# Partially Reconfigurable Point-to-Point Interconnects in Virtex-II Pro FPGAs

Jae Young Hur, Stephan Wong, and Stamatis Vassiliadis

Computer Engineering Lab., TU Delft, The Netherlands
http://ce.et.tudelft.nl

**Abstract.** Conventional rigid router-based networks on chip incur certain overheads due to huge occupied logic resources and topology embedding, i.e., the mapping of a logical network topology to a physical one. In this paper, we present an implementation of partially reconfigurable point-to-point ($\rho$-P2P) interconnects in FPGA to overcome the mentioned overheads. In the presented implementation, arbitrary topologies are realized by changing the $\rho$-P2P interconnects. In our experiments, we considered parallel merge sort and Cannon's matrix multiplication to generate network traffic to evaluate our implementation. Furthermore, we have implemented a 2D-mesh packet switched network to serve as a reference to compare our results with. Our experiment shows that the utilization of on-demand $\rho$-P2P interconnects performs 2× better and occupies 70% less area compared to the reference mesh network. Furthermore, the reconfiguration latency is significantly reduced using the Xilinx module-based partial reconfiguration technique. Finally, our experiments suggest that higher performance gains can be achieved as the problem size increases.

## 1 Introduction

In modern on-chip multi-core systems, the communication latency of the network interconnects is increasingly becoming a significant factor hampering performance. Consequently, network-on-chips (NoCs) as a design paradigm has been introduced to deal with such latencies and related issues. At the same time, NoCs provide improved scalability and an increased modularity[1][2][3][4][5]. However, these multi-core systems still incorporate rigid interconnection networks, i.e., mostly utilizing a 2D-mesh as the underlying physical network topology combined with packet routers. More specifically, the interconnection network will be fixed in the design stage leading to the modification of algorithms to suit the underlying topology or the embedding of the logical network (intended by the algorithm) onto the physical interconnection network. In both cases, reduced performance is the result. The topology embedding techniques are well-studied [6] and usually require the introduction of a router module to handle network dilations and congestions. Furthermore, worm-hole flow control for the packet switched network (PSN) is widely used due to its insensitivity to multi-hop delays. As a result, these systems that still utilize rigid network interconnects have

P.C. Diniz et al. (Eds.): ARC 2007, LNCS 4419, pp. 49–60, 2007.

the following limitations. First, the programmer must have intricate knowledge of the underlying physical network in order to fully exploit it. Second, performance is lost due to topology embedding that is likely to increase communication delays and/or create traffic congestions. Third, a router (with virtual channels) occupies significant on-chip resources.

Our work is motivated by several key observations. First, different applications (or algorithms) generate different traffic patterns that require different topologies to handle them in the best possible manner. Therefore, the ability to 'construct' topologies on-demand (at application start time or even during run-time) is likely to improve performance. Second, direct point-to-point connections eliminate the communications overheads of utilizing packet switched networks. Traditionally, P2P networks were not popular due to complex wiring and scalability issues. However, modern reconfigurable hardware fabrics contain rich intra-chip wiring resources and additionally provide a capability to change the interconnections (possibly) in run-time [7]. Figure 1(1) shows a Configurable Logic Block (CLB) cell in the Virtex-II Pro device indicating that the wiring resources occupy more than 70% of the cell. The Virtex-II Pro xc2vp30 device contains 3680 CLBs and abundant wires such as long, hex, double, and direct lines. Moreover, the interconnect wires do not require logic slices, which are valuable computational resources. Therefore, we aim to design and implement a (dynamically) adaptive and scalable interconnection network suited to meet demands of specific applications. Figure 1(2) depicts our general approach. Our system adaptively provides demanding interconnection networks that realize the traffic patterns. The reconfigurable interconnects consist of set of on-chip wiring resources. The presented $\rho$-P2P implementation combines the advantages of high performance of traditional P2P networks and the flexibility of reconfigurable interconnects.

