Short Papers

Topology Synthesis of Cascaded Crossbar Switches

Minje Jun, *Student Member, IEEE*, Sungjoo Yoo, *Member, IEEE*, and Eui-Young Chung, *Member, IEEE*

***Abstract*—Performance requirements of on-chip network increase as system-on-chips (SoCs) are becoming more and more complex. For high-performance applications, crossbar switch-based networks are replacing the traditional shared buses as the backbone networks in SoCs. In this paper, we tackle the topology design of on-chip networks with crossbar switches in a cascaded fashion. We also resolve the unacceptable complexity of our previous method based on mixed integer linear programming by a heuristic method. Experimental results show that the proposed method overcomes the frequency limitation of the single crossbar-based design, particularly when the wire delay effect is considered. The proposed heuristic method also achieves more area reduction (up to 69.5%) over the existing methods, and finds as good solutions as the exact method while the synthesis time is saved by orders of magnitude.**

***Index Terms*—Embedded systems, on-chip networks, synthesis, system-on-a-chip (SoC).**

I. Introduction

On-chip network design determines the overall system performance of high-end data-intensive chips. Despite the cost effectiveness of a shared bus and its hierarchical variants, they cannot satisfy the communication requirement as the design complexity of modern system-on-chips (SoCs) increases due to the tighter bandwidth and latency constraints. Several works proposed the enhancement of the traditional on-chip buses from the arbitration [1], [2] and the architecture perspectives [3], [4]. The work in [5] analyzed the effect of bridges, protocols, and topologies over different traffic patterns in multilayered bus architecture.

Recently, the industry has productized crossbar switch (often called, bus matrix) solutions for SoC, such as ARM PL301 and Sonics SMX [6], [7]. In academia, authors in [8] proposed a technique for partially connected bus matrix generation to improve the area and clock frequency of a crossbar switch by eliminating unnecessary connections. Authors in [9] and [10] showed further improvements considering other design parameters such as arbitration schemes, buffer size, and memory allocation. The work in [11] reduced the area and power consumption of the central crossbar backbone by clustering masters and slaves into local buses. However, these approaches have a limitation on their topology, i.e., single crossbar, and thus they are limited to small and moderate size on-chip network designs from the performance perspective.

Manuscript received February 14, 2008; revised July 15, 2008 and November 18, 2008. Current version published May 20, 2009. This work was supported in part by Grant R01-2006-000-10156-0 from the Basic Research Program of the Korea Science and Engineering Foundation, by “System IC 2010” project of Korea Ministry of Knowledge Economy, and by IDEC (IC Design Education Center). This paper was recommended by Associate Editor L. Benini.

M. Jun and E.-Y. Chung are with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea (e-mail: mjun@dtl.yonsei.ac.kr; eychung@yonsei.ac.kr).

S. Yoo is with the Department of Electronic and Electrical Engineering, Pohang University of Science and Technology, Pohang 790-784, Korea (e-mail: sungjoo.yoo@postech.ac.kr).

Digital Object Identifier 10.1109/TCAD.2009.2017079

A structured network called cascaded crossbar switch network, which consists of multiple arbitrary-sized crossbar switches, is proposed in [12] and [13]. Since the physical characteristics of a crossbar switch, such as area, delay, and power consumption, are degraded as the switch size grows, careful consideration is needed when determining the crossbar network topology, particularly to meet the frequency requirement. The authors in [12] presented a novel representation called traffic group encoding (TGE) and used simulated annealing for topology synthesis, including bridges and pipeline stages. Tackling the optimality of this method, the authors in [13] employed mixed integer linear programming (MILP) upon the adjacency matrix representation to find the global optimum. However, it suffered from the computational complexity.

This paper first presents a modified MILP formulation and introduces new features to reinforce our previous work [13]. Then, a heuristic method, which is the main contribution of the work, will be presented to tackle the computational complexity. We expect that the reinforced MILP method will be applied to relatively small problems for the optimal solution, and the proposed heuristic method be used for even larger problems to find decent solutions in a reasonable time.

II. Problem Definition

We adopt the following four assumptions from our previous work [13]: 1) single frequency of the crossbar switch network; 2) single path between two communicating terminals; 3) single intercrossbar connection; and 4) one pipeline stage between intercrossbar connection.

