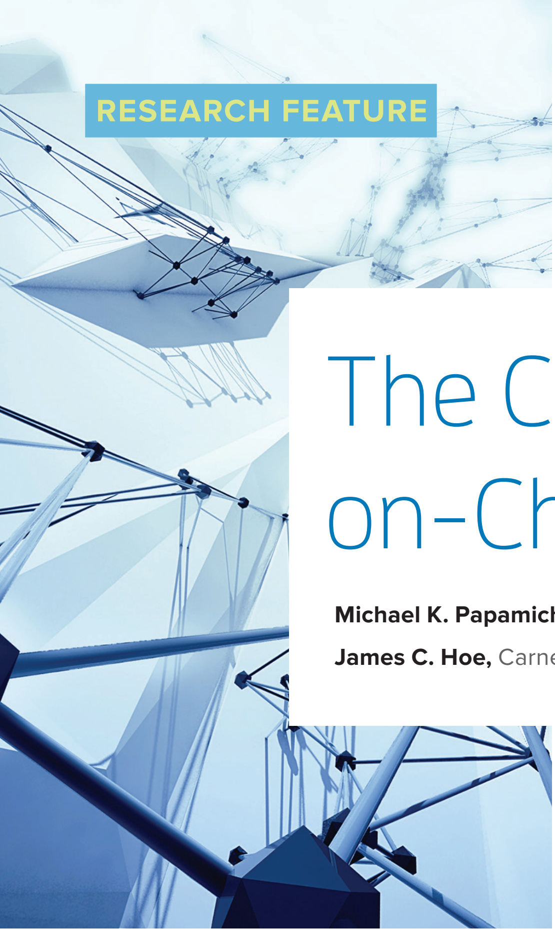


RESEARCH FEATURE

The CONNECT Network-on-Chip Generator

Michael K. Papamichael, Microsoft Research
James C. Hoe, Carnegie Mellon University

Efficiently supporting the communication needs of systems on chip with tens to hundreds of interacting modules requires a systematic and flexible network-on-chip (NoC) infrastructure. The freely available CONNECT generator lets users quickly navigate a range of design parameters to produce tailored NoC design instances in Verilog. To date, it has generated nearly 4,000 designs.

Modern integrated circuits (ICs) can contain billions of transistors organized as hundreds of interacting modules that form a system on chip (SoC). This scale is making module intercommunication increasingly complex, which is pushing interconnects into a more central chip-design role. Chips with a handful of major modules can still rely on ad hoc point-to-point wires or a shared bus, but such approaches do not scale to the extent needed for current SoC communication needs. Consequently, there is a push to develop more sophisticated and scalable interconnect schemes, such as networks on chip (NoCs), which, as the name implies, implement a dedicated network of links and routers that act as the chip's communication substrate.[1]

NoC proliferation, in turn, has stimulated a surge in NoC-related research—from low-level hardware implementation to application issues, such as providing quality-of-service (QoS) guarantees. As a result, NoC designs form a broad landscape that mirrors the diverse communication needs and requirements of applications that run on SoCs.

Despite vast and varied design choices and the lack of a one-size-fits-all solution to the interconnect problem, the NoC intellectual property (IP) blocks—prevalidated, reusable hardware modules that implement a specific function—currently available to the research community are limited. Most are fixed NoC instances or target only a few design aspects. NoC IP blocks, examples of which are described in the "Network-on-Chip Tools and Frameworks" sidebar, typically come in the form of structural register-transfer level (RTL) design modules. As such, their parameterization is limited by the expressiveness of current hardware description languages (HDLs), such as Verilog or VHDL. Users generally can choose either a single topology configuration or a configuration from a small set. They must also deal with low-level details, such as editing RTL code to set parameters or configure a router and writing additional RTL code to arrange routers in the chosen topology and populate routing tables.