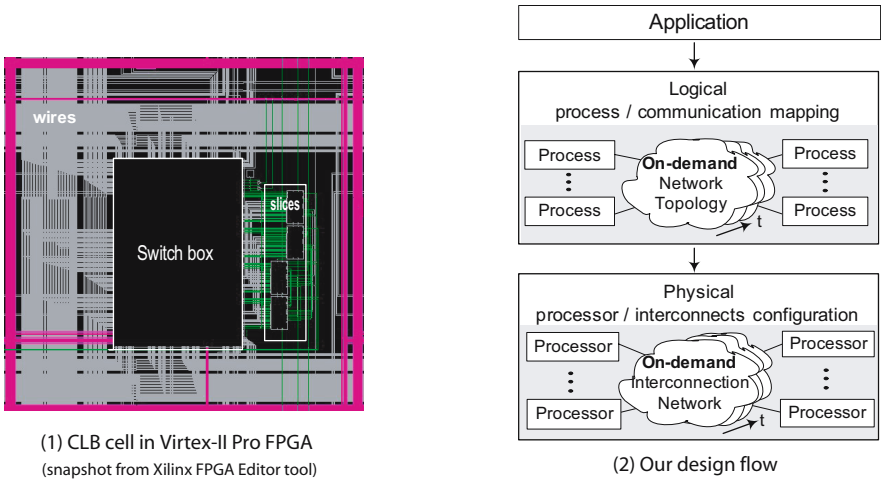


**Fig. 1.** Our approach

In this work, we implement 2D-mesh PSNs and our $\rho$-P2P interconnects. We perform a comparative study to re-examine the performance gain and the area reduction of the $\rho$-P2P network over the PSN in modern reconfigurable platforms. We propose to utilize partial reconfiguration techniques to realize the $\rho$-P2P networks. The main contributions of this work are:

- An arbitrary topology can be rapidly configured by updating small-sized partial bitstreams, which is verified by an actual prototyping. The proof-of-concept experiment in Virtex-II Pro device using the Xilinx module-based partial reconfiguration technique shows that the actual system performance gain increases as the problem size increases.
- The experiments show that our $\rho$-P2P network performs 2× better and occupies 70% less area compared to a mesh-based PSN for the considered applications.

The organization of this paper is as follows. In Section 2, related work is described. System designs and their hardware implementations are described in Sections 3 and 4. In Section 5, conclusions are drawn.

## 2 Related Work

The general concept of on-demand reconfigurable networks was introduced in [8]. In this paper, we present $\rho$-P2P networks to realize on-demand interconnects as an implementation methodology using modern FPGA technology. Recently, a number of NoCs have been proposed [1][2][3][4][5]. These networks employ rigid networks fixed at design time. Additionally, a packet router with virtual channels occupy significant on-chip logic resources. As an example, 16 routers with 4 virtual channels in [4] occupy 25481 slices, while a common device such as the xc2vp30 Virtex-II Pro contains in total only 13696 slices. Those networks route packets over multi-hop routers and consequently we consider a wormhole flow control based PSN as a reference. Our approach is different from the afore-mentioned networks in that the interconnection network consists of directly connected native on-chip wires. Our $\rho$-P2P network is different from the traditional point-to-point network, such as a systolic array, in that the physical interconnects of our work are reconfigurable. Our work is similar to [9] in that an on-chip packet processing overhead is discussed. While [9] considers a butterfly fat tree topology and does not discuss topology embedding, our work presents a network design to avoid packet processing overheads together with topology embedding. A partial reconfiguration technique for the NoC design is presented in [10]. While [10] presents dynamic insertion and removal of switches, our work presents configurations of physical interconnects. Reconfigurations of buses are presented in [11] and our work is similar to [11] in that the topology is dynamically changed. However, the buses in [11] are static as they use pre-established lines, we use only the required wiring resources.

## 3 Design and Implementation

### 3.1 2D-Mesh PSN

We consider a 2D-mesh PSN as a reference network as depicted in Figure 2(1). The PSN system consists of processing nodes which are interconnected in a rigid 2D-mesh topology. Each node is composed of a processing element (PE), a dual-ported memory, a network interface, and a router. Each node communicates utilizing a handshaking protocol. The packet is composed of 3 control flits and a variable amount of data flits as depicted in Figure 2(2). The first flit is a header containing the target node address. The second flit is the physical memory address of the remote node to store the transmitted data. The trailer indicates the last flit of a packet. Figure 2(3) depicts how a packet is processed. A flit-based worm-hole flow control is adopted for more efficient buffer utilization. When a packet arrives, the switch controller checks the packet header in the buffer. If the current node address and a target node address are different, the switch controller forwards the packet to the appropriate port. If current node address and a target node address are identical, the switch controller checks how many local memory ports are idle. The switch controller permits up to two packets to simultaneously access a local dual-ported memory. This feature is useful, provided that two-operand computations are common in many applications. Figure 2(4) depicts how a 2D-mesh node is organized. The network interface deals with packet generation, assembly, and memory management. The memory is private to the PE and the packet size is determined by the PE using trailer signal. In this way, direct memory access (DMA) of burst data transfers is supported. Each port entails full-duplex bidirectional links. A buffer is contained in each input port and the XY routing algorithm is utilized for its simplicity. A buffer accommodates 8 flits, while the individual flit width is 16 bit. The packets are arbitrated based on the round robin scheduling policy. The processors are wrapped inside a router in order for the dual-ported memory to simultaneously store the 2 incoming packet payloads. Consequently, the system conforms to the non-uniform memory access (NUMA) model. Our implementation is similar to [3][5] in that the same flow control and routing scheme are used. However, our PSN system in this work differs in the following ways. First, variable length burst packets are directly communicated between distributed memories. Second, dual-ported embedded memories are utilized to simultaneously accommodate two incoming packets. Third, a topology embedding (e.g., binary tree traffic embedding using [12]) has been implemented. A single packet containing $N_l$ elements requires the following amount of cycles to conduct a source-to-destination communication:

$$L_l = N_l + \#hop \cdot L_h + P_{overhead}, \tag{1}$$

where $L_l$ refers to the communication latency in number of cycles to transfer $N_l$ elements. $L_h$ denotes the header latency per router. The $P_{overhead}$ refers to the packetization overhead. $\#hop$ refers to the number of intermediate routers.

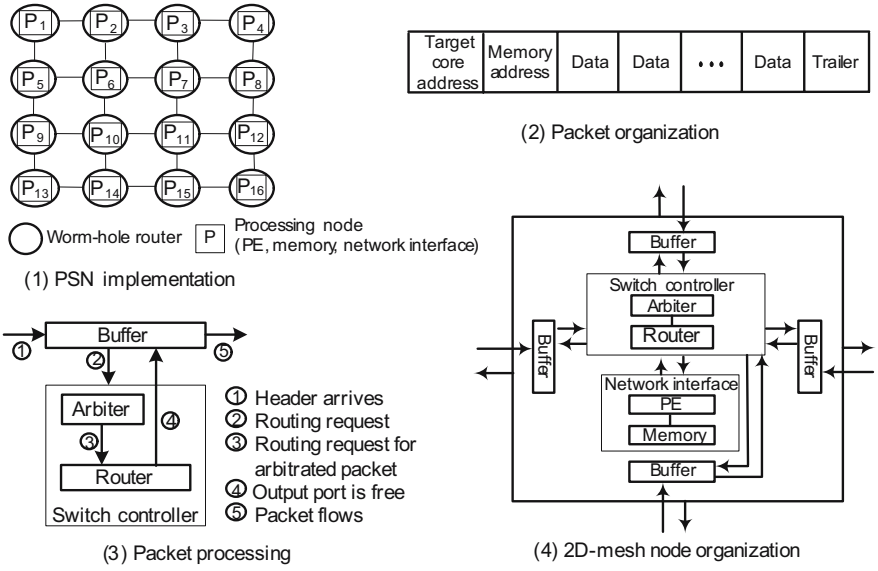


**Fig. 2.** The 2D-mesh PSN system organization

### 3.2 $\rho$-P2P Network

The proposed $\rho$-P2P system consists of directly interconnected PEs as depicted in Figure 3(1). Figure 3(2) shows the device floorplan. PEs are located in a static region and interconnects in the reconfigurable region are adaptively (re)configured on demand. The interconnects in the reconfigurable region correspond to the required wiring resources. The bus macro region separates the reconfigurable region from the static region and consists of symmetrical tri-state buffer arrays. The left-hand side belongs to the reconfigurable region and the right-hand side belongs to the static region. The interconnects are enabled or disabled by controlling the tri-state buffers. Two tri-state buffers constitute a single-bit wire line. The reconfigurable region and bus macros constitute the topology component. The topology component is modular and can be replaced by other topologies. In our experiments, the computational region remains fixed over time and only the communication region is reconfigurable. Those regions span whole vertical columns since the Virtex-II Pro device is configured in full column by column. When the required communication patterns change, the physical interconnects can be quickly changed. We exploit dynamic and partial reconfiguration techniques [13] of Xilinx FPGAs to adaptively configure $\rho$-P2P interconnects. Figure 3(3) depicts the exemplified reconfiguration steps, where the processors are initially interconnected in a 2D-mesh. After that, the on-demand topology is (re)configured by updating partial bitstreams only for the reconfigurable topology component during the respective time $t_2 - t_1, t_3 - t_2$. The network topology is implemented as a partially and dynamically reconfigurable

component of the system. $N_l$ elements requires $N_l$ cycles to conduct a source-to-destination communication. This approach can be generally applied to the recent reconfigurable hardware, such as partially reconfigurable Xilinx Virtex series devices, though we target Virtex-II Pro in this work.