The assumption 4) came from the observation of the results from the method in [12], where most of intercrossbar cascading paths are pipelined with one pipeline stage module.

Definition 1: A Communication Requirement Graph (CRG) $G(V_\mathrm{M}, V_\mathrm{S}, E, freq_\mathrm{lb})$ is an undirected graph, where $v_\mathrm{m} \in V_\mathrm{M}$ represents a master node, $v_\mathrm{s} \in V_\mathrm{S}$ a slave node, and $e_\mathrm{m,s} \in E$ an *edge* between the master and slave. $w(e_\mathrm{m,s})$ denotes the required bandwidth between v_m and v_s, and $d(e_\mathrm{m,s})$ the latency constraint. $freq_\mathrm{lb}$ is the required frequency lower bound.

The topology synthesis problem of cascaded crossbar switch network is represented with three 2-D binary decision variable matrices, MX (master-to-crossbar), SX (slave-to-crossbar), and XX (intercrossbar), where the element is one if the connection is established and otherwise zero. A crossbar switch network $T(V_\mathrm{M}, V_\mathrm{S}, X, MX, SX, XX)$ (in short, $T(MX, SX, XX)$) is composed of master node set (V_M), slave node set (V_S), crossbar set (X), and the connections among them, i.e., MX, SX, and XX. We also adopt the baseline of the definitions from our previous work [13], and omit the details, except for the modified or added parts.

The problem of synthesizing the topology of cascaded crossbar switch network can be defined as follows: **given** CRG $G(V_\mathrm{M}, V_\mathrm{S}, E, freq_\mathrm{lb})$, and the physical characteristics of crossbar switches, **find** $T(MX, SX, XX)$ that optimizes the design goal, such as area minimization, **such that** the communication requirements $w(e_\mathrm{m,s})$, $d(e_\mathrm{m,s})$, and $freq_\mathrm{lb}$ are satisfied.

III. MILP Formulation

The MILP formulation gives the exact solution for the given design parameters: the maximum number of available crossbar switches $|X|$

and the maximum cascading depth N. Here, we introduce only the enhanced formulations from the original one [13]. The enhancement is threefold: dealing with latency constraints in time unit, considering dynamic traffic patterns, and considering the wire delay effect in the topology synthesis.

A. Latency Constraints in Time Unit

To overcome the impreciseness of the previous latency constraints in hop count level, we modify the latency constraint to be handled in time unit. Let $H_{\mathrm{m,s}}$ be the hop count between a master v_{m} and a slave v_{s}. If v_{m} and v_{s} are connected through depth-n path, $H_{\mathrm{m,s}} = n$. Since at most one of D^ns[1] is one for a $(v_{\mathrm{m}}, v_{\mathrm{s}})$ pair by the single path assumption, $H_{\mathrm{m,s}}$ can be calculated as follows:

$$\forall (v_{\mathrm{m}}, v_{\mathrm{s}}) \text{ with } w(e_{\mathrm{m,s}}) > 0$$
$$H_{\mathrm{m,s}} = \sum_{n=1}^{N} \left(\sum_{v_{\mathrm{m}} \in V_{\mathrm{M}}} \sum_{v_{\mathrm{s}} \in V_{\mathrm{S}}} \sum_{P^n \in \mathbb{P}^n} n \times D^n_{m,P^n,s} \right) \tag{1}$$

where $\mathbb{P}^n$ is the set which has the permutation of n crossbar switch indices as its elements, such that $\mathbb{P}^n = \{(x_{c_0}, x_{c_1}, \ldots, x_{c_{n-1}}) | 0 \leq c_0 < c_1 < \cdots < c_{n-1} \leq |X| - 1\}$ and P^n denotes an element of $\mathbb{P}^n$.

Since the per-hop traversal time must be within $1/freq_{\mathrm{lb}}$, the delay of the path between v_{m} and v_{s} is not larger than $H_{\mathrm{m,s}}/freq_{\mathrm{lb}}$, and it must not exceed $d(e_{\mathrm{m,s}})$.