To provide more options for both NoC novices and experts, we developed CONNECT, a flexible NoC IP generator that produces high-quality synthesizable NoC implementations in Verilog. Synthesizable designs are

NETWORK-ON-CHIP TOOLS AND FRAMEWORKS

The rapid growth of research interest in networks on chip (NoCs) as well as their commercial application has led to the development of several tools and frameworks for NoC design. Commercial interconnect solutions include system-on-chip (SoC) oriented products, such as Spidergon's STNoC, Arteris' FlexNoC, ARM's AMBA, and Sonics' NoC tools. Commercial interconnect architectures commonly used in a field-programmable gate array (FPGA) environment include ARM's AXI, which is in modern Xilinx FPGAs, or Altera's Qsys.

In addition to CONNECT, several freely available synthesizable NoC options are available, primarily the result of academic and research efforts. The Open Source Network-on-Chip Router RTL Project (http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router) provides a high-quality Verilog implementation of a state-of-the-art virtual channel router. Netmaker (www-dyn.cl.cam.ac.uk/~rdm34/wiki/) comprises a library of various synthesizable NoC components, along with supporting material and scripts to run simulations under different traffic patterns. Atlas (https://corfu.pucrs.br/redmine/projects/atlas), NoCem (http://opencores.org/project,nocem), and the Open Source Low Latency Network-on-Chip (NoC) Router RTL Project (http://nocrouter.codeplex.com) provide RTL code for building NoCs with mesh and torus topologies. The OpenSoC Fabric project (www.opensocfabric.org) provides a parameterized Chisel-based NoC implementation that supports mesh and flattened butterfly topologies. Bluetiles and Bluetree, both part of Blueshell (https://rtslab.wikispaces.com/Blueshell), are mesh and tree NoC implementations in Bluespec System Verilog for connecting processor cores to each other and with memory.

There are also tools to facilitate the evaluation and implementation of NoC choices. NOCBENCH (www.tkt.cs.tut.fi/research/nocbench) includes a set of hardware and software models and tools to help evaluate NoC designs. FPGA NoC Designer (www.eecg.utoronto.ca/~mohamed/noc_designer.html) provides implementation estimates for hard and soft FPGA-resident NoCs.

described in enough detail that electronic design automation (EDA) tools can implement them in hardware such as field-programmable gate arrays (FPGAs) or application-specific ICs (ASICs). Our main goal was to drastically reduce the complexity involved in configuring and generating working NoC designs by offering a push-button solution for generating a wide range of NoC configurations. CONNECT generates the requested NoC designs on demand, providing choices in a range of key parameters like topology, router architecture, flow control, allocation algorithms, pipelining options, and buffer size.

CONNECT, which evolved from a tool we created to support our NoC design exploration research,[2] was publicly released in 2012 as a Web-based NoC-generation service (www.ece.cmu.edu/calcm/connect) to support researchers and hardware designers in rapid NoC experimentation and prototyping. Our motivation for developing and releasing CONNECT in this form was to create a powerful and user-friendly tool that expands the NoC options available to academic and other research communities.

