System-Level Design of Networks-on-Chip for Heterogeneous Systems-on-Chip

(Invited Paper)

Young Jin Yoon*
Department of Computer Science
Columbia University
New York, New York
youngjin@cs.columbia.edu

Paolo Mantovani
Department of Computer Science
Columbia University
New York, New York
paolo@cs.columbia.edu

Luca P. Carloni
Department of Computer Science
Columbia University
New York, New York
luca@cs.columbia.edu

ABSTRACT

The network-on-Chip (NoC) is a critical subsystem for many large-scale systems-on-chip (SoC). We present a complete framework for the design and optimization of NoCs at the system-level. By combining a library of pre-designed configurable NoC modules specified in SystemC with high-level synthesis, we can generate a variety of alternative 2D-Mesh NoC architectures for a given SoC. We also support the automatic synthesis of network interfaces to translate between IP-specific messages and NoC flits. We demonstrate our approach with the design-space exploration of two complete SoCs running complex applications on a high-end FPGA board.

CCS CONCEPTS

• **Networks → Network on chip**; *Network components*; • **Hardware → Network on chip**;

KEYWORDS

Network-on-Chip, System-Level Design, Synthesizable SystemC

ACM Reference Format:
Young Jin Yoon, Paolo Mantovani, and Luca P. Carloni. 2017. System-Level Design of Networks-on-Chip for Heterogeneous Systems-on-Chip. In *Proceedings of NOCS'17, Seoul, Republic of Korea, October 19–20, 2017,* 6 pages.
https://doi.org/10.1145/3130218.3130238

1 INTRODUCTION

Networks-on-chip (NoC) play a critical role in the integration of components in large-scale systems-on-chip (SoC) at design time, and have a major impact on their performance at run time. Over the last few years, the research community has produced many different frameworks and tools for NoC design and optimization [7, 14, 16, 17]. Most of these approaches provide some degree of parameterization which allows designers to optimize the NoC architecture for the target SoC and the given ASIC or FPGA technology.

We leveraged this aggregate research experience for the development of *ICON (Interconnect Customizer for the On-chip Network).*

*Young Jin is now with Intel Corporation, Hillsboro, OR.

NOCS'17, October 19–20, 2017, Seoul, Republic of Korea

ACM ISBN 978-1-4503-4984-0/17/10.
https://doi.org/10.1145/3130218.3130238

Components	Parameters	Subcomponents
Input/Output units	Queue size, Number of VCs	Routing unit, flow-control unit
Virtual channel (VC) allocator	Input/output-first, wavefront	Input/Output arbiters
Switch (SW) allocator	VC and output-first, wavefront	VC arbiters, output arbiters
Allocator unit	Independent, speculative	VC/SW allocators
Router	RC/SA/VA/ST/VT pipelined	Input/Output units allocator unit, crossbar
Physical network	Flit width, topology	Routers, channels, network interfaces (NI)

Table 1: NoC parameters and sub-components in ICON.

ICON is a new framework for the design and optimization of NoCs at the system level. Some of its distinguished features include: support for virtual channels for message-class isolation, which is critical for the prevention of protocol deadlock [20], the ability to generate NoC architectures that combine multiple physical networks with multiple virtual channels [23], and the ability to explore the NoC design space by varying the NoC parameters in a non-uniform way (e.g. to have different numbers of virtual channels per input port in a router [9]). The generation of NoCs with ICON relies on a rich library of parameterized components that can be combined in a modular way to create complex NoC subsystems and, ultimately, a complete NoC architecture tailored to the target SoC. Table 1 reports a list of the key components that can be used to generate a variety of router micro-architectures.

ICON promotes system-level design as it allows the automatic generation of NoC architectures specified in SystemC. These generated specifications can be integrated with full-system simulators, known as virtual platforms, as well as synthesized with high-level synthesis (HLS) tools to produce corresponding RTL implementations. Makefiles and scripts for synthesis, simulation, and co-simulation across various levels of abstraction are automatically generated along with the SystemC source code. By bringing the description of the NoC to a higher level, ICON enables the exploration of a broader design space through the combination of system-level parameters with micro-architectural settings for the HLS tool. Also, the compatibility with virtual platforms allows fast full-system simulation, which is crucial to increase the number of design points that can be evaluated.

