International Journal of Electronics
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713599654

Partially reconfigurable point-to-point FPGA interconnects

Jae Young Hur [a]; Stephan Wong [a]; Stamatis Vassiliadis [a]
[a] Computer Engineering Lab, TU Delft, The Netherlands

Online Publication Date: 01 July 2008

To cite this Article Young Hur, Jae, Wong, Stephan and Vassiliadis, Stamatis(2008)'Partially reconfigurable point-to-point FPGA interconnects',International Journal of Electronics,95:7,725 — 742

To link to this Article: DOI: 10.1080/00207210801924586

URL: http://dx.doi.org/10.1080/00207210801924586

International Journal of Electronics
Vol. 95, No. 7, July 2008, 725–742

Taylor & Francis
Taylor & Francis Group

Partially reconfigurable point-to-point FPGA interconnects

Jae Young Hur*, Stephan Wong and Stamatis Vassiliadis†

Computer Engineering Lab., TU Delft, The Netherlands

(*Received 14 September 2007; final version received 28 November 2007*)

We present a novel use of wiring flexibility in modern FPGA technology in order to implement an on-demand network topology. Conventional rigid router-based networks on chip incur certain overheads due to huge logic resources occupation and topology embedding. In this work, we implement partially reconfigurable point-to-point (ρ-P2P) interconnects to alleviate such overheads. In our implementation, arbitrary topologies can be realised by updating a partial bitstream for the ρ-P2P interconnects. We consider parallel merge sort, Cannon's matrix multiplication, and wavelet applications to generate network traffic. Furthermore, we implement a packet switched network to serve as a reference. The experiments show that the utilisation of our P2P interconnects performs 2 times better and occupies 70% less area when compared to the reference network. Furthermore, the topology 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.

Keywords: FPGA; interconnects; networks on chip; topology; partial reconfiguration

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 (Dally and Brian 2001). However, these multi-core systems still incorporate rigid interconnection networks, i.e., mostly utilising a 2D-mesh as the underlying physical network topology combined with packet routers. More specifically, it is necessary for the designer to either (i) modify algorithms to suit the underlying fixed topology or (ii) embed the logical topology (intended by the algorithm) onto the physical topology, as depicted in Figure 1(a). In both cases, reduced performance is the result. The topology embedding techniques are well-studied (Leighton 1992) 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 utilise

*Corresponding author. Email: J.Y.Hur@ewi.tudelft.nl

ISSN 0020–7217 print/ISSN 1362–3060 online

DOI: 10.1080/00207210801924586
http://www.informaworld.com

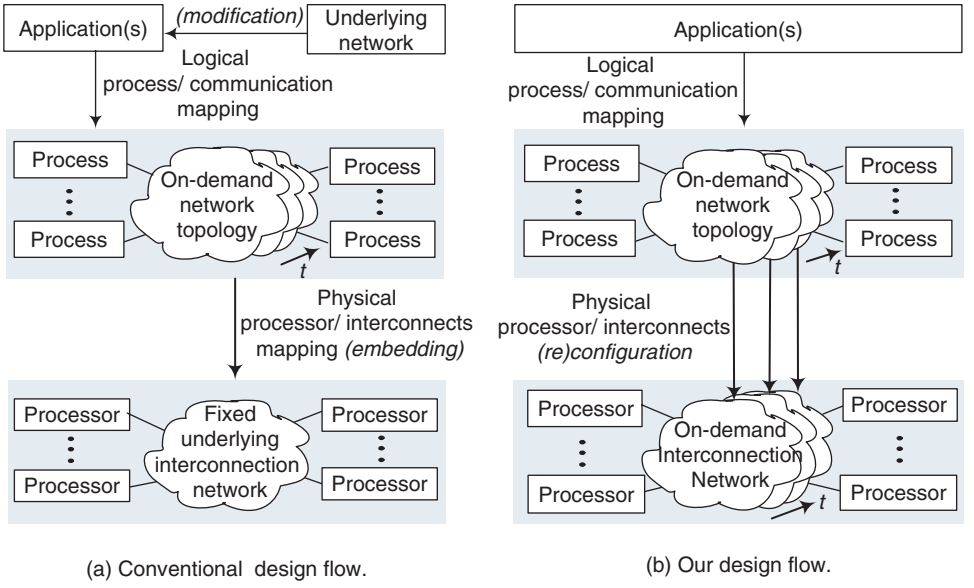

Figure 1. Running application(s) on static (a) and on-demand (b) interconnection networks.

rigid network interconnects have the following limitations. First, the programmer must have intricate knowledge of the underlying physical network in order to fully exploit it. Second, performance is deteriorated due to topology embedding that is likely to increase communication delays and/or create traffic congestions. Third, a router (usually with virtual channels) occupies significant on-chip resources. Therefore, we aim to design and implement a (dynamically) adaptive and scalable interconnection network to alleviate these problems. Figure 1(b) depicts our general approach. Our system adaptively changes the interconnection network to better suit different traffic patterns traffic patterns. The ability to 'construct' topologies on-demand (at application start time or even during run-time) is likely to improve performance.

