Exploiting Multiple Switch Libraries in Topology Synthesis of On-Chip Interconnection Network

Minje Jun*, Sungroh Yoon†, and Eui-Young Chung*

*School of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea.
†School of Electrical Engineering, Korea University, Seoul, Korea.
*{jjuninho, eychung}@yonsei.ac.kr †sryoon@korea.ac.kr

***Abstract*—On-chip interconnection network is a crucial design component in high-performance System-on-Chips (SoCs). Many of previous works have focused on the automation of its topology design, since the topology largely determines its overall performance. For this purpose, they mostly require a switch library which includes all possible switch configurations (*e.g.* the number of in/output ports and data width) with their implementation costs such as delay, area, and power. More precisely, they characterize the switches by synthesizing them with a common design objective (*e.g.* minimizing area) and common design constraints for a given gate-level design library. The implementation costs are used in evaluating the topologies throughout the topology synthesis. The major drawback of single switch library approach is that it forces the topology synthesis methods to search the best topology with the assumption that all the switches comprising a topology will be implemented (synthesized) with a common design objective and common design constraints. Such assumption prevents them from exploring diverse combinations of the switches for a topology from the implementation perspective. To tackle this issue, we propose a topology synthesis method with multiple switch libraries, where the switch libraries are prepared with different design objectives and design constraints. The experimental results show that the power consumption and the area of optimal topologies can be saved by up to 67.1% and 27.2%, respectively, by the proposed method with negligible synthesis time overhead.**

I. Introduction

On-chip communication is becoming more and more crucial to the overall system performance due to the growing size of SoCs and the adoption of data-intensive applications, such as HD-videos and 3D games. The shared bus architecture and its variants, such as hierarchical buses with bridges, faced their limitation on supporting the large size and the complex communication requirements of contemporary SoCs.

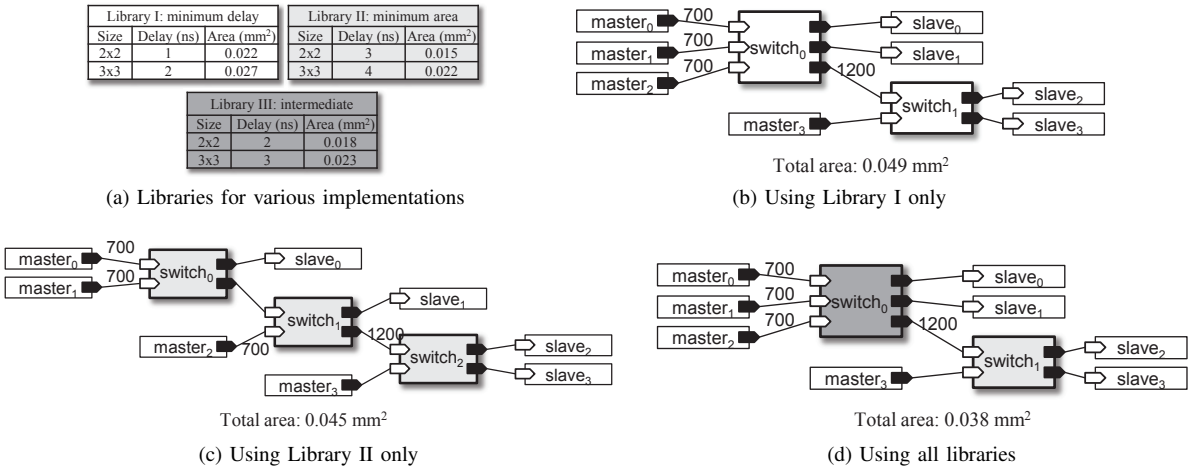
Bus matrix (also called crossbar) is nowadays one of the most popular solutions as a backbone communication resource to handle the large amount of on-chip traffics of high-performance SoCs. Its point-to-point nature enables multiple concurrent communications among the IPs, but it also accompanies large power and area overhead in addition to the speed limitation mainly due to the internal wires and arbiter logics. Several methods were proposed to minimize these drawbacks by clustering IPs and/or eliminating unnecessary connections in the crossbar [1]–[5]. However, they also showed a limitation on the scalability since the single crossbar needs to connect the ever increasing number of IP clusters, i.e. local networks.

