, we cannot predict the operating frequency (between 100 to 400 MHz) or data width of our design modules. Each module in a design could be different. We fix the embedded NoC frequency to its maximum speed (up to 1 GHz) and use a fabric port to match the fabric bandwidth to the NoC bandwidth at each NoC router, thus allowing each design module to run at an independent speed and data width. This fabric port is hardwired to NoC routers on one side and connects—through programmable multiplexers—to the FPGA fabric interconnect on the other side, which allows it to reach any module configured on the FPGA. Inside the fabric port, we use time-domain multiplexing logic to adapt the data width, and an asynchronous first-in, first-out (FIFO) buffer for clock-domain crossing. We also include voltage-domain-crossing circuitry to be able to run an embedded NoC at a lower voltage than the rest of the FPGA, effectively trading excess NoC speed for energy efficiency. We study the implementation of NoC components[8,9] and focus on the question of which components should be implemented as "hard" or "soft."

- *Soft*: Constructs a digital circuit by programming the reconfigurable FPGA fabric. This entails programming components, such as logic blocks, memories, and the soft interconnect (short wires and programmable multiplexers) between them.
- *Hard*: Fabricates a specialized (embedded) module on the FPGA silicon using ASIC technology. Although this approach rids part of the FPGA of its reconfigurability and requires an architectural change, a hard block is much more efficient than its soft counterpart. The interconnect also can be hardened. "Hard links" are dedicated metal connections that contain no multiplexers; thus, they are faster than soft links but are nonconfigurable.

Routers are 30× smaller, 6× faster, and use 14× less power when implemented hard rather than soft,[8,9] and the case for hardening them is compelling. For links, the answer is not as decisive; we therefore propose two new NoC architectures and summarize the strengths of each in Table 1.

Mixed NoC: Hard routers and soft links

In this NoC architecture, we embed hard routers on the FPGA and connect them via the soft FPGA interconnect. A key question is: How does a new hard block connect to the programmable FPGA interconnect? We use the same structure (multiplexer size) as a logic block (LB), thereby guaranteeing the same interconnection flexibility. This setup ensures that a hard router will not cause any interconnection hot spots or present a difficult target for CAD tools. Figure 2 shows a detailed illustration of such an embedded router; we ensure that the router area is a multiple of logic blocks to simplify the layout. After accounting for the programmable multiplexers to the soft interconnect, mixed NoCs are on average 20× smaller, 5× faster, and 9× more power-efficient than fully soft NoCs.[8]

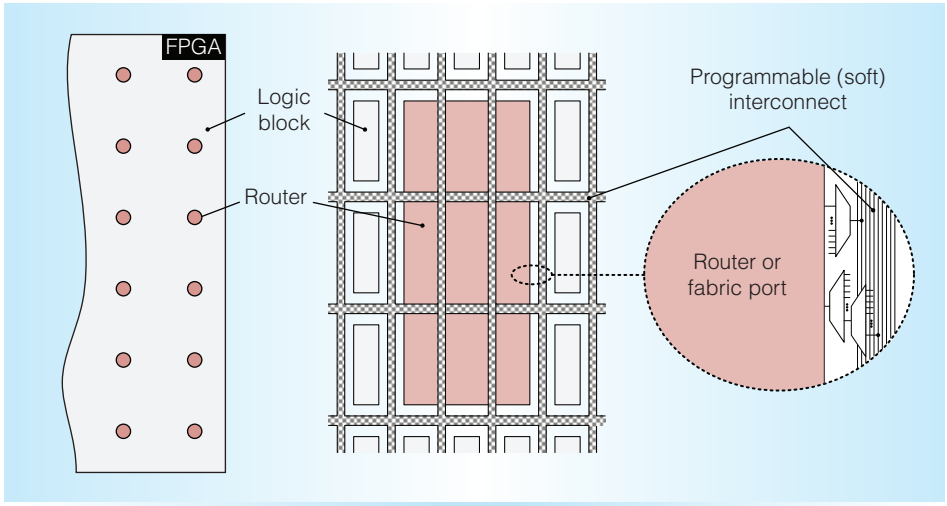

Figure 2. Floorplan of a hard router with soft links embedded in the FPGA fabric (drawn to scale). A typical hard router (32-bit width, 5 ports, 2 virtual channels) occupies the area equivalent to approximately 9 logic blocks, and it connects to the FPGA's soft interconnect through programmable multiplexers.