In this work, we implement an arbitrary topology by updating small-sized partial bitstreams for the point-to-point (P2P) interconnects. Our work is motivated by several key observations. First, different applications generate different traffic patterns that require different topologies to handle them in the best possible manner. Second, the topology that practical applications require is in many cases simple (Bertozzi et al. 2005), which implies that the area of the reconfigurable network can be relatively small. Third, a parallel application is actually specified by a point-to-point data flow (or task) graph. Fourth, direct P2P network connections eliminate the aforementioned overheads of utilising packet switched networks. Fifth, modern reconfigurable hardware fabrics contain rich intra-chip wiring resources and additionally provide a capability to change the interconnections (possibly) in run-time. More than 70% of the area of current state-of-the art FPGAs is occupied by the millions of wiring segments, such as long, hex, double, and direct lines.

In order to realise the reconfigurable point-to-point (ρ-P2P) networks, we utilise Xilinx partial reconfiguration techniques (Xilinx 2004). As a comparative study, we implement

PSNs to re-examine the performance gain and the area reduction of our P2P network over the PSN in an FPGA platform. The main contributions of this work are as follows.

- The experiments show that our P2P network performs 2× better and occupies 70% less area compared to a mesh-based PSN for the considered applications. The actual system performance gain increases as the problem size increases.
- The proof-of-concept implementation provides the feasibility of the ρ-P2P interconnects. Experiments show that topology can be reconfigured in hundreds of microseconds for small sized network.

The organisation of this paper is as follows: in § 2, related work is described. Hardware implementations of the reference network and the ρ-P2P network are described in § 3 and 4, respectively. Finally, conclusions are drawn in § 5.

2. Related work

Vassiliadis and Sourdis (2006) introduced a general concept of on-demand reconfigurable interconnects, in which the interconnections of a parallel system are established on demand before or during program execution. In this paper, our ρ-P2P network is presented to realise on-demand interconnects as an viable implementation methodology using modern FPGA technology.

A number of ASIC (SoC)-targeted NoCs have been proposed, as surveyed by Bjerregaard and Mahadevan (2006). Typically, packet routers constitute tiled NoC architectures. These ASIC-targeted NoCs employ rigid underlying networks, in which the network parameters are fixed at fabrication time. Subsequently, the main drawback of these ASIC-targeted NoCs is limited adaptivity. In addition, a shared bus is still a typical network infrastructure, even though the sequential nature of communications is not desirable.

A number of FPGA-based systems employ packet switched networks (Zeferino and Susin 2003, Moraes et al. 2004, Marescaux et al. 2004, Bartic et al. 2005). Subsequently, we consider a packet switched network as a reference to compare with our approach. These NoCs are usually specified in technology-independent synthesisable HDL (or SystemC) and targets reconfigurable logic. Consequently, the NoCs constitute an overlay network on top of pre-fabricated fine-grained reconfigurable fabrics in FPGAs. Marescaux et al. (2004) designed a network for partially reconfigurable modules supported by an operating system. Bartic (2005) presented a topology adaptive parameterised router network, in which an on-demand physical topology is constructed between packet routers. Our topology construction method differs from the method of Bartic et al. (2005), in that the topology of our work is defined by the direct interconnection between processors. Kapre et al. (2006) presents time-multiplexed switches, which constitute overlay networks with a butterfly fat tree topology. Our work is similar to the work of Kapre et al. (2006) in that we discuss overheads of on-chip packet switched networks. While Kapre et al. (2006) do not discuss topology embedding, our work presents a network implementation to avoid packet processing overheads together with topology embedding.

