AN AREA-EFFICIENT PARTIALLY RECONFIGURABLE CROSSBAR SWITCH WITH LOW RECONFIGURATION DELAY

Chin Hau Hoo and Akash Kumar

Department of Electrical and Computer Engineering
National University of Singapore, Singapore
Corresponding author email address: eleak@nus.edu.sg

ABSTRACT

With the increasing number of processors in Multiprocessor System-on-Chips (MPSoCs), Network-on-Chips (NoCs) are replacing conventional buses as the interprocessor communication architecture. Since different use cases might be running on MPSoCs, there is a need for dynamically reconfigurable NoC. However, most dynamically reconfigurable NoCs have a large area overhead due to the additional reconfiguration logic. While recently some dynamically reconfigurable NoCs have been proposed based on partial reconfiguration (PR), they have high reconfiguration delay and require off-line bitstream generation for all possible scenarios. The problem lies with the design of the crossbar switch, which is the fundamental component of a NoC. In this paper, a novel partially reconfigurable crossbar switch design with low area requirement, low reconfiguration delay and runtime bitstream generation is presented. The crossbar switch is built from lookup tables (LUTs), and reconfiguration is done by modifying the LUTs' content through PR. Reconfiguration delay is minimized by constraining the placement of the LUTs into the least number of configurable logic block columns and identifying the configuration frames that are responsible for LUTs' content. The novel crossbar switch design achieves an area saving of up to 84% and reconfiguration delay minimization of up to 78%. It can be used to realize any network topology, and Clos, Benes and single stage crossbar topologies are evaluated in the paper.

Index Terms: Reconfigurable architectures, Network-on-Chip, partial reconfiguration, crossbar switch, spatial division multiplexing.

1. INTRODUCTION

With increasing chip density, it is now possible to pack more transistors into a single chip as dictated by Moore's Law. This results in the creation of Multiprocessor System-on-Chip (MPSoC) where processors, memory, and other IO devices are fabricated onto a single chip. As the number of processors in an MPSoC increases, the bottleneck switches from computation to communication [1]. In fact, traditional bus is no longer able to meet the bandwidth and latency requirement of MPSoC due to its decrease in performance as the number of processors contending for it increases. Therefore, Network-on-Chip (NoC) has been proposed as an alternative to buses.

Early NoC designs are based on packet switching that provides only best-effort service. In other words, the NoCs do not provide guaranteed bandwidth and bounded latency for data transfers. Guaranteed quality of service is usually achieved through resource reservation with techniques such as time division multiplexing (TDM) and spatial division multiplexing (SDM). The NoCs proposed in [2] and [3] are examples of SDM based NoC while Æthereal [4] and Nostrum [5] are examples of TDM based NoC.

In addition to providing guaranteed quality of service, NoCs have to be dynamically reconfigurable because MPSoCs need to handle multiple use cases [6]. A use case is a combination of applications that are running concurrently on an MPSoC. The use cases often have different requirements in terms of connectivity, bandwidth and latency, and NoCs have to cater to those requirements. Besides, it is desirable that applications that are common between two use cases are not interrupted during reconfiguration. In multimedia systems such as smart phones, one might expect the music to continue playing when switching from web browsing to word processing. Therefore, dynamically reconfigurable NoCs should allow glitch-free use case switching as well.

There are two methods of achieving dynamic reconfiguration. The first and more widely adopted approach involves adding reconfiguration logic to the network interfaces (NIs) and routers but the logic consumes extra area [7][8][9]. The second and more recent method requires the use of partially reconfigurable FPGA devices such as the Xilinx Virtex series. Partial reconfiguration (PR) allows some logic blocks on an FPGA to be programmed at runtime without affecting the operation of other logic blocks [10]. However, current PR based NoCs have high reconfiguration delay, require large storage space for partial bitstreams and can only cater to predefined use cases. Besides, some network topologies used by PR based NoCs do not allow for glitch-free use case switching. Benes network is an example of such topology.

Contributions: In this paper, a novel partially reconifigurable crossbar switch design is presented as a solution to the aforementioned problems. The crossbar switch consists

of multiplexers built from lookup tables (LUTs), and connections are set up by modifying the LUTs' content through PR. This results in significant saving in terms of area because dedicated control logic in the crossbar switch is no longer required. An $\mathcal{O}(logN)$ algorithm is also described to set up connections in the crossbar switch. Finally, methods of minimizing the reconfiguration delay of the crossbar switch are proposed.