1) Latency Constraint: The total delay between a pair of master and slave should not exceed the given latency constraint

$$\forall (v_{\mathrm{m}}, v_{\mathrm{s}}) \text{ with } w(e_{\mathrm{m,s}}) > 0$$
$$H_{\mathrm{m,s}}/freq_{\mathrm{lb}} \leq d(e_{\mathrm{m,s}}). \tag{2}$$

B. Dealing With Dynamic Traffic Information

Here, we explain how to support the case of dynamic traffic where the traffic requirement varies over time. Our approach to handle the dynamic traffic pattern is similar to the method in [11] where the traffic is given for each time window. We first extend the CRG to dynamic CRG (dCRG) which contains the temporal traffic information, and then modify the constraints so that the bandwidth and the latency requirement are satisfied for each time window.

Definition 1-1: A dCRG $G(V_{\mathrm{M}}, V_{\mathrm{S}}, E, \Delta, freq_{\mathrm{lb}})$ is an extended CRG where $w^{\delta}(e_{\mathrm{m,s}})$ denotes the required bandwidth between v_{m} and v_{s}, and $d^{\delta}(e_{\mathrm{m,s}})$ the latency constraint in the time window $\delta \in \Delta$.

We set the bandwidth and latency constraints for each time window. This will multiply the number of linear constraints by the number of total time windows, while the number of decision variables remains the same. The feasibility of the topology holds if the bandwidth and latency constraints are satisfied for each time window.

1) Dynamic Bandwidth Constraint: The total bandwidth flowing on each link must not exceed its peak bandwidth for each time window

$$\forall x_p, x_q \in X, p < q, \text{ and } \delta \in \Delta$$
$$E^{\delta}_{x_p,x_q} \leq freq_{\mathrm{lb}} \times (linkwidth) \tag{3}$$

where $E^{\delta}_{x_p,x_q}$ is the total bandwidth on the link between crossbar x_p and x_q in time window δ, which is a dimensionwise extension of E_{x_p,x_q} in [13].

[1]$D^n_{\mathrm{m},P^n,\mathrm{s}}$ is a binary decision variable matrix which is one if $v_{\mathrm{m}} \in V_{\mathrm{M}}$ and $v_{\mathrm{s}} \in V_{\mathrm{S}}$ are connected through the crossbar switches in P^n. Refer to [13] for more details.

2) Dynamic Latency Constraint: The total delay of a connection hopping the cascaded crossbars must not exceed the given delay for each time window

$$\forall (v_{\mathrm{m}} \in V_{\mathrm{M}}, v_{\mathrm{s}} \in V_{\mathrm{S}}, \delta \in \Delta) \text{ with } w^{\delta}(e_{\mathrm{m,s}}) > 0$$
$$H_{\mathrm{m,s}}/freq_{\mathrm{lb}} \leq d^{\delta}(e_{\mathrm{m,s}}). \tag{4}$$

C. Wire Delay Effect in Network Topology Synthesis

In the cascaded crossbar network, the actual data traversal time during one clock cycle (hereafter, we call it hop delay) is the sum of: 1) delay from the starting pipeline stage (or master–slave interface) to the crossbar; 2) crossbar delay; and 3) delay from the crossbar to the destination pipeline stage (or slave–master interface). The longest hop delay in the crossbar network determines the network clock frequency. In order to consider the wire delay effect, we need to obtain the distances between the masters and slaves by floorplanning, and include this information as a part of the input wire delay-included CRG (wCRG).

Definition 1-2: A wCRG $G(V_{\mathrm{M}}, V_{\mathrm{S}}, E, freq_{\mathrm{lb}})$ is an extended CRG (or dCRG) where $dw(e_{\mathrm{m,s}})$ denotes the wire delay between v_{m} and v_{s}.

Here, we assume that if the path is established through multiple hops, the wire delay is evenly allocated to each hop. This assumption makes sense because we can locate the pipeline stage where the assumption holds, since the size of pipeline stage is negligibly small compared to cores and smaller even than the smallest crossbar switch.

1) Assumption of Even Wire Delay Distribution: When a master v_{m} and a slave v_{s} are connected through a depth-n connection, the wire delay between them $dw(e_{\mathrm{m,s}})$ is evenly distributed to each hop, such that the hop delay equals to the sum of each crossbar delay and $dw(e_{\mathrm{m,s}})/n$.