A crossbar switch or point-to-point network are currently widely utilised for FPGA interconnects. Nikolov et al. (2006) developed a multiprocessor system that accommodates an overlay crossbar network, where crossbar interconnects or P2P network can be

systematically constructed. A full crossbar fabric provides a dynamic topology reconfiguration feature, because all-to-all topologies are accommodated in a crossbar. However, we do not consider these all-to-all crossbars in this work, since number of wires increases in a quadratic manner as number of ports increases. Patel et al. (2006) and Chang et al. (2005), presented P2P networks in high performance FPGA-based multiprocessor systems. In these systems, a direct point-to-point link (for intra-FPGA using Fast Simplex Link), fully connected all-to-all networks (for inter-FPGA using Rocket IO), and commercial ethernet switch (for inter-board) are utilised. Our ρ-P2P network is different from these traditional point-to-point networks in that the physical interconnects of our work are partially reconfigurable.

Brebner and Levi (2003) and Huebner et al. (2004) implemented networks, utilising a modern partial and dynamic reconfiguration technology. Brebner and Levi (2003) developed a large-sized (928×928 bits) all-to-all crossbar network utilising native programmable interconnect switches and look-up tables (LUTs). Huebner et al. (2004) first presented a LUT-based bus macro implementation. Our work is similar to the work of Huebner et al. (2004) in that the LUT-based bus macro is utilised. However, the utilisation of these LUTs is moderately different from ours. In Huebner et al. (2004), the buses are static and these LUTs are operated as a network itself. In other words, these bus macros interconnect computation components. On the other hand, LUTs in our work operate as anchor points to decouple computation components and communication networks. In addition, we use only necessary wiring resources to implement on-demand topologies. Our work is also close to the work of Raaijmakers and Wong (2007), Braun et al. (2007) in that partial reconfiguration of P2P wires is presented. Raaijmakers and Wong (2007) presented a dynamic wiring without using bus macros and also presents an in-house tool to realise on-line routing. While a bitstream manipulation tool by Raaijmakers and Wong (2007) targets only Virtex-II Pro xc2vp30 device, we utilise bus macros and a mainstream tool chain. Subsequently, our approach can be utilised in state-of-the-art Xilinx devices, such as Virtex-4 and 5. Braun et al. (2007) presents wire reconfiguration for newly placed modules and crossbar interconnects controlled by LUTs. Our approach differs from the work of Braun et al. (2007), in that we establish only necessary interconnects.

3. Design and implementation

In this section, we present a comparative study between the reference network and the proposed ρ-P2P interconnects. A hardware implementation of the reference 2D-mesh packet switched network (PSN) is presented in § 3.1. The proposed ρ-P2P interconnects are presented in § 3.2. In § 3.3, a performance analysis is presented for the considered network traffic.

3.1 *Packet switched networks*

We consider the PSN as a reference network as depicted in Figure 2(a). The PSN system consists of processing nodes which are interconnected in a rigid topology. Each node comprises a processing element (PE), a dual-ported memory, a network interface, and a router. Each node communicates utilising a handshaking protocol. Each packet is composed of 3 control flits and a variable amount of data flits, as depicted in Figure 2(b). 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.

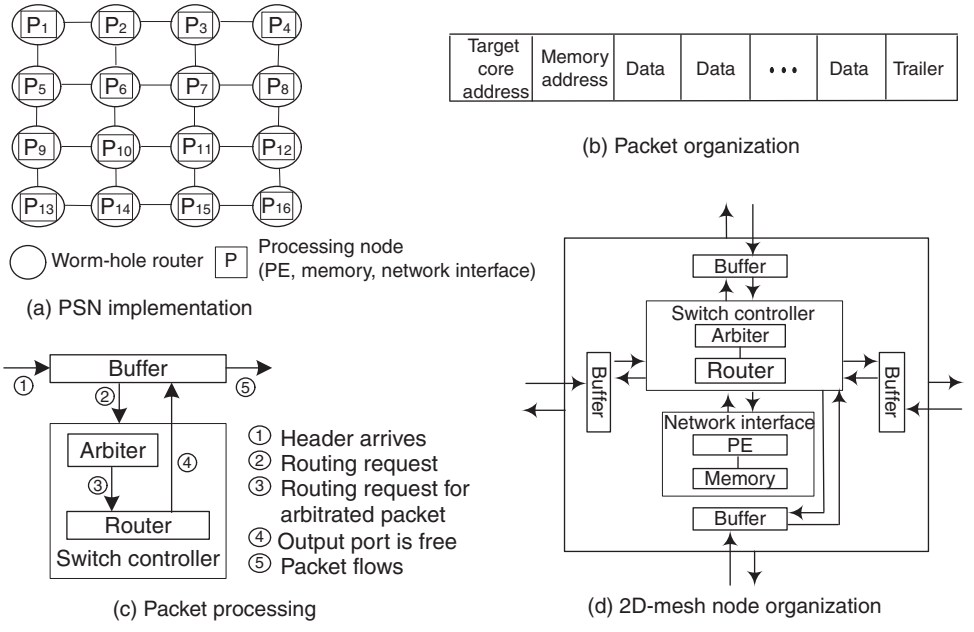

Figure 2. The 2D-mesh PSN system organisation.