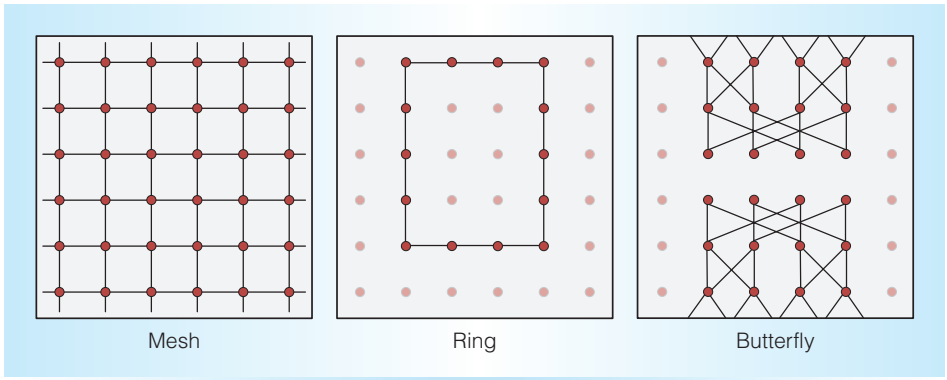

Figure 3. Examples of different topologies that can be implemented using the soft links in a mixed NoC. Any topology can be implemented using the FPGA's programmable interconnect, limited only by the number of ports available on the hard routers.

Although this NoC achieves a major increase in efficiency and performance, it remains highly configurable, which is a hallmark of FPGAs, by virtue of the soft links. Figure 3 shows how different network topologies can be configured from the soft links, including topologies that use only a subset of the available routers or two separate NoCs. Furthermore, mixed NoCs do not stress the soft interconnect much; the maximum utilization of FPGA interconnect by soft NoC links in a localized region is approximately 11 percent. However, the speed of this type of NoC is limited by the soft interconnect, which is often slower than the fast hard routers.

Hard NoC: Hard routers and hard links

This NoC architecture involves hardening both the routers and the links as shown in Figure 4. Routers are connected to other routers using dedicated hard links; however, routers still interface to the FPGA fabric through programmable multiplexers connected to the soft interconnect. The NoC topology is no longer configurable, but the hard links save area (because they require no

