Routing Aware Switch Hardware Customization for Networks on Chips

Paolo Meloni[§], Srinivasan Murali[⋆], Salvatore Carta[¶], Massimo Camplani[§], Luigi Raffo[§], Giovanni De Micheli[†]
[§]DIEE/University of Cagliari, Cagliari, Italy, {paolo.meloni@diee.unica.it, luigi@diee.unica.it}
[⋆]CSL/Stanford University, Stanford, USA, smurali@stanford.edu
[¶]DMI/University of Cagliari, Cagliari, Italy, salvatore@unica.it
[†] LSI/EPFL, Lausanne, Switzerland, giovanni.demicheli@epfl.ch

***Abstract*— Networks on Chip (NoC) has been proposed as a scalable and reusable solution for interconnecting the ever-growing number of processor/memory cores on a single silicon die. As the hardware complexity of a NoC is significant, methods for designing a NoC with low hardware overhead, matching the application requirements are essential. In this work, we present a method for reducing the hardware complexity of the NoC by automatically configuring the architecture of the NoC switches to suit the application traffic characteristics. The crossbar matrix and the arbiters of each switch in the NoC design are customized to support the traffic flows utilizing that switch. This application-specific switch customization is integrated with an existing design flow, which automates NoC topology synthesis, mapping, RTL code and physical layout generation. Several experimental studies on NoC benchmark designs are carried out, which show that the proposed switch customization technique leads to large reduction in the NoC switch area (28% on average) and power consumption (21% on average). Moreover, the critical paths of the switches reduce significantly, thereby leading to a significant speed-up of the NoC design.**

I. Introduction

To interconnect the ever-growing number of processors, memories and hardware devices that are integrated onto the chip, scalable *Networks on Chips* are required [1]- [3]. Today's communication architectures are based on single or multiple layers of buses and cannot sustain the increasing communication requirements of such complex *Systems on Chips (SoCs)*. The use of a NoC results in a scalable, modular and efficient interconnection architecture for the SoC.

The SoCs are usually application-specific, with the application tasks statically mapped onto the various processor/hardware cores in the design. Hence the communication traffic characteristics of the SoC can be obtained statically and the interconnect can be tailor-made to suit the application traffic pattern. This is true from SoC designs that are small to the state-of-the art SoCs, such as the Philips Nexperia platform [4], ST Nomadik [5]. For NoCs to be feasible in today's SoC designs, a NoC architecture with low hardware overhead is required. The NoC architecture typically has a large area overhead when compared to the current bus-based systems. Designing methods that reduce the NoC hardware overhead, while satisfying the application characteristics of SoCs comprise an important research area in the NoC domain.

A typical NoC switch consists of input ports, arbiters, crossbar matrix, output ports and buffers (present at the input/output ports). In existing NoC designs, the internal architecture of all the switches in the NoC are uniform, with all the input ports of a switch connected to all its output ports, through the crossbar matrix and arbiters. Such an architecture is needed when the packet routes cannot be determined at design time, so that during run-time, data from any input port can be sent to any output port of the switch. However, in most NoC designs deterministic routing is employed, where the routes for the packets of the various traffic flows are obtained at design time and the route selection is usually performed during the NoC topology synthesis phase [20]. Thus, the general architecture leads to an over-design of the NoC switches, resulting in large area-power overhead for the NoC.

In this work, we present a method for reducing the hardware complexity of the NoC by automatically configuring the architecture of the NoC switches to match the designed routes and the given application traffic pattern. While there are several research works that have addressed several aspects of the application-specific NoC design process (refer Section II for details), to the best of our knowledge this is the first work that presents a method for automatically customizing the architecture of the switches in the NoC design.

In the proposed customization method, we tune the crossbar matrix and arbiters of each switch in the NoC. That is, based on the designed routes, our method automatically prunes the set of input-to-output connections in the crossbar matrix and arbiters that are not utilized in each switch of the NoC. We integrate the proposed routing-aware switch architecture customization method with an existing design flow, which automates NoC topology synthesis, mapping, RTL code and physical layout generation.

Several experimental studies on SoC benchmark designs are carried out, which show that the proposed switch customization technique leads to large reduction in NoC switch area (28% on average) and power consumption (21% on average).