To tackle the limitation, on-chip interconnection networks composed of multiple switching components, so called Network-on-Chip (NoC), were proposed. Also, many of NoC topology synthesis methods have been proposed [6]–[8], [10]–[17]. These methods require the preparation of a switch library which includes all possible switch configurations and their implementation costs such as delay, area, and power. More precisely, a switch configuration (i.e. a certain number of in/output ports and data width) is characterized for a set of metrics such as area, delay, and power consumption by synthesizing it for a given gate-level library. The information from the characterization (implementation cost) is used in evaluating the topologies during the topology synthesis. Generally, these methods perform the characterization of all the switches in the library with a common design objective and common design constraints. In other words, they assume that the switches comprising a topology will be implemented in the same way. For instance, the switch library in [6] consists of the switches which were optimized for minimizing delay. On the other hand, the switches used in [7] were commonly optimized for minimizing area with a minimum frequency constraint.

Such assumption may guide the topology synthesis to choose a sub-optimal topology by ignoring the combinations of the switches from the implementation perspective. In our work, we aim at tackling this problem by considering various switch implementations. An analogy can be found in logic synthesis. It requires a gate-level design library which includes several implementation choices for a single logic function (*e.g.* several 2-input NAND gates with different delays, areas, and powers). The technology mapping utilizes these gates to find the optimal implementation combination of gates for the minimized logic expressions. To the best of the authors' knowledge, this is the first work which tackles the diverse implementation possibilities of switches and applies them to the on-chip network topology synthesis.

The rest of this paper is organized as follows. We first summarize the previous works in Section II, and then present a motivational example of our work in Section III. In Section IV and Section V, we describe the details of the proposed method and show its effectiveness through the extensive experimental results, respectively. We conclude our work in Section VI.

978-3-9810801-6-2/DATE10 © 2010 EDAA



(a) Libraries for various implementations

(b) Using Library I only

(c) Using Library II only

(d) Using all libraries

Fig. 1: An NoC topology synthesis example w/ and w/o considering the implementation diversity.

II. Previous Works

Crossbar-based solutions have been proposed to overcome the performance limitation of on-chip shared bus architecture, by providing a dedicated bus to each slave. The works in [1]–[5] proposed the central crossbar minimization methods by clustering some masters and slaves into local subsystems and eliminating unnecessary paths inside the crossbar. However, these solutions are not free from the limitation of crossbar, since the growing system size will enlarge the central crossbar and its speed will become unacceptably slow. These methods are less sensitive to the problem raised in this work, since there is only a single crossbar.

To overcome the limitation, the cascaded crossbar network and the automatic synthesis methods were proposed. In this architecture, the central crossbar is replaced by multiple smaller ones which are connected to each other. In [10], [11], they assume that the network is composed of the switches of a single size, for example, all switches are 5-ported. The power consumption is modeled at each input/output port with linear regression with respect to the packet injection ratio. Even though a single switch configuration builds a topology, different implementations of switches may yield a better topology, hence these methods can be benefited by the proposed method.

In [6]–[8], they consider topologies which consist of various switch configurations. For this purpose, they prepare a switch library by synthesizing all possible switch configurations. Also, their implementation costs were annotated through static timing/power analysis. Similar switch libraries were also used in [12]–[17]. These methods did not consider the fact that a switch can be implemented in many different ways, *e.g.* fastest or most area/power-efficient ones. These methods can be largely improved by the proposed method, since there are more possible implementation combinations of switches compared to the aforementioned methods.

Unlike the existing methods, we characterize a switch for its various implementations and take advantage of it in the topology design. Our method is orthogonal to the topology synthesis method, i.e. it can be easily plugged into most of the existing topology synthesis algorithms.