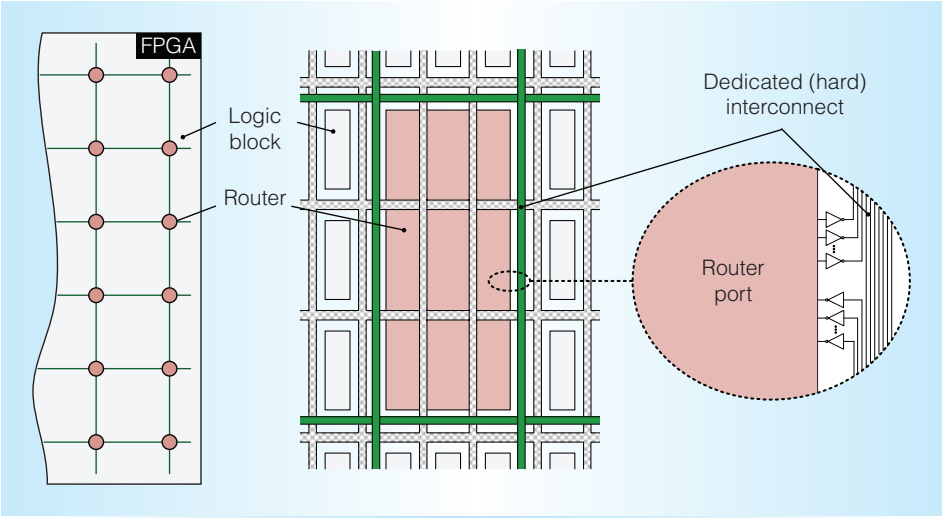

Figure 4. Floorplan of a hard router with hard links embedded in the FPGA fabric (drawn to scale). A typical hard router (32-bit width, 5 ports, 2 virtual channels) occupies the area equivalent to 9 logic blocks. Hard links, or dedicated metal wires with repeaters, connect hard routers in a fixed topology.

multiplexers) and can run at higher speeds compared with soft links, allowing the NoC to achieve the router's maximum frequency. After accounting for the wire drivers and the programmable multiplexers needed at the router-to-FPGA-fabric ports, this NoC is on average 23× smaller, 6× faster, and consumes 11× less power than a soft NoC.

With hard routers and hard links, an NoC is almost completely separate from the FPGA fabric, connecting only through router-to-fabric ports. This approach makes it easy to use a separate, lower-voltage power grid for the NoC, letting us trade excess NoC speed for power efficiency. We therefore propose a low-power version of the hard NoC that operates at a lower voltage. For example, 65-nm Stratix III FPGAs use a supply voltage of 1.1 V, whereas our low-power NoC saves 33 percent dynamic power by operating at 0.9 V in the same process.[9]

Embedded NoCs versus current interconnect solutions

FPGA designers currently use soft buses to create the communication infrastructure of large systems (see the "Communication Infrastructure for FPGA Computing" sidebar for more information). The resulting hierarchical bus consists primarily of wide pipelined multiplexers configured out of FPGA logic blocks and a soft interconnect (as shown in Figure 5). We propose changing the FPGA architecture to include an embedded (mixed or hard) NoC instead. Consequently, we must compare the efficiency of the soft bus-based interconnect to the proposed embedded NoC. Previous NoC versus bus comparisons have shown that NoCs become the more efficient interconnect only for large systems;[10] however, because we compare a hard NoC to a soft bus, we find the NoC exceeds bus efficiency even for small systems.

The communication infrastructure is used to connect an application to DDRx memory, which is necessary for all sizeable FPGA systems. We compare the efficiency of the application-tailored soft buses generated by Qsys (Altera's system integration tool) to our embedded NoC, when used to connect to external memory. By varying the number of modules accessing memory, we emulate different applications and study interconnects of different sizes and interconnection capabilities. We also vary the physical distance between the traffic generators and the memory they are trying to access, from "close" to "far," meaning on opposite ends of the chip.

Communication Infrastructure for FPGA Computing

High-level synthesis lets designers use programming languages similar to Java or C to code for FPGAs instead of hardware description languages like Verilog, which, in turn, facilitates reconfigurable computing. Two examples are Maxeler's MaxCompiler and Altera's OpenCL Compiler, which have been used to accelerate applications related to geosciences,[1] financial analysis,[2] video compression,[3] and information filtering[4] while demonstrating order-of-magnitude gains in performance and power efficiency compared to multicore CPUs and GPUs.

Figure A shows the anatomy of a typical computing application implemented on an FPGA, such as a Maxeler or Altera system. It consists of application-tailored computing kernels, on-chip memory, and at least two connections to external devices—a host CPU through PCIe links and off-chip memory through DDRx interfaces. These components form the basic infrastructure necessary for operation, and the customization of the application kernels and the memory hierarchy allows the implementation of different applications. The application designer creates a custom *communication infrastructure* out of the FPGA's low-level programmable logic and interconnect to connect these diverse sets of blocks. This infrastructure must at least be capable of buffering data, adapting width and frequency, and distributing data across the chip. Its design is challenging for many reasons.

