Advance Access publication on February 26, 2010
doi:10.1093/comjnl/bxq020

Design of On-Chip Crossbar Network Topology Using Chained Edge Partitioning

MINJE JUN AND EUI-YOUNG CHUNG*

School of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea
**Corresponding author: eychung@yonsei.ac.kr, eychung@paran.com*

This paper proposes an efficient topology synthesis method for on-chip interconnection network based on crossbar switches. The efficiency of topology synthesis methods is often measured by two metrics—the quality of the synthesized topology and synthesis time. These two metrics are critically determined by the definition of the topology design space and the exploration method. Furthermore, an efficient representing method for the design space is required to tightly link the design space and the exploration method. Even though topology synthesis methods have actively been researched, most of the previous methods were not deep in thought for these factors. Unlike the previous methods, we propose a topology synthesis method with a careful consideration of these factors. Our method efficiently defines the design space by a technique called *chained edge partitioning*, in conjunction with a representing method for the points in the space, called *enhanced restricted growth function*. We also provide an exploration method which well incorporates with the aforementioned search space. To prove the effectiveness of our method, we compared our method with previous methods. The experimental results show that our method outperforms the compared methods by up to 49.8% and 104.6× in the quality of the synthesized topology and the synthesis time, respectively.

Keywords: on-chip network; topology; synthesis

Received 15 July 2009; revised 12 January 2009
Handling editor: Sung Woo Chung

1. INTRODUCTION

The design of communication architecture in system-on-chips (SoCs) has become more challenging due to the advent of data-intensive applications such as high-definition videos and 3D games. These applications urge the adoption of multiple processors into SoCs (so-called MPSoCs) to deal with huge computation requirement. These applications also produce high volume of communications among the computation cores and memories, causing high pressure on their backbone on-chip interconnection network. Such heavy traffics cannot be accommodated by the traditional shared bus architecture and its variants (i.e. multi-layer bus architecture), since their operating clock frequency relative to that of the computing units becomes worse as the process technology scales down.

Bus matrix (also called crossbar) is rapidly replacing the traditional shared busses in contemporary SoC design, since it provides higher bandwidth than the shared busses by employing the point-to-point communication architecture. In other words, a dedicated communication path is allocated to each pair of masters and slaves. With this architecture, it is possible to achieve higher bandwidth by allowing multiple concurrent communications among the computing units and memories. However, its advantage is diminished when the bus matrix becomes large, since its operating clock frequency is inversely proportional to its size which is closely related to the number of its input and output ports. Partial crossbar solutions were proposed to tackle this problem by removing unnecessary connections inside the crossbar and clustering masters and slaves into local buses [1–5]. However, they are not free from the limitation of the single crossbar solution since even the partial crossbar will become unacceptably slow due to ever-increasing numbers of masters and slaves.

To overcome the limitation, on-chip interconnection network has been proposed as a promising solution to large scale systems. There are two classes of studies on on-chip interconnection network. One is for general purpose multi-core systems and the other is for application-specific multi-core systems. The former basically concerns on the flow control, routing scheme and other design parameters (e.g. buffer size), while adopting the regular topology such as mesh and torus. This architecture is typically called chip-multi-processor (CMP) architecture. On the other hand, the latter (most of embedded MPSoCs are in this class) even concerns on the network topology, since the area and power costs of an on-chip interconnection network are non-marginal. It is well known that the topology of an on-chip interconnection network largely determines these metrics.

Compared with the CMP architecture, there are potential opportunities in optimizing (or customizing) the topology of the application-specific on-chip network for area and power by utilizing the following properties: (i) it is application- or domain-specific (i.e. it targets a limited set of applications), (ii) the traffic pattern among the cores and memories are not symmetric and (iii) the computing cores are heterogeneous (some of them are hardware accelerators) and have real-time constraints. By exploiting the aforementioned properties, it is possible to create custom-tailored irregular on-chip interconnection networks that are more cost-effective than the regular ones.

Due to the above reason, optimizing (or customizing) the on-chip network topology has become a critical design step of modern embedded MPSoCs. Recently, several works proposed a cascaded crossbar switch architecture and the corresponding topology synthesis methods in [6–8]. In this architecture, a single large central crossbar (or partial-crossbar) switch is replaced by multiple smaller crossbar switches which are connected in a cascaded fashion. Figure 1 contrasts the single partial crossbar switch network and the cascaded crossbar switch network with irregular topology. The white rectangles denoted by 'M' are master units (e.g. CPU) and those denoted by 'S' are slave units (e.g. memory). Also, the gray rectangles with inner connections are crossbar switches. A critical problem in this architecture is to find an optimal topology which pays the minimum cost (area, power consumption or both) compared with other possible choices, while satisfying the given communication constraints (bandwidth and latency). It is challenging since there are huge possible topological choices for a given specification. This is the motivation of the topology synthesis of the cascaded crossbar switch network which automatically determines the connections among the master units and slave units by the appropriate selections and connections of crossbar switches. It has been shown that the cost and the performance of a design critically depends on the quality of the topology synthesis methods [6–8].

Many custom topology synthesis methods for network-on-chip have been proposed in [9–15]. These works usually assume peer-to-peer communication and packet-switched network, while the cascaded crossbar networks assume master–slave communication and circuit-switching. Even though their natures are somewhat different from each other, their topology synthesis problems are quite similar in the sense that they are both to determine the connections among the IPs and switches.

The efficiency of a topology synthesis method is typically measured by two metrics—the quality of the synthesized network and synthesis time. However, the previous works were not deep in thought to identify the major factors affecting these two metrics. More specifically, the aforementioned two metrics are critically affected by the definition of the topology search space (or design space) and its exploration method. Since the design space and the exploration method have different abstraction levels, a representing method is required to translate the design space into the data structure of the exploration method. Hence, the synthesis quality and time are also affected by the representing method. Unfortunately, previous methods focused on only part of these three factors, thereby missing at least one of them in their implementation. We will discuss the details of previous methods in Section 2 with respect to these three factors.

In this work, we propose a topology synthesis method that is designed with the careful consideration of these three factors. The contribution of our work can be summarized as follows. First, we define the search space by a technique called *chained edge partitioning* (CEP) in conjunction with the *enhanced restricted growth function* (ERGF). With these techniques,

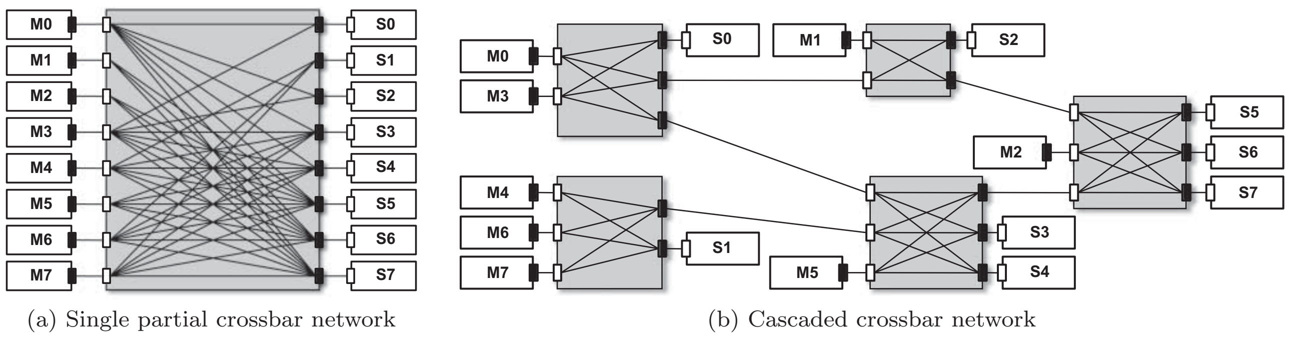

M0
M1
M2
M3
M4
M5
M6
M7
S0
S1
S2
S3
S4
S5
S6
S7
(a) Single partial crossbar network
M0
M3
S0
M1
S2
S5
M2
S6
S7
M4
M6
M7
S1
M5
S3
S4
(b) Cascaded crossbar network

FIGURE 1. An example of cascaded crossbar network.

the design space only includes the *legal* design points. A topology at each legal design point guarantees that it provides all the paths required by the specification. In other words, our method automatically prunes out the topologies in which the required paths are not provided by the efficient definition of the search space. Second, we propose a representing method called ERGF which helps to identify the isomorphic design points to avoid re-evaluation of the same topology. Finally, we propose an exploration method which well incorporates with the aforementioned search space.