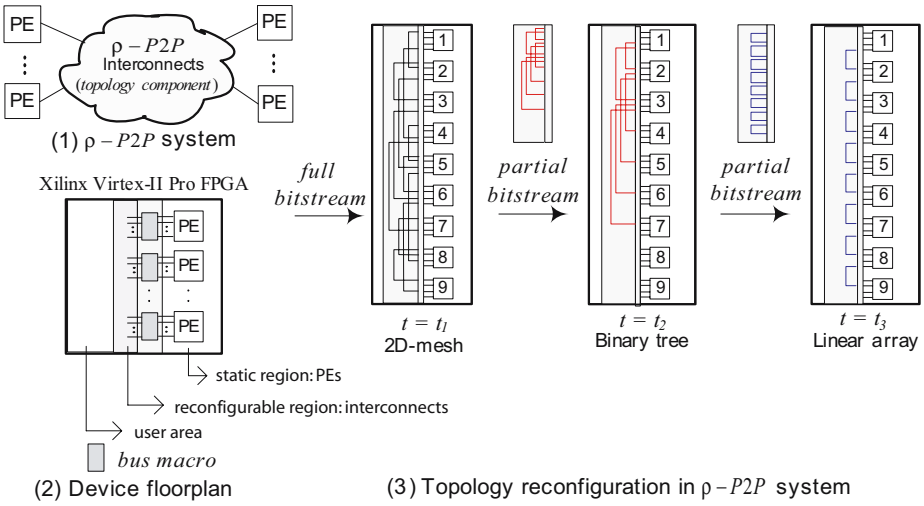


**Fig. 3.** The $\rho$-P2P system

The layout of static region is identical for each system configuration and remains unchanged during the interconnects reconfiguration. The small sized topology components can be reconfigured while processors are in operation within the static region. The reconfiguration latencies are directly proportional to corresponding partial bitstream sizes. The bitstream size is determined by required on-chip resources. In Virtex-II Pro, a bitstream is loaded in a column basis, with the smallest load unit being a configuration frame of the bitstream. Different network configurations are implemented in VHDL using the bus macros. Figure 4 depicts the design of a topology component using bus macro region as an example. Virtex-II Pro xc2vp30 contains abundant (6848) tri-state buffers which can be used for bus macro designs. As can be observed from Figure 4, the network topology can be implemented using tri-state buffer configurations. The interconnects do not require any logic resources such as slices, while the look-up tables (LUTs) in the bus macro regions are only required for power and ground signals of tri-state buffers. Since the data is communicated in a point-to-point fashion without packetization and multi-hop routing overheads, faster communication can be achieved compared to PSN. The area is also significantly reduced, provided that the interconnection network can be realized by configuring hard-wired bus-macros and using on-chip wires. Additionally, the partial bitstream is automatically generated using the Xilinx modular design flow.

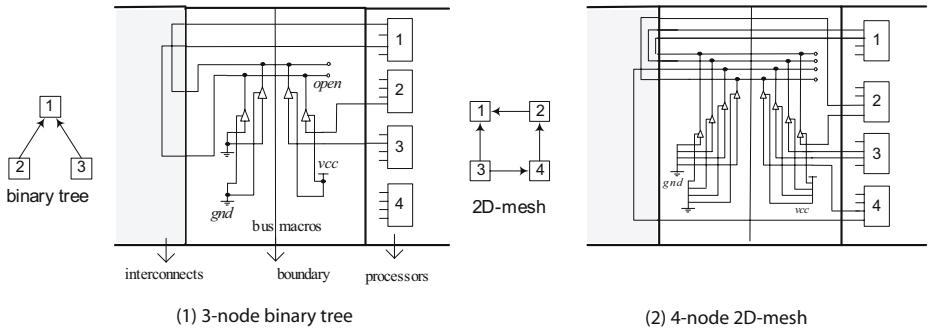


**Fig. 4.** The topology implementation using bus macros

### 3.3 Configuration Examples

In order to evaluate the presented network, we consider two types of workloads. The parallel merge sort (binary tree, burst traffic) and the Cannon's matrix multiplication (2D-torus, element-wise traffic) have been chosen as network traffics. Those algorithms are common and the implemented PEs are small in area such that larger networks can be realized in a single chip.