After summarizing the most related NoC research in Section 2, we present the overall architecture of ICON and its unique features in Section 3. Then, in Section 4 we demonstrate some of the capabilities of ICON by generating 36 different NoC configurations that can be seamlessly integrated in two SoCs, which we designed and

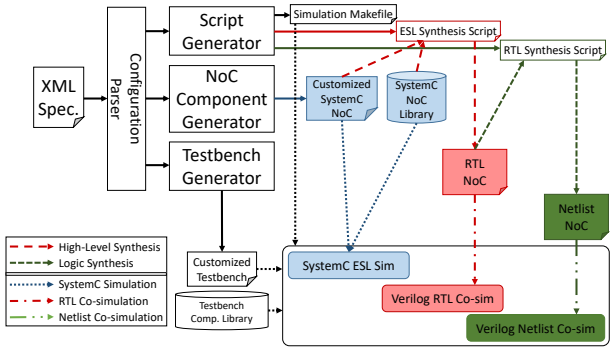


Figure 1: The ICON synthesis and simulation flows.

implemented on an FPGA board. We present a comparative analysis of the resources utilization and performance evaluation across these NoC configurations for the two SoC designs while running real workloads. We also report estimates on area occupation and throughput for a corresponding ASIC implementation tested with synthetic traffic patterns.

2 RELATED WORK

How to design low-latency and high-bandwidth architectures by combining flexible and configurable parameterized components has been the focus of many papers in the NoC literature.

Mullin *et al.* proposed low-latency virtual-channel routers with a free virtual channel queue and VA/SA speculation that offer a high degree of design flexibility in SystemVerilog [14]. Kumar *et al.* demonstrated a 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator scheme that improves the matching efficiency by allowing multiple requests per clock cycle and keeping track of previously conflicted requests [11]. Becker presented a state-of-art parameterized virtual channel router RTL with a new adaptive backpressure mechanism that improves the utilization of the router input buffers [3]. Dall'Osso *et al.* developed *×pipes* as a scalable and high-performance NoC architecture, where parameterizable SystemC component specifications are instantiated and connected to create various NoCs [5]. Stergiou *et al.* improved this architecture by presenting *×pipes Lite*, a synthesizable parameterizable NoC component library that includes OCP 2.0 compatible network interfaces, and by providing a companion synthesis and optimization flow [22]. Fatollahi-Fard *et al.* developed *OpenSoC Fabric* [7], a tool that simplifies the generations of NoCs from parameterized specification by leveraging the properties (abstract data types, inheritance, etc.) of Chisel hardware description language [1].

A large portion of NoC research focused on FPGAs. Lee *et al.* analyzed the performance sensitivity to various NoC parameters for FPGA-based NoCs [12]. Kapre *et al.* presented a detailed analysis of packet-switch vs time-multiplexed FPGA overlay networks [10]. Schelle *et al.* presented NoCem, an architecture based on composing simple router blocks to build large NoCs on FPGAs [18]. Hilton *et al.* proposed PNoC, a flexible circuit-switched NoC for FPGA-based systems [8]. Shelburne *et al.* proposed MetaWire to emulate a NoC on FPGAs [19]. Lu *et al.* presented a cost-effective low-latency NoC router for FPGA [13]. Papamichael *et al.* developed the *CONfigurable NEtwork Creation Tool (CONNECT)* [17] that combines Bluespec SystemVerilog [15] and a web-based front-end to generate a fast FPGA-friendly NoC based on a simple but flexible fully-parameterized router architecture.

In developing ICON we kept in mind the lessons from many of these works. Given the common emphasis on system-level design, our work has perhaps most commonalities with the CONNECT project. However, we trade off some optimization in favor of more flexible framework that targets both ASIC and FPGA technologies.

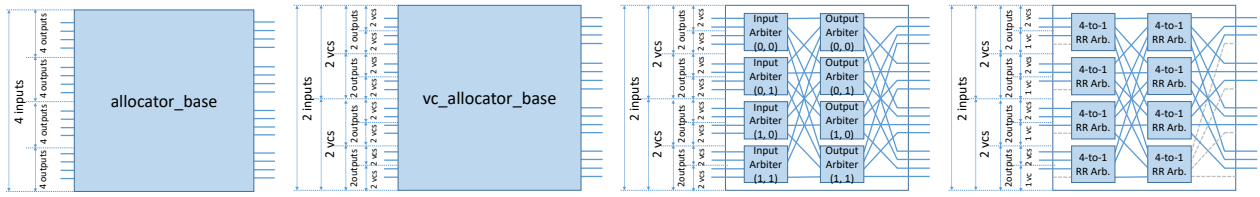
Distinctively, ICON is the first system-level framework that can generate hybrid NoC architectures which combine virtual channels with multiple physical planes. In addition, ICON pushes the design entry point to the system level in a way that it enables the exploration of a broader design space and the evaluation of a very large number of design points in such space.