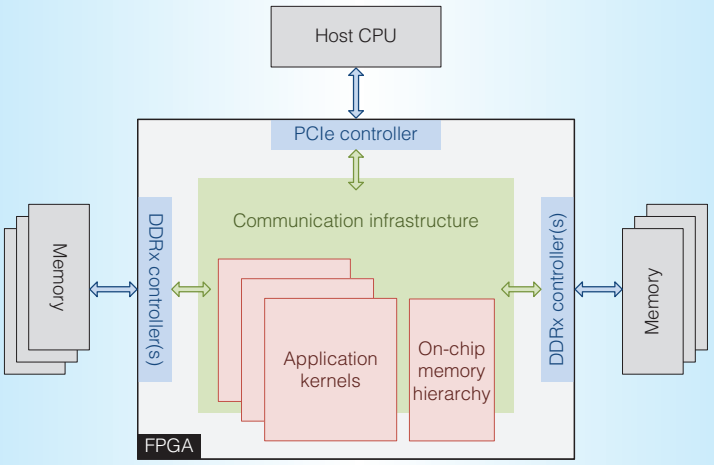

Figure A. Architecture of a typical reconfigurable computing system. An FPGA is connected through PCIe to a host processor and through DDRx controllers to external memory. "Communication infrastructure" refers to the interconnect that connects the application kernels to external devices and on-chip memory.

First, this interconnect's operating frequency must be high enough to satisfy the stringent operating frequencies of each I/O interface (satisfying a design's timing constraints is called *timing closure*). Second, this interconnect often spans the entire physical dimension of the FPGA to connect I/O controllers and application kernels on opposite ends of the chip, making timing closure more difficult and necessitating the addition of pipeline registers. Third, for a different FPGA device, this communication infrastructure could require redesign because it targets different hardware; thus, the whole application usually is not portable between devices without further hardware modifications. Additionally, to implement all the required functionality, this interconnect becomes power hungry and area intensive, because it is configured out of the FPGA's general-purpose logic blocks and interconnect. The design of this communication infrastructure is a hindrance to reconfigurable computing on FPGAs, because it involves tedious and time-consuming low-level hardware design and is not portable between devices.

References

1. O. Lindtjorn et al., "Beyond Traditional Microprocessors for Geoscience High-Performance Computing Applications," *IEEE Micro,* vol. 31, no. 2, 2011, pp. 41-49.
2. G.C.T. Chow et al., "A Mixed Precision Monte Carlo Methodology for Reconfigurable Accelerator Systems," *Proc. 20th ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays* (FPGA 12), 2012, pp. 57-66.
3. D. Chen and D. Singh, "Fractal Video Compression in OpenCL: An Evaluation of CPUs, GPUs, and FPGAs as Acceleration Platform," *Proc. 18th Asia and South Pacific Design Automation Conf.* (ASP-DAC 13), 2013, pp. 297-304.
4. D. Chen and D. Singh, "Using OpenCL to Evaluate the Efficiency of CPUs, GPUs and FPGAs for Information Filtering," *Proc. 22nd Int'l Conf. Field Programmable Logic and Applications* (FPL 12), 2012, pp. 5-12.

The system-level interconnect may span a large distance for two reasons; either the FPGA is full and its modules are scattered across the chip, or I/Os on opposite ends of the chip are connected to one another. This physical remoteness makes generating the Qsys bus-based interconnect that ties everything together more difficult.