III. Motivation

The on-chip network design is to integrate the IPs such that their communication requirements such as bandwidth and latency are supported by the network, while minimizing the target design cost such as delay, area, and power. Although difference exists to some extent, the existing iterative on-chip network topology synthesis methods usually consist of the following steps - 1) generate candidate topology, 2) evaluate its cost, 3) update the up-to-now solution if the current candidate is the best, and 4) iterate until there is no candidate left. In this procedure, the topology is evaluated with the pre-characterized implementation costs of the switches. In the existing methods, a switch of a certain configuration is characterized only for a single possible implementation, *e.g.* a 5x5 switch has delay and area values of 5ns and 0.09mm^2, respectively. However, a switch can be implemented in various ways. For this reason, characterizing a switch with only a single set of metrics may lead the topology synthesis to finish with the sub-optimal topology.

Fig. 1 shows an example of the topology synthesis with the single switch library and multiple switch libraries. In Fig. 1, the 'master' nodes are the initiators of the communications (*e.g.* CPUs), while the 'slave' nodes are the targets of the communications (*e.g.* memories). The edge represents the physical link between the two nodes and the numbers annotated on the edges are the required bandwidths between the connected nodes in MB/s. Suppose that we are to optimize the network for a system so that the area consumed by the switches is minimized. Assuming that the link having the bandwidth of 1200MB/s is the most heavily loaded one in the network and the link is 32bit-wide, the required frequency is 300Mhz. Fig. 1a shows the libraries which are characterized for the different implementation objectives. Library I has the implementation costs of the switches when they are optimized for minimizing the delay (for short, fastest implementations and denoted as 'Fastest'). The switches in Library II are optimized

for minimizing the area (for short, smallest implementations and denoted as 'Smallest'). Similarly, the switches in Library III are optimized for an intermediate point of the other two libraries. Fig. 1b and 1c show the synthesized topologies using only Library I and Library II, respectively. In Fig. 1b ('Fastest'), both 2x2 and 3x3 switches satisfy the required frequency and thus the network can be constructed with one 3x3 switch and one 2x2 switch, yielding the area of 0.049mm^2.

If only the smallest implementations are considered, a 3x3 switch cannot satisfy the required frequency since its delay is 4ns, so the network must use three 2x2 switches to satisfy the frequency requirement as shown in Fig. 1c. Although the network in Fig. 1c uses more switches than that in Fig. 1b, the total area is smaller, since 1) it uses only 2x2 switches and 2) its area in Library II is smaller than that in Library I.

These two cases represent the conventional topology synthesis with a single switch library. They also show that a switch library largely affects the quality of the topology. The comparison of Fig. 1b and 1c may be intuitive, since the 'Smallest' library yields a better topology than the 'Fastest' library from the area perspective. Also, people may believe that Fig. 1c would be the best topology from the area perspective, since it is from the 'Smallest' library. However, the example in Fig. 1d outperforms the topology shown in Fig. 1c thanks to the topology synthesis with multiple libraries. In Fig. 1d, all three libraries are considered for topology synthesis. A 3x3 switch and a 2x2 switch are picked from Library III and Library II to create a minimum area topology, yielding the area of 0.038mm^2. All single switch library including Library III cannot beat the result of multiple libraries in this example.

The single library approach becomes more complicated when the design objective is power, since the power minimization of a switch should be accompanied with its operating frequency (or delay) constraint. However, the target operating frequency of a switch cannot be intuitively decided since the network frequency is available only after the topology is determined. On the other hand, the power cost of a topology only can be evaluated from the power cost of switches, causing a cyclic dependency between the switch library and the topology synthesis. In other words, it is hard to prepare a switch library optimized for power unlike the switch library for area. The proposed method resolves this issue by considering multiple libraries which are dynamically accessed during the evaluation phase of topology candidates. The details will be presented in the following section.

IV. Proposed Method

A. Overview

This paper concerns the problem of the topology synthesis for application-specific NoCs, as in [6]–[8], [10]. Fig. 2 shows a typical flow for NoC topology synthesis. Given the communication requirements and the placement of IPs, this synthesis flow iteratively generates topology candidates and evaluates them in terms of some specific objective(s) such as area and/or power consumption. The flow continues until no more candidate is left or a given termination condition is met.