3 THE ICON FRAMEWORK

The main advantage of using ICON is to generate multiple different NoCs, integrate them into existing SoCs, and create new NoC components with minimal effort. Most of this flexibility is achieved by allowing users to mix-and-match several heterogeneous instances of each sub-component listed in Table 1 to build customized NoC components. Following a user-defined topology and connection scheme, these components are then automatically connected to generate the desired NoC configuration. In addition, ICON generates the necessary simulation environment and testbench for validation, which can be reused across all NoC configurations generated with both pre-configured and custom sub-components. Furthermore, users can extend the set of configuration parameters available to ICON. For example, a user can add definitions of round-robin or random-based arbiters to create new types of virtual channel (VC) allocators. At a higher level in the NoC hierarchy, these allocators can be selected to build different types of routers.

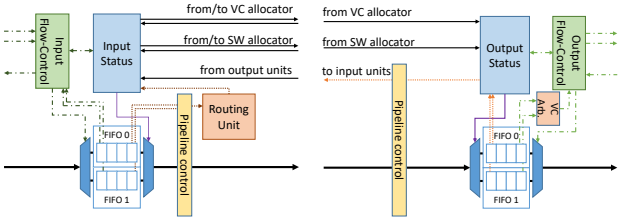
Beside the NoC generation, ICON automatically creates network interfaces according to the message types and message classes specified for the IP components of the SoC. Hence, users can mix-and-match different NoC configurations without changing IP component specification. Alternatively, the same NoC can be used for multiple SoCs, each with a specific set of message types and message classes. All customized NoC components can be seamlessly integrated. The communication behavior of the same type of components, *i.e. a component group*, is pre-defined in ICON. Testbenches and synthesis scripts can be shared for a component group. This simplifies the validation of user-defined NoC components and their integration into the target system.

ICON consists of six main parts: configuration parser, script generator, NoC component generator, testbench generator, the SystemC NoC library and the testbench component library. Fig. 1 illustrates the high-level relationships between these parts and the flow that ICON follows to generate the NoC design and the corresponding scripts for synthesis and simulation. Starting from the user-provided specification of the NoC through an XML template, the parser instantiates the necessary objects to build the NoC architecture with the desired configuration. The objects are then sent to the three generators that produce the actual NoC design, together with the scripts for synthesis and simulation, and the SystemC testbenches to validate the design. With the parameter-specific or customized SystemC code from the NoC component generator, the user can launch first HLS and then logic synthesis using the `tcl`



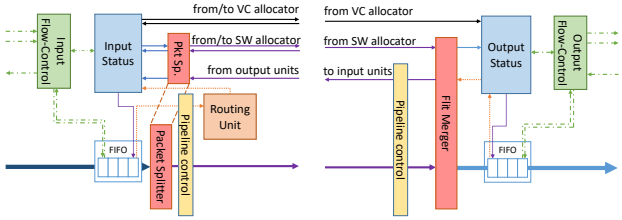
(a) `allocator_base` (b) `vc_allocator_base` (c) input-first vc allocator (d) customized vc allocator

Figure 2: An example of object-oriented and parameterized module implementation with the virtual channel allocator.



(a) router input unit (b) router output unit

Figure 3: Input and output units of routers with 2 VCs.



(a) source NI input unit (b) dest NI output unit

Figure 4: Input and output units of network interfaces.

scripts from the script generator. The synthesized RTL and netlist can then be co-simulated with the same testbenches by using the generated Makefiles. The SystemC testbench component library is equipped with the set of synthetic traffic models commonly used to evaluate NoCs. These traffic models can be controlled with simulation configurations specified in the XML specification.

SystemC NoC Component Library. The SystemC NoC library contains a rich set of components and sub-components that are specified based on object-oriented programming and that can be combined hierarchically to obtain a variety of NoC architectures. Table 1 gives an example of the many components and sub-components for the router and their hierarchical relationships. The router class is one of the main classes and is defined as a collection of input units, output units, VC and SW allocators, and crossbars in the NoC component library. All these sub-components are defined as C++ template parameters in the router class to provide the flexibility of combining various sub-component implementations to build a router. A component like the router can have a uniform microarchitecture, where every sub-component is configured with the same parameter values, or a non-uniform architecture. An example of the latter is a router which supports different numbers of virtual channels across different inputs. The NoC component generator instantiates a predefined design from the library for a uniform microarchitecture, while it creates a customized SystemC class at runtime for non-uniform microarchitectures.