The rest of this paper is organized as follows. In Section 2, we summarize the previous methods with their drawbacks. In Section 3, we define our topology synthesis problem. In Section 4, we present the proposed method in detail. In Section 5, we show the effectiveness of our method by comparing it with previous methods and Section 6 concludes our work.

2. RELATED WORKS

Several works have been proposed to improve the bandwidth and latency of the shared bus architecture by employing sophisticated arbitration schemes [16–18] and by physically segmenting the bus to enable concurrent communications [19, 20]. Medardoni *et al.* [21] analyzed the effect of the protocol, bridge design and traffic pattern on the hierarchical bus architecture and Drinič *et al.* [22] proposed the automated design method for hierarchical bus architecture. Although these methods greatly improves the performance of the shared bus-based architectures, the shared nature of the bus limits the performance and the scalability.

The works in the early stage of the switch-based on-chip network topology synthesis mostly focused on the exploration method, hence the search space was not well refined. Many of them used adjacency matrix-based representation to specify the entire search space. Using the adjacency matrix, they represent the connections between the masters, slaves and switches [7–10]. These works mostly solved the topology synthesis problem with the mathematical formulations which yield the exact solution. Even though the synthesis quality of their methods is optimal, the synthesis time is prohibitively large due to the inefficient search space which includes a lot of infeasible design points. For this reason, their methods are applicable to only small-scale problems and should be extended in a heuristic manner for larger scale designs. For instance, the works in [9, 10] proposed topology synthesis methods using mixed-integer linear programming (MILP) where the decision variables are the elements of the adjacency matrices. The refinement on the search space was more elaborated in [7], but they still suffered from the long synthesis time. Later, they proposed a heuristic method based on the input merging technique [8].

The works in [11, 12] proposed clustering (or partitioning)-based heuristic algorithms. They were aware of the synthesis time issue and put limits on the search space. The refined search space often excludes a part of feasible design points, yielding inferior synthesis quality. The work in [14] improved the exploration efficiency by considering the management policy of the explored design points. They record the explored design points in a *tabu list* and check this list first before the new design point is evaluated. The advantage of this approach becomes marginal when a long *tabu list* is employed, since the full search of the *tabu list* for every design point incurs a large computation overhead.

The works in [6, 15] made a good contribution in the search space reduction. They formulated the network topology synthesis problem as clustering (or partitioning) of communication edges. The former proposed a method named *traffic group encoding* in which an ordered set of edge sets is always interpretable to a legal implementation. Since the reduced search space is still large, they explore the search space using simulated annealing with random moves. Such random moves often visits the design points already explored, which depreciate the impact of the search space reduction. Also, the synthesis quality is a concern when the deign time is tightly constrained. On the other hand, Yan and Lin [15] claimed that the topology synthesis problem can be mapped to a partitioning problem of the given communication edges. Their method is limited to direct network topologies, thereby missing more efficient indirect network topologies.

3. PROBLEM DEFINITION

In this section, we define the topology synthesis problem and introduce the notations used for the definition. In our formulation, we consider a cascaded crossbar switch architecture which is irregular from the topological point of view. The on-chip network topology synthesis problem is to determine the connections between the masters (more precisely, ports with master interface on IPs) and the switches, between the slaves (more precisely, ports with slave interface on IPs) and the switches and between the different switches, while the communication requirements (bandwidth and latency) specified on each master-slave pair are satisfied. The objective of the synthesis is to minimize the cost, which can be the area, the power consumption or both. The communication requirements of a given application is the input of our method and given as a graph called *communication requirement graph* (CRG).

(i) A CRG is a directed bipartite graph $G(V_M, V_S, E)$ where $v_m \in V_M$ denotes a master, and $v_s \in V_S$ denotes a slave. An edge $e \in E$ denotes a communication between v_m and v_s. src(e) and dst(e) are the source (master) and the destination (slave) of the edge 'e', respectively. Also, bw(e) and lat(e) denote the bandwidth and latency constraint of the edge 'e', respectively.

Since the cost of the synthesized design is determined by the cost of the switches used in the network, we also need the physical information of the switches of various sizes.

(ii) $XA_{i,j}$, $XP_{i,j}$ and $XF_{i,j}$ are the area, power consumption and frequency of a crossbar switch with i input ports and j output ports. ρ is the area of the pipeline stage unit. We obtain this information from the register transfer-level synthesis of the crossbar switches and pipeline stage.

A topology is *feasible* if it satisfies all the given communication requirements, and it is *infeasible* otherwise. Also, we call a topology *legal* if it provides the required paths regardless of the bandwidth and latency constraint satisfaction, and *illegal* otherwise.

We adopt the basic assumptions for the cascaded crossbar network, such as single path routing and single link between two switches, from [7]. In this work, we consider AMBA 3.0 AXI as the target on-chip interconnection architecture, hence the details of the target architecture can be found in [23]. Note that, however, our method is not limited to a specific architecture but applicable to other architectures with the minor modifications, if necessary.

Then, our topology synthesis problem is to find a *feasible* topology, given the CRG and the physical information of the network components (XA, XP, XF and ρ), such that the target cost is minimized.

4. PROPOSED TOPOLOGY SYNTHESIS METHOD

4.1. Overview

The fundamental issue in on-chip topology synthesis is attributed to the large design space. Hence, it is crucial to tightly define the design space and efficiently explore the design space. Furthermore, the representing method of a design space is closely related to the quality of the exploration method. For instance, a bad representing method will produce several different representations for an identical design point.

To tackle these points, we first define a topology design space, using a novel method called CEP (see Section 4.2). An ideal design space contains all feasible solutions, while excluding all infeasible or trivial solutions as many as possible. The aim of CEP is to define a design space as close as the ideal design space and we discuss the details of CEP in Section 4.2.

Second, we propose a representing method of a CEP-based design space called ERGF. ERGF translates each design point in the design space into a unique set of number sequences (Section 4.3). Such uniqueness eliminates the isomorphic representations, hence protecting to revisit the design points already explored. As will be shown in Section 4.3, a number sequence generated by ERGF (ERG sequence for short) corresponds to a unique partitioning of edges in a certain stage, and thus a set of ERG sequences of all stages corresponds to a unique design point in the CEP-based design space.

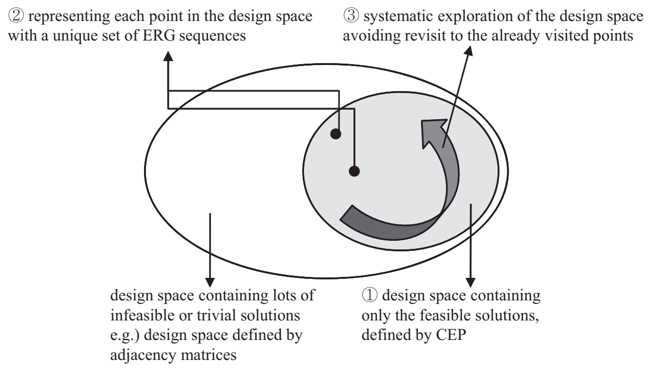

② representing each point in the design space with a unique set of ERG sequences
③ systematic exploration of the design space avoiding revisit to the already visited points
design space containing lots of infeasible or trivial solutions e.g.) design space defined by adjacency matrices
① design space containing only the feasible solutions, defined by CEP

FIGURE 2. The interactions of three key factors of the proposed synthesis method.

Finally, we present an efficient design space exploration method based on ERGF (Section 4.4). Even though the design space definition and the representing method are efficient, a poor exploration method can depreciate their benefits. For example, the exploration may guide us to visit the same design point repeatedly, while ignoring some other design points in the design space throughout the synthesis process. With the proposed exploration method, it is possible to visit every design point (with the highest effort) in the design space only once, thereby never wasting time to evaluate a single design point more than twice.

Figure 2 shows the interactions of the aforementioned three key factors. To summarize, CEP contributes to the topology synthesis by tightening the design space, whereas ERGF helps the exploration method by providing unique representation for each design point. The ERGF exploration method efficiently searches the design space by avoiding the revisit to the design points already explored.