**Parallel Merge Sort:** The logical traffic pattern of the parallel merge sort is as follows. Firstly, sequential sort is performed at the bottom-most nodes. Secondly, the parent nodes sort elements taken from child nodes until the root node finishes the task. Sequential and parallel computation require $O(n \log n)$, $O(n)$ steps, respectively, and communication requires $O(n)$ steps, for the problem size $n$. Sequential PEs are identical for $\rho$-P2P and PSN systems. Consider $p$ processors performing the parallel merge sort of $N$ integers. System cycles are derived as shown in Equations (2a),(2b), where MS_PSN, MS_P2P refer to the number of cycles when the merge sort is operated in the PSN, $\rho$-P2P, respectively. In PSN system, the system cycles are calculated by (computation cycles) + (communication cycles). Consider a complete binary tree with $p$-processors, $\alpha_q \frac{2N}{p+1} \log \frac{2N}{p+1}$ cycles are required for a sequential sort. In PSN, when the sequential PE points to a remote memory address and commands a message transaction, the network interface generates a packet containing $\frac{2N}{p+1}$ elements. After that, each packet requires $(\beta_s \frac{2N}{p+1} + \#hop \cdot L_h + P_{overhead})$ cycles to move up to upper nodes. When each packet arrives, the network interface assembles the packets and stores the elements in local memory. Upper nodes perform a parallel sort, in which each element requires $\alpha_s$ cycle(s) for the parallel computation. $(\log_2(p+1)-1)$ corresponds to the level of binary tree. In $\rho$-P2P system, $\alpha_p N$ cycles are required for the parallel computation for $N$ total elements, in which $\alpha_p$ cycles are required to perform load, compare, forward operations for each element. $\beta_p N$ are required for communications. In case $\alpha_p$ and $\beta_p$ are same, all the communication cycles $\beta_p N$ are completely hidden by the computation, since communication and computation can be overlapped.

$$MS\_PSN = \alpha_q \frac{2N}{p+1}\log\frac{2N}{p+1} + 2\alpha_s N(1-\frac{2}{p+1}) + \beta_s N(1-\frac{2}{p+1}) + (\log_2(p+1)-1)(\#hop \cdot L_h + P_{overhead}) \quad (2a)$$

$$MS\_P2P = \alpha_q \frac{2N}{p+1}\log\frac{2N}{p+1} + \alpha_p N \quad (2b)$$

**Cannon's Matrix Multiplication:** The logical 2D-torus traffic pattern of the Cannon's matrix multiplication is as follows. Firstly, a scalar multiplication is performed at each node and secondly, the intermediate result is transferred to left/upper node. These two steps are repeated until the task is finished. A sequential computation requires $O(m^3)$ steps for the matrix of size $m \times m$. Each PE is assumed to contain a single multiplier. Consider $\sqrt{p} \times \sqrt{p}$ processors and 2 symmetric matrices with size $M \times M$. System cycles are derived as shown in Equations (3a), (3b), where MM_PSN, MM_P2P refer to the number of cycles when the matrix multiplication is operated in the PSN, $\rho$-P2P, respectively. In $\rho$-P2P system, there are $\sqrt{p}$ phases of computations and $\frac{M^3}{p\sqrt{p}}$ computation cycles are required for each phase. PEs require $\alpha_p$ cycles to perform multiply, add, transfer operations for each element. The communication is performed between directly connected neighbor PEs, or #hop is 1. Totally $\alpha_p \frac{M^3}{p}$ cycles are required for the system cycles. In 2D-mesh PSN system, $L_h$, $P_{overhead}$ and worst case $\sqrt{p}$ hops for each packet are required. Additionally, the communication is directly performed between distributed memories.

$$MM\_PSN = \alpha_q \frac{M^3}{p} + \sqrt{p}\,(\beta_s \frac{M^2}{p} + \#hop \cdot L_h + P_{overhead}) \quad (3a)$$

$$MM\_P2P = \alpha_p \frac{M^3}{p} \quad (3b)$$

## 4 Experimental Results

In this work, 3 experiments have been conducted. First, the system cycles are analytically derived from Equation (2) and (3) to measure the performance gain of $\rho-$P2P over PSN in the reconfigurable hardware. The coefficients are obtained from the highly optimized hardware implementations. For the parallel merge sort, $\alpha_q = 2.6$, $\alpha_s = 2.3$, $\beta_s = 1$ , $L_h = 6$ , $\alpha_p = 2$ , $\beta_p = 2$ have been obtained. For the matrix multiplication, $\alpha_q = 2$, $\alpha_p = 1$, $\beta_s = 1$ are obtained. $\alpha_q$ is 2, since the PE requires 2 cycles to access a local memory, while $\alpha_p$ is 1, since the data is communicated directly between PEs. We can fairly compare $\rho-$P2P and PSN, since the computational latency is same for both systems. Figure 5(1) depicts the system cycles for different problem sizes. As can be observed, the $\rho-$P2P network performs on average 2× better (Note: graphs have a log scale). $\rho-$P2P is better in performance, since PSN suffers from a multi-hop communication latency, a packetization overhead while $\rho-$P2P provides a single-hop communication. Figure 5(2) shows the performance gain of $\rho-$P2P over PSN