Let $CostDelay$ be a decision variable which is the inverse of the network clock frequency. Without considering wire delay, it is determined by the slowest crossbar switch in the network, but it is now dependent on the wire delay as well. Since the maximum hop delay in the crossbar network determines the network frequency, $CostDelay$ must be larger than or equal to the maximum hop delay. We can obtain the $CostDelay$ as follows.

2) Delay Constraints: $CostDelay$ must be larger than or equal to the maximum hop delay and less than or equal to the inverse of $freq_{\mathrm{lb}}$

$$\forall x \in X$$
$$Delay_x + \sum_{v_{\mathrm{m}} \in V_{\mathrm{M}}} \sum_{v_{\mathrm{s}} \in V_{S}} \sum_{P^n \in \mathbb{P}^n_x} D^n_{m,P^n,s} \times dw(e_{\mathrm{m,s}})/n$$
$$\leq CostDelay \leq 1/freq_{\mathrm{lb}} \tag{5}$$

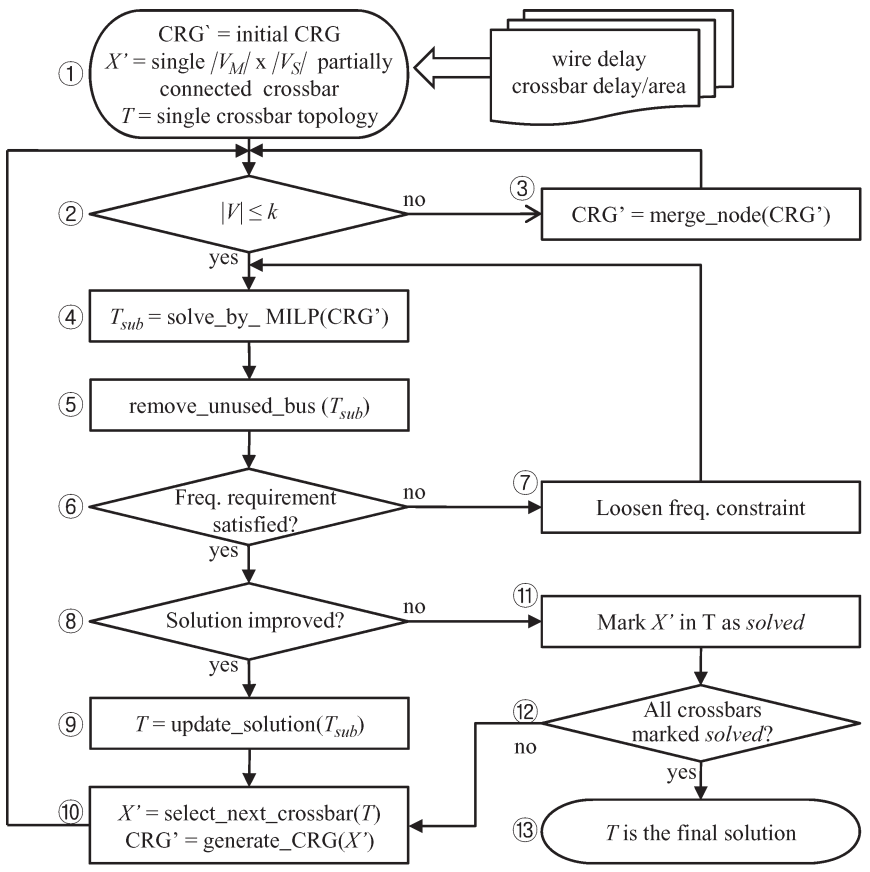
where $\mathbb{P}^n_x$ is the subset of $\mathbb{P}^n$ whose elements contain the crossbar x, and $Delay_x$ is the delay of crossbar x.

IV. Heuristic Method: MIRO

In this section, we present a heuristic method called *Merging Input-based Repetitive Optimization* (MIRO for short) that is based on the MILP method. The method enables to tackle much larger problems than the MILP method that suffers from exponential complexity on the problem size, i.e., the number of masters and slaves, N, and $|X|$.

A. Procedure of the Proposed Heuristic Method

The heuristic method tackles the problem in a divide and conquer fashion. The problem complexity is first reduced by merging masters and/or slaves into groups called supernodes. Then, an optimal crossbar network is obtained for each of the supernodes by applying the MILP method.