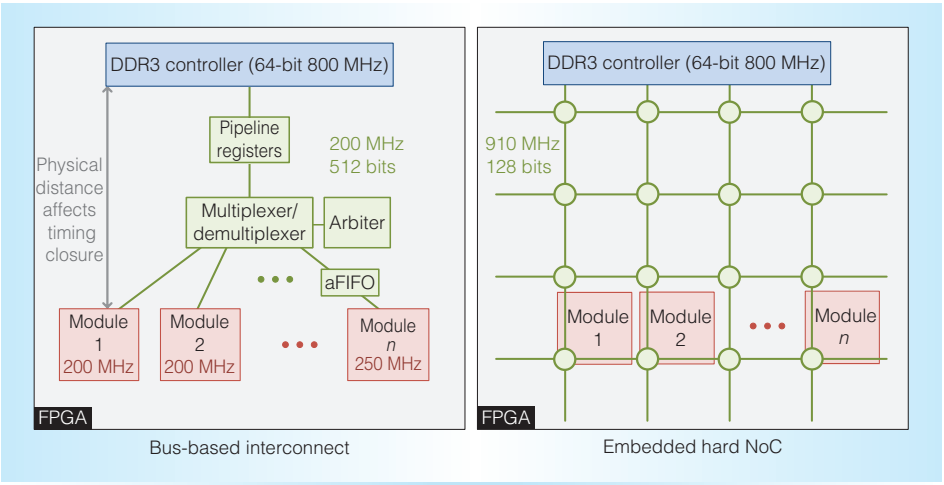

Figure 5. Connecting multiple modules to DDR3 memory using a bus-based interconnect or the proposed embedded NoC. We use random traffic generators for these modules to emulate an application's compute kernels or on-chip memory.

Table 2. Design steps and interconnect overhead of implemented systems.

System size	Physical proximity	Design effort	Frequency	Area	Power
Small (3 modules)	Close	—	× 187 MHz	95 LBs	17 mW
		Max tool effort (plus physical synthesis)	× 194 MHz	95 LBs	17 mW
		Auto interconnect pipelining (3 stages)	✓ 227 MHz	139 LBs	75 mW
	Far	—	× 92 MHz	96 LBs	48 mW
		Max tool effort (plus physical synthesis)	× 96 MHz	96 LBs	51 mW
		Auto interconnect pipelining (4 stages)	✓ 205 MHz	299 LBs	199 mW
Large (9 modules)	Far	—	× 70 MHz	249 LBs	49 mW
		Max tool effort (plus physical synthesis)	× 70 MHz	250 LBs	49 mW
		Auto interconnect pipelining (4 stages)	× 198 MHz	757 LBs	302 mW
		Manual interconnect pipelining (1 stage)	✓ 216 MHz	801 LBs	307 mW

Design effort

To highlight the design effort necessary to implement different bus-based systems, Table 2 lists the steps taken to meet timing constraints using Qsys. A small system, with three modules located physically close to external memory, does not meet timing with default settings. Typically, designers first would switch on all optimization options in the CAD tools, causing a major increase in compile time to improve timing closure. Turning on extra optimization improved design frequency but still fell short of the target by 6 MHz. We then enabled pipelining at various points in the bus-based interconnect between modules and external memory. Only with three pipeline stages did we meet timing, and this result came at a 45 percent area penalty and 340 percent power increase. Placing the modules physically farther away from memory makes the bus physically larger and therefore more difficult to design and less efficient: more than 2× the area and power. Furthermore, larger systems with nine modules are still more difficult and time-consuming to design even with sophisticated tools like Qsys. Even after

enabling the maximum interconnect pipelining, we must identify the critical path and manually insert an additional stage of pipeline registers to reach 200 MHz. This timing closure process is largely ad hoc and relies on trial and error and designer experience. Our proposition eliminates this time-consuming process; an embedded NoC is predesigned to distribute data over the entire FPGA chip while meeting the timing and bandwidth requirements of interfaces such as DDR3.