DESIGN PARAMETERS

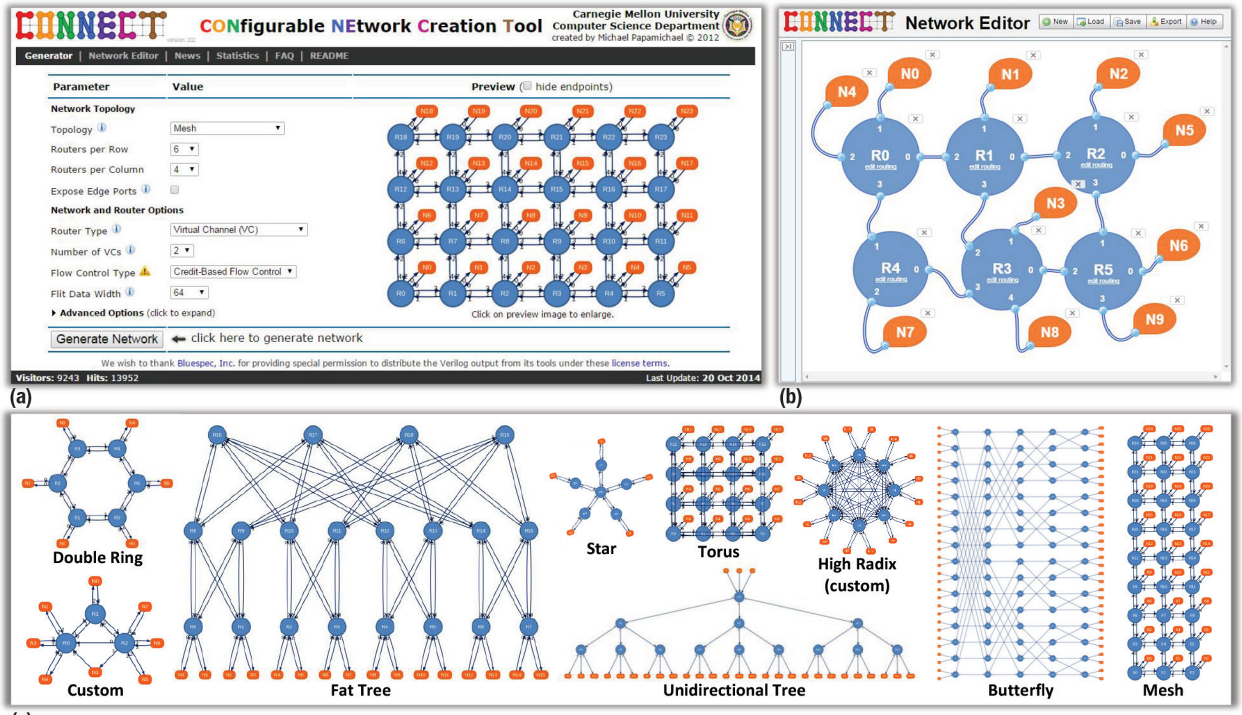
As Figure 1 shows, users see a range of design parameters through a front-end interface that consists of multiple high-level configuration interfaces dynamically updated to guide users while they interact with the generator. CONNECT's main interface supports a wide range of common network topologies, provides hardware implementation estimates, and lets users preview each candidate network's router and endpoint arrangement.

Network topologies

In our experience, most CONNECT users are SoC and application-level designers looking for an interconnect solution that fits with their design. For these users, CONNECT provides a wide range of common preconfigured unidirectional and bidirectional topologies, including single switch, ring, double ring, star, mesh, torus, fat tree, fully connected, butterfly, and distribution/aggregation tree. We are continuing to expand the variety of provided topologies and configuration options in response to user requests.

Each supported topology family includes its own scaling and configuration parameters. CONNECT also populates the routing tables in each network using a default routing scheme that aligns with the selected topology variant, which the user can override. With this wide range of topology and configuration options, users can rapidly custom-configure an NoC with minimal effort.

CONNECT supports advanced users through its network editor (see Figure 1b), which lets them use a graphical interface to create custom topologies.


CONNECT CONfigurable NEtwork Creation Tool
Carnegie Mellon University Computer Science Department created by Michael Papamichael © 2012

Parameter	Value
Network Topology	
Topology	Mesh
Routers per Row	6
Routers per Column	4
Expose Edge Ports	☐
Network and Router Options	
Router Type	Virtual Channel (VC)
Number of VCs	2
Flow Control Type	Credit-Based Flow Control
Flit Data Width	64

Click on preview image to enlarge.

We wish to thank Bluespec, Inc. for providing special permission to distribute the Verilog output from its tools under these license terms.

Visitors: 9243 Hits: 13952 Last Update: 20 Oct 2014

FIGURE 1. CONNECT interface: (a) Web-based generator interface, (b) network editor, and (c) samples of preconfigured topologies and two user-defined custom topologies. Users can rapidly switch among design parameters, such as router architecture, flow control, allocation algorithms, pipelining options, and buffer size.

Through a network-specification language, expert users can instantiate routers of different radix, mix unidirectional and bidirectional links, and attach multiple (or no) endpoints to each router in the generated network. With this high degree of customization, users can build networks that precisely match their application's connectivity and communication characteristics.

Router architectures

Figure 2 is an abstract block diagram of the basic structure of the virtual channel (VC) router—one of three router architectures that CONNECT provides. The other two are virtual output queued (VOQ) and input queued (IQ). The variants differ in their internal logic and buffer organization, but in all three, input and output interfaces are either connected to network endpoints or form links with other routers in the network.