By sharing the same interface across different implementations, NoC components in ICON can be seamlessly combined into a bigger component. Fig. 2 illustrates an example of how these common interfaces are specified for the case of virtual channel allocators. All allocators are derived from `allocator_base` (Fig. 2(a)), and the number of input and output (I/O) virtual channels are specified in `vc_allocator_base` (Fig. 2(b)). When using uniform sub-components to create a large component, ICON leverages SystemC template parameters. For example, the input-first VC allocator [6] is derived from `vc_allocator_base`, and contains multiple arbiters in the I/O stages (Fig. 2(c)). For each I/O stage, the type of arbiter is specified as a template parameter for the input-first VC allocator implementation in the NoC component library. If multiple non-uniform sub-components need to be instantiated in a component, *e.g.* different number of output VCs per output unit, the front-end SystemC generator dynamically produces SystemC classes by inheriting common interfaces defined in the SystemC NoC library. For example, to create the allocator of Fig. 2(d) derived from the one of Fig. 2(c), the template parameters for I/O arbiters are specified as 4-to-1 round-robin arbiters based on the XML specification, and some of unused VCs (gray lines) are bound to constants.

Input and Output Units. Fig. 3 illustrates how the I/O units are implemented in the SystemC NoC library. Both the I/O units consist of flow-control, status control, and pipeline control modules with optional FIFOs to store flits. In addition, an input unit contains a routing unit to calculate the designated output port based on the destination information in the header flit. The routing unit in Fig. 3(a) not only produces the output port of the flit, but also provides possible output VCs with the message class of the input VCs. By providing extra information for the output VCs at the routing stage, input units avoid sending unnecessary requests to the VC allocator. Therefore, a generic VC allocator implementation can be used without any modification for the message-class isolation. Instead of managing the granted inputs and outputs and their VC information with a centralized status logic, ICON relies on distributed VC and flow management between I/O units. A distributed design makes it easier to instantiate non-uniform I/O ports. It also helps to control the status of non-uniform I/O ports that characterizes a network interface.

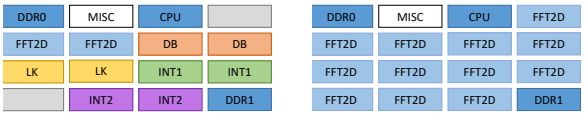
Network Interfaces. In order to support multiple physical networks [23], message-class isolation [20], and non-uniform packet specification, we designed network interfaces in ICON as routers with non-uniform data types for the input or output ports. Thanks to the parameterized and component-based design, the implementation of the I/O unit for both source and destination network interfaces reuses most of the router sub-component implementations in

```
<network_type name="example2x2">
  <source_network_interfaces num_src="4">
    <source_network_interface index="0" type="sni"/>
    <source_network_interface index="1" type="sni"/>
    <source_network_interface index="2" type="sni"/>
    <source_network_interface index="3" type="sni"/>
  </source_network_interfaces>
  <destination_network_interfaces num_dest="4">
    <destination_network_interface index="0" type="dni"/>
    <destination_network_interface index="1" type="dni"/>
    <destination_network_interface index="2" type="dni"/>
    <destination_network_interface index="3" type="dni"/>
  </destination_network_interfaces>
  <routers num_routers="4">
    <router index="0" type="r2x2"/>
    <router index="1" type="r2x2"/>
    <router index="2" type="r2x2"/>
    <router index="3" type="r2x2"/>
  </routers>
  <channels>
    <channel type="ch" src_ni="0" src_port="0" dest_router="0" dest_port="4"/>
    <channel type="ch" src_ni="1" src_port="0" dest_router="1" dest_port="4"/>
    <channel type="ch" src_ni="2" src_port="0" dest_router="2" dest_port="4"/>
    <channel type="ch" src_ni="3" src_port="0" dest_router="3" dest_port="4"/>
    <channel type="ch" src_router="0" src_port="4" dest_ni="0" dest_port="0"/>
    <channel type="ch" src_router="1" src_port="4" dest_ni="1" dest_port="0"/>
    <channel type="ch" src_router="2" src_port="4" dest_ni="2" dest_port="0"/>
    <channel type="ch" src_router="3" src_port="4" dest_ni="3" dest_port="0"/>
    <channel type="ch" src_router="0" src_port="1" dest_router="1" dest_port="0"/>
    <channel type="ch" src_router="0" src_port="3" dest_router="2" dest_port="2"/>
    <channel type="ch" src_router="1" src_port="0" dest_router="0" dest_port="1"/>
    <channel type="ch" src_router="1" src_port="3" dest_router="3" dest_port="2"/>
    <channel type="ch" src_router="2" src_port="1" dest_router="3" dest_port="0"/>
    <channel type="ch" src_router="2" src_port="2" dest_router="0" dest_port="3"/>
    <channel type="ch" src_router="3" src_port="0" dest_router="2" dest_port="1"/>
    <channel type="ch" src_router="3" src_port="2" dest_router="1" dest_port="3"/>
  <channels>
<network_type>
```

Figure 5: Example of 2×2 NoC XML specification for ICON.