The rest of the paper is organized as follows. Section 2 provides some background on the concepts that will be used in the rest of the paper. Section 3 describes various related works and their limitations. Section 4 describes the architecture of the new crossbar switch. Section 5 evaluates the performance of the new architecture. Section 6 concludes the paper and highlights future work.

2. BACKGROUND

2.1. Introduction to SDM-based Networks

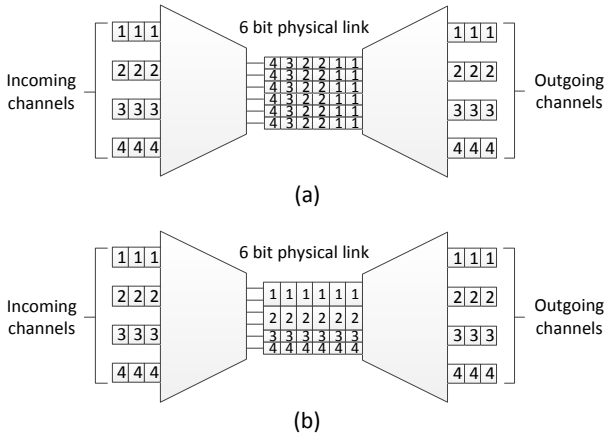

Fig. 1. Comparison between TDM (top) and SDM (bottom)

To provide guaranteed bandwidth and latency, virtual connections (VCs) are created between processing elements (PEs). VCs are built upon the concept of either TDM or SDM [3]. Figure 1 shows how a 6-bit physical link can be shared between four incoming channels using TDM and SDM. In TDM, a time slot is allocated to each incoming channel such that each channel has exclusive access to the physical link during its allocated time slot. The duration of time slot can vary among the incoming channels to allow different channels to have different amount of bandwidth. In SDM, a subset of the physical link is allocated exclusively to an incoming channel for the whole lifetime of the connection. However, the data has to be serialized before transmission. For example, in Figure 1(b), the data from channel 1 and 2 are serialized to 2-bit while the data from channel 3 and 4 are serialized to 1-bit.

In order to implement TDM, a multiplexer is commonly used. The configuration of the multiplexer has to be changed every time slot to grant an incoming channel access to the physical link [3]. If the multiplexer was made partially reconfigurable, the FPGA would have to be reconfigured every time slot. In contrast, a SDM network needs to be configured only during the setup of the connection. Therefore, SDM is chosen over TDM to provide guaranteed bandwidth and latency for PR based NoC because the non-trivial delay of PR is amortized over the connection duration.

2.2. Non-Blocking Networks

There are two types of non-blocking network, strictly non-blocking and rearrangeably non-blocking. A strictly non-blocking network allows a new connection to be set up without rerouting existing connections. On the other hand, a rearrangeably non-blocking network may require existing connections to be rerouted before a new connection can be setup.

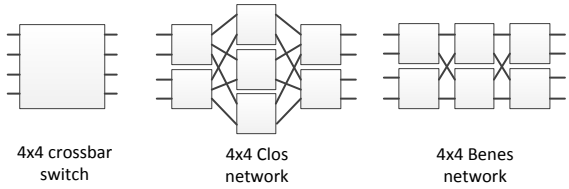

Fig. 2. 4×4 non-blocking networks

Figure 2 shows how a 4×4 network can be built as a single crossbar switch, Clos network [11], and Benes network [12]. A crossbar switch and Clos network are examples of strictly non-blocking network while Benes network is an example of a rearrangeably non-blocking network. Clos network is a three-stage network that was originally designed to reduce the total number of cross points of a telephone switching network to less than that of a single stage crossbar switch of equivalent size. It can be characterized by a triple (m, n, r) where m is the number of middle stage switches, n is the number of input (output) ports on each input (output) switch, and r is the number of input and output switches. Clos [11] has proven that his network is strictly non-blocking if and only if $m \geq 2n - 1$.