The trailer indicates the last flit of a packet. Figure 2(c) depicts how a packet is processed in the router. A flit-based worm-hole flow control is adopted for more efficient buffer utilisation. 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 the current node address and a target node address are identical, the switch controller checks how many local memory ports are idle and forwards the packet to the memory of a local PE.

The switch controller permits up to two packets to simultaneously access the local dual-ported memory. This feature is useful, since two-operand computations are common in many applications. Figure 2(d) depicts how each node is organised. 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 a trailer signal. In this way, direct memory access (DMA) of burst data transfers can be supported. Each port incorporates full-duplex bidirectional links. A buffer is located at each input port and the XY routing algorithm is utilised for its simplicity. A buffer accommodates 8 flits, while the individual flit width is 16 bits. The packets are arbitrated based on the round robin scheduling policy. The PEs are wrapped inside a router in order for the dual-ported memory to simultaneously store 2 incoming packet payloads. Our implementation is similar to the network of Moraes et al. (2004) in that the same flow control and routing scheme are used. However, our PSN differs in the following ways. First, variable length burst packets are directly communicated between distributed memories. Second, dual-ported embedded memories are utilised to simultaneously accommodate two incoming packets. Third, a topology embedding (e.g., binary tree traffic embedding using a mapping algorithm of Lee and Chi (1996)) has been implemented. A single

Table 1. Router implementations for different topologies (DOR: dimension ordered routing).

Switch	# ports	Algorithm	Area(slices)	Freq. (MHz)	Peak BW(Gbps)
Binary tree	4	Horowitz	389	136.4	4.36
2D-mesh	5	DOR	626	137.6	5.5
3D Hypercube	4	DOR	472	162.6	5.2
4D Hypercube	5	DOR	674	142.3	5.69
5D Hypercube	6	DOR	826	137	6.58
6D Hypercube	7	DOR	973	133.9	7.5
Linear array	3	DOR	318	178.2	4.28
Ring	3	DOR	318	178.2	4.28
2D Torus	5	DOR	630	136.6	5.46

packet containing N_l elements requires the following amount of cycles to conduct a source-to-destination communication:

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

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.

Similarly, we can implement PSNs with different topologies. Table 1 shows a router implementation for the PSN with different topologies. Different topologies require different routing schemes. Horowitz' routing algorithm (Horowitz and Zorat 1981) is used to implement binary tree topology. Dimension-ordered routing algorithm is used for other topologies. When these overlay networks are mapped onto a target FPGA technology, the physical network performance is determined by the design itself as well as the quality of mapping/placement/routing methodology. The main drawback of these NoCs is increased cost. As shown in Table 1, single router occupies 2%~7% of xc2vp30 Virtex-II Pro device. The area overhead is significantly increased when virtual channels are implemented to maintain a quality-of-service. As an example, 16 routers with 4 virtual channels in Mello et al. (2005) occupy 25481 slices, while a common device such as the xc2vp30 Virtex-II Pro contains only 13696 slices in total. Furthermore, these networks route packets over (mostly) multi-hop routers, which increase a communication latency. In order to alleviate these problems, ρ-P2P network is presented in next section.

3.2 *ρ-P2P network*

Overview: Dynamically changing the network topology is often beneficial. These dynamic (re)wiring can be utilised in the following two cases. First, when new functionalities are plugged in the device, it is required to interconnect to (or between) these new modules. Some application scenarios that utilise the dynamic module replacement are presented in the work of Braun et al. (2007). Second, the dynamic wiring can be beneficial for an application that contains multiple subroutines, in which communication patterns of these subroutines are temporally local. As an example, Linpack benchmark application mainly consists of two routines, namely *dgefa* (factorisation of matrix) and *dgesl* (solving linear equation with back-substitution). Main subroutines in *dgefa* routine are *idamax* (finding maximum) and *dgemm* (matrix multiplication). Additionally, *dgemm* is a main