II. Previous Work

Several different architectures and tool flows have been presented for designing NoCs [6]- [13]. The design of the switch architecture has been investigated in [14]- [18]. An

1-4244-0391-X/06/$20.00 ©2006 IEEE

analysis of power consumption of different switch fabrics in network routers has been presented in [19]. However, a method to customize the different switches is not presented in the work.

The problem of application-specific NoC topology and route generation has received considerable attention [20]- [26]. In [20]- [23], mapping applications onto regular NoC topologies have been presented. In [24]- [26], methods for designing irregular (customized) application-specific NoC topologies have been presented. An approach for application-specific long range link insertion has been presented in [27] and a stochastic approach for sizing the switch buffers has been presented in [28].

However, none of these earlier works have addressed the issue of switch crossbar matrix and arbiter customization for NoCs. Our approach is complementary to most of these existing works on topology design, link insertion and buffer sizing and can be used in conjunction with them.

III. Reference design flow

The customization method proposed in this work is integrated with an existing NoC design flow (shown in Figure 1) that guides the SoC designer to achieve the most power/performance efficient NoC architecture, starting from the application characteristics. More in detail, the flow is composed of three major design steps: NoC topology synthesis, NoC instantiation and back-end implementation.

- **Step 1: Topology synthesis:**
  In the first step, *SUNFLOOR* [29], a custom CAD tool is used to synthesize the most power/performance efficient NoC topology that satisfies the application requirements. The application traffic characteristics, size of the cores, and the area and power models for the network components are obtained as inputs to the synthesis engine. The tool generates different NoC switches and maps the cores onto the switches. During the synthesis process, it determines deadlock-free routes for the different traffic flows of the application. The tool also produces the 2D floorplan of the synthesized NoC topology. The output produced by the tool is a text file that defines the synthesized topology, which is fed to the next step.
- **Step 2: Topology instantiation:**
  In the second step, another custom tool $\times$*pipesCompiler* [30], reads the topology definition file generated by SUNFLOOR and instantiates the RTL description of the NoC components using $\times$*pipes* [9], a pre-designed SystemC RTL component library. The modules in the component library support a large number of instantiation parameters, such as different input/output ports, buffer sizes, etc. The tool also interconnects the RTL description of the processor/memory cores of the SoC (which are pre-designed components, taken as user inputs) with the RTL code of the network components. The output of this step is the RTL design of the entire NoC that can be simulated and synthesized.
- **Step 3: Back-end implementation:**
  This step includes the utilization of a commercial tool-chain to implement the designed topology at the layout level. The RTL description of the interconnect from the previous step is synthesized and the placement&routing of the design is performed using the commercial tool, Cadence SoC Encounter [31]. The post-layout design is then verified for functional correctness and is used for obtaining accurate performance estimations of the design.

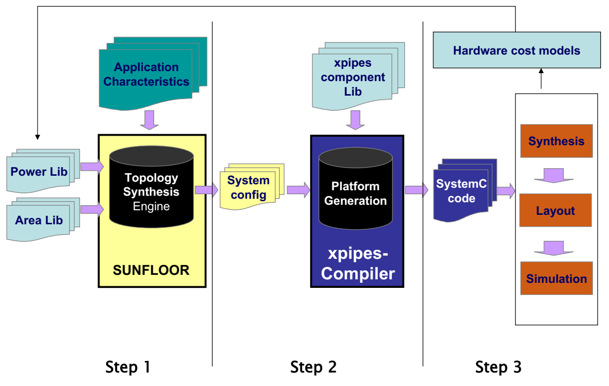


Fig. 1. Design flow overview

IV. Routing aware hardware optimization

The objective of the proposed switch customization method is to automatically remove those resources that are not utilized in the NoC from the interconnect hardware, based on the designed topology and the paths that are selected for routing the different traffic flows. The base-line switch architecture of the $\times$pipes library, before applying the switch customization, is composed of four main blocks:

- **Input ports:** At each input port, the incoming flit and control signals are latched onto registers, so that the critical path of the switch is reduced.
- **Crossbar:** The switch crossbar includes several fully connected sets of multiplexers, one per output port, which connect all the input ports to each output port. The control signals that are required for data selection for each multiplexer are generated by the arbiters of the corresponding output port.
- **Arbiters:** Each output port of the switch has an arbiter that grants access to one of the input ports requesting the corresponding output port, based on some arbitration policy.
- **Output ports:** In the $\times$pipes architecture, output buffering is employed, where the flit buffers are present at the output ports of the switch. The output ports also accommodate the combinational logic needed to handle the flow-control operations.