in terms of the execution time. As the problem size increases, the performance gain in the actual amount of cycles also increases. Assuming the systems operate at 100MHz, the performance gain is obtained by (PSN system cycles - P2P system cycles)×10ns. Figure 5 depicts the performance gain that can grow up to 120 ms, 1347 ms for the merge sort, matrix multiplication, respectively. It can be noted that the Virtex-II Pro xc2vp30 device requires 26ms as a whole chip reconfiguration time [7]. The experiment suggests that partially reconfigured on-demand interconnects can be more beneficial for large problem sizes, since the reconfiguration latency is relatively smaller.

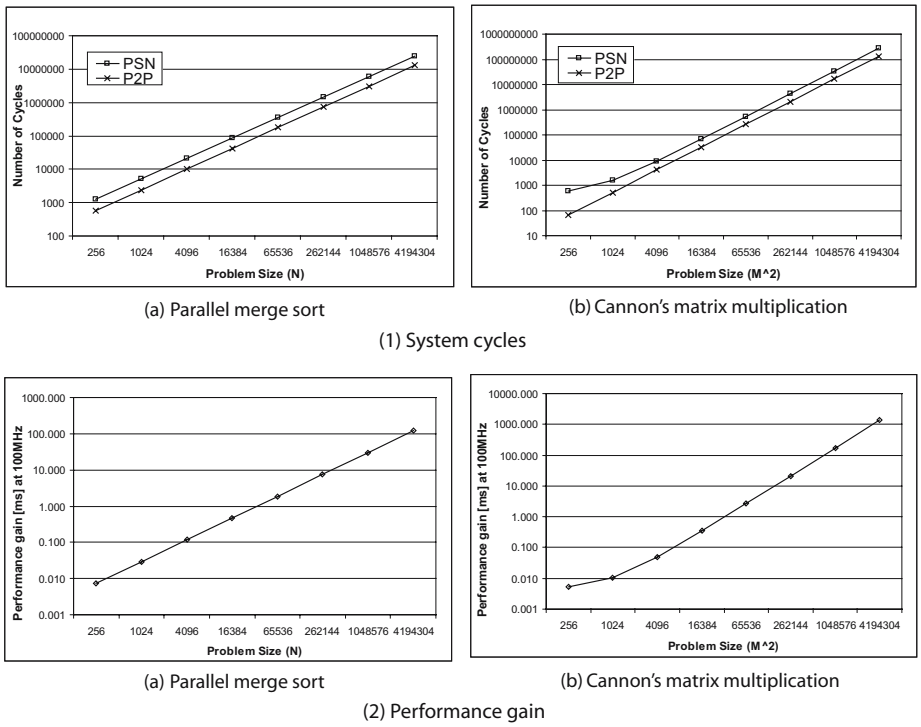


**Fig. 5.** System cycles and performance gains of $\rho$-P2P and PSN

Second, PSN and $\rho-$P2P systems have been implemented in VHDL, placed and routed using the Xilinx ISE tool, in order to evaluate the system cycle models in Equations (2),(3). The systems have been implemented for N=512 (merge sort) and M=16 (matrix multiplication). Each PE has been implemented in an application-specific finite state machine, occupying 1% of area. Virtex-II Pro xc2vp100-6 has been used as a target device in order to experiment on larger networks. The implementation results for the merge sort are presented in Figure 6(1), in which network size, type of network, topology, number of