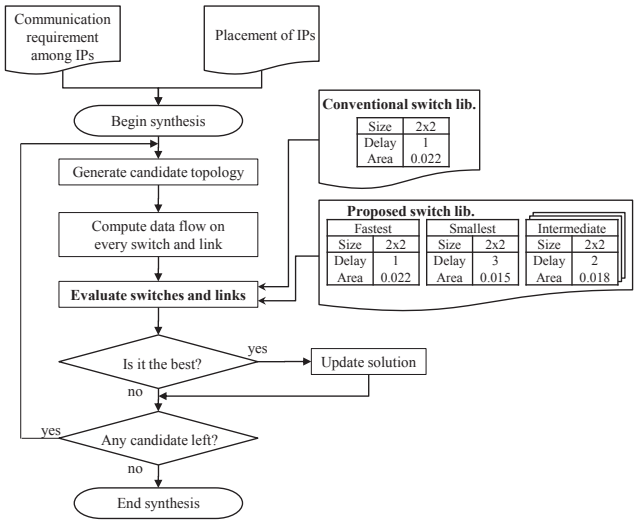


Fig. 2: A generic NoC synthesis flow

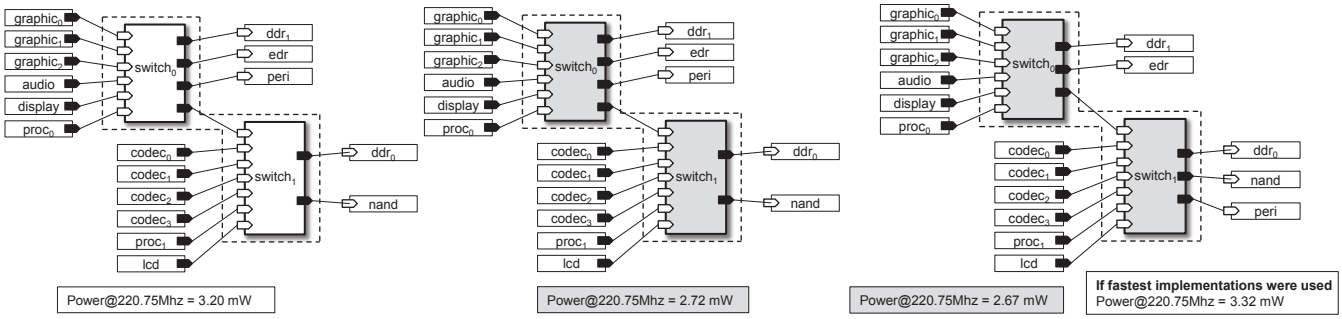
The focus of this paper is on how to evaluate topology candidates, rather than on specific algorithms to generate them. In particular, we are interested in evaluating topologies of switches, not links. As indicated in Fig. 2, the conventional approaches evaluate switches by using a simple library, in which each switch is characterized only once for a single implementation. In contrast, the proposed method employs a more sophisticated library, in which a switch is characterized multiple times for various implementations. By utilizing this library of multiply characterized switches, we aim at finding the optimal implementation for each switch in the network.

B. Characterizing Switches for Multiple Implementations

Fig. 3 presents the proposed switch characterization flow. The first step of this flow is to generate the RTL code that implements a given switch configuration. Then, the flow is split into two branches. One branch is for synthesizing the switch with the objective of minimizing its delay without any power and area constraints. The other is to synthesize the switch with the objective of minimizing its area without any delay constraint.[1] Each of the two synthesis results is then fed into the typical ASIC analysis flow with either pre- or post-layout accuracy. At the completion of timing and power[2] analysis, the minimum and maximum delays (i.e. *mindelay* and *maxdelay*) of the switch are available. A synthesis script is generated with a delay constraint between mindelay and maxdelay. This constraint can be selected randomly or within a fixed interval. The RTL code is then synthesized with the selected delay constraint, followed by timing and power analysis with either pre- or post-layout accuracy. It is designer's choice how many implementations of a switch configuration are produced.

[1]We also tried another branch for minimizing power consumption. According to our experiments, this branch produces almost identical results to that by the branch for minimizing area and is thus omitted in the flow.

[2]For the power characterization of switches, we adopt the switch power modeling used in [9].



(a) Fastest (b) Fastest+Post (c) In-process

Fig. 4: An example of NoC topology synthesis.