We use a procedure that analyzes the designed topology and routing paths, and evaluates what input ports actually communicate packets to the different output ports of each switch in the design. Then, we remove those set of input-to-output connections from the crossbar matrix and arbiters that are not used by the chosen routing paths.

TABLE I
SWITCH ROUTING TABLE EXAMPLE

	input 0	*input 1*	*input 2*	*input 3*
output port 0	x			
output port 1		x		
output port 2			x	x
output port 3	x		x	x

Example 1: As an example, let us consider the set of input-to-output connections that are required at a particular switch (a 4×4 switch) of a NoC (refer Table I), which are obtained from the routing paths established by the topology synthesis procedure. In the table, the presence of a cross signifies that the input-to-output connection is utilized by the routing paths. In Figure 2(a), we present a traditional architecture for this switch, where all the input ports are connected to all the output ports of the switch. In Figure 2(b), we present the switch architecture obtained by the proposed method, where the crossbar matrix and arbiters are customized to match the required input-to-output connections of the designed routes. The switch customization for this example, leads to 56.25% reduction in the input-to-output connections of the switch.

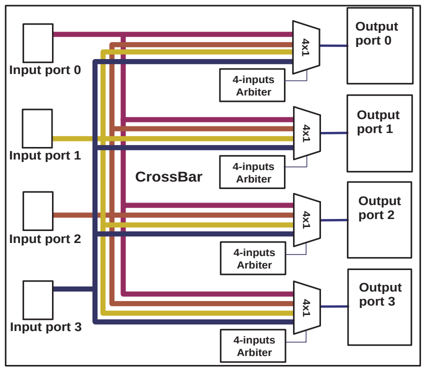
To achieve this switch customization, we operated at two points of the design flow, i.e. at the component library level (*hardware level*) and at the RTL code generation level (*software level*). These levels are explained in detail in the following sub-sections:

A. Hardware-Level Customization Support

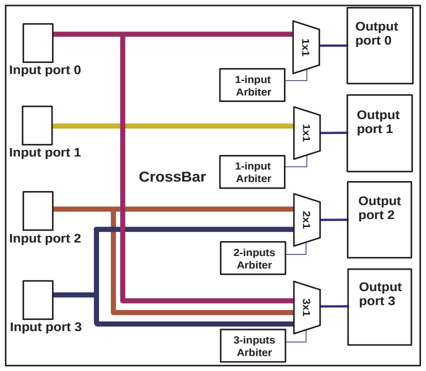
We added a layer to the configuration of the switch building blocks in the $\times$pipes SystemC RTL macros, specifying a set of parameters for each instance of the multiplexer and the arbiter. More in detail, before the enhancement, the multiplexers were configured automatically, connecting all the input ports to each output port. After the customization enhancement, each multiplexer module is also configured in terms of the number of input ports that need to be connected to each output port. In the same way, each arbiter is configured for the number of ports for which it has to arbitrate for. While the individual multiplexer and arbiter modules are parameterized in this fashion, a tool chain is needed to instantiate the different modules for the designed NoC and to interconnect the sub-blocks with the switch top module. To achieve this, we extend the $\times$pipesCompiler tool at the software level to customize, instantiate and interconnect the different modules together.