the NoC component library. Specifically, a source network interface is implemented as a specialized router where the input unit accepts packets and produces multiple flits, while a destination network interface is implemented as a specialized router where the output unit collects multiple flits to produce a packet. Fig. 4 illustrates the specialized I/O units to build a network interface. Compared to the router I/O units shown in Fig 3, all components are the same, with the exception of the packet splitter and the flit merger. Starting from the user specification of the packet format for the source and destination, ICON creates a SystemC module that implements a custom channel. The latter is characterized by a specific interface implemented with the list of input ports (`sc_in`) and output ports (`sc_out`) for the module. This channel is also used as a data type to create status, flow-control, and FIFOs for the I/O units. Packet splitters and flit mergers are attached to these components to translate a packet from/to multiple flits. Since the flit is the base of the control mechanism between I/O units, the packet splitter and flit merger must manage the request and grant signals between the input status and the switch allocator. For example, upon receiving a packet from the input queue, the packet splitter creates requests and manages grants for the switch allocator until the entire packet is sent to the output unit as a sequence of multiple flits. After sending the last flit of a packet, the packet splitter sends a grant signal back to the input status to indicate the complete transmission. Similarly, flit mergers keep collecting flits from input units to build a packet and send a grant signal to the output status to indicate when a valid packet is ready.

Network Generation. Fig. 5 shows the example of an XML tree that defines a simple 2x2 2D-Mesh NoC. A user can specify routers with `router`, and network interfaces with `source_network_interface` and `destination_network_interface` XML elements. Links are specified as `channel` with the connection information. Based on this specification, ICON generates a class with fully customized `sc_in` and `sc_out` for the network interfaces, and instantiates and connects all subcomponents (routers, network interfaces, and channels).



(a) Heterogeneous SoC (b) Homogeneous SoC

Figure 6: High-level floorplan of the two SoC case studies.

Symbol	Desc.	Values	Notes
F	Flits	8, 16, 32	flit width for all physical networks
N	Networks	1, 2, 5	number of physical networks
V	VCs	1, 2, 3, 5	number of virtual channels per physical network
P	Pipelines	2, 4	pipeline configurations for all routers in the network
Q	Queues	2, 4	queue size of all input units of all routers

Table 2: NoC configuration parameters.

Message Class	From → To	N-V Assignments					
		1N-5V		2N-2/3V		5N-1V	
		N	V	N	V	N	V
REQ	CPU → MEM	0	0	0	0	0	0
RES	MEM → CPU		1	1	0	1	
REQ	MEM → ACC		2	0	1	2	
MISC	–		3	1	1	3	
RES	ACC → MEM		4		2	4	

Table 3: Message classes and their N-V assignments

4 EXPERIMENTAL RESULTS

To demonstrate the capabilities of the ICON framework in exploring the NoC design space for a target SoC, we designed two complete SoCs as instances of Embedded Scalable Platforms [4]. As shown in Fig. 6, each SoC contains a Leon3 CPU running Linux and 2 DDR-3 DRAM controllers together with a set of accelerators: 10 accelerators for 5 distinct application kernels from the Perfect benchmark suite [2] in the *heterogeneous SoC* and 12 copies of the FFT-2D accelerator in the *homogeneous* SoC.

For each SoC, we used ICON to generate 36 different NoC designs by combining the 5 parameters of Table 2. While every combination of parameter values is supported, we limit ourselves to three possible combinations for the number N of physical networks and the number V of virtual channels. Table 3 reports how these three configurations support the five distinct message classes that are needed to enable the various independent transactions in the SoC while avoiding protocol deadlock [20]: two for CPU-memory transfers, two for accelerator-memory transfers and one for accelerator configuration and interrupt requests. Note that ICON allows us to use different numbers of VCs per physical network, *e.g.* 2 for the network 0 and 3 for network 1 with 2N-2/3V. All NoC configurations share a 4×4 2D-mesh network topology with XY dimension-order routing and credit-based flow control.

Each of the 36 NoC designs given in SystemC was synthesized into a corresponding Verilog design by using Cadence C-to-Silicon. Then, we used two distinct back-end flows, one for ASIC and another for FPGA, to obtain final implementations for each NoC.

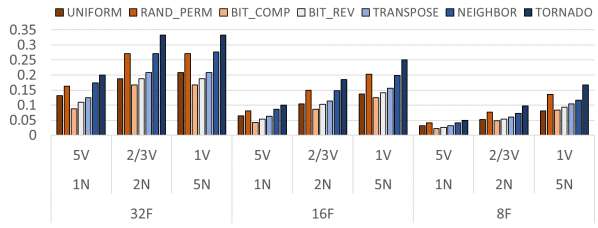


Figure 7: Saturation throughput of NoCs ($P = 2, Q = 2$).