4.2. Feasible design space definition

In the proposed method, we map the application-specific network topology synthesis problem to the successive partitioning of the communication edges for a fixed number of stages. The number of stages means the maximum number of switches a communication can traverse, usually called hop counts. We name the method CEP after its behavior. The benefit of CEP is that it contains only the legal topologies with the help of ERGF. Note that the legality of a topology is a necessary condition for the feasibility, i.e. if a topology is illegal, it is definitely infeasible. Since CEP implicitly prunes the illegal design space, the search space for the feasible topologies can be better confined. Before explaining the procedure, we first define several terminologies.

DEFINITION 1. *A partition p_i^k is a set of edges which is generated in kth stage of CEP where i is the partition index.*

```
input     $E^0 = E$ in CRG
          maximum number of stage $N$
initial   stage index $k = 1$

(step 1)  Generate $P^k$, a partition set of $E^{k-1}$.
(step 2)  Generate new edge set $E^k$ from each
          partition in $P^k$ to the slaves.
(step 3)  Increment $k$ by 1
(step 4)  If $k = N$, stop and evaluate the topology.
          Otherwise go back to (step 1)
```

FIGURE 3. The procedure of generating a topology by CEP.

A partition corresponds to a switch node in the topology graph T, except p_0^k. For all stages, p_0^k is a bypass partition which transparently passes the edges to the next stage, and is not translated to a real switch.

Definition 2. *$e_i^k \in E^k$ is an edge generated after the partitioning in kth stage such that $\mathrm{src}(e_i^k)$ is one of the switches generated in the kth stage and $\mathrm{dst}(e_i^k)$ is one of slave in the CRG. E^0 is same as the E in CRG.*

Definition 3. *A partition set P^k is a set of partitions in kth stage of CEP which is obtained by the edge partitioning. The edge partitioning is grouping the edges into the one or more subsets such that the sets are disjoint and exhaustive, i.e. $p_i^k \cap p_j^k = \emptyset$ if $i \neq j$ and $\cup_i p_i^k = E^{k-1}$. Φ^k is the set of all possible partition sets in kth stage, whose element P_i^k denotes a unique partition set in the stage. For convenience, we omit the index i if it is not of importance.*

Definition 4. *A topology T is an N-tuple of partition sets such that $T = (P^1, P^2, \ldots, P^N)$, where N is the given number of stages. Ω is the set of all the topologies, whose element T_i is a unique point in the set.*

The procedure of generating a topology instance by CEP is shown in Fig. 3.

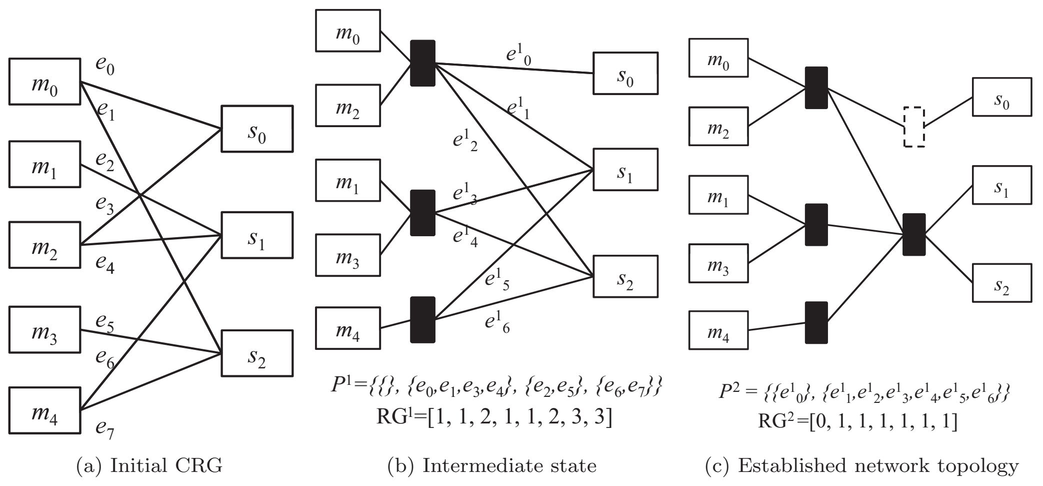
The procedure takes the set of edges in the CRG (E^0) and the maximum number of stage N. The physical meaning of N is the maximum number of crossbars that a traffic between a master and a slave can traverse in the network. It can also be understood as the maximum hop count physically bounded by the network. Hence, the master–slave pairs in a single network may have different number of crossbars in their paths. For example, if we set N to 3, some master–slave pairs may have three crossbars in their paths, while other pairs may have one or two crossbars. With these inputs, it performs edge partitioning and generates the partition set of the first stage P^1 (Step 1). Next, the new edges are generated from each partition p_i^1 to the slaves, and these edges make up the new set of edges E^1. When a new edge is generated from the partition of the previous stage, its bandwidth and latency are calculated as follows.

$$\mathrm{bw}(e_i^k) = \sum_{e \in D_i^k} \mathrm{bw}(e) \tag{1}$$

$$\mathrm{lat}(e_i^k) = \mathrm{MIN}_{e \in D_i^k} \mathrm{lat}(e) \tag{2}$$

where $D_i^k = \{e \mid e \in E^{k-1} \wedge \mathrm{dst}(e) = \mathrm{dst}(e_i^k) \wedge e \in \mathrm{src}(e_i^k)\}$. At this point, the connections between the masters and the switches in the first stage is determined. If $N = 1$, the procedure stops and evaluates the current topology, but if $N > 1$, it performs the partitioning of E^1 to determine the connections between the switches in the first stage and second stage. This procedure is repeated until the Nth stage connections are determined (Steps 3 and 4).

Figure 4 shows an example of this procedure with the maximum number of stages of 2. Figure 4a is the initial CRG where m_0 to m_4 represents the masters, s_0 to s_2 the slaves and e_0 to e_7 the edges. First, the edges in the CRG are partitioned into arbitrary number of partitions, where each partition represents



m_0
m_1
m_2
m_3
m_4
e_0
e_1
e_2
e_3
e_4
e_5
e_6
e_7
s_0
s_1
s_2
(a) Initial CRG
e^1_0
e^1_1
e^1_2
e^1_3
e^1_4
e^1_5
e^1_6
$P^1=\{\{\}, \{e_0,e_1,e_3,e_4\}, \{e_2,e_5\}, \{e_6,e_7\}\}$
$RG^1=[1, 1, 2, 1, 1, 2, 3, 3]$
(b) Intermediate state
$P^2 = \{\{e^1_0\}, \{e^1_1,e^1_2,e^1_3,e^1_4,e^1_5,e^1_6\}\}$
$RG^2=[0, 1, 1, 1, 1, 1, 1]$
(c) Established network topology

FIGURE 4. Establishing network topology by using CEP.

a switch to which the allocated edges are connected. Since a master can be connected to exactly one switch, the edges from the same master must be in the same partition in the first stage. Let's assume that e_0, e_1, e_3 and e_4 are allocated to the same partition (p_1^1), e_2 and e_5 to the second (p_2^1) and e_6 and e_7 to the third (p_3^1). Then, the resulting graph is shown in Fig. 4b, where the filled rectangles are the switches and e_0^1 to e_6^1 are newly generated edges by (Step 2) in Fig. 3. Note that, when a new edge is generated from the partition of the previous stage, the bandwidth is accumulated and the latency is taken as the minimum. For example, $\mathrm{bw}(e_0^1) = \mathrm{bw}(e_0) + \mathrm{bw}(e_3)$, and $\mathrm{lat}(e_0^1 = \mathrm{MIN}(\mathrm{lat}(e_0),\ \mathrm{lat}(e_3))$. In this state, the edges between masters and switches are physical links but those between switches and slaves are still abstract edges, not physical links.

In the next step, the same partitioning process is performed with these edges e_0^1 to e_6^1. Since we are assuming two-stage network, the switches which will be generated now must give legal connections to the slaves. In other words, since a slave must also be connected to a single switch just as a master, the edges to the same slaves must be assigned to the same partition. In this example, we assign e_0^1 to the *bypass partition* (p_0^2), and the other edges to the first partition (p_1^2). A bypass partition just bypasses the edges allocated to it to the next stage (in this example, to the slave s_0) and is not translated to a real switch. The resulting network topology is shown in Fig. 4c. Note that if we assume more stages than two, in the intermediate stages (i.e. $1 < k < N$), the edges from (to) the same switch (slaves) do not need to be partitioned into the same partition (see Section 4.3).