① CRG` = initial CRG
X' = single |V_M| x |V_S| partially connected crossbar
T = single crossbar topology
wire delay
crossbar delay/area
② |V| ≤ k
no
③ CRG' = merge_node(CRG')
yes
④ T_sub = solve_by_ MILP(CRG')
⑤ remove_unused_bus (T_sub)
⑥ Freq. requirement satisfied?
no
⑦ Loosen freq. constraint
yes
⑧ Solution improved?
no
⑪ Mark X' in T as solved
yes
⑨ T = update_solution(T_sub)
⑫ All crossbars marked solved?
no
yes
⑩ X' = select_next_crossbar(T)
CRG' = generate_CRG(X')
⑬ T is the final solution

Fig. 1. MIRO Process.

Fig. 1 shows the entire flow of MIRO. The MIRO process takes a CRG (which can also be dCRG or wCRG) and starts from the single partial crossbar topology (step 1 in Fig. 1). If the number of nodes (master and slave nodes) is more than the user parameter k, the merge process is performed to the current CRG until the CRG has no more than k nodes (step 2–3). After the merge process, the MILP method is applied to the merged CRG with the maximum cascading depth N of 2 (step 4) since the time-saving purpose of the heuristic approach fades with N larger than this value. However, note that the process does not restrict the cascading depth in the final solution. After the MILP step outputs the subtopology T_{sub}, the postprocessing is performed to remove unused paths in the subtopology (step 5). Next, the process checks to see if the result satisfies the frequency requirement, and if not, the MILP step is applied again with a lowered frequency requirement (step 7). Since our approach is based on splitting a crossbar into multiple ones repetitively, the frequency lower bound is not necessarily satisfied from the first time, but it must be at the final solution. If the obtained result gives an improvement with respect to the existing solution (in the first loop, the initial single partial crossbar switch), then the obtained subtopology replaces the single crossbar switch (step 9). The flow improves the solution iteratively. For this iterative improvement, a new crossbar switch (and masters, slaves, and/or supernodes connected to it) is selected as a subnetwork from the obtained topology, and then a new CRG is constructed for the selected subnetwork (step 10). The merging and MILP formulation are again applied to the new CRG (step 2–5). If the solution obtained from the MILP formulation does not give any improvement, then the corresponding crossbar is marked as "solved" (step 11), and it is no longer selected in step 10. If all the crossbars are marked as "solved," the process ends with reporting the current topology T as the solution (step 12–13).

B. Node Merging in CRG

The function merge_node() in Fig. 1 takes a CRG as its input, merges two nodes (masters, slaves, or supernodes) of the same type (i.e., masters are merged only with masters) in the CRG, and outputs another CRG consisting of the merged node(s) and remaining nodes. When two or more nodes are merged into one, it forms a master supernode M or a slave supernode S. M and S are subsets of V_{M} and V_{S}, respectively. When two nodes are merged, the edges belonging to those nodes are also merged. At that time, the bandwidths of those edges are accumulated while the minimum latency becomes the latency constraint of the new edge.

The baseline policies of merge process are as follows.

1) **Merge policy #1**: Maximize the link sharing by merging two masters (slaves) which have the most similar connectivity to slaves (masters).
2) **Merge policy #2**: Avoid link overloading by minimizing the heaviest edge weight in the resulting CRG.

We define metrics called *edge correlation*, denoted by $\rho_{i,j}$, for merge policy #1 and *max weight*, denoted by $\sigma_{i,j}$ for merge policy #2. The rationale of merge policy #1 is to merge as many masters (slaves) with the same communicating slaves (masters) as possible. The benefit will be to reduce the number of required links in the final topology for the resulting CRG. Merge policy #2 is to encourage merging low-bandwidth masters (slaves), in other words, masters (or slaves) with high bandwidth will survive without being merged with other masters (or slaves). This will increase the opportunity of finding better solutions by allowing a topology to satisfy the bandwidth constraint more easily.