Experiments with ASIC Design Flow. We performed logic synthesis targeting a $45nm$ technology and 500Mhz clock frequency. We simulated the ASIC implementations using the Makefiles and testbenches generated by ICON for the seven "classic" synthetic traffic patterns: Uniform, Random Permutation, Bit Complement, Bit Reverse, Transpose, Neighbor, and Tornado [6]. Fig. 7 reports the results in terms of saturation throughput for all configurations with $P = 2$ and $Q = 2$. Across all traffic patterns the throughput changes considerably depending on the flit width. For the same flit width, 5N-1V, which has a bisection bandwidth that is five times bigger than 1N-5V, provides the highest throughput. The saturation throughput is higher for the simulations with the Random Permutation, Neighbor, and Tornado patterns than in the other cases because on average the destination of the generated traffic is closer to the source. Fig. 8 shows the area-performance trade-off of the NoC configurations for different flit-width values.

Experiments with FPGA Designs. We combined the generated NoC Verilog designs with those for the two SoCs of Fig. 6 and performed logic synthesis for a Xilinx Virtex-7 XC7V2000T FPGA with two DDR-3 extension boards for a target frequency of 80MHz.

For each SoC we run a multi-threaded application that uses Linux to invoke all accelerators (via their device drivers) so that they run simultaneously and, therefore, compete for access to the NoC and DDR-3 controllers. Fig. 9 reports the execution time of the application (normalized with respect to the simplest configuration) and the SoC area occupation for many different NoC configurations. Specifically, it shows the impact of varying the flit width (F) in a NoC with 1 physical network (N=1), 5 virtual channels (V=5), a 4-stage pipeline (P=4) and 2 different queue sizes (Q={2,4}). When raising F from 8 to 16, the application for the heterogeneous SoC takes a time that is 86.55% (for Q=2) and 87.57% (for Q=4) of the case for F=8 in exchange for modest area increases (3.1% and 4.3%, respectively). The execution time of the corresponding application on the homogeneous SoC becomes 78.24% and 78.98% of the case with F=8 (with 4.11% and 5.55% of area increase, respectively). While the performance improvement obtained by doubling the flit width from 8 bits to 16 bits is considerable, this is not the case when doubling it again from 16 to 32 bits. For both the F=16 and F=32 configurations, the NoCs are not saturated and the zero-load latency has a bigger impact than the contention latency. The main reason is the long communication delay on the off-chip channels between the DDR-3 controllers and DRAM. The average throughput on this channel is about 2.72 bits per clock cycle for both the F=16 and F=32 configurations while it decreases to 2.48 for the F=8 configuration when the on-chip links become more congested and the NoC becomes the system bottleneck.

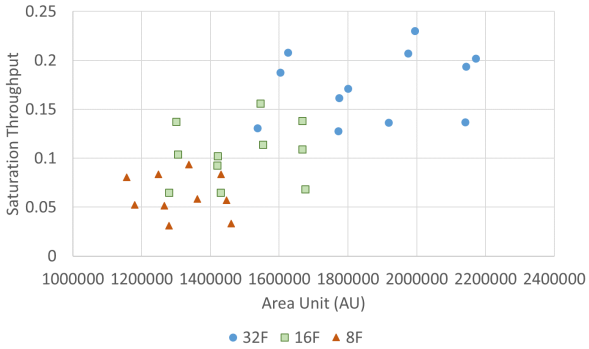


Figure 8: ASIC experiments: area/performance trade-offs.

Fig 10 reports the normalized execution time and area comparisons for the 3 different combinations of numbers of physical networks and virtual channels (N=5 and V=1, N=2 and V=2/3, N=1 and V=5) specified in Table 3. Overall, the first configuration is better from an area viewpoint, while the differences in performance are minimal. Fig. 11 summarizes the area and performance trade-offs across all the configurations from the previous two figures as well as the rest of the 36 configurations that we tested for this SoC case study. For the heterogeneous SoC, the Pareto curve includes 4 NoC configurations: `8F-5N-1V-2P-2Q`, `16F-5N-1V-4P-2Q`, `16F-5N-1V-2P-2Q`, and `16F-2N-2/3V-4P-2Q`. For the homogeneous SoC, the Pareto curve consists of 3 configurations: `8F-5N-1V-2P-2Q`, `16F-1N-5V-4P-2Q`, and `16F-2N-2/3V-2P-2Q`. This set of results shows how ICON can be used to quickly generate and evaluate several network design points. Each design can be seamlessly integrated into a complex heterogeneous SoC without modifying any of the computing IP blocks present in the system. Further, ICON allows us to identify the configuration parameters that have a larger impact on performance for the specific target SoC. Exploring such a large design space and gathering accurate information from a full-system evaluation would not have been possible without the ICON automation framework.