In CEP, a partition set corresponds to a unique connection at the stage: if two partition sets are different, it means that at least an edge in the first partition set is in the different partition in the second partition set. It means that the edge is connected to the different switch, and therefore the physical realization of the two partition sets are distinctive. Also, with a limited number of stages N, it can represent all the possible topologies. To prove it, we need to first understand that the partition set at a certain stage can represent all possible connections between the current and the previous stages. Suppose three extreme cases: (i) all edges are assigned to the same non-bypass partition, (ii) all edges are assigned to different partitions and (iii) all edges are assigned to the bypass partition. The first case corresponds to the single switch connection where the masters or switches in the previous stage are connected to the single switch in the current stage. The second case is where each edge is assigned to its dedicated switch, thus the number of switches is the maximum.[1] The last case is where all edges are assigned to the bypass partition, thus the entire stage is transparent. Then, the partition sets between these extreme cases can represent all the possible connections between the current and the previous stages. Since the design space of CEP is all the possible combinations of the partition sets of all the stages, it contains all the possible topologies for the given number of stages. Note that since a stage can entirely be bypassed, the design space with larger N comprises that with smaller N. Even though not every partition set can be translated to a legal implementation (e.g. all edges are bypassed for all stages), these illegal partition sets can easily be detected and avoided by the simple rules before the expensive evaluation phase, which will be discussed in Section 4.4.

[1] In our method, this case is prohibited since it results in trivial connection, i.e. generating 1×1 switches (see Section 4.3).

4.3. Representation for ordered design points with ERGF

As aforementioned, a partition set can be realized into a unique connection at a certain stage, and it is obviously a desired property to explore the design space efficiently. However, this benefit can be depreciated if the same partition set is repeatedly generated while exploring the design space. Therefore, we need a representing method with which we can easily know which partition set is already visited or not. In this section, we propose a representation for the partition set which is the enhanced variant of the *restricted growth function*. The original restricted growth function [24] generates a sequence S according to the following rule.

$$rg_{i+1} \leq 1 + \max(rg_1, rg_2, \ldots, rg_i), \tag{3}$$

where rg_i is an element of the sequence S, and the length of the sequence is the same as the number of elements in the set to be partitioned. In the set partitioning with the restricted growth function, the element re_i corresponds to the partition index to which the element i is allocated. For example, a sequence (0, 1, 1) represents that the first element of the set is allocated to the partition 0, and the second and the third elements to the partition 1. A restricted growth sequence (i.e. the sequence generated by restricted growth function) can be realized into a unique partition set [24]. Note that, since (0, 1, 1), (1, 2, 2) and (2, 3, 3) result in the same partition set (there are two partitions and the allocations of edges are the same), the first element of the restricted growth sequence is always fixed to the minimum value (typically 0 or 1) for all the partition sets.

However, the restricted growth function is not suitable for the partitioning of our concern because we use the partition 0 as a special purpose, the bypass partition. For example, if the first edge is not allocated to the bypass partition (i.e. rg_1 is larger than zero), then no other edges can be bypassed because the other elements cannot be less than rg_1. On the other hand, if we use zero as the minimum value of the sequence, then rg_1 should be always zero for all sequences, which means that the first edge is always bypassed.

In order to represent the partitioning of our concern, we propose the ERGF. In ERGF, we add six modifications (one relaxation and five restrictions) to the restricted growth function, as follows:

(i) Relaxation: a dummy element re_0 is assumed before the beginning of the sequence and it is fixed to zero. With this modification, the first element of the sequence rg_1 can be either zero or one, and any element in the sequence can be zero even if all the preceding elements are greater than zero.

(ii) Restriction 1: the edges from (to) the same master (slave) must be allocated to the same partition. The purpose of the restriction is to avoid the infeasible topology generation. Since a master (slave) means a ports on an IP, it cannot be connected to multiple switches.

(iii) Restriction 2: at last stage, the edges allocated to the bypass partition must be from the same node (either master node or switch node). The purpose of the restriction is to avoid the infeasible topology generation. If the restriction is violated, multiple links from different sources are directly connected to a slave, which is infeasible since a slave represents a single port with slave interface.

(iv) Restriction 3: at the last stage, the edges from master nodes (not from the switch nodes) cannot be allocated to the bypass partition. The purpose of the restriction is to avoid the infeasible topology generation. If the restriction is violated, a master can directly be connected to a slave. A direct connection can be possible if the master and the slave communicate only each other, but if so, that kind of local connection will be isolated from the complex on-chip network topology design.

(v) Restriction 4: each number except zero must be appeared at least twice in the sequence. The purpose of the restriction is to avoid generation of 1×1 switch. A partition with only one edge assigned to it will be realized into a switch with single input port and single output port, which is meaningless. Since the bypass partition passes the allocated edges to the next stage, not being realized into a switch, zero can be appeared arbitrary number of times in the sequence.

(vi) Restriction 5: the edges from a switch which has only one slave interface port cannot be allocated to the same non-bypass partition. The purpose of the restriction is to avoid generation of 1×1 switch. For example, if the first to third edges are from the same switch in the previous stage which has only one slave interface port, the ERGF bans the sequence (1, 1, 1,…).

As an example of using ERGF, the partition sets of the edges and the corresponding ERG sequences (ERG sequence means a sequence generated by ERGF) are shown in Fig. 4b and c.

By using the ERGF, we can represent only the legal partition sets. Like the original restricted growth function, each ERG sequence corresponds to a unique partition set. Also, due to its convenience of handling, we can easily control the generation of the partition sets which have been already visited. With the help of these properties of ERGF, we can efficiently explore the huge partitioning space. The formal definitions are as follows.

Definition 5. $\mathrm{RG}^k = [rg_1^k, rg_2^k, \ldots, rg_n^k]$, *where* n *is the number of elements to be partitioned, is an ERG sequence in* k*th stage.* Ψ^k *is the set of all the possible ERG sequences in the stage* k*, and* RG_i^k *is a unique element in the set. For convenience, we omit the index* k *and/or* i *if they/it are/is not of importance.*

Definition 6. V *is a* N*-tuple of the ERG sequences of all the stages such that* $V = (\mathrm{RG}^1, \mathrm{RG}^2, \ldots, \mathrm{RG}^N)$. Θ *is the set of all the possible* V*'s and* V_i *denotes a unique element in* Θ.

Then, there are one-to-one correspondences between $\mathrm{RG}_i^k \in \Psi^k$ and $P_i^k \in \Phi^k$, and $T_i \in \Omega$ and $V_i \in \Theta$.

Definition 7. *An ERG sequence* RG_i *is greater than* RG_j *(*$\mathrm{RG}_i > \mathrm{RG}_j$*) if there exists* $1 \le q \le n$ *such that* $rg_{i,q} > rg_{j,q}$ *and* $rg_{i,r} \ge rg_{j,r}$ *for all* $r < q$*, where* $rg_{i,q}$ *is the* q*th element of* RG_i *and* n *is the length of the sequence. It can be also said that* RG_j *is smaller than* RG_i*. Then, incrementing an ERG sequence means changing the sequence so that the new sequence is greater than the current sequence. Also incrementing an ERG sequence by one means incrementing the sequence so that there exists no other sequence which is greater than the current sequence and smaller than the new sequence.*

Incrementing RG^k is analogous to incrementing an n-ary number, where the first element (rg_1) corresponds to the most significant digit, and the last element (rg_n, if there are total n edges) to the least significant digit. For example, if there are four edges to be partitioned, the smallest sequence is (0, 0, 0, 0) and the largest is (1, 2, 2, 1) (by Restriction 4) unless it violates the other restrictions of ERGF.

Definition 8. *Incrementing* V *means incrementing any* RG^k *in* V*. Incrementing* V *by one is to find the largest* k' *in the range* $1 \le k \le N$*, increment* $\mathrm{RG}^{k'}$ *by one, and reset* RG^k*'s for* $k' < k \le N$.

Incrementing V is also analogous to incrementing an n-ary number where the first element of the tuple RG^1 corresponds to the most significant digit and the last element RG^N to the least significant digit.

4.4. Topology design space exploration

Here we introduce how to explore the network topology design space with CEP and ERGF. Basically, we explore the topology design space by incrementing RG^k's and V. Since RG^k's and V are always in their increasing direction, the same $V_i \in \Theta$ is visited only once.