TABLE I
RESULT OF SINGLE$_{P.C.}$, TGE+S.A., MILP, AND MIRO

	w/o wire delay						w/ wire delay	
	SINGLE$_{P.C.}$	TGE+S.A.	MILP	MIRO	Imprv over TGE	Imprv over MILP	SINGLE$_{P.C.}$	MIRO
App I	0.189	0.301 (28.9)	0.189 (653.9)	0.189 (9.6)	37.2%	0%	fail	0.435
App II	0.290	0.516 (58.3)	0.290 (86400)	0.290 (30.7)	43.8%	0%	fail	0.875
App III	0.259	0.359 (81.2)	0.259 (129600)	0.259 (120.2)	27.9%	0%	fail	0.414
App IV	fail	0.572 (54.3)	0.566 (timeout)	0.462 (82.1)	19.2%	18.4%	fail	0.619
App V	fail	6.073 (3231.5)	fail (timeout)	1.855 (363.4)	69.5%	n/a	fail	2.500

Unless the unit is specified, the values represents network area in mm^2, and those in the parenthesis represents the synthesis time in second

The edge correlation $\rho_{i,j}$ represents the number of communication targets (masters or slaves) shared by the node (or supernode) i and the other node (or supernode) j. It is calculated as follows:

$$\rho_{i,j} = \sum_{v_k \in V_{\mathrm{M}} \text{ or } V_{\mathrm{S}}} \tau_{i,j,k} \tag{6}$$

where $v_k \in V_{\mathrm{S}}$ holds when i and j are masters (supernodes) and $v_k \in V_{\mathrm{M}}$ holds for the opposite case. $\tau_{i,j,k}$ is defined to be one if nodes (or supernodes) i and j both have an edge to the node (or a node in the supernode) k in the original CRG, and zero otherwise.

The max weight $\sigma_{i,j}$ is the maximum edge weight (i.e., the maximum of bandwidth requirements) in the resulting CRG where nodes (or supernodes) i and j are merged. Max weight is important since it affects the minimum required frequency and also limits the acceptable switch size.

Finally, the merge cost $C_{i,j}$ is defined as follows:

$$C_{i,j} = \frac{1 + \sigma_{i,j} \times u\left(avg\left(w(E)\right) - heaviness\_threshold\right)}{\rho_{i,j}} \tag{7}$$

where $avg(w(E))$ is the average edge weight of the edges in the input CRG, and $heaviness\_threshold$ is the parameter at which the input traffic is considered to be heavy or light. We initially set the value to 30% of the required link bandwidth, and we will refer to the parameter as the *heaviness threshold ratio*. We will show the effect of this parameter in the experiment. The node merge process is to find a pair of nodes (or supernodes) which yields the smallest $C_{i,j}$.

V. Experimental Results

A. Experimental Setup

We applied our methods to five real-world applications. The first application is an mpeg4 decoder example in [14] with latency constraints added to some edges arbitrarily (App I). The second one is a +20 Mgate design with 12 masters and 4 slaves (App II) in [13]. The third one is a multimedia player with 12 masters and 5 slaves (App III), and the forth one is a mobile application processor SoC (App IV) with 14 masters and 5 slaves. The last application (App V) is a game SoC example with 38 masters and 8 slaves. All the backbone bus interfaces in App I to App IV have 32-bit data width, and those in App V have 64-bit data width. For these five applications, the required clock frequency is 200 MHz.

We obtain the physical characteristics of crossbar switches by generating RTL code from AMBA Designer [6] and synthesizing it using a proprietary 90-nm process technology library. We obtained the wire delay between masters and slaves by running Parquet [15], a system level floorplanner.

The MILP formulation is solved using XPRESS-MP [16] with the objective of minimizing the network area, composed of the crossbars and the pipeline stages. The proposed method is implemented in C++, and the counterpart method in [12] (TGE+S.A.) is implemented in Java.

B. Evaluation of Proposed Architecture and Methods

The result is summarized in Table I. Without wire delay consideration, single partial crossbar solution (SINGLE$_{\mathrm{P.C.}}$ for short) is the best solution for the first three applications, but fails to satisfy the 200 MHz of frequency requirement for App IV and App V. On the other hand, the proposed architecture (particularly the results from MIRO) gives feasible solutions resolving the frequency problem. The necessity of the proposed architecture is even more prominent when the wire delay is taken into account. SINGLE$_{\mathrm{P.C.}}$ fails to satisfy the frequency requirements for all the applications, while MIRO finds feasible crossbar networks for all the five applications.