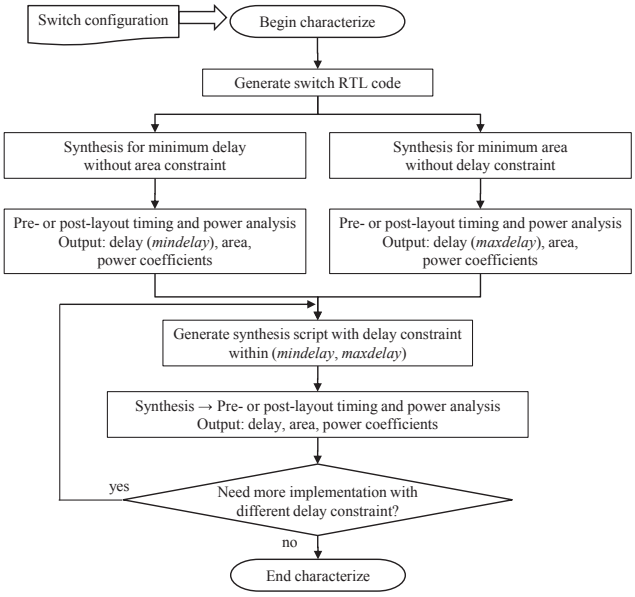


Fig. 3: Proposed switch characterization flow

C. Topology Synthesis Considering Multiple Switch Implementations

Here we explain how to apply the proposed switch characterization methodology to the NoC topology synthesis flow. We consider three different policies for exploiting the implementation diversity of switches.

• The *post replacement* policy (also called 'Post') is to replace the switches in the topology synthesized with the conventional characterization with their best fitting implementations. Since the implementation replacement occurs after the topology is determined, the network topology is not changed.

• The *re-synthesis* policy (also termed 'Resyn') progresses as follows: First, the synthesis is performed with the conventional switch characterization. Next, the best fitting implementations of the switches are selected for the frequency determined by the topology. Lastly, the synthesis is performed again with the switch implementations selected in the previous step. Since this policy performs the synthesis twice, the final topology may differ from that produced in the first step.

• The *in-process replacement* policy (also called 'In-process') is to find the best fitting implementations for every topology candidate during the topology synthesis. That is, for a topology candidate which is being evaluated, each switch in the topology is evaluated for all of its implementations and is matched to the best one.

Although the time for switch evaluation grows proportionally with the number of implementations considered, it is normally not the computational bottleneck in the whole topology synthesis process, as will be shown in Section V.

D. An Example

Fig. 4 shows three different synthesis results for one of our test cases (mobile_mmp; see Section V) whose synthesis objective is minimizing power consumption. Fig. 4a is the synthesis result with the conventional characterization where the switches are characterized for implementations having the minimum delay (this implementation is thus denoted as 'Fastest'). The frequency determined by the topology is 220.75Mhz, and the power consumption is 3.20mW. Fig. 4b shows the result after performing the post replacement (i.e. Fastest followed by Post) where the switch_0 and switch_1 are replaced with the implementations that are the most power-efficient at 220.75Mhz. The power consumption is reduced from 3.20mW to 2.72mW by the post replacement. Lastly, Fig. 4c presents the result with the in-process replacement policy used.[3] The only topological difference among the alternatives is the location of the 'peri' node, which is connected to switch_0 in Fig. 4a and 4b and to switch_1 in Fig. 4c. The in-process replacement resulted in the best result with the power consumption of 2.67mW. However, with only the fastest implementations, the topology in Fig. 4c consumes more power (3.32mW) than that in Fig. 4a, and this case is therefore discarded by the synthesis algorithm.

V. Experimental Result

A. Settings

For the switch characterization, we designed an RTL generator for an output-registered switch, whose interface is

[3]Fastest+Resyn produced the same result as this application.

compatible with the AMBA3 AXI protocol [18]; the address and data widths used are both 32bit. We assume circuit-switched networks where only the output ports (i.e. slave port) of the switches are registered. We generated the RTL code of switches sized from 1x2 to 12x12, synthesized the code with Synopsys Design Compiler and performed pre-layout timing and power analysis with Synopsys Primetime. The delay range (i.e. the interval between the minimum and maximum delays of a switch) was divided by 0.5ns intervals; *e.g.* if the difference between the minimum and maximum delays is 5ns, there would be 9 intermediate implementations in total.