B. Software-Level Customization Support

We defined a software layer that is integrated with the $\times$pipesCompiler tool in the design flow, that analyzes the routing paths and finds the unneeded hardware in the switches. The software thus:



(a) Traditional design



(b) Customized design

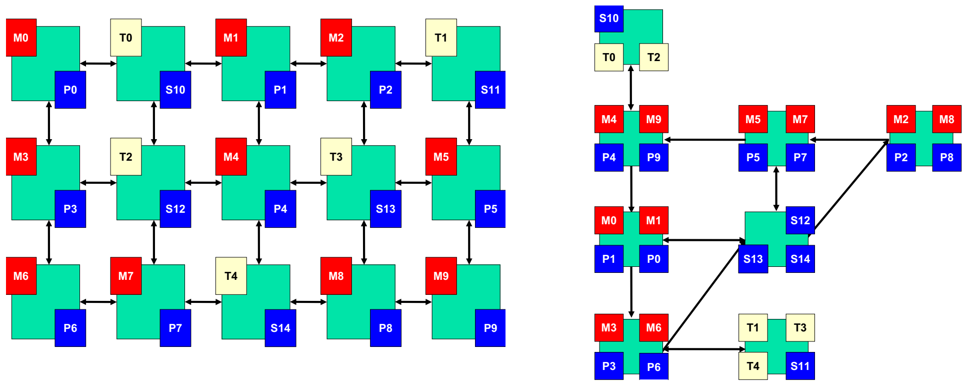
Fig. 2. Switch architecture before and after the routing aware customization

- parses the topology definition file from SUNFLOOR, extracting the information about every single routing path,
- for each arbiter and multiplexer, defines a set of parameters that summarize how many and which input ports require access to the related output port,
- for all the switches, generates a top module that instantiates all the sub-blocks and defines the connectivity between them according to the routing needs.

The output of this switch customization enhanced NoC instantiation tool is the RTL description of the customized NoC design that can be simulated and synthesized

V. EXPERIMENTAL RESULTS

We apply our switch customization method on the NoCs designed for several SoC applications: *Multimedia design (MULT-*30 *cores), IMage Processing design 1 (IMP1-*25 *cores), IMage Processing design-2 (IMP2-*21 *cores), FFT based SoC (FFT-*29 *cores), Data Processing SoC (DP-*15 *cores) and SoC implementing a DES encryption system (DES-*19 *cores)*. We explain the details of one of the applications (MULT, the largest of the benchmarks) and the customization impact on the different designs in the next sub-sections.



(a) Mesh topology (b) Custom topology

Fig. 3. Mesh and application-specific custom topologies for the MULT benchmark. The P0-P9 are the processors, T0-T4 are the hardware cores, M0-M9 are the private memories and S10-S14 are the shared devices. The shaded boxes connecting the cores are the switches in the design.

TABLE II

TOTAL SWITCH AREA OF THE DESIGNS

Topology	*Total area base-line* (mm^2)	*Total area customized* (mm^2)	*Area reduction* (%)	*# I-to-O links reduction* (%)
5×3 Mesh	0.73	0.51	30.14	69.63
Custom	0.45	0.31	31.11	66.38

TABLE III

COMBINATIONAL AREA OF THE SWITCHES FOR THE DESIGNS

Topology	*Comb. area base-line* (mm^2)	*Comb. area customized* (mm^2)	*Comb. area reduction* (%)
5 × 3 Mesh	0.32	0.158	50.63
Custom	0.22	0.09	59.09

TABLE IV

SWITCH POWER CONSUMPTION FOR THE DESIGNS

Topology	*Power base-line* (mW)	*Power customized* (mW)	*Power reduction* (%)
5×3 Mesh	66.2	29.4	55.6
Custom	36.3	28.6	21.2

A. *Experiments on the Multimedia benchmark*

In this subsection, we present the designs obtained by our approach for two different topologies (a regular and a custom NoC topology) that are used to interconnect the cores of the MULT SoC benchmark. We use such different topologies to show the generality of the proposed customization method. The MULT SoC consists of fifteen processor/hardware cores, ten private memories for the different processors and five shared slave devices. The regular topology is a 5×3 mesh, which is hand-designed to suit the application characteristics of the benchmark (presented in Figure 3(a)) [9]. The second topology is an application-specific custom topology obtained from the SUNFLOOR tool (presented in Figure 3(b)). To achieve the optimum network throughput, 3 flit buffers are utilized at each output port [9]. The total area and the area of the combinational blocks of the switches for the two topologies, for the base-line design and for the design where the proposed switch customization technique is applied are shown in Tables II and III. The area numbers are obtained from synthesizing the RTL code of the different switches in the design. As seen from the table, the use of the switch customization technique leads to a large reduction (an average of 30.63%) in the total switch area of the design. As the proposed procedure optimizes the switch crossbar and multiplexers, which are predominantly combinational blocks, a large savings in the combinational area of the switches is obtained, which leads to the significant total switch area savings.