For all router variants, users can configure low-level details, such as pipelining, allocator type, and routing, which lets them explore tradeoffs in performance, area, and frequency, as well as meet specific traffic prioritization and fairness goals. CONNECT also provides two interfacing options (credit-based and peek) that implement different flow-control protocols to better match the communication assumptions and requirements of a given application. In credit-based flow control, routers exchange credits and maintain credit counters to keep track of buffer availability; in peek flow control, routers directly expose their buffer occupancy to upstream routers, which eliminates the need for credit-counting logic and storage.[2]

Virtual channel. The VC router supports a variable number of VCs and organizes incoming traffic at each input into separate buffers on the basis of VC information carried by packets. NoCs employ VCs to provide the abstraction of multiple logical channels over a physical underlying channel. VCs are useful in implementing protocols that require enforcing QoS guarantees, such as traffic isolation and message-class prioritization, for example, prioritizing responses over requests to prevent deadlock.[3] They can also help increase network performance by mitigating the effects of head-of-line blocking—in which the first packet in a queue blocks the remaining waiting packets that would otherwise be making progress.[4]

Virtual output queued. The VOQ router can offer the highest performance out of the three supported architectures. For each input, it steers incoming traffic into per-output dedicated buffers, eliminating the effects of head-of-line blocking and offering very high levels of performance that can approach that of an ideal (but impractical) output-queued router architecture.[5] VOQ routers are well suited for demanding applications with heavy

TABLE 1. Results of implementing CONNECT-generated network-on-chip design modules in FPGA and ASIC environments.*

Network details					LX760 FPGA implementation			ASIC implementation (32 nm)		
Topology_endpoints	Link width	Ports per router	No. of routers (variant)	No. of VCs	Logic area (% of LUTs)	Max. freq. (MHz)	Peak bisection bandwidth (Gbps)	Area (μm^2)	Max. freq. (GHz)	Peak bisection bandwidth (Gbps)
Ring_128	128	2	128 (IQ)	N/A	8.9	364	93	561,181	5.2	1,342
Ring_64	64	2	64 (VC)	2	3.4	327	42	305,159	4.5	572
DoubleRing_16	48	3	16 (VC)	4	2.0	236	45	110,492	3.5	680
DoubleRing_32	64	3	32 (VC)	2	2.9	241	62	142,797	3.8	978
FatTree_16	32	4	20 (VOQ)	N/A	1.9	203	104	22,070	4.9	2,484
Mesh_16 (4 × 4)	32	5	16 (VC)	4	3.2	181	46	79,537	3.6	914
Mesh_48 (6 × 8)	24	5	48 (VC)	2	7.4	169	32	104,500	4.4	842
Torus_20 (4 × 5)	64	5	16 (VOQ)	N/A	6.4	180	184	49,712	5.2	5,361
FullyConnected_8	32	8	8 (VC)	2	3.1	132	34	19,687	4.0	1,034
HighRadixCustom_16	48	9	8 (IQ)	N/A	4.3	121	47	31,219	5.2	1,990

*μm^2: square micrometers; ASIC: application-specific integrated circuit; FPGA: field-programmable gate array; Gbps: gigabits per second; IQ: input queued; LUTs: lookup tables; VCs: virtual channels; VOQ: virtual output queued.

communication requirements and less structured traffic patterns that would still suffer from head-of-line blocking using a conventional VC-based router.

Input queued. The IQ router uses a single buffer per router input, making it well suited for building simple barebones NoCs for applications that require basic connectivity with low hardware cost. NoCs built around IQ routers are a good match for applications with non-critical or simple communication needs that do not require the isolation, prioritization, or higher performance that VC and VOQ router architectures provide.