For the topology candidate generation and design space exploration, we used a topology synthesis method similar to [6] with enhanced topology representation and exploration techniques. The details of the topology synthesis method used is irrelevant and is thus omitted; the proposed idea of using multiple switch implementations is widely applicable to many other existing topology synthesis methods.

We tested the proposed idea with five realistic applications and two very large synthetic applications. The first application (mpeg4decode) is from [14], the second (multimedia_soc) from [7], and the third to fifth (mobile_mmp, mobile_ap, and game_soc) from [9]. The two synthetic applications (45x10 and 65x14) were generated by combining two or more of the above realistic applications. Each application was synthesized with the objective of minimizing power, area or a weighted sum of power and area.

The switch characterization was performed on a virtual machine of 512MB memory without multithreading, and the topology synthesis was done in the host machine that has a 2.5Ghz Core2Quad processor and 2GB memory. The entire characterization took about a week, but we expect that the time can be greatly reduced by parallelization.

B. Results

Fig. 5 shows the NoC topologies synthesized by the conventional and proposed switch characterization methods with the three different synthesis objectives mentioned in Section V-A. The bars marked as Fastest and Smallest represent the results from using the conventional algorithms, and all the other bars are the results from using the proposed approach.[4] The height of a bar represents power consumption normalized to the height of the bar representing In-process.

Presented in Fig. 5a is the result when the synthesis objective is minimizing power consumption. It is worth noting that Fastest shows lower power consumption than Smallest (except mobile_ap and the failed cases of Smallest), even though each switch implementation used in Smallest is typically more power-efficient than that in Fastest. The reason is that Smallest uses a larger number of smaller switches since the switch implementations in Smallest are much slower than those in Fastest. Smallest uses five switches while Fastest uses only two for mpeg4decode. This tendency becomes more prominent as the application synthesized has heavier bandwidth and thus requires a higher network clock frequency. The mpeg4decode application has the highest bandwidth and requires a clock frequency of at least 340.75Mhz whereas mobile_ap has the lowest bandwidth requiring a clock frequency of 157.5Mhz.

The Post policy achieves considerable power saving up to 15.0% (for mobile_mmp) when combined with Fastest, while the power saving is only up to 3.9% when combined with Smallest. The reason is that the implementations used in Fastest have a great amount of margin to be optimized for power consumption since they are initially optimized for speed rather than for power, while the implementations in Smallest are already more power-efficient than those used in Fastest.

The Resyn policy can even further reduce the power consumption by up to 2.0% with respect to Post (for mobile_mmp) when combined with Fastest. When combined with Smallest, the power saving by Resyn is much larger than the power savings by Resyn combined with Fastest, showing up to 67.1% of reduction (for mpeg4decode). This power saving comes from reducing the number of switches during the second synthesis. The In-process policy does not produce better results than the Resyn policy in terms of power minimization. Overall, Resyn can reduce power consumption by up to 16.6% (mobile_mmp) and 9.5% on average in comparison with Fastest. Compared with Smallest (excluding the failing cases), Resyn can save power by up to 67.1% (mpeg4decoe) and 33.5% on average.