Next, we will verify the effectiveness of the proposed methods, the exact method (for short, MILP) and the heuristic method MIRO, by comparing them with the existing method TGE+S.A. The comparison here is performed without considering the wire delay effect because there is no wire delay consideration in TGE+S.A. Since the solution spaces of TGE+S.A and our methods are different, we first manipulated the TGE+S.A. such that the design space is equivalent to ours. For MILP, we set the maximum cascading depth N to three and the number of available crossbar switches $|X|$ to six. Moreover, we set the MILP timeout deadline to 48 hours. For MIRO, we use N of two and $|X|$ of four for each MILP run, and set the merge threshold k and heaviness threshold ratio to 10% and 30%, respectively.

As shown in Table I, both of the proposed methods find better solutions than TGE+S.A. for all the cases, except for App V where MILP fails to find a feasible solution. MIRO reduces area from TGE+S.A. for the five applications by 42.7% on average, ranging from 19.2% (App IV) to 69.5% (App V). We can also find that MIRO resolves the complexity issue of MILP while giving as good solutions as MILP (or sometimes better solutions than MILP when MILP is timed out).

C. Analysis on the Effects of MIRO Parameters

In this section, we will report the analysis on the effects of MIRO parameters: the merge threshold k and the heaviness threshold ratio. Regarding the remaining parameter of MIRO $|X|$, we could not find any notable improvement in solution quality when $|X|$ is larger than four. The reason is that since the input CRG is compressed into a quite small problem before being applied to MILP, the resulting subtopology hardly has more than four crossbars. The synthesis is performed considering the wire delay effect.

First, we repeated MIRO synthesis process changing the merge threshold k from 3 to 12. The result is shown in Fig. 2(a) and (b). The values on the dashed lines in Fig. 2(a) are normalized to the maximum area obtained for each application, and the bold line is their average. Note that there is no feasible solution found for several applications with small k, e.g., at least nine of k is required for App II. The results show that the solution quality tends to be improved as k increases, even though the results for App II and App V show that it is not always true. Since MIRO operates in a repetitive manner for each switch, there can be a situation where splitting a crossbar into greater number of

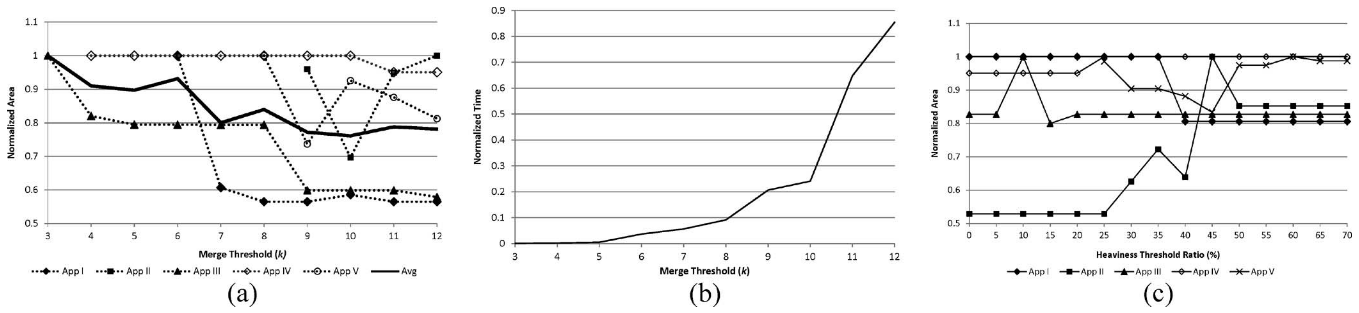


Fig. 2. Sensitivity on the MIRO parameters. (a) Normalized area over k. (b) Normalized time over k. (c) Normalized area over heaviness threshold.

smaller ones in the present iteration limits splitting crossbars at the next iterations.

Fig. 2(b) shows the average normalized runtime for different values of k. As k increases from 3 to 12, while the area reduction is improved by 24.0%, on average, as shown in Fig. 2(a), the synthesis time skyrockets by more than 900x. For this reason, it is desired to set k large enough to find a feasible solution, but not too large to make the synthesis time acceptable.