nodes, system cycles, system area, clock frequency and system execution time are shown. The sequential merge sort requires $11981(\approx 2.6 \times 512 \times \log_2 512)$ cycles in the implementation. To implement a PSN, the binary tree is embedded in 2D-mesh using the algorithm of [12]. The binary tree P2P system reduces on average 67% area and 37% of execution time compared to PSN. In Figure 6(1), PSN and $\rho-$P2P for the same network size can be fairly compared, since an actual number of PEs which participate in the computation is the same. Figure 6(2) shows the comparison of PSN and $\rho-$P2P for the matrix multiplication. The sequential matrix multiplication requires $12546(\approx 3 \times 16^3)$ cycles. Embedded hardwired $18 \times 18$ bit multiplier, an adder and a simple control unit are implemented for each PE, which is identical for $\rho$-P2P and 2D-mesh PSN systems. 2D-torus P2P performs Cannon's matrix multiplication with a single-hop communication. Each PE performs an integer multiplication in a single cycle. In $\rho-$P2P, 94.6% of an execution time is reduced, since in PSN with 16×16 PEs, in the worst case 16 hops are required to transfer packets, while single cycle per each hop is required. Additionally, 82% of area is reduced, since complex router modules are eliminated.

| Size | Network | Topology | #nodes | #cycles | Area | | Max. | Execution time | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | #slices | reduction(%) | Freq. (MHz) | [us] | reduction(%) |
| | Sequential | - | 1 | 11981 | 491 | - | 125.5 | 95.4 | - |
| 1 | PSN | 2D-mesh | 4 | 6429 | 2821 | 60.9 | 117.3 | 54.8 | 9.0 |
| | P2P | Binary tree | 3 | 6154 | 1102 | | 123.5 | 49.8 | |
| 2 | PSN | 2D-mesh | 9 | 4276 | 6828 | 66.1 | 116.4 | 36.7 | 25.6 |
| | P2P | Binary tree | 7 | 3338 | 2316 | | 122.1 | 27.3 | |
| 3 | PSN | 2D-mesh | 16 | 3415 | 15533 | 68.8 | 113.3 | 30.1 | 40.9 |
| | P2P | Binary tree | 15 | 2063 | 4851 | | 115.8 | 17.8 | |
| 4 | PSN | 2D-mesh | 36 | 3094 | 33915 | 71.0 | 111.1 | 27.8 | 48.7 |
| | P2P | Binary tree | 31 | 1623 | 9852 | | 113.7 | 14.3 | |
| 5 | PSN | 2D-mesh | 64 | 2873 | 64057 | 69.1 | 105.5 | 27.2 | 58.9 |
| | P2P | Binary tree | 63 | 1247 | 19791 | | 111.4 | 11.2 | |

(1) Merge sort (N=512)

| Network | Topology | #nodes | #cycles | Area | | Max. | Execution time | |
|---|---|---|---|---|---|---|---|---|
| | | | | #slices | reduction(%) | Freq. (MHz) | [us] | reduction(%) |
| Sequential | - | 1 | 12546 | 124 | - | 125.9 | 99.7 | - |
| PSN | 2D-mesh | 256 | 300 | 44094 | 82.2 | 100.2 | 3.0 | 94.6 |
| P2P | 2D-torus | 256 | 16 | 7837 | | 99.4 | 0.2 | |

(2) Matrix multiplication (M=16, p=256)

**Fig. 6.** Implementation results

Third, intended as a proof-of-concept, the run-time reconfiguration of the interconnects has been realized on the Virtex-II Pro xc2vp30 in the Digilent XUP-V2P prototyping board[14]. Figure 7 demonstrates the procedure of the partial run-time reconfiguration. The 2D-mesh and the binary tree interconnects have been reconfigured using Xilinx module-based partial reconfiguration

technique [13] in a boundary scan mode. Actual network interconnects in Figure 7(2) and (4) correspond to the topology components depicted in Figure 3. The small sized topology components can be reconfigured while processors are in operation. As an example, we reconfigure the binary tree interconnects by updating partial bitstream (Figure 7(4)). The layout of static region (Figure 7(1)) is identical for each system configuration and remains unchanged during the interconnects reconfiguration. In this experiment, the partial bitstream size in number of frames is 79 out of 1757, indicating that 4.4% of the reconfiguration latency is required compared to the full reconfiguration. It can be observed that the reconfiguration latency is significantly reduced by utilizing the partial bitstream. Furthermore, 162 LUTs (out of 27396) of logic resources and 240 TBUFs (out of 6848) primitives were used for bus macros. Regarding wiring resources, 231 switch boxes (out of 4592) are used for the binary tree partial bitstream. The experiment clearly shows that the occupied logic resources are significantly reduced by utilizing $\rho$-P2P interconnects.