The pseudo-code for the synthesis procedure is shown in Fig. 5. The procedure takes the application's communication requirement (i.e. CRG), the physical information of the switches and the maximum number of stages N as inputs. The synthesis starts by generating the initial ERG sequence RG^k's for each stage $1 \le k \le N$, and the corresponding topology (lines 1–3). The initial RG^k's are all zero sequences, except the RG^N is all one sequence. This set of sequences represents the single switch topology, where the edges bypass the first $N - 1$ stages and are connected to the only switch in the last stage. Note that these initial ERG sequences make V the smallest. After the

```
Procedure CEP_kernel
library  CRG, $XF$, $XP$, $XA$, $\rho$
input    maximum number of stages $N$
         effort parameter $\gamma$
output   network topology

1:  $E^0 = E$ in CRG;
2:  generate initial $RG^k$'s for $1 \leq k \leq N$;
3:  generate initial topology;
4:  while (!$V$ →maximum_reached)
5:    for ($k = N$; $k \geq 1$; $k--$)
6:      if (!$RG^k$ →maximum_reached)
7:        $k' = k$;
8:        break;
9:      endif
10:   endfor
11:   for ($k = k'$; $k \leq N$; $k++$)
12:     increment($RG^k$, $\gamma$);
13:     allocate $e \in E^{k-1}$ to switches;
14:     generate $E^k$;
15:     if ($k \neq N$) reset $RG^{k+1}$;
16:   endfor
17:   evaluate_topology;
18:   if current topology is the best then save it;
19: endwhile
```

FIGURE 5. Topology synthesis procedure with CEP and ERGF.

initial topology is generated, it searches which stage is not fully searched, i.e. which RG^k can be incremented, from the last stage to the first stage (lines 5–10). It is analogous to incrementing the less significant digit prior to the more significant one when incrementing a number. If the stage is found, let's say the stage k', the sequence of that stage RG$^{k'}$ is updated, which means that the connections between the stage $k' - 1$ and k' is changed. When the RG$^{k'}$ is updated, the RG^k's for $k' < k$ are reset to the minimum sequence since the connections of the previous stages are changed (lines 11–16). As an analogy, it is similar to incrementing a decimal number 109 to 110, where the last digit is reached its maximum, so increment the second digit by one, and then last digit is reset to the smallest value. At the end of the loop from lines 11 to 16, a candidate topology is generated. At line 17, the candidate topology is first examined for its feasibility, i.e. the bandwidth and latency constraint satisfaction, and evaluated for the cost such as area and power consumption. According to the cost function given by the designer, the best topology is saved in line 18. This operation is repeatedly performed until the V is reached its maximum.

Even though we can explore the partition sets of each stage only once without revisiting already visited one, the number of partition sets increases dramatically as the number of elements increases. In the set partitioning problem, the number of partition sets of the set having n elements equals nth Bell Number [24], which is greater than 4M when $n = 10$. Therefore, exploring the entire search space of CEP will be very time-consuming for large problems, even though a lot of partition sets are pruned by ERGF. Instead, we randomly increment the ERG sequence based on probability, not increment one by one. Since this random increment misses some design space, we compensate it by performing the synthesis procedure for several iterations.

The random increment of the ERG sequence is controlled by the effort parameter γ. Two different properties of the exploration are determined by γ: (i) the probability of incrementing the ERG sequences one by one or randomly, (ii) the probability of each element of the ERG sequence to be incremented when the random increment is used. For the first property, γ directly indicates the probability of incrementing the sequence one by one, thus the random increment occurs with the probability of $1 - \gamma$. For the second property, γ determines the probability of each element to be incremented with the following relation.

$$p(rg_i) = \alpha \times i^{\gamma}, \tag{4}$$

where $p(re_i)$ is the probability of rg_i to be incremented by one, and α is the fraction to make $\sum_i p(rg_i) = 1$. Note that the exploration skips larger space with smaller γ. If $\gamma = 1$, the exploration becomes exhaustive search. On the other hand, if $\gamma = 0$, the ERG sequences are always incremented randomly and all the elements are incremented with the uniform probability.

The synthesis procedure for $\gamma < 1$ is shown in Fig. 6. In the procedure, we perform the procedure CEP_kernel in Fig. 5 for multiple times to compensate the randomness. However, this can fade the benefit of the proposed exploration method by allowing the revisit to the already visited points throughout the iterations. To resolve it, we isolate the design space in a certain iteration by forcing some elements in the ERG sequences to certain values. For example, if we perform five iterations, the first two elements of RG^1 can be forced to (0,0), (0,1), (1,0), (1,1) and (1,2) in each iteration, respectively. We call these

```
Procedure CEP_rand
input   $N, \gamma < 1$
        number of iterations $K$
output network topology

1:  for ($n = 2$; $n \leq N$; $n++$)
2:    for ($k = 1$; $k \leq K$; $k++$)
3:      CEP_kernel($n, \gamma$);
4:      if current topology is the best then save it;
5:    endfor
6:    if a feasible solution is found then break;
7:  endfor
```

FIGURE 6. Synthesis procedure for $\gamma < 1$.

TABLE 1. Effect of N.

	Area (mm^2)		Power (mW)		Time (s)	
N	2	3	2	3	2	3
App I	fail	0.472	fail	17.15	0.095	0.7
App II	0.495	0.491	8.027	8.015	1.401	27.195
App III	0.487	0.491	2.991	3.038	3.789	43.968
App IV	0.541	0.542	3.249	3.328	9.667	76.417

sequences as *compulsion sequences*. If the number of iterations exceeds the number of compulsion sequences, we can either increase the length of the compulsion sequence or wrap around to the first sequence. In latter case, the revisit problem can occur, but the worst case number of revisit to the same point is reduced to the number of iterations divided by the number of possible compulsion sequences (five in the above example).

Also, given the maximum number of stages N, the procedure CEP_kernel is repeatedly performed while increasing the number of stages from 2 to N until a feasible solution is found, so that the design space with smaller number of stages (or cascading depth denoted as n) can be explored first.[2] N, the maximum number of stages, can be determined by the experience of user. However, it can be set to the infinity when the user is ignorant of the given design and/or our method. Hence, CEP_rand finds a feasible solution which has the smallest number of stages among all possible feasible solutions. Note that we do not need CEP_rand when $\gamma = 1$, since this setting drives the CEP_kernel to exhaustively search every possible topology candidate in the design space with N number of stages. The basic rationale of CEP_rand is that smaller hop count is favored for low latency and low power consumption from the network's perspective. The experimental results shown in Table 1 support the rationale, since the increase of the cascading depth marginally improves the solution quality, when our method finds a feasible solution with a smaller cascading depth.

To summarize, our method provides a control mechanism to trade off the solution quality and the computation time by controlling the parameters—γ, N, and the number of iterations like other heuristic methods. For instance, the work in [6] uses a simulated annealing-based exploration method which takes parameters such as acceptance probability and termination condition, and the work in [14] uses tabu search method which takes the size of tabu list as its parameter.

[2]We ignore $N = 1$ since it is trivial. If $N = 1$, the problem is just to distinguish disjoint communications (i.e. local traffics) from each other in CRG. If there is no disjoint communication, the single switch topology is the only legal solution.

5. EXPERIMENTAL RESULTS

5.1. Experimental settings

We evaluate the proposed method in terms of synthesis time and the solution quality. We compare the proposed method (CEP) with the methods in [7] (MILP) [8] (MIRO) and [25] (TGE + GEN). The first two methods represent the topology synthesis problem with adjacency matrices and solve it by MILP. The work in [8] combines the MILP with the input-merging technique and solves the large problem in a divide-and-conquer fashion, to reduce the synthesis time. TGE+GEN represents the topology design space with traffic group encoding and explores the design space with genetic algorithm. Since the design space of TGE+GEN involves the insertion of frequency and data-width conversion bridges, while the other methods assume the single frequency and single data-width of the network, we manipulate the inputs to TGE + GEN so that the bridges are not generated, and neglect the cost of the components which are not considered in our design space. We will also show the effect of the design parameters—the maximum number of stages N, effort parameter γ and the number of iterations, on the solution quality and the synthesis time.