3. RELATED WORKS

Joseph *et al.* [2] describe a dynamically reconfigurable SDM based NoC with a circuit switched data network and a packet switched control network. The control network is responsible for configuring the routers in the data network and the NIs. Focus was placed on reducing the amount of resource used by the NIs and the routers by replacing a 32-to-*N* bit serializer with *N* 32-to-1 bit serializers and a *k*-way router with a 1-way router. However, the routers still incur area overhead due to the need to support dynamic reconfiguration. As described later in Section 5.1, the area overhead is eliminated by making the router partially reconfigurable. In addition, the 1-way router design limits routing flexibility, resulting in cases where a use case's connection require-

ment cannot be satisfied. Besides, the link allocation has to be done at design time. Therefore, the limitations prevent the NoC from adapting to new runtime requirements.

DyNoC [13], CuNoC [14] and CuNoChi [15] are attempts of applying PR to the design of NoC. However, they suffer from various limitations. Regular partially reconfigurable regions (PRRs) are defined on the FPGA, and each of them can be reconfigured as a router or a PE. Therefore, the minimum size of the PRR is the size of the largest PE in the system. As a result, there is a waste of resources when a PRR is reconfigured as a router since PE is usually much larger than a router. In addition, the reconfiguration delay is high because the whole PRR has to be reconfigured even though the router uses just a fraction of it. Devaux *et al.* [16] propose a static fat tree network with partially reconfigurable routers known as R2NoC in which the routing logic is replaced with direct links, resulting in significant area saving. Each router is placed in an independent PRR, and the PRRs are defined specifically for routers only. Therefore, the size of the PRR can be tailored to the size of the router. Although connections can be established at runtime, the reconfiguration delay is non-trivial due to the large number of routers that needs to be reconfigured, and the authors did not attempt to minimize the delay. A fat tree network is essentially a folded Benes network that is known to be only rearrangeably non-blocking. Therefore, there is a possibility of the network stalling when a new connection has to be made. This results in a loss of bandwidth that is proportional to the reconfiguration delay. Therefore, the high reconfiguration delay outweighs the saving in area. Hur *et al.* [17] also leverage on PR to create direct links similar to R2NoC but the network is not flexible in that the partial bitstreams for each use case have to be generated at design time.

Young *et al.* [18] describe an approach to efficiently build a large crossbar switch on a Xilinx FPGA using the switch matrix. The crossbar switch is configured through PR. However, the drawback of the approach is that the JBit library required is deprecated and does not support devices other than Virtex-II. In addition, Brant and Lemieux [19] describe how a LUT can be configured as a multiplexer but methods of building larger multiplexers and minimizing reconfiguration delay have not been explored.

4. PARTIALLY RECONFIGURABLE CROSSBAR SWITCH ARCHITECTURE

4.1. Building Multiplexers without Dedicated Selector Pins

A crossbar switch is the basic building block of any network. On a Xilinx FPGA, it can be realized using logic elements called configurable logic blocks (CLBs). CLBs are arranged in columns, and their composition varies across different families of FPGA. In the case of a Virtex-6 FPGA, a CLB is made up of two slices that are connected to a switch matrix. Each slice in turn contains four 6-input LUTs, which can be used to implement any 6-input logic function. A crossbar switch can be built using either the switch matrix alone or a combination of the LUTs and the switch matrix. Both approaches have their own advantages and limitations.

The switch matrix has abundant routing resources which are perfect for creating a crossbar switch. However, there are two major problems that limit the feasibility of this approach. Firstly, the switch matrix cannot be instantiated in VHDL/Verilog. Therefore, cascading the switch matrices to build a larger crossbar switch can be tedious and error prone because high level design is impossible. Secondly, runtime reconfiguration of the switch matrix would require the knowledge of the mapping between the programmable interconnect points in the switch matrix and the bits in the partial bitstream. However, the mapping is undocumented by Xilinx. Due to the reasons described above, the second approach of implementing the crossbar switch using LUTs is used.