Area

Figure 6 shows how the area and power of bus-based interconnects and embedded NoCs vary with the number of modules. We compare a Qsys-generated, 512-bit, soft bus to a 16-node, 128-bit, hard NoC embedded onto the FPGA, as shown in Figure 5. Our previous work[8,9] details the methodology for finding the area, speed, and power of hard NoCs; we follow the same methodology and build on it in this work. For example, we compute hard NoC power for our benchmarks by first simulating the benchmarks to find the total data bandwidth flowing through the NoC; we then multiply the total bandwidth by our "Power per BW" metric that we computed for hard NoCs.[9]

The plotted area of the hard NoC is independent of the design size up to 16 nodes, because we must prefabricate the entire NoC onto the FPGA chip and always pay its complete area cost. Nevertheless, the NoC's entire area is less than that of the optimized bus-based interconnect for designs with more than three modules accessing DDR3 memory. Moreover, buses that are generated by Qsys become larger and more difficult to design with more connected modules, consuming up to 6 percent of the logic blocks on large FPGAs to connect to a single external memory. Bus-based and NoC-based communication can be combined into one system, maintaining the high configurability that's crucial to FPGAs. For example, if more than 16 modules access an external memory, two modules may be combined using a small soft bus to share one NoC fabric port. Additionally, portions of a design with low bandwidth can also choose to use a bus for communication.

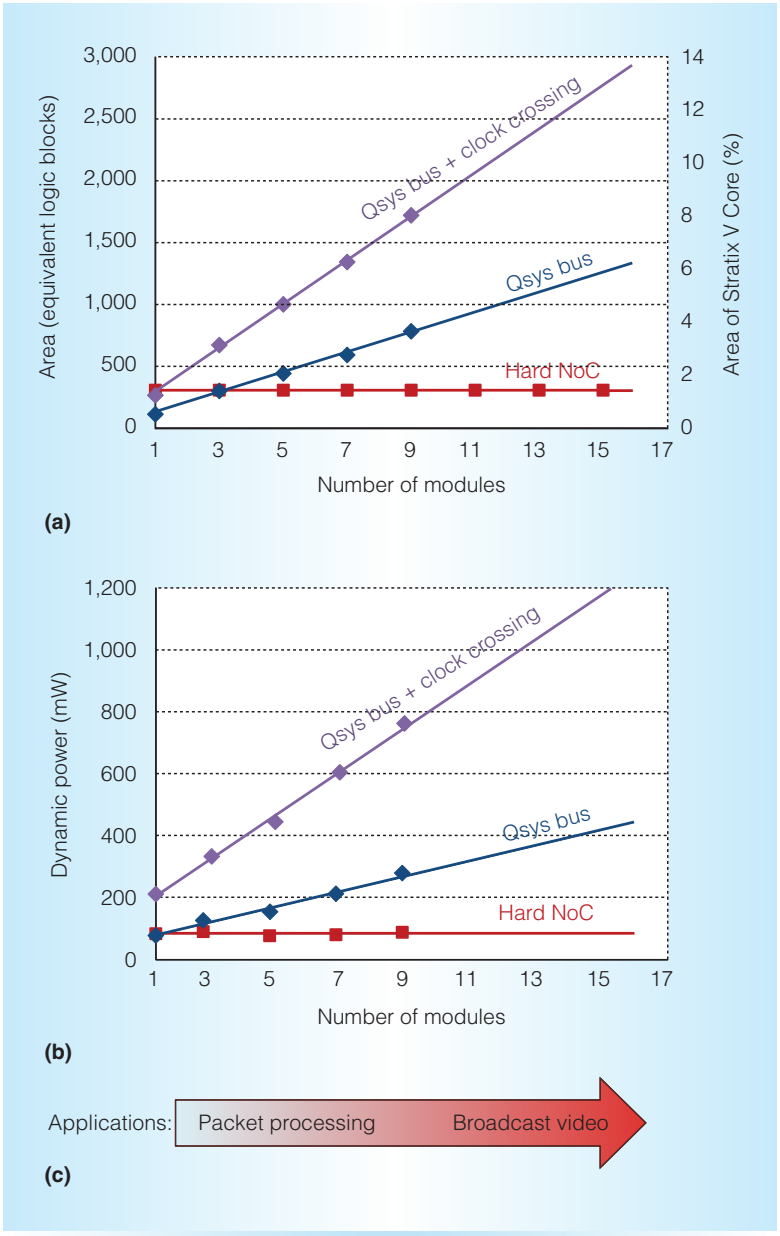

Figure 6. Comparison of area and power of Qsys bus-based interconnect and embedded NoCs with a varying number of modules accessing external memory. Example applications corresponding to the number of modules are annotated.