The power consumption of the different architectures are shown in Table IV. The power numbers are based on the switching activities of the components, which are obtained from functional simulations. For the power consumption estimations, we also consider the accurate switching resistance and capacitance values of the components that are obtained from the post-synthesis net-lists of the NoC designs. The synthesis experiments were performed using Synopsys Design Compiler [32], with 0.13μ technology library, an operating frequency of 500 MHz and an operating voltage of 1.2 V. We also utilize clock-gating in the architectures, so that the elements that are not heavily utilized have a lower power consumption value. The proposed customization technique also leads to a large reduction in the power consumption of the switches (an average of 38.4%) for both the topologies.

B. *Experiments on SoC benchmarks*

We synthesized the best topologies for the different SoC designs using SUNFLOOR. The area, power and input-to-output connection savings for the different designs for the customized architecture, when compared to the base-line architecture are shown in Figure 4. For all the designs, the switch customization technique leads to significant reduction in the switch area (28% on average) and power consumption (21% on average) values.

Finally, as the proposed customization technique leads to a reduction in the crossbar and arbiter complexity, the critical

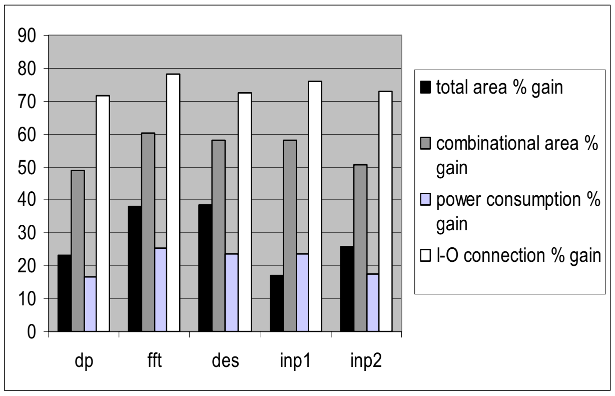


Fig. 4. Switch area, power and input-to-output connection savings for the SoC designs

path of the switches also reduces significantly. This is because, the critical path of the switch in the $\times$pipes architecture is the path from the input ports to the output ports, that traverses the arbiters and the crossbar multiplexers. As an example, for a 4×4 base-line switch architecture, the critical path is around 1 ns, while for the customized 4×4 switch architecture, on average, the critical path is around 0.57 ns. These timing values are obtained from RTL synthesis of the switches. Thus, the customization technique also leads to a significant speed-up (of 75%) of the NoC.

VI. Conclusions

Networks on Chips (NoC) based interconnects are required to meet the large communication complexity of current and future *Systems on Chips (SoCs)*. For NoCs to be feasible in today's SoC designs, a NoC architecture with low hardware overhead is required. In this paper, we have proposed a method for reducing the NoC hardware overhead by automatically customizing the architecture of the switches of the NoC to match the designed topology and routing paths. The customization process is integrated with an existing tool chain, thereby automating NoC topology synthesis, customization, RTL code generation and physical design processes. The customization process leads to large reduction in network area and power consumption. Moreover, the critical paths of the switches reduce significantly, thereby leading to a significant speed-up of the NoC design. In future, we plan to extend the customization method for sizing the buffers of the NoC switches as well.

VII. Acknowledgments

This research is supported by the Semiconductor Research Corporation (SRC project number 1188), a grant by the US National Science Foundation (NSF, under contract CCR-0305718) and a grant by the Fonds National Suisse (FNS, Grant 20021-109450/1).