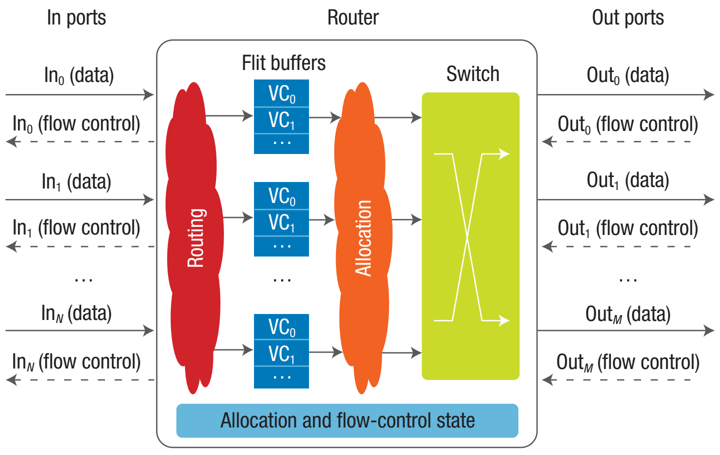


FIGURE 2. Architecture of a CONNECT virtual channel (VC) router. Communication with other routers happens through input and output port interfaces. Each port interface consists of two channels; one channel for receiving (or sending) data and one side channel running in the opposite direction for flow control.

QUALITY OF GENERATED NOCS

Because NoCs are typically used or studied as part of larger designs with interacting modules that exhibit diverse communication characteristics and often impose stringent constraints on hardware resources, generated NoCs must be high quality and map well to the available hardware resources. All CONNECT NoC IPs—including any debugging and instrumentation structures—consist of fully synthesizable Verilog descriptions that map efficiently to both FPGAs and ASICs. Generated NoCs account for unique FPGA implementation characteristics, ensuring that they can coexist and share resources with other hardware-resident components in an FPGA environment with tight resource constraints.

The results in Table 1 give an idea of how the generated NOCs map to FPGAs and ASICs. The results are for select realistic configurations; all networks assume a flit-buffer depth of

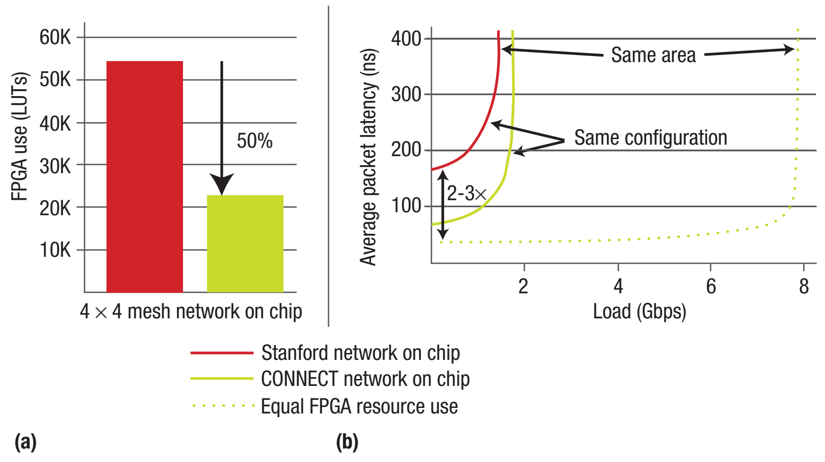


FIGURE 3. Comparison of a 4 × 4 mesh network on chip (NoC) using CONNECT-generated RTL and the Stanford open source router RTL. (a) FPGA resource use (measured as the number of lookup tables [LUTs]) is 50 percent more efficient in the CONNECT NoC. (b) Assuming uniform traffic at 100 MHz, network performance is also significantly better. When the CONNECT NoC uses the same FPGA resources (same area) as the Stanford NoC, the CONNECT NoC can handle four times the load with two to three times lower idle latency.

eight flow-control digits (flits), which are the basic unit of resource allocation and flow control within the network; employ peek flow control; and have the router core pipeline option. However, they vary in number of endpoints, link width, number of routers and router architecture, and number of VCs (for VC routers). The maximum frequency, area, and peak bisection bandwidth given in the table are based on synthesis results for FPGA and ASIC implementations. The table shows FPGA logic area as a percentage of total FPGA lookup table (LUT) capacity. All the sample networks fit well within 10 percent of the moderately sized Xilinx LX760 FPGA.