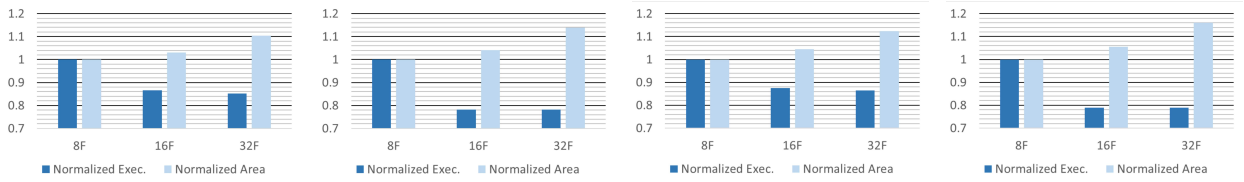
5 CONCLUSIONS

We presented ICON, a complete system-level design framework for the specification, synthesis and design-space exploration of NoCs for heterogeneous SoCs. We demonstrated ICON capabilities with a variety of experiments including the complete full-system designs of two SoCs on FPGAs. Future work includes extending ICON to support industry standards (e.g. AMBA-AXI) and open-source protocols (OCP) and to augment its testbench library with statistical NoC models like those proposed by Soteriou *et al.* [21].

Acknowledgements. This work was supported in part by DARPA (C#: R0011-13-C-0003), the National Science Foundation (A#: 1219001), and C-FAR (C#: 2013-MA-2384), an SRC STARnet center.

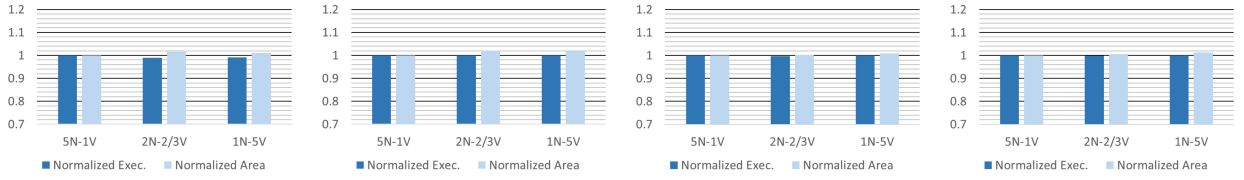
REFERENCES

[1] J. Bachrach et al. 2012. Chisel: Constructing hardware in a Scala Embedded Language. In *Design Automation Conference (DAC)*. 1212–1221.
[2] K. Barker et al. 2013. *PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual.* Pacific Northwest National Laboratory and Georgia Tech Research Institute. http://hpc.pnnl.gov/projects/PERFECT/.
[3] D. Becker. 2012. *Efficient Microarchitecture for Network-on-Chip Routers.* Ph.D. Dissertation. Stanford University.



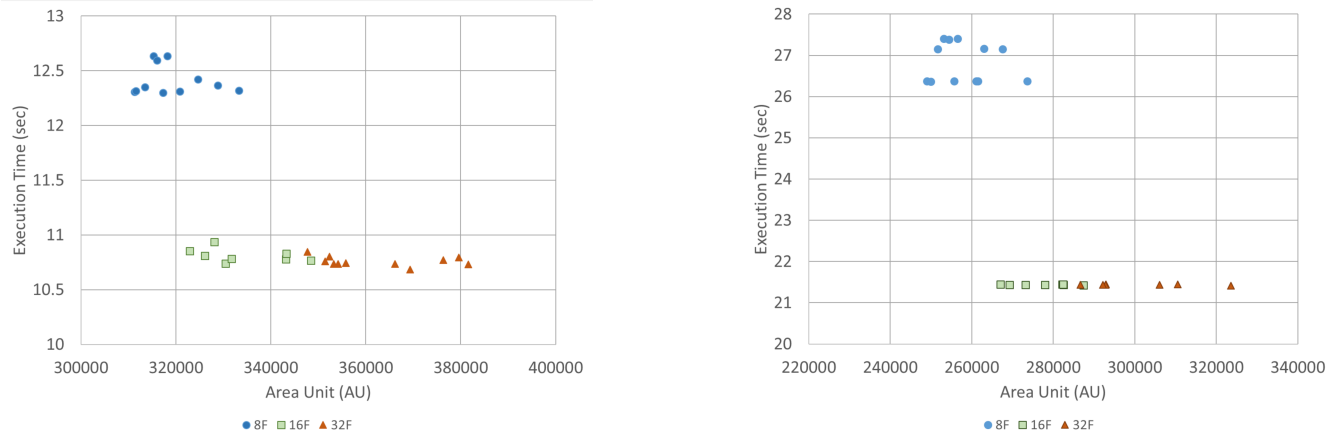
(a) Heterogeneous 1N-5V-4P-2Q. (b) Homogeneous 1N-5V-4P-2Q. (c) Heterogeneous 1N-5V-4P-4Q (d) Homogeneous 1N-5V-4P-4Q

Figure 9: Normalized execution time and area comparison as function of the flit width (8/16/32 bits).