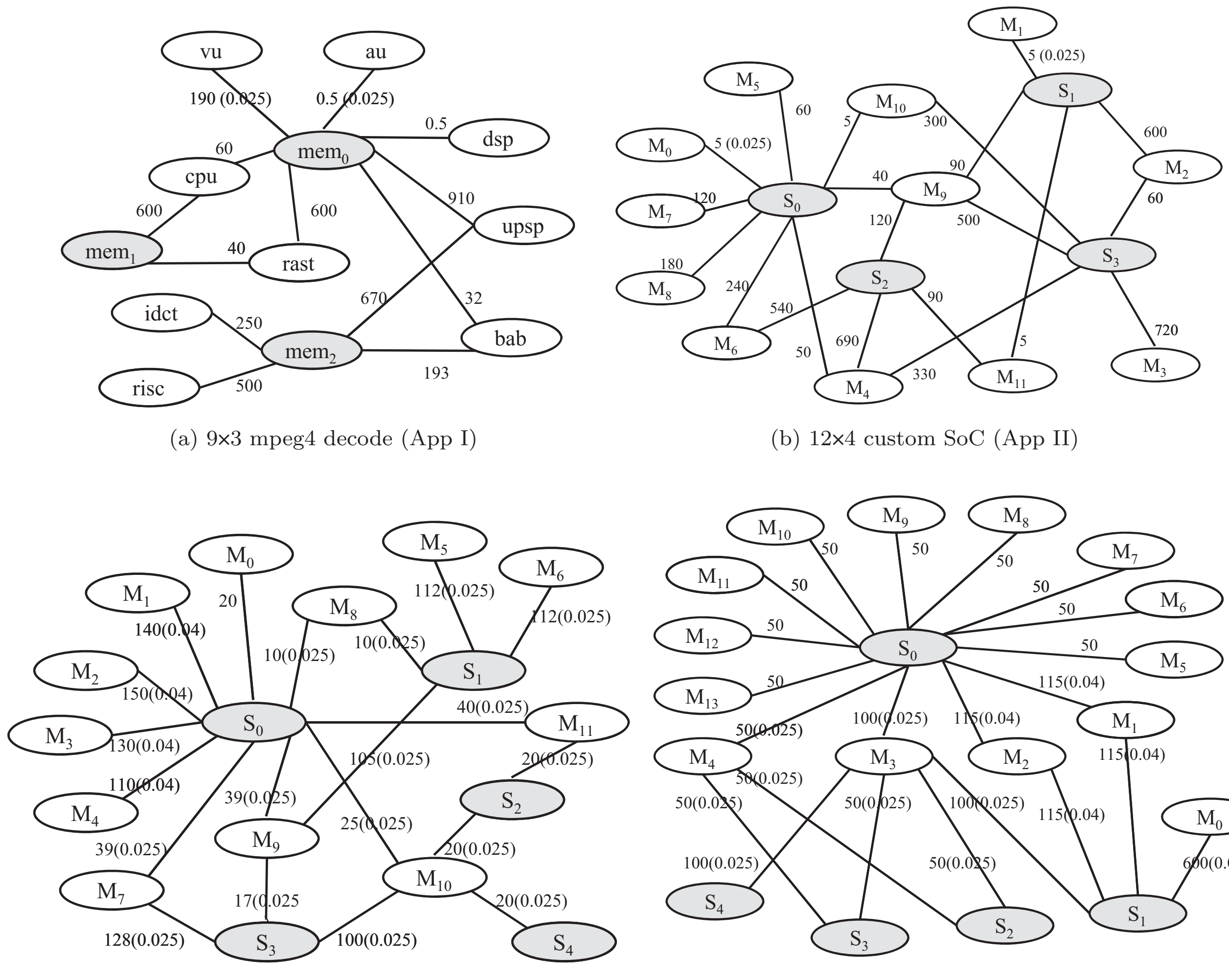
We apply the methods to four real-world applications which were used in the compared works. The first one is an mpeg4 decoder application (App I) in [10],[3] which has nine masters and three memories. The second one is an multimedia SoC application (App II) which has 12 masters and 4 slaves [7]. The masters and slaves include video codecs, graphics IPs, ARM11, DDR and NAND Flash memory. The third application is a mobile multimedia player (App III) having 12 masters and 5 slaves [25], and the fourth is a mobile application processor application (App IV) with 14 masters and 5 slaves [25]. The CRGs for the four applications are shown in Fig. 7. The white ovals are masters and shaded ovals are slaves. The solid lines among them are edges, where the denoted values are the bandwidths in MB/s and those in the parenthesis are latencies in microseconds.

For the physical information of the switches, we generated Verilog RTL codes for AXI [23] crossbar switches, sizing from 1×2 to 9×16, and pipeline register, and synthesized them using Synopsys Design Compiler with 90 nm process library. CEP, MILP and MIRO are implemented in C++, and TGE+GEN is implemented in Java. We performed the topology synthesis for two design goals: minimizing the area and the power consumed by the crossbars and pipeline components. The power model in [25] is used for the power calculation.

5.2. Sensitivity analysis on design parameters

In this section, we examine the sensitivity of the proposed method on the design parameters, i.e. the maximum number

[3]The bandwidth is arbitrarily divided into read and write, and the latencies are also added arbitrarily.



(a) 9×3 mpeg4 decode (App I)

(b) 12×4 custom SoC (App II)

(c) 12×5 mobile multimedia player (App III)

(d) 14×5 mobile application processor (App IV)

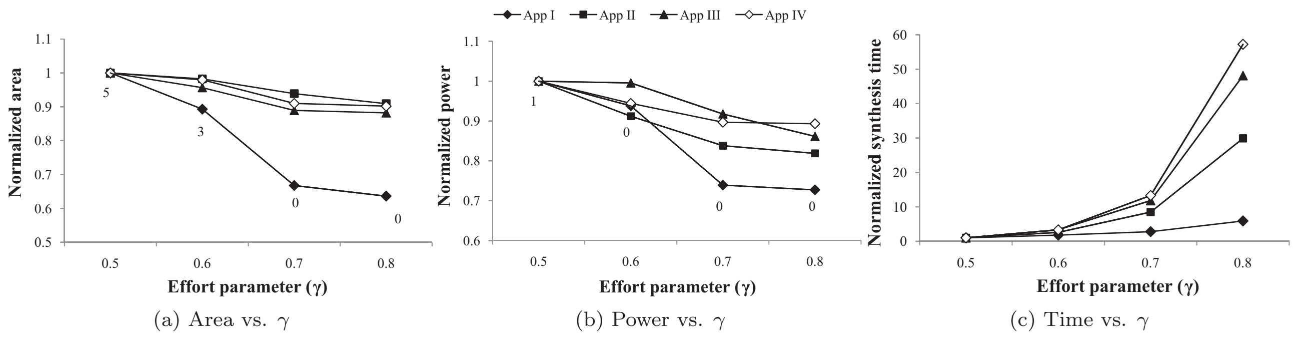
FIGURE 7. CRGs of the four applications.

of stages N, the effort parameter γ and the number of iterations when random increment is used.

Figures 8a–c show the effect of γ on the solution quality and the synthesis time, with the number of iterations fixed to 15. N is set to 2 for all applications except for App I. For App I, N is set to 3 since no solution is found with smaller N. Even though the number of masters and slaves of App I is the smallest among the applications, it requires the frequency of 340.75 MHz (the channel is 32 bit wide) to satisfy its bandwidth requirement, while the other applications require <250 MHz. The reported area values are from the synthesis with the objective of area minimization, while the power values are with power minimization. We run the synthesis 10 times for each application[4] and took their average. In Figure 8a and b, a number is annotated to each evaluation point of App I to indicate the number of 'fail' runs out of 10 runs. A run is said to be fail, if it does not find any feasible solution. Rather than excluding the fail runs for computing the average area, we penalize the fail runs by setting its area and power to the twice the maximum value among the non-fail runs. Except App I, no fail happened for the other applications.

As γ increases from 0.5 to 0.8, the area (power) is saved by 36.4 (37.3), 9.1 (18.1), 11.8 (13.8) and 9.8% (10.7%) for App I, App II, App III and App IV, respectively. Except App I where the penalty for the fail runs greatly affects the average value, 10.2% (14.2%) of improvement is achieved on average by increasing γ from 0.5 to 0.8. While the improvement is <20%, the synthesis time is increased by up to 57.2× (App IV) and 35.3× on average, as shown in Fig. 8c. Thus, it is necessary to trade off the solution quality and the synthesis time. We pick the γ of 0.7 as a tradeoff point. In this point, the solution quality is only degraded by 2.5% and the synthesis speedup is 3.5, on average against when γ is 0.8.