We changed the heaviness threshold ratio from zero to 70% by 5% with k fixed to ten. The normalized area is shown in Fig. 2(c). As the heaviness threshold gets larger, the edge correlation ρ becomes dominant in the merge process, while in the opposite case the maximum edge weight σ becomes dominant. The result shows the application dependence of the parameter since the tendency is quite opposite for App I and App II and IV, while App III shows no tendency to this parameter. Even though there is no globally optimal point for the heaviness threshold, its effect on the solution quality is considerable. The area saving varies up to 47.1% (App II) against this parameter, and 21.6% on average. Therefore, the heaviness threshold must be chosen thoroughly for better cascaded crossbar network design.

VI. Conclusion

In this paper, we enhance our previously proposed exact method and propose an efficient heuristic method called MIRO for the topology synthesis of cascaded crossbar switch networks. The experimental results show that MIRO finds as good solutions as the exact method, or even better solutions when the exact method is timed out, while the synthesis time is reduced from days (timed out with two days of deadline) to minutes (at most about 6 min). In addition, MIRO shows more area reduction than the existing method TGE+S.A. [12] by up to 69.4% and 42.7% on average. The two proposed alternative topology synthesis methods help designers tradeoff the design time of the on-chip communication network and its quality depending on the problem size and the design time constraint.

References

[1] M. Jun, K. Bang, H. J. Lee, and E. Y. Chang, “Slack-based bus arbitration scheme for soft real-time constrained embedded systems,” in *Proc. ASPDAC*, 2007, pp. 159–164.

[2] K. Lahiri, A. Raghunathan, and G. Lakshminarayana, “LOTTERYBUS: A new high-performance communication architecture for system-on-chip designs,” in *Proc. DAC*, 2001, pp. 15–20.

[3] R. Lu and C. K. Koh, “SAMBA-BUS: A high performance bus architecture for system-on-chips,” in *Proc. ICCAD*, 2003, pp. 8–12.

[4] K. Sekar, K. Lahiri, A. Raghunathan, and S. Dey, “FLEXBUS: A high-performance system-on-chip communication architecture with a dynamically configurable topology,” in *Proc. DAC*, 2005, pp. 571–574.

[5] S. Medardoni, M. Ruggiero, D. Bertozzi, L. Benini, G. Strano, and C. Pistritto, “Capturing the interaction of the communication, memory and I-O subsystems in memory-centric industrial MPSoC platforms,” in *Proc. DATE*, 2007, pp. 660–665.

[6] *ARM*. [Online]. Available: www.arm.com

[7] Sonics Inc. [Online]. Available: www.sonicsinc.com

[8] S. Murali and G. De Micheli, “An application-specific design methodology for STbus crossbar generation,” in *Proc. DATE*, 2005, pp. 1176–1181.

[9] S. Pasricha, N. Dutt, and M. Ben-Romdhane, “Constraint-driven bus matrix synthesis for MPSoC,” in *Proc. ASPDAC*, 2006, pp. 30–35.

[10] S. Pasricha and N. Dutt, “COSMECA: Application specific co-synthesis of memory and communication architectures for MPSoC,” in *Proc. DATE*, 2006, pp. 700–705.

[11] S. Murali, L. Benini, and G. De Micheli, “An application-specific design methodology for on-chip crossbar generation,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 7, pp. 1283–1296, Jul. 2007.

[12] J. Yoo, S. Yoo, and K. Choi, “Communication architecture synthesis of cascaded bus matrix,” in *Proc. ASPDAC*, 2007, pp. 171–177.

[13] M. Jun, S. Yoo, and E. Y. Chung, “Mixed integer linear programming-based optimal topology synthesis of cascaded crossbar switches,” in *Proc. ASPDAC*, 2008, pp. 583–588.

[14] K. Srinivasan, K. S. Chatha, and G. Konjevod, “An automated technique for topology and route generation of application specific on-chip interconnection networks,” in *Proc. ICCAD*, 2005, pp. 231–237.

[15] S. N. Adya and I. L. Markov, “Fixed-outline floorplanning: Enabling hierarchical design,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.

[16] *Dash Optimization*. [Online]. Available: www.dashoptimization.com