(a) Heterogeneous 8F-4P-2Q. (b) Homogeneous 8F-4P-2Q. (c) Heterogeneous 32F-4P-2Q. (d) Homogeneous 32F-4P-2Q.

Figure 10: Execution time and area comparison of 1N-5V, 2N-2/3V, and 5N-1V NoCs configurations (with $P = 4$ and $Q = 2$).



(a) Heterogeneous SoC (b) Homogeneous SoC

Figure 11: FPGA experiments: area/performance trade-offs.

[4] L.P. Carloni. 2016. The Case for Embedded Scalable Platforms. In *Design Automation Conference (DAC)*. 17:1–17:6.

[5] M. Dall'Osso et al. 2003. ×pipes: A Latency Insensitive Parameterized Network-on-Chip Architecture for Multi-Processor SoCs. 45–48.

[6] W. J. Dally and B. Towles. 2004. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann.

[7] F. Fatollahi-Fard et al. 2016. OpenSoC Fabric: On-chip network generator. In *Intl. Symp. on Perf. Analysis of Systems and Software (ISPASS)*. 194–203.

[8] C. Hilton and B. Nelson. 2006. PNoC: a Flexible Circuit-Switched NoC for FPGA-based systems. *IEE Proc. - Computers and Digital Techniques* 153, 3 (2006), 181–188.

[9] T. C. Huang et al. 2007. Virtual Channels Planning for Networks-on-Chip. In *Intl. Symp. on Quality Electronic Design (ISQED)*. 879–884.

[10] N. Kapre et al. 2006. Packet Switched vs. Time Multiplexed FPGA Overlay Networks. In *IEEE Symp. on Field-Programmable Custom Computing Machines*. 205–216.

[11] A. Kumar et al. 2007. A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS. In *Intl. Conf. on Computer Design (ICCD)*.

[12] J. Lee and L. Shannon. 2009. The Effect of Node Size, Heterogeneity, and Network Size on FPGA-based NoCs. In *Intl. Conf. on Field-Programmable Tech.* 479–482.

[13] Y. Lu et al. 2011. Generic Low-Latency NoC Router Architecture for FPGA Computing Systems. In *Intl. Conf. on Field Programmable Logic and Applications*. 82–89.

[14] R. Mullins et al. 2004. Low-Latency Virtual-Channel Routers for On-Chip Networks. In *Intl. Symp. on Computer architecture (ISCA)*. 188–107.

[15] R. Nikhil. 2004. Bluespec System Verilog: Efficient, Correct RTL from High-Level Specifications. In *Intl. Conf. on Formal Methods and Models for Co-Design*. 69–70.

[16] J. Öberg and F. Robino. 2011. A NoC System Generator for the Sea-of-Cores Era. In *Proc. of the 8th FPGA World Conference*. 4:1–4:6.

[17] M. Papamichael and J. Hoe. 2012. CONNECT: Re-Examining Conventional Wisdom for Designing NoCs in the Context of FPGAs. In *Intl. Symp. on Field Programmable Gate Arrays*. 37–46.

[18] G. Schelle and D. Grunwald. 2008. Exploring FPGA Network on Chip Implementations across Various Application and Network Loads. In *Intl. Conf. on Field Programmable Logic and Applications*. 41–46.

[19] M. Shelburne et al. 2008. MetaWire: Using FPGA Configuration Circuitry to Emulate a Network-on-Chip. In *Intl. Conf. on Field Programmable Logic and Applications*. 257–262.

[20] Y. H. Song and T. M. Pinkston. 2003. A Progressive Approach to Handling Message-Dependent Deadlock in Parallel Computer Systems. *IEEE Trans. on Parallel and Distributed Systems* 14, 3 (2003), 259–275.

[21] V. Soteriou et al. 2006. A Statistical Traffic Model for On-Chip Interconnection Networks. In *Intl. Symp. on Modeling, Analysis, and Simulation*. 104–116.

[22] S. Stergiou et al. 2005. ×pipes Lite: A Synthesis Oriented Design Library for Networks on Chip. In *Conf. on Design, Automation and Test in Europe (DATE)*. 1188–1193.

[23] Y. Yoon et al. 2010. Virtual Channels vs. Multiple Physical Networks: a Comparative Analysis. In *Design Automation Conference (DAC)*. 162–165.