The embedded NoC includes clock-crossing and data-width adapters at its fabric ports, letting us connect modules running at any independent clock frequency. This approach is common in commercial designs because IP modules could have any frequency within the limits of the FPGA, rather than matching the DDR3 fabric interface

frequency of 200 MHz. Figure 6 plots the area of Qsys buses when the connected modules operate at 250 MHz and thus require asynchronous FIFOs to connect to the bus. These FIFOs are as wide as the bus (512 bits) and hence incur a large area overhead; in fact, the area required by a one-module Qsys system with clock crossing is approximately equal to the complete hard NoC area.

Each link of the hard NoC can carry the entire bandwidth of an 800-MHz, 64-bit DDR3 channel; the NoC is underutilized in our study. Consequently, it can support connectivity to multiple I/O interfaces at once with no additional area; this capability further widens its efficiency advantage over buses for more complex systems. For example, an FPGA in a Maxeler system connects to six DDR3 controllers.

Dynamic power

Figure 6 shows that NoC dynamic power is also somewhat constant compared with the number of modules accessing one DDR3 interface; this is true for two reasons. First, we always move the same bandwidth on the NoC divided equally among all accessing modules, so the interconnect itself always transports the same total bandwidth. Second, we average the power for our systems when their modules are placed physically close and far from the DDR3 interface, meaning that the same bandwidth moves a similar average distance for all systems. We use the same assumptions for Qsys buses. These buses, however, become more power hungry with larger systems, because the bus itself uses more resources—such as wider multiplexers and additional soft interconnect—when spanning a larger distance to connect to a higher number of modules. Furthermore, we increasingly need to add power-hungry pipeline registers for larger Qsys buses to meet the 200-MHz timing requirement. In contrast, the embedded NoC routers are unchanged when we increase the number of modules. As Figure 6 shows, the embedded NoC is consistently more power efficient than tailored soft buses. The power gap widens further when we include clock-crossing FIFOs in a Qsys-generated bus; whereas, an embedded NoC already performs clock-crossing in its fabric ports.

Latency

The latency of Qsys buses varies according to the level of pipelining; in our systems, two or three pipeline stages typically are required. The average latency is therefore 12.5 ns. The embedded NoC requires a higher number of cycles; the average number of hops for our 16-node NoC is 3.5, and the latency per hop is three cycles in a lightly loaded system, such as the single-DDR3 system we study, leading to a latency of 11.5 ns. The fast 910-MHz NoC clock puts NoC latency on par with buses despite the higher number of cycles.

We have discussed the usage of NoCs for interconnection to DDR3 memory, which is naturally latency tolerant. Other I/O interfaces, such as PCIe, can also tolerate variable latency, making the NoC suitable for connecting to these important I/Os. However, the NoC can also be used for interconnecting application kernels to one another if they follow a latency-insensitive design (LID) methodology.[11] Although LID is not suitable for all applications, we believe it can not only mitigate scalability limitations imposed by large silicon chips[11] but also boost the productivity of reconfigurable computing. Combining embedded NoCs and LID can allow application kernels to be portable and easily composed, without having to redesign for a different timing constraint each time a designer creates a new system or moves between devices. This approach makes hardware design more modular with the communication within an application abstracted in the NoC. Additionally, it mirrors how software design allows simple application composition using functions from different libraries, with communication abstracted through shared memory. Our future work will focus on the design tools necessary to target NoC-augmented FPGAs with LID.