References

[1] L.Benini, G.De Micheli, “Networks on Chips: A New SoC Paradigm”, IEEE Computers, pp. 70-78, Jan. 2002.
[2] D.Wingard, “MicroNetwork-Based Integration for SoCs”, Proc. DAC, pp. 673-677, Jun 2001.
[3] M. Sgroi et al., “Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design”, Proc. DAC, pp. 667-672, June 2001.
[4] S. Dutta, R. Jensen, A. Rieckmann, “Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems”, IEEE Design and Test of Computers, pp. 21-31, Vol. 18, No. 5, Sep/Oct 2001.
[5] A. Artieri, V. D Alto, R. Chesson, M. Hopkins, M. C. Rossi, “Nomadik Open Multimedia Platform for Next-generation Mobile Devices”, STMicroelectronics Technical Article TA305, 2003, available at http://www.st.com
[6] S. Kumar et al., “A Network on Chip Architecture and Design Methodology”, Proc. ISVLSI, pp. 117-122, April 2002
[7] P. Guerrier, A. Greiner, “A generic architecture for on-chip packet-switched interconnections”, Proc. DATE, pp. 250-256, March 2000.
[8] K. Goossens et al., “The Aethereal network on chip: Concepts, architectures, and implementations”, IEEE Design and Test of Computers, Vol. 22(5), pp. 21-31, Sept-Oct 2005.
[9] S. Stergiou et al., “xpipes Lite: A Synthesis Oriented Design Library For Networks on Chips”, Proc. DATE, pp. 1188-1193, March 2005.
[10] I. Saastamoinen et al., ”Proteo interconnect IPs for networks-on-chip”. In Proc. IP Based SoC Design, France, 2002.
[11] E. Bolotin et al., “QNoC: QoS architecture and design process for network on chip”, Journal of Systems Architecture, Feb. 2004.
[12] K. Goossens et al., ”A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification”, DATE 2005.
[13] D. Bertozzi et al., ”NoC Synthesis Flow for Customized Domain Specific Multi-Processor Systems-on-Chip”, IEEE Transactions on Parallel and Distributed Systems, Feb 2005.
[14] P. Pande et al., “Design of a Switch for Network on Chip Applications”, pp. 217-220, Porc. ISCAS 2003.
[15] C. Zeferino et al., “A Parameterizable Interconnect Switch for Networks-on-Chip”, pp. 204-209, Proc. SBCCI 2004.
[16] T. Bjerregaard, J. Spars, “A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chips”, Proc. DATE 2005.
[17] J. M. Kim et al., “A Low-Latency Router Supporting Adaptivity for On-Chip Interconnects”, pp. 559-564, Proc. DAC, June 2005.
[18] K. Lee et al., “A High-Speed and Lightweight On-Chip Crossbar Switch Scheduler for On-Chip Interconnection Networks”, ESSCIRC 2004.
[19] T. Ye et al., “Analysis of Power Consumption on Switch Fabrics in Network Routers”, Proc. DAC 2002.
[20] J. Hu, R. Marculescu, “Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures”, Proc. DATE, March 2003.
[21] S. Murali, G. De Micheli, “SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs”, Proc. DAC 2004.
[22] W. Hu et al., “Thermal Aware Placement for NoC Architectures”, pp. 430-437, Proc. ICCD 2004.
[23] A. Hansson et al., “A unified approach to constrained mapping and routing on network-on-chip architectures”, pp. 75-80, Proc. ISSS 2005.
[24] A.Pinto et al., “Efficient Synthesis of Networks on Chip”, ICCD 2003, pp. 146-150, Oct 2003.
[25] T. Ahonen et al. ”Topology Optimization for Application Specific Networks on Chip”, Proc. SLIP 04.
[26] K. Srinivasan et al., “An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks”, Proc. ICCAD ’05.
[27] U. Y. Ogras, R. Marculescu, “Application-Specific Network-on-Chip Architecture Customization via Long-Range Link Insertion”, Proc. ICCAD 2005.
[28] J. Hu, R. Marculescu, “Application Specific Buffer Space Allocation for Networks on Chip Router Design”, Proc. ICCAD 2004.
[29] S. Murali et al., “Designing Application-Specific Networks on Chips with Floorplan Information”.
[30] A. Jalabert et al., “$\times$pipesCompiler: A Tool for Instantiating Application-Specific Networks on Chips”, Proc. DATE 2004.
[31] www.cadence.com
[32] www.synopsys.com