The network details that give optimal results on FPGAs can differ drastically from those that map well to ASICs because the two implementation environments contrast sharply in relative speed and cost of logic, wires, and memory primitives. CONNECT accounts for those differences by considering each implementation environment's unique mapping and operating characteristics, which enables it to use resources more efficiently. For example, by accounting for FPGAs' dense configurable routing substrate, on-chip storage peculiarities, and frequency limitations, it can produce NoCs that use FPGA resources very efficiently.

FPGA resource use

To assess the importance of FPGA specialization, we compared a 4 × 4 mesh built using the Stanford open source router RTL,[6] a high-quality virtual channel router implementation in synthesizable Verilog, with an identically configured 4 × 4 mesh NoC generated by CONNECT.

Figure 3 shows the extent to which CONNECT's flexibility gives it an advantage over other NoC design options that cannot tune results to the FPGAs' implementation environment.[2] In this example, CONNECT-generated NoCs have comparable network performance at one-half the FPGA resource cost; or alternatively, three to four times higher network performance with approximately the same FPGA resource cost.

Application-specific NoCs

CONNECT users can also create efficient application-specific NoCs. In an end-to-end study on FPGA application deployment, we experimented with compiler-guided development of application-specific NoCs. The customized NoCs generated through CONNECT reduced FPGA resource usage (for the interconnect) by almost an order of magnitude while retaining the same application performance levels as a baseline generic NoC.[7] For example, when we switched from a baseline generic mesh interconnect to a custom application-tuned, tree-based interconnect, we realized overall efficiency (throughput over area) gains from 37 to 48 percent for two FPGA-based applications: dense matrix multiply and Black–Scholes.

Tuning to design and performance constraints

Figure 4 demonstrates the coverage possible with CONNECT NoCs, showing 256 64-endpoint NoC configurations, which are taken from a pool of about 30,000 synthesized design variants that target a commercial 65-nm ASIC library. All these CONNECT NoCs are functionally equivalent and therefore interchangeable from an application perspective. The NoCs in the figure represent only a small subset of the NoC configurations available with CONNECT, yet they already show a variance of two to three orders of magnitude across the three metrics (power, area, performance). These results underline the importance of being able to tune and optimize an NoC to meet specific design and performance constraints.

USER AND DESIGN STATISTICS

As of late October 2015, the CONNECT website has seen more than 12,000 unique visitors and the service has generated on the order of 4,000 networks for 900 users in 50 countries (www.ece.cmu.edu/calcm/connect/stats). Research papers by CONNECT users have reported employing CONNECT to generate NoCs for design research or for production IP in design projects. Figure 5 shows network-generation statistics, such as user type and topologies selected. More detailed statistics and usage information are available on the CONNECT website.

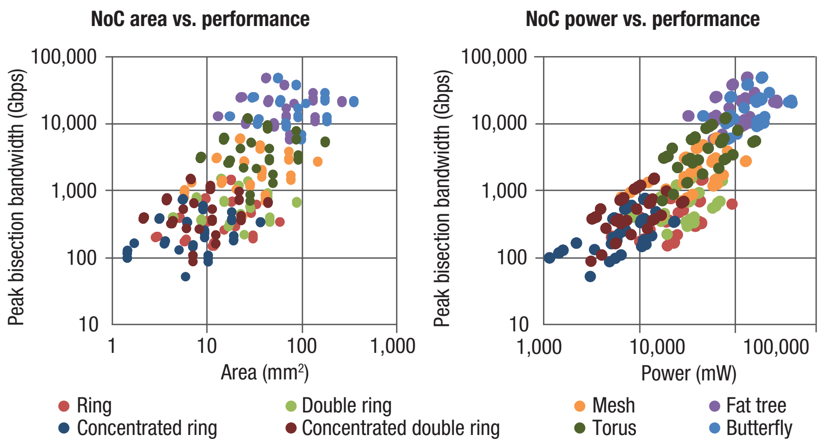


FIGURE 4. Area, power, and performance results for various 64-endpoint NoC configurations targeting a commercial 65-nm ASIC technology node. This small subset already demonstrates wide variance across area, power, and performance metrics, which shows the importance of being able to tune an NoC to meet design and performance constraints.