High design effort remains a major barrier to reconfigurable computing with FPGAs. A major hurdle is the design—using the low-level FPGA interconnect—of the communication infrastructure that ties together different parts of a computing application. We propose augmenting the FPGA with an embedded NoC to simplify

system-level interconnection while maintaining the FPGA's configurability. A high-bandwidth NoC can be integrated onto a modern FPGA for a core area increase of only 1.4 percent. When compared with the application-tailored buses currently used for system integration, embedded NoCs are more efficient in terms of area and power, even though they are overprovisioned in capability and bandwidth. MICRO

Acknowledgments

We thank Daniel Becker, Natalie Enright Jerger, Desh Singh, Dana How, and David Lewis for their helpful discussion and feedback. This work is funded by Altera, NSERC, and Vanier CGS.

References

1. O. Lindtjorn et al., "Beyond Traditional Microprocessors for Geoscience High-Performance Computing Applications," *IEEE Micro,* vol. 31, no. 2, 2011, pp. 41-49.
2. G.C.T. Chow et al., "A Mixed Precision Monte Carlo Methodology for Reconfigurable Accelerator Systems," *Proc. 20th ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays* (FPGA 12), 2012, pp. 57-66.
3. D. Chen and D. Singh, "Fractal Video Compression in OpenCL: An Evaluation of CPUs, GPUs, and FPGAs as Acceleration Platform," *Proc. 18th Asia and South Pacific Design Automation Conf.* (ASP-DAC 13), 2013, pp. 297-304.
4. D. Chen and D. Singh, "Using OpenCL to Evaluate the Efficiency of CPUs, GPUs and FPGAs for Information Filtering," *Proc. 22nd Int'l Conf. Field Programmable Logic and Applications* (FPL 12), 2012, pp. 5-12.
5. W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proc. 38th Design Automation Conf.* (DAC 01), 2001, pp. 684-689.
6. L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *Computer,* vol. 35, no. 1, 2002, pp. 70-78.
7. D.U. Becker, "Efficient Microarchitecture for Network-on-Chip Router," PhD dissertation, Dept. of Electrical Engineering, Stanford University, 2012.
8. M.S. Abdelfattah and V. Betz, "Design Tradeoffs for Hard and Soft FPGA-based Networks-on-Chip," *Proc. Int'l Conf. Field-Programmable Technology* (FPT 12), 2012, pp. 95-103.
9. M.S. Abdelfattah and V. Betz, "The Power of Communication: Energy-Efficient NoCs for FPGAs," *Proc. 23rd Int'l Conf. Field Programmable Logic and Applications* (FPL 13), 2013, pp. 1-8.
10. C.A. Zeferino et al., "A Study on Communication Issues for Systems-on-Chip," *Proc. 15th Symp. Integrated Circuits and System Design* (ICSD 02), 2002, pp. 121-126.
11. L. Carloni and A. Sangiovanni-Vincentelli, "Coping with Latency in SOC Design," *IEEE Micro,* vol. 22, no. 5, 2002, pp. 24-35.

Mohamed S. Abdelfattah is a PhD candidate in the Department of Electrical and Computer Engineering at the University of Toronto. His research interests include new communication architectures and CAD for high-performance FPGAs. Abdelfattah has an MSc in electrical and computer engineering from the University of Stuttgart.

Vaughn Betz is an associate professor in the Department of Electrical and Computer Engineering at the University of Toronto. His research interests include FPGA architecture and CAD, and the use of reconfigurable hardware to accelerate computation. He has a PhD in Electrical and Computer Engineering from the University of Toronto.

Direct questions and comments about this article to Mohamed S. Abdelfattah, University of Toronto, Electrical and Computer Engineering, 10 King's College Road, Toronto, ON, Canada, M5S 3G4; mohamed@eecg.utoronto.ca.