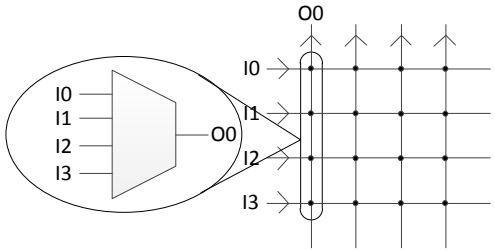

Fig. 3. Multiplexer as the building block of a crossbar switch

Each output port of an $N \times N$ crossbar switch can be built using an N-to-1 multiplexer as shown in Figure 3. Therefore, an $N \times N$ crossbar switch requires N number of N-to-1 multiplexers, one for each output of the switch. A multiplexer can be realized on an FPGA by using LUTs. Conventionally, an N-to-1 multiplexer requires $log_2 N$ number of selector pins. Therefore, a Virtex-6 LUT can implement a 4-to-1 multiplexer by using 4 out of the 6 inputs of the LUT as data inputs and the remaining 2 inputs as selector inputs. However, with PR, dedicated selector inputs are not required.

The input pins of the LUT address one of the 64 entries of the LUT. Each entry of the LUT contains 1 bit of data. For example, when the value of the input pins is *000000*, the output contains the value stored at the first entry of the LUT. Therefore, by carefully selecting the content of each entry, any of the LUT's input pins can be multiplexed to the output without the need of dedicated selector pins. The value of each entry equals to the value of the n-th bit of the entry's address, where n is the index of the LUT input pin to be multiplexed.

Figure 4 shows an example of how the LUTs' content can be configured to act as a multiplexer. A 2-input LUT has 4 entries. To multiplex *I0* or *I1*, the value of each entry equals to the 0th or 1st bit of the corresponding entry's address respectively.

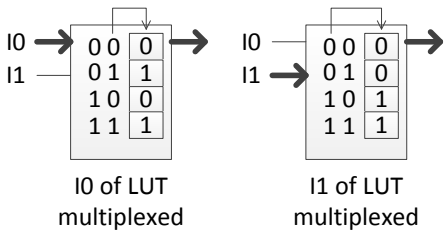

Fig. 4. Configuring *2*-input LUTs as multiplexers

Listing 1. LUT6 instantiation in VHDL with I0 being multiplexed initially

```
LUT6_0 : LUT6
  generic map (INIT => X"AAAAAAAAAAAAAAAA")
  port map (
    O => crossbar_out,
    I0 => crossbar_in(0),
    I1 => crossbar_in(1),
    I2 => crossbar_in(2),
    I3 => crossbar_in(3),
    I4 => crossbar_in(4),
    I5 => crossbar_in(5)
  );
```

To build an *N*-to-1 multiplexer with *k*-input LUTs where $N > k$, multiple *k*-input LUTs can be cascaded in the form of a *k*-ary complete tree. Cascading can be done easily because LUT is a primitive that can be instantiated in VHDL or Verilog. The completeness criterion for the tree is required to simplify the reconfiguration process. Listing 1 shows an example of how a LUT can be instantiated in VHDL where the first input of the LUT (*I0*) is mapped to the crossbar switch's first input port (*crossbar_in(0)*), the second input of the LUT is mapped to the crossbar switch's second input port and so on. However, Xilinx's Place and Route tool may not preserve the pin mappings in order to improve timing. Therefore, the LOCK_PINS constraint has to be applied to every LUT instance that is connected to the crossbar switch input ports to ensure the mapping is preserved. This is important because the connection establishment algorithm described in Section 4.2 requires the mapping between the crossbar switch input ports and the LUT input pins.

4.2. Connection Establishment through Partial Reconfiguration

To establish a new connection between an input port and an output port of the crossbar switch, the multiplexer that is associated with the output port has to be reconfigured. Since the multiplexer is realized as a *k*-ary complete tree as described in Section 4.1, setting up a new connection involves finding a path from the leaf node that is associated with the input port to the root node. The completeness of the tree allows path finding to be done using simple division and remainder operation as shown in Algorithm 1.

The *k*-to-1 multiplexer that is associated with the input to be multiplexed at each level of the tree can be determined by dividing the input index at the corresponding level with the tree radix. Then, the input to be selected by the *k*-to-1 multiplexer determined in the previous step can be found by finding the remainder of the input index when divided by the tree radix. Since the division and remainder operations have to be executed once for every level of the tree, and the number of levels of the tree is of $\mathcal{O}(logN)$, the routing algorithm has a very desirable complexity of $\mathcal{O}(logN)$ for each connection setup. In the worst case where all connections of the crossbar switch have to be changed, the complexity is $\mathcal{O}(NlogN)$.