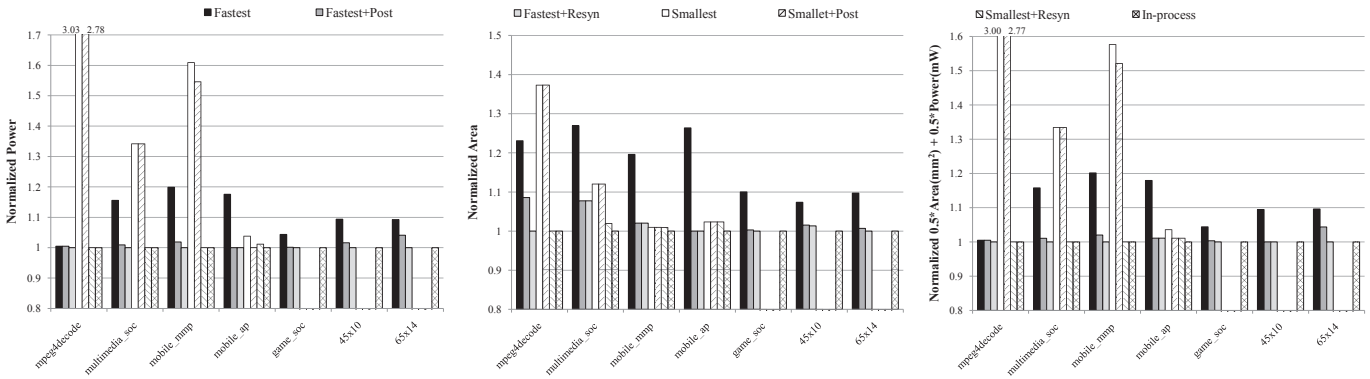
Fig. 5b shows the result when the synthesis objective is minimizing area. The tendency observed is similar to that in Fig. 5a, but we can find the benefit of adopting the In-process policy for multimedia_soc, mobile_mmp, and 45x10. In-process can save area by up to 21.2% (multimedia_soc) and 14.6% on average compared with Fastest, and by up to 27.2% and 10.3% on average over Smallest.

Fig. 5c presents the result when the synthesis objective is to minimize a weighted sum of area and power, namely $0.5 \times$ area (mm^2)$+0.5\times$power (mW). In this case, In-process shows better performance than Fastest+Resyn for mobile_ap.

Since we evaluate switches in their multiple implementations, the overhead in synthesis time may be of concern. However, according to our experiments, the post replacement took less than 0.1 seconds for all the applications, while the entire synthesis time ranged from 2.5 (mpeg4decode) to 1488 seconds (65x14). Fig. 6 shows the synthesis time of Post, Resyn, and In-process, each of which is combined with Fastest. The objective used is power minimization, and the bars are normalized to the height of Fastest+Post. Intuitively, In-process should incur the greatest overhead since it needs to evaluate a switch for all of its implementations. However, Resyn showed the longest synthesis time. The reason is that Resyn should perform the synthesis twice with different switch characterizations. In-process only incurs negligible overhead compared with Post.[5] This result confirms that the switch

[4]Regardless of the objective used, Smallest failed to find a feasible solution for game_soc, 45x10 and 65x14, and thus Smallest+Post and Smallest+Resyn could not be performed; the corresponding bars do not appear in the plots.

[5]Interestingly, for mobile_ap and 45x10, the synthesis time of In-process is shorter than that of Post, which is theoretically impossible. We conjecture that this is due to some unexpected variabilities in experiments such as OS process scheduling events during synthesis.



(a) Power consumption (b) Area (c) Weighted sum of area (mm^2) and power (mW)

Fig. 5: Comparison of various switch characterization on NoC topology synthesis

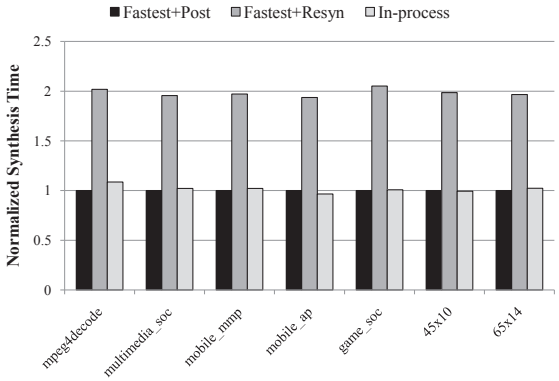


Fig. 6: Comparison of the synthesis time

evaluation step is not the computational bottleneck of the whole topology synthesis, as mentioned in Section IV-C.

VI. Conclusion

We proposed a methodology to consider multiple implementations of switches for NoC topology synthesis. Most existing approaches aim at determining the network topology of NoCs without considering diverse implementation possibilities of switches and may thus produce suboptimal solutions. By adopting the proposed technique, the burden on switch characterization may increase. However, the characterization step is done only once, whereas the NoC synthesis typically needs to be done multiple times during the front- and back-end design iterations. The effectiveness of our approach was demonstrated by the experiments performed on realistic and synthetic examples. We observed that the propose in-process replacement policy can save power and area by up to 67.1% and 27.2%, respectively, over conventional methods, with negligible computational overhead.

