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

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s). ACM Reference Format:

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.

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 NoC 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

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

Besides 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
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
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 sub-components (routers, network interfaces, and channels).

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.
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 bit width. For the same bit 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 bit-width values.

**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 bit width. For the same bit 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 bit-width values.

We presented ICON, a complete system-level design framework for heterogeneous Systems-on-Chip. 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**


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

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

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