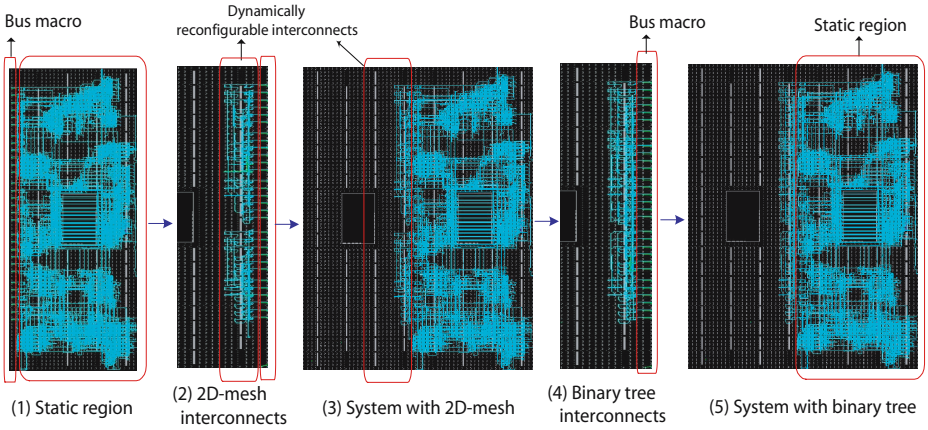


**Fig. 7.** Partial run-time reconfiguration

## 5 Conclusions

In this work, we proposed the partially reconfigurable interconnects on demand. Our $\rho$-P2P networks have been implemented and evaluated by comparing them with the 2D-mesh PSNs. We showed that the $\rho$-P2P network provides a better performance and reduced area by avoiding the use of complex routers. We also showed that the bitstream size is significantly reduced using partial reconfiguration of interconnects. Finally, our experiment projects adequate performance benefits to offset the reconfiguration latency as the problem size increases. Therefore, systems facilitating processors directly interconnected with the proposed reconfigurable interconnects can be suitable for our general approach.

**Acknowledgement.** This work was supported by the Dutch Science Foundation (STW) in the context of the Architecture, Programming, and Exploration of Networks-on-Chip based Embedded System Platforms (ARTEMISIA), project number LES.6389.

## References

1. W. J. Dally and T. Brian, "Route Packets, Not Wires: On-Chip Interconnection Networks," Proceedings of 38th International Conference on Design Automation Conference (DAC'01), pp. 684–689, Jun 2001.
2. T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-chip," ACM Computing Surveys, vol. 38, no. 1, pp. 1–51, Mar 2006.
3. F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: an Infrastructure for Low Area Overhead Packet-switching Netwoks on Chip," Integration, the VLSI Journal, vol. 38, no. 1, pp. 69–93, Oct 2004.
4. A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC," Proceedings of 18th Symposium on Integrated Circuits and Systems Design (SBCCI'05), pp. 178–183, Sep 2005.
5. A. Mello, L. Möller, N. Calazans, and F. Moraes, "MultiNoC: A Multiprocessing System Enabled by a Network on Chip," Proceedings of International Conference on Design, Automation and Test in Europe (DATE'05), pp. 234–239, Mar 2005.
6. F.T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays · Trees · Hypercubes. Morgan Kaufmann Publishers, Inc., 1992.
7. Virtex-II Pro Handbook, http://www.xilinx.com.
8. S. Vassiliadis and I. Sourdis, "FLUX Networks: Interconnects on Demand," Proceedings of International Conference on Computer Systems Architectures Modelling and Simulation (IC-SAMOS'06), pp. 160–167, Jul 2006.
9. N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, and A. DeHon, "Packet Switched vs Time Multiplexed FPGA Overlay Networks," Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06), pp. 205–216, Apr 2006.
10. T. Pionteck, R. Koch, and C. Albrecht, "Applying Partial Reconfiguration to Networks-on-Chips," Proceedings of 16th International Conference on Field Programmable Logic and Applications (FPL'06), pp. 155–160, Aug 2006.
11. M. Huebner, M. Ullmann, L. Braun, A. Klausmann, and J. Becker, "Scalable Application-Dependent Network on Chip Adaptivity for Dynamical Reconfigurable Real-Time Systems," Proceedings of 14th International Conference on Field Programmable Logic and Applications (FPL'04), pp. 1037–1041, Aug 2004.
12. S.-K. Lee and H.-A. Choi, "Embedding of Complete Binary Tree into Meshes with Row-Column Routing," IEEE Transactions on Parallel and Distributed Systems, vol.7, no.5, pp. 493–497, May 1996.
13. "Two Flows for Partial Reconfiguration: Module Based or Difference Based," Xilinx Application Note, xapp 290, Sep 2004.
14. Digilent, Inc., http://www.digilentinc.com/XUPV2P.