Acknowledgement

This work was supported in part by the IT R&D program of MKE/IITA 2009-S-005-01 (Development of Configurable Devices & S/W Environment), by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (No. 2009-0068969, KRF-2007-313-D00578), and by Yonsei University Institute of TMS Information Technology, a Brain Korea 21 program, Korea.

References

[1] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Constraint-Driven Bus Matrix Synthesis for MPSoC," *Proc. ASPDAC*, 2006, pp. 30-35.

[2] S. Murali and G. De Micheli, "An Application-Specific Design Methodology for STbus Crossbar Generation," *Proc. DATE*, 2005, pp. 1176-1181.

[3] S. Pasricha and N. Dutt, "COSMECA: Application Specific Co-synthesis of Memory and Communication Architectures for MPSoC," *Proc. DATE*, 2006, pp. 700-705.

[4] S. Murali, L. Benini, and G. De Micheli, "An Application-Specific Design Methodology for On-Chip Crossbar Generation," *IEEE Trans. on CAD*, vol. 26, pp. 1283-1296, Jul. 2007.

[5] S. Pasricha, N. Dutt, and F.J. Kurdahi, "Dynamically Reconfigurable On-Chip Communication Architectures for Multi Use- Case Chip Multiprocessor Applications," *Proc. ASPDAC*, 2009, pp. 25-30.

[6] J. Yoo, D. Lee, S. Yoo, and K. Choi, "Communication Architecture Synthesis of Cascaded Bus Matrix," *Proc. ASPDAC*, 2007, pp. 171-177.

[7] M. Jun, S. Yoo, and E. Y. Chung, "Mixed Integer Linear Programming-based Optimal Topology Synthesis of Cascaded Crossbar Switches," *Proc. ASPDAC*, 2008, pp. 583-588.

[8] M. Jun, S. Yoo, and E. Y. Chung, "Topology Synthesis of cascaded crossbar switches," *IEEE Trans. on CAD*, vol. 28, pp. 926-930, Jun. 2009.

[9] J. Yoo, S. Yoo, and K. Choi, "Topology/Floorplan/Pipeline Co-Design of Cascaded Crossbar Bus," *IEEE Trans. on VLSI*, vol. 17, pp. 1034-1047, Aug. 2009.

[10] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear Programming based Techniques for Synthesis of Network-on-Chip Architectures," *IEEE Trans. on VLSI*, vol. 14, pp. 407-420, Apr. 2006.

[11] K. Srinivasan, K. S. Chatha, and G. Konjevod, "An Automated Technique for Topology and Route Generation of Application Specific On-chip Interconnection Networks," *Proc. ICCAD*, 2005, pp. 231-237.

[12] S. Murail, P. Meloni, F. Angionili, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo, "Designing Application-Specific Networks on Chips with Floorplan Information," *Proc. ICCAD*, 2006, pp. 355-362.

[13] S. Murali, L. Benini, and G. De Micheli, "Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees," *Proc. ASPDAC*, 2005, pp. 27-32.

[14] S. Murali and G. De Micheli, "SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs," *Proc. DAC*, 2004, pp. 914-919.

[15] A. Pinto, L. P. Carloni and A. L. Sangiovanni Vincentelli, "A Methodology for Constraint-Driven Synthesis of On-Chip Communications," *IEEE Trans. on CAD*, vol. 28, pp. 364-377, Mar. 2009.

[16] J. Chan and S. Parameswaren, "NoCOUT : NoC Topology Generation with Mixed Packet-Switched and Point-to-Point Networks," *Proc. ASP-DAC*, 2008, pp. 265-270.

[17] S. Yan and B. Lin, "Application-Specific Network-on-Chip Architecture Synthesis Based on Set Partitions and Steiner Trees," *Proc. ASPDAC*, 2008, pp. 277-282.

[18] ARM, online *http://www.arm.com*.