USER-CENTRIC DESIGN PRINCIPLES

To ensure that CONNECT would be a practical research and design tool for all user levels, we consciously engineered certain design principles across all its aspects—from the user interface to the hardware-generation engine.

Interfaces that match expertise

CONNECT offers a variety of configuration and generation interfaces that are tailored to the user's expertise. The main (Web-based) interface provides preconfigured common topologies and settings as well as the opportunity to create a custom topology through the network editor. Expert users can employ a custom specification language that eliminates the need for low-level bug-prone RTL coding, as well as a non-GUI command-line front end that generates NoC instances by remotely connecting to the CONNECT framework.

CONNECT's interfaces also have automated features that help users avoid erroneous configurations—a common problem when dealing with complex highly parameterized IP blocks. Options are dynamically updated as the user makes selections, and routing tables are automatically populated unless the user chooses to override that feature. The interface guides the user through previews of the topology and endpoint arrangement and provides feedback,

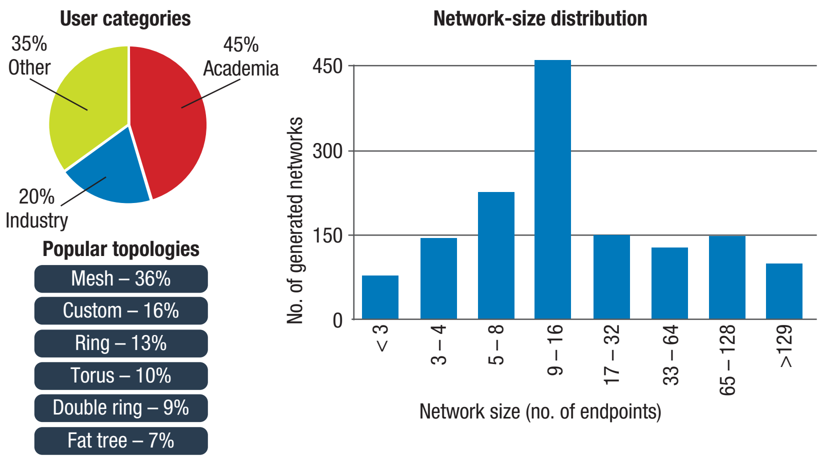


FIGURE 5. CONNECT network-generation statistics showing user categories, popular topologies, and network-size distribution as of March 2015.

such as cost and performance estimates, as well as tips on how different options affect hardware implementation.

Support for rapid prototyping and exploration

A simple set of link-level interfaces are common among all CONNECT-generated NoCs, which allows easy integration within a design project. From a user perspective, the NoC appears to be a plug-and-play black box module that receives and delivers packets. This simplifies rapid prototyping and exploring design choices, because all CONNECT NoCs with the same number of endpoints are interface-compatible at the network boundary.

CONNECT's support of routing-table updates not only opens up interesting research directions, such as experimenting with adaptive routing techniques, but also considerably reduces experiment turnaround time. Users can build a system around a dense, highly connected topology and then modify the routing scheme to emulate other topologies; for example, by overlaying a ring or mesh on top of a torus. There is no need to repeat hardware synthesis, eliminating hours of work.

ABOUT THE AUTHORS

MICHAEL K. PAPAMICHAEL is a researcher in the Computer Architecture Group at Microsoft Research. His research interests include computer architecture, on-chip interconnects, and methodologies to facilitate hardware specialization. While conducting the research reported in this article, he was a doctoral student in computer science at Carnegie Mellon University (CMU). Papamichael received a PhD in computer science from CMU. He is a member of IEEE and ACM. Contact him at papamix@microsoft.com.

JAMES C. HOE is a professor of electrical and computer engineering at CMU. His research interests include computer architecture, digital systems, and high-level synthesis. Hoe received a PhD in electrical engineering and computer science from MIT. He is a Fellow of IEEE and member of ACM. Contact him at jhoe@cmu.edu.

Easy endpoint implementation

In addition to the plug-and-play nature of the generated networks, CONNECT provides features aimed specifically at easing endpoint implementation. For example, with peek instead of credit flow control, endpoints can directly observe buffer occupancy of neighboring routers instead of having to exchange and keep track of credits. Virtual links guarantee the contiguous transmission and delivery of multiflit packets, which eliminates the need for reassembly logic and buffering at the receiving endpoints. These features push complexity back into the network—complexity that the network endpoints would handle otherwise.