Algorithm 1 Determining the *k*-to-1 multiplexers to be modified to setup a new connection

Input: crossbar input port index *Idx*, radix of the tree *k*, number of levels of the tree *numOfTreeLevels*
Output: logical identifier of the *k*-to-1 multiplexers to be modified

1: **if** $Idx \geq numOfNodesAtLastLevel \times k$ **then**
2: $\quad Idx \leftarrow Idx - [numOfNodesAtLastLevel \times (k-1)]$
3: $\quad startLevel \leftarrow numOfTreeLevels - 1$
4: **else**
5: $\quad startLevel \leftarrow numOfTreeLevels$
6: **end if**
7: $modifiedMux \leftarrow \emptyset$
8: **for** $i \leftarrow startLevel$ to 0 **do**
9: $\quad muxIndex \leftarrow \lfloor Idx/k \rfloor$
10: $\quad muxPortIndex \leftarrow Idx\%k$
11: $\quad modifiedMux \leftarrow modifiedMux \cup \{(i, muxIndex, muxPortIndex)\}$
12: $\quad Idx \leftarrow muxIndex$
13: **end for**
14: **return** $modifiedMux$

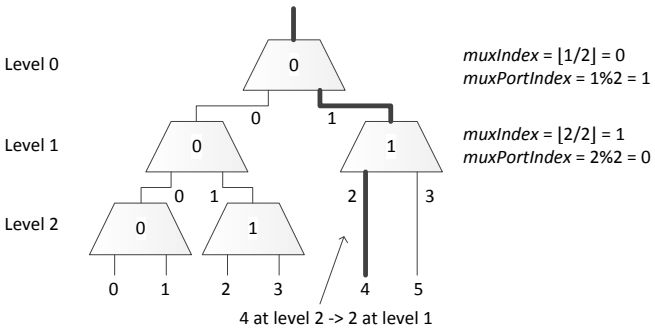

Fig. 5. Example of applying Algorithm 1

Figure 5 illustrates the result of applying Algorithm 1 to a complete binary tree ($k = 2$) with 3 levels to provide a total of 6 inputs.

After determining the logical configuration of the multiplexers that are required to setup a new connection, partial bitstreams are generated at runtime to reconfigure the crossbar switch. Runtime bitstream generation is possible because the data required to configure a LUT is position independent. For example, LUTs that are located at different

columns of the device require the same data to configure them as multiplexers. In addition, a *k*-input LUT has only *k* different configurations when it implements a *k*-to-1 multiplexer. Therefore, only *k* number of different configurations need to be determined and stored at design time. The *k* different configurations are then used to incrementally generate the required configuration of a crossbar switch at runtime. In contrast, the conventional method of reconfiguring an $N \times N$ crossbar switch requires $N!$ partial bitstreams to be generated and stored at design time.

PR is performed through the process described in Algorithm 2 using Internal Configuration Access Port (ICAP). The algorithm does not use the PR function provided by Xilinx to modify a LUT's content because the function is inefficient. The smallest addressable unit in the FPGA configuration memory space is known as a configuration frame, and the LUTs in a column of CLB are reconfigured by 8 frames as shown in Figure 6(a). Therefore, modifying a single LUT requires all the frames to be written even though only a small segment is modified. Instead, Algorithm 2 relies on the aforementioned *k*-configuration database and the mapping shown in Figure 6 to fill a temporary frame data buffer with the configuration of multiplexers to be modified. Then, the buffer is written to the ICAP to reconfigure the multiplexers. Therefore, the overhead of writing redundant data to the ICAP is eliminated.