[4]It means that the procedure CEP_rand in Fig. 6 is performed 10 times.



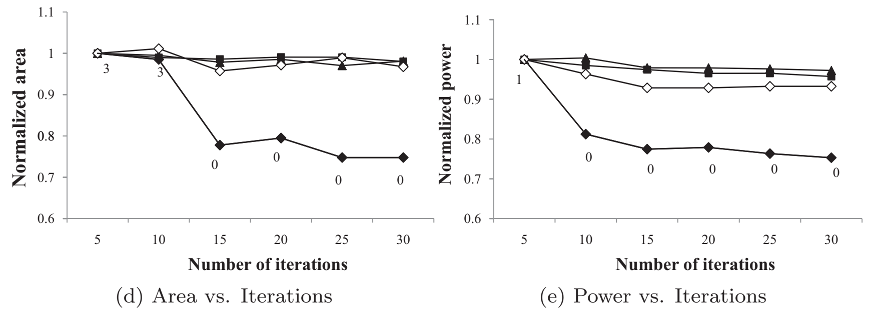


FIGURE 8. Sensitivity on the design parameters, γ and the number of iterations.

Next, we examined the effect of the number of iterations on the solution quality. We fix the first two elements of the ERG sequence in the first stage, i.e. rg_1^1 and rg_2^1. The possible pairs of (rg_1^1 and rg_2^1) are (0,0), (0,1), (1,0), (1,1) and (1,2). Each of them is allocated to each iteration to force the five iterations to search five different sub-design spaces. Since this setting dedicates each iteration to each sub-design space, there is no chance to visit the same design point with our method. However, if the number of iterations is larger than 5, some of the sub-design spaces will be explored multiple times. More precisely, the exploring frequency of each sub-design space is the number of iterations over the number of sub-design spaces. If the exploring frequency of a sub-design space is larger than 1, a design point in the sub-design space may be visited more than once by different iterations. For this reason, the efficiency of the preceded iteration is higher than that of the succeeded iteration. Figure 8d well supports this claim. Except App I, N and γ are fixed to 2 and 0.7, respectively. For App I, we set N to 3 and γ is unchanged. Throughout 10 runs, we performed the area and power sensitivity analyses of the proposed method with respect to the number of iterations as shown in Fig. 8d and e, respectively. As in Fig. 8a and b, the number of fail runs is annotated to each evaluation point of App I. Increasing the number of iterations improves the area $<5\%$ and power $<10\%$ in the applications other than App I. The observation indicates that single iteration for each sub-design space produces a fairly good solution thanks to the tight design space. It also indicates that the efficiency of the succeeded iterations is much lower than that of the first iteration due to the revisits of already explored design points. Note that the fail penalty at low number of iterations distorts the same observation in App I.

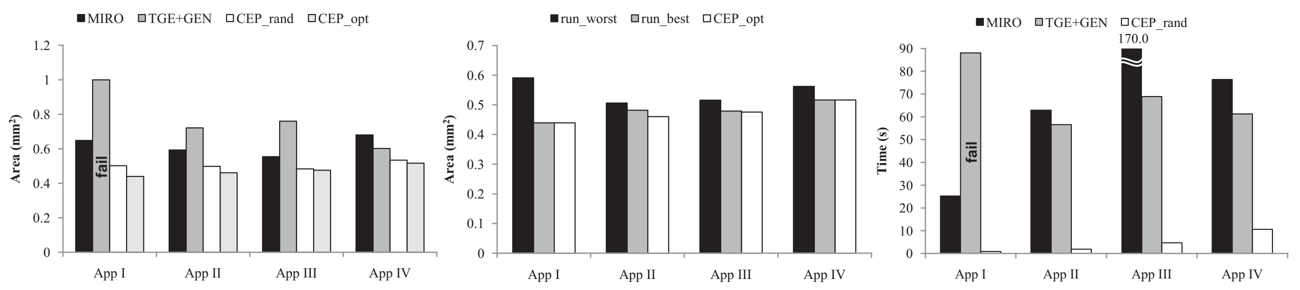
TABLE 2. Variance over ten synthesis runs.

	Norm. area			Norm. power			Norm. time		
	min	max	std.	min	max	std.	min	max	std.
App I	0.88	1.18	0.10	0.88	1.11	0.07	0.85	1.14	0.08
App II	0.97	1.02	0.01	0.98	1.06	0.03	0.75	1.19	0.12
App III	0.99	1.07	0.02	0.89	1.11	0.07	0.92	1.08	0.05
App IV	0.97	1.05	0.03	1.00	1.04	0.01	0.91	1.23	0.09

The results from the 10 synthesis runs can vary due to the randomness which exists when γ is <1. If the variance is too large, the reliability of the synthesis method can be depreciated. Table 2 shows the minimum values, maximum values and standard deviations of the area, power and computation time obtained over the 10 synthesis runs. The area and power are

TABLE 3. Synthesis time (in s) comparison of CEP_opt and MILP.

App	App I	App II	App III	App IV
N	3	2	2	2
MILP	71 280	82 128	Timeout	Timeout
CEP_opt	3755	148	571	33 942



(a) Area (vs. existing methods) (b) Area (vs. best and worst) (c) Synthesis time

FIGURE 9. Synthesis quality and time comparison.

normalized to their average, and the standard deviation is obtained for the normalized values. The standard deviations of the synthesis quality (i.e. area and power) are within 0.10, and those of the synthesis time are within 0.12 for all the test cases. These deviations are practically acceptable and it proves the reliability of the proposed method.

Finally, we examined the effect of N by performing synthesis, while changing N from 2 to 3. When $N = 3$ is used, we manipulated the procedure CEP_rand in Fig. 6 such that the loop for $n = 3$ is performed even if a feasible solution has already been found with $n = 2$, in order to examine the effect of searching larger design space. We fixed γ and the number of iterations to 0.7 and 15, respectively. The result is shown in Table 1. As aforementioned, no solution was found for App I when N is 2, while it is found when N is 3. On the other hand, we could not find the benefit of using stages more than 2 for the other applications, while the synthesis time is increased by 7.4× (App I) to 19.4× (App II). Therefore, N of 2 is enough for App II–IV.

In further experiments, we take the values 2, 0.7 and 15 for the parameters N, γ, and the number of iterations, respectively, except App I for which N of 3 is used.

5.3. Comparison with the existing methods

In this section, we compare the proposed method with the existing methods MILP, MIRO and TGE+GEN in terms of the area of the synthesized network and the elapsed time to perform the synthesis. The power consumption is not compared here since MILP and MIRO do not support the power calculation. However, we believe that the area and time are enough to compare the efficiency of the synthesis methods.

In order to measure how efficiently our method defines the design space, we performed the exhaustive search for the design space defined by setting γ to 1 in our method (CEP_opt) and compared its synthesis time with that of MILP. Since both methods find the optimal solution for the given number of stages, only the synthesis time is compared. The result is shown in the Table 3. We set the timeout deadline to 24 h. The result is shown in Table 3. The second row of Table 3 is the maximum number of stages N. The result shows that MILP finds the solutions only for the first two applications, but fails to find a solution within the given timeout deadline for the other applications. Moreover, its synthesis time is huge, 22 h on average, even for App I and App II. On the other hand, CEP_opt takes much smaller amount of time than MILP, completing the search in 2.5 min and 9.4 h for App II and App IV, respectively. The great time saving is mainly due to the efficiently defined search space. While the design space of MILP is defined by the several adjacency matrices where a large portion of design points yields illegal solutions, any design point in our design space can be realized into a legal topology.

Next, we compare the synthesis time and the solution quality of the three heuristic (i.e. not guaranteeing the optimality) methods, CEP with random increment (CEP_rand), MIRO and TGE + GEN. The result is shown in Fig. 9.[5] First, Fig. 9a shows the synthesized area of the three methods and the optimal solution found by CEP_opt. The value for CEP_rand is the averaged value over 10 runs. The result shows that CEP_rand gives better solutions than the compared methods. Specifically, the area is saved by up to 22.6 and 18.3% on average, over MIRO, and by up to 36.4 and 26.2% on average, over TGE+GEN. Note that TGE+GEN failed to find a feasible solution for App I, so it is not counted in calculating the average improvement ratio. Compared with CEP_opt, CEP_rand shows up to 14% (App I) and on average 6.8% of quality degradation. To appreciate the solution quality deviations of 10 runs, we compared the best and worst cases of CEP_rand with CEP_opt, which is shown in Fig. 9b. It shows that CEP_rand found the same solution with CEP_opt for App I and App IV during the 10 runs. Also, the average area of CEP_rand in the best case is slightly larger (2.6%) than that of CEP_opt. Note that running the program 10 times takes <2 min for all the applications.