In addition, each generated NoC is accompanied by documentation, test benches, and scripts, as well as user-editable routing and topology files—all custom-generated to match the specific NoC configuration.

Simplified access

Because CONNECT comprises many components developed using a variety of tools, releasing it to users as a self-maintained package would have required installing tools and libraries, setting up the environment, acquiring licenses, and continued upkeep, among other issues. Releasing CONNECT as a service eliminates these requirements and simplifies user access—the only requirement for generating CONNECT NoC designs is an Internet connection. Moreover, having a distribution point allows us to quickly and transparently deliver fixes or improvements.

Our experience with building and releasing CONNECT has given us invaluable insight into multiple aspects of IP development, dissemination, and use. For NoC design experts, CONNECT eliminates the time and effort they would have spent coding and debugging the RTL for an NoC design. Instead, they simply dial in exactly the configuration they are looking for.

However, user feedback reveals that most CONNECT users are not NoC experts and hence do not know what configuration to ask for. Often, they do not understand all the NoC parameters that CONNECT offers.

Indeed, our generation statistics show that most CONNECT users configure very few high-level parameters—often suboptimally—typically, topology or datapath width. Most leave options such as router architecture or allocator type untouched, despite the significant impact of these parameters on cost, performance, and correctness. This problem continues after non-expert NoC users integrate a CONNECT NoC into their design because users at this level likely cannot properly diagnose performance and correctness issues, such as degraded performance or deadlock from suboptimal router architecture or allocator choice.

Thus, despite the ease of use that IP generators promise, a wide knowledge gap persists between domain experts who develop the IP and non–domain experts who use the IP. This observation has motivated our work on Pandora,[8] a new IP design paradigm in which IP blocks not only capture the microarchitectural and structural design views but also encapsulate additional knowledge that the IP author might have. For example, by incorporating IP author knowledge in genetic algorithms, we have been able to extend CONNECT to vastly accelerate and automate NoC parameter tuning.[9] We believe our work on CONNECT and Pandora extensions are significant steps toward enabling the more efficient use of IPs and IP generators. C

ACKNOWLEDGMENTS

Funding for this work was provided in part by National Science Foundation grant CCF-1012851.

REFERENCES

1. W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proc. 38th ACM/IEEE Design Automation Conf.* (DAC 01), 2001, pp. 684–689.
2. M.K. Papamichael and J.C. Hoe, "CONNECT: Re-Examining Conventional Wisdom for Designing NoCs in the Context of FPGAs," *Proc. 20th ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays* (FPGA 12), 2012, pp. 37–46.
3. W. Dally and C. Seitz, "Deadlock-Free

Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, vol. C-36, no. 5, 1987, pp. 547–553.
4. M. Karol, M. Hluchyj, and S. Morgan, "Input Versus Output Queuing on a Space-Division Packet Switch," *IEEE Trans. Comm.*, vol. 35, no. 12, 1987, pp. 1347–1356.
5. W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2003.
6. D.U. Becker, "Efficient Microarchitecture for Network-on-Chip Routers," PhD dissertation, Dept. Electrical Eng., Stanford Univ., 2012.
7. E.S. Chung and M.K. Papamichael, "ShrinkWrap: Compiler-Enabled Optimization and Customization of Soft Memory Interconnects," *Proc. 21st IEEE Int'l Symp. Field-Programmable Custom Computing Machines* (FCCM 13), 2013, pp. 113–116.
8. M.K. Papamichael, "Pandora: Facilitating IP Development for Hardware Specialization," PhD dissertation, Dept. Computer Science, Carnegie Mellon Univ., 2015.
9. M.K. Papamichael, P. Milder, and J. C. Hoe, "Nautilus: Fast Automated IP Design Space Search Using Guided Genetic Algorithms," *Proc. 52nd ACM/EDAC/IEEE Design Automation Conf.* (DAC 15), 2015; doi: 10.1145/2744769.2744875.