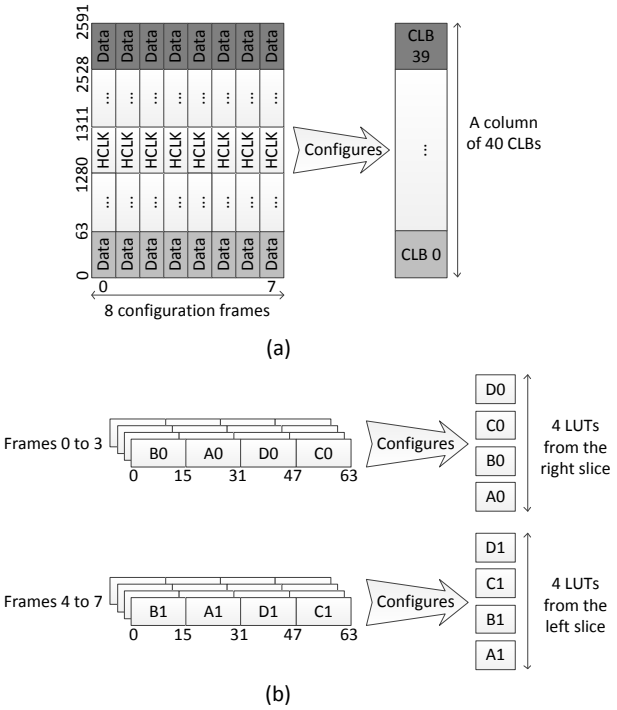

Fig. 6. Structure of Virtex-6 configuration frames

Algorithm 2 Reconfiguring FPGA with new network configuration

1: $M \leftarrow getAllModifiedMux()$
2: clear $frameData$
3: **for all** $mux \in M$ **do**
4: update $frameData$ with configuration of mux
5: **end for**
6: write $frameData$ to FPGA through ICAP

4.3. Minimizing Reconfiguration Delay

Since the FPGA is reconfigured column-wise, the reconfiguration delay can be minimized by constraining the placement of the crossbar switch into the least number of columns of CLB with sufficient LUTs to implement the multiplexers required by the crossbar switch. This is done by creating an AREA_GROUP constraint based on the number of LUTs required by the crossbar switch. The number can be calculated analytically as described in Section 5.2.

In addition to the AREA_GROUP constraint, reconfiguration delay can be minimized by writing only the frames that are responsible for the configuration of the content of the LUTs. To fully reconfigure a column of CLB, 36 frames are required. However, only 8 frames need to be written to reconfigure the LUTs' content, resulting in a speedup of up to 78%.

A network rarely consists of only one crossbar switch. For example, each wire in a SDM network is connected by an independent crossbar switch. Applications with high bandwidth requirement often requires links that are wider than 1-bit. Therefore, multiple crossbar switch requires the same configuration, and reconfiguration delay can be minimized by exploiting the common configuration among crossbar switches. The optimization is achieved by utilizing the Multiple Frame Write (MFW) command of the ICAP module. The command allows the frame data that is written to the FDRI register to be transferred to multiple different frame addresses as specified by the FAR register. Therefore, *N* frames can be configured with the same data by writing the data only once to the FDRI register instead of writing it *N* times, resulting in a significant minimization of reconfiguration delay. In order to effectively utilize the MFW command, the AREA_GROUP constraint must be applied to each crossbar switch in a way that they occupy non-overlapping CLB region, and in turn, non-overlapping frames. In this case, when two or more crossbar switches have the same configuration, the reconfiguration delay can be minimized with the MFW command because the configuration data has to be written only once to the FDRI register, and it is transferred to all the switches which require the same configuration.

5. RESULTS AND ANALYSIS

To evaluate its area requirement and reconfiguration delay, the novel crossbar switch design has been implemented in a SDM based NoC on a Virtex-6 FPGA ML605 Evaluation

Board with Xilinx ISE 13.2.

5.1. Area Saving of a Partially Reconfigurable Router

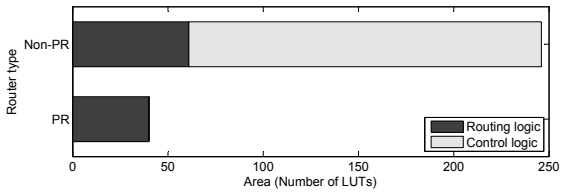

Fig. 7. Area requirement of non-PR and PR router

The non-PR 1-way router described in [2] consists of w 5×5 crossbar switches where w is the width of the link between the routers and a control logic to set the configuration of the crossbar switches. As shown in Figure 7, the control logic of the non-PR router incurs a significant area overhead of 75%. When the router in [2] is implemented with partially reconfigurable 5×5 crossbar switches, the per-router control logic is replaced with a global reconfiguration controller, which configures the entire network and consumes 799 LUTs. While the overhead seems significant, it should be noted that it is independent of the network size. In contrast, the area overhead of the design described in [2] scales linearly with the network size. In addition, the routing logic of PR based router consumes only 40 LUTs as compared to 61 LUTs required by Joseph's routing logic. This is because PR based multiplexers require no dedicated input selection pins as described in Section 4.1. Therefore, the 5-to-1 multiplexer in the 5×5 crossbar switches can be built with less LUTs when PR is utilized.