More importantly, CEP_rand outperforms the compared methods with much less synthesis time as shown in Fig. 9c. Quantitatively speaking, the speedup of CEP_rand over MIRO is up to 36.5 and 26.9 on average. Also, the speedup over the TGE+GEN is up to 104.6 and 38.9 on average.

[5]We do not compare them for power, since some of the methods compared do not provide the power optimization.

6. CONCLUSION

We proposed a novel topology synthesis method and its efficiency was proved by comparing it with the existing methods. The strength of our method is attributed to three factors—a tight design space, a representing method of the design space and the corresponding exploration method. Even though many works have been done in the on-chip network topology synthesis, few works have covered all these three factors together in a single framework. The proposed design space definition, CEP in conjunction with ERGF, contains only the legal solutions, and excludes some trivial solutions such as topologies having 1×1 switch(es). Also, the proposed exploration method based on ERGF resolves the revisiting problem which has been a major drawback of many previous methods. The experimental results indicate that the proposed method outperforms the compared methods in terms of both synthesis quality and speed, by showing up to 49.8% and $104.6\times$ of quality and speed improvement, respectively.

FUNDING

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] Pasricha, S., Dutt, N. and Ben-Romdhane, M. (2006) Constraint-driven Bus Matrix Synthesis for MPSoC. *Proc. ASPDAC 2006*, Yokohama, Japan, January, 24–27, pp.30–35. IEEE Press, Piscataway, NJ, USA.

[2] Murali, S. and De Micheli, G. (2005) An Application-Specific Design Methodology for STbus Crossbar Generation. *Proc. DATE 2005*, Munich, Germany, March, 7–11, pp. 1176–1181. IEEE Computer Society, Washington, DC, USA.

[3] Pasricha, S. and Dutt, N. (2006) COSMECA: Application Specific Co-synthesis of Memory and Communication Architectures for MPSoC. *Proc. DATE 2006*, Munich, Germany, March 6–10, pp. 700–705. European Design and Automation Association, 3001 Leuven, Belgium.

[4] Murali, S., Benini, L. and De Micheli, G. (2007) An application-specific design methodology for on-chip crossbar generation. *IEEE Trans. CAD*, **26**, 1283–1296.

[5] Pasricha, S., Dutt, N. and Kurdahi, F.J. (2009) Dynamically Reconfigurable On-Chip Communication Architectures for Multi Use-Case Chip Multiprocessor Applications. *Proc. ASPDAC 2009*, Yokohama, Japan, January 19–22, pp. 25–30. IEEE Press, Piscataway, NJ, USA.

[6] Yoo, J., Lee, D., Yoo, S. and Choi, K. (2007) Communication Architecture Synthesis of Cascaded Bus Matrix. *Proc. ASPDAC 2007*, Yokohama, Japan, January 23–26, pp. 171–177. IEEE Computer Society, Washington, DC, USA.

[7] Jun, M., Yoo, S. and Chung, E.Y. (2008) Mixed Integer Linear Programming-based Optimal Topology Synthesis of Cascaded Crossbar Switches. *Proc. ASPDAC 2008*, Seoul, Korea, March 21–24, pp. 583–588. IEEE Computer Society Press, Los Alamitos, CA, USA.

[8] Jun, M., Yoo, S. and Chung, E.Y. (2009) Topology synthesis of cascaded crossbar switches. *IEEE Trans. CAD*, **28**, 926–930.

[9] Srinivasan, K., Chatha, K.S. and Konjevod, G. (2006) Linear programming based techniques for synthesis of network-on-chip architectures. *IEEE Trans. TVLSI*, **14**, 407–420.

[10] Srinivasan, K., Chatha, K.S. and Konjevod, G. (2005) An Automated Technique for Topology and Route Generation of Application Specific On-chip Interconnection Networks. *Proc. ICCAD 2005*, San Jose, CA, USA, November 6–10, pp. 231–237. IEEE Computer Society, Washington, DC, USA.

[11] Murail, S., Meloni, P., Angionili, F., Atienza, D., Carta, S., Benini, L., De Micheli, G. and Raffo, L. (2006) Designing Application-specific Networks on Chips with Floorplan Information. *Proc. ICCAD 2006*, San Jose, CA, USA, November 5–9, pp. 355–362. ACM, New York, NY, USA.

[12] Murali, S., Benini, L. and De Micheli, G. (2005) Mapping and Physical Planning of Networks-on-chip Architectures with Quality-of-service Guarantees. *Proc. ASPDAC 2005*, Shanghai, China, January 18–21, pp. 27–32. ACM, New York, NY, USA.

[13] Pinto, A., Carloni, L.P. and Sangiovanni Vincentelli, A.L. (2009) A Methodology for Constraint-driven Synthesis of On-chip Communications. *IEEE Trans. CAD*, **28**, 364–377.

[14] Chan, J. and Parameswaren, S. (2008) NoCOUT : NoC Topology Generation with Mixed Packet-switched and Point-to-Point Networks. *Proc. ASPDAC 2008*, Seoul, Korea, March 21–24, pp. 265–270. IEEE Computer Society Press, Los Alamitos, CA, USA.

[15] Yan, S. and Lin, B. (2008) Application-specific Network-on-chip Architecture Synthesis based on Set Partitions and Steiner Trees. *Proc. ASPDAC 2008*, Seoul, Korea, March 21–24, pp. 277–282. IEEE Computer Society Press, Los Alamitos, CA, USA.

[16] Lahiri, K., Raghunathan, A. and Lakshminarayana, G. (2001) LOTTERY — BUS: A New High Performance Communication Architecture for System-on-Chip Designs. *Proc. DAC 2001*, Las Vegas, NV, USA, pp. 15–20. ACM, New York, NY, USA.

[17] Jun, M., Bang, K., Lee, H.J., Chang, N. and Chung, E.Y. (2007) Slack-based Bus Arbitration Scheme for Soft Real-time Constrained Embedded Systems. *Proc. ASPDAC 2007*, Yokohama, Japan, January 23–26, pp. 159–164. IEEE Computer Society, Washington, DC, USA.

[18] Lin, B.C., Lee, G.W., Huang, J.D. and Jou, J.Y. (2007) A Precise Bandwidth Control Arbitration Algorithm for Hard Real-time SoC Buses, *Proc. ASPDAC 2007*, Yokohama, Japan, January 23–26, pp. 165–170. IEEE Computer Society, Washington, DC, USA.

[19] Lu, R. and Koh, C.K. (2003) SAMBA-BUS: A High Performance Bus Architecture for System-on-chips. *Proc. ICCAD 2003*, San Jose, CA, USA, November 9–13, pp. 8–12. IEEE Computer Society, Washington, DC, USA.

[20] Sekar, K., Lahiri, K., Raghunathan, A. and Dey, S. (2005) FLEXBUS: A High-performance System-on-chip Communication Architecture with a Dynamically Configurable Topology. *Proc. DAC 2005*, Anaheim, CA, USA, June 13–17, pp. 571–574. ACM, New York, NY, USA.

[21] Medardoni, S., Ruggiero, M., Bertozzi, D., Benini, L., Strano, G. and Pistritto, C. (2007) Capturing the Interaction of the Communication, Memory and I/O Subsystems in Memory-centric Industrial MPSoC Platforms. *Proc. DATE 2007*, Nice, France, April 16–20, pp. 660–665. EDA Consortium, San Jose, CA, USA.

[22] Driniȅ, M., Kirovski, D., Megerian, S. and Potkonjak, M. (2006) Latency-guided on-chip bus network design, *IEEE Trans. CAD*, **25**, 2663–26673.

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

[24] Milne, S.C. (1978) A q-analog of restricted growth functions, Dobinski's equality, and Charlier polynomials. *Trans. Am. Math. Soc.*, **245**, 89–118.

[25] Yoo, J. Yoo, S. and Choi, K. (2009) Topology/floorplan/pipeline co-design of cascaded crossbar bus. *IEEE Trans. VLSI*, **17**, 1034–1047.