5.2. Area Complexity of Various Network Topologies

Table 1. Number of cross points of various network topologies

Topology	Number of cross points
Single $N \times N$ crossbar	N^2
$N \times N$ Clos	$6N^{\frac{3}{2}} - 3N$
$N \times N$ Benes	$2N(2log_2N - 1)$

Clos and Benes network were designed to reduce the total number of cross points in the telephone switching network to less than that of a single stage crossbar switch as shown in Table 1. The results in Figure 8 shows how the number of cross points actually translates to the number of LUTs required by the new PR based crossbar switch design. The PEs in the network are connected with 4-bit duplex links, and the n parameter of the Clos network is equal to 4. The area requirement is determined analytically by considering the number of LUTs that needs to be instantiated in

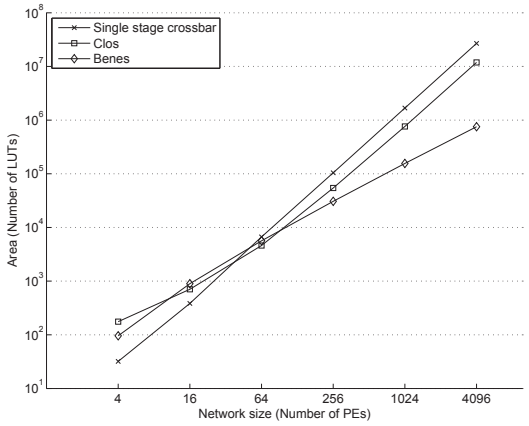

Fig. 8. Area requirement of various network topologies

HDL to build the network. A few observations can be made from the graph.

For network sizes of 64 and above, the order of the number of cross points provides a good approximation on how fast the area requirement on an FPGA increases. As shown in the graph, the area required by a single stage crossbar and a Clos network grows much faster than that of a Benes network, which is consistent with the order of the number of cross points shown in Table 1. However, for smaller network sizes, the trend is different.

Although its area complexity is the highest, the single stage crossbar switch requires the least area for network sizes of 4 and 16. This is because a single stage crossbar switch has only a path diversity of one while Benes and Clos network have higher path diversity. For small networks, the area overhead of Benes and Clos network at providing higher path diversity outweighs their smaller area complexity.

A Benes network is built from 2x2 crossbar switches. However, a LUT in a Virtex-6 FPGA can implement a 6-to-1 multiplexer, which is larger than the 2-to-1 multiplexers required in the 2x2 crossbar switch. For small networks, the area overhead introduced by underutilized LUTs outweighs the smaller area complexity of the Benes network. As a result, the Benes network consumes the largest area at a network size of 16 as shown in the graph although it is rearrangeably non-blocking.

5.3. Reconfiguration Delay

Current PR based NoCs [13][14][15][16] rely on Xilinx's PR design flow where each of the routers in the NoC is placed a single PRR. This approach incurs a significant reconfiguration overhead that increases as the area requirement of the router decreases. Lower area requirements of router allows more routers to be placed in a single column of CLB and configured at the same time. However, a partial bitstream generated by Xilinx's PR design flow only reconfigures a single router regardless of the number of routers in

a single CLB column. To illustrate the difference in terms of reconfiguration delay between Xilinx's flow and the reconfiguration process described in Section 4.2, the design proposed in R2NoC [16] is considered due to its lightweight router. A single Virtex-6 CLB column with 320 LUTs can implement up to 40 routers with 1-bit ports based on R2NoC's design. However, each partial bitstream modifies the configuration of a single router while preserving the configuration of others. Therefore, 40 partial bitstreams needs to be written to the ICAP to reconfigure all the routers in a single column. In addition, each partial bitstream configures 36 frames because R2NoC's design requires the switch matrix to be reconfigured as well. As a result, a total of 1440 frames need to be written while the reconfiguration process described in Section 4.2 requires 8 frames only.

6. CONCLUSION

In this paper, an area-efficient crossbar switch has been presented with an area saving of up to 84%. In addition, a fast algorithm for setting up a new connection in the crossbar switch with a complexity of $\mathcal{O}(logN)$ is also proposed. Finally, reconfiguration delay of the crossbar switch is minimized up to 78% with a custom reconfiguration procedure.

In the future, other network topologies will be built upon the proposed crossbar switch architecture, and their performance will be evaluated. Furthermore, the possibility of harnessing the rich routing logic on the switch matrix to build a more efficient crossbar switch will be explored.

Acknowledgment

The authors would like to thank the Xilinx University Program (XUP) for donating the PR license. This work was supported by Singapore Ministry of Education Academic Research Fund Tier 1 with grant number R-263-000-655-133.

7. REFERENCES

[1] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[2] Z. Yang, A. Kumar, and Y. Ha, "An area-efficient dynamically reconfigurable spatial division multiplexing network-on-chip with static throughput guarantee," in *Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010, pp. 389–392.

[3] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor, "Concepts and implementation of spatial division multiplexing for guaranteed throughput in networks-on-chip," *Computers, IEEE Transactions on*, vol. 57, no. 9, pp. 1182–1195, 2008.

[4] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: concepts, architectures, and implementations," *Design & Test of Computers, IEEE*, vol. 22, no. 5, pp. 414–421, 2005.

[5] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2. IEEE, 2004, pp. 890–895.

[6] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal, "Multiprocessor systems synthesis for multiple use-cases of multiple applications on fpga," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 13, no. 3, p. 40, 2008.

[7] L. Devaux, S. Sassi, S. Pillement, D. Chillet, and D. Demigny, "Flexible interconnection network for dynamically and partially reconfigurable architectures," *International Journal of Reconfigurable Computing*, vol. 2010, p. 6, 2010.

[8] M. Modarressi, H. Sarbazi-Azad, and A. Tavakkol, "An efficient dynamically reconfigurable on-chip network architecture," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. IEEE, 2010, pp. 166–169.

[9] B. Ahmad, A. Erdogan, and S. Khawam, "Architecture of a dynamically reconfigurable NoC for adaptive reconfigurable mpsoc," in *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on*. IEEE, 2006, pp. 405–411.

[10] *Partial Reconfiguration User Guide*, Xilinx, Inc., 2012, v13.4. [Online]. Available: http://www.xilinx.com/

[11] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, pp. 406–424, 1953.

[12] V. E. Benes, "Optimal rearrangeable multistage connecting networks," *Bell System Technical Journal*, pp. 1641–1656, 1964.

[13] C. Bobda, M. Majer, D. Koch, A. Ahmadinia, and J. Teich, "A dynamic NoC approach for communication in reconfigurable devices," *Field Programmable Logic and Application*, pp. 1032–1036, 2004.

[14] S. Jovanovic, C. Tanougast, S. Weber, and C. Bobda, "Cunoc: A scalable dynamic NoC for dynamically reconfigurable fpgas," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*. IEEE, pp. 753–756.

[15] T. Pionteck, R. Koch, and C. Albrecht, "Applying partial reconfiguration to networks-on-chips," in *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*. IEEE, pp. 1–6.

[16] L. Devaux, S. Pillement, D. Chillet, and D. Demigny, "R2noc: dynamically reconfigurable routers for flexible networks on chip," in *2010 International Conference on Reconfigurable Computing and FPGAs*. IEEE, 2010, pp. 376–381.

[17] J. Hur, S. Wong, and S. Vassiliadis, "Partially reconfigurable point-to-point interconnects in virtex-ii pro fpgas," *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 49–60, 2007.

[18] S. Young, P. Alfke, C. Fewer, S. McMillan, B. Blodget, and D. Levi, "A high i/o reconfigurable crossbar switch," in *Proc. of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 03)*, 2003, pp. 3–10.

[19] A. Brant and G. Lemieux, "Zuma: An open fpga overlay architecture," in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2012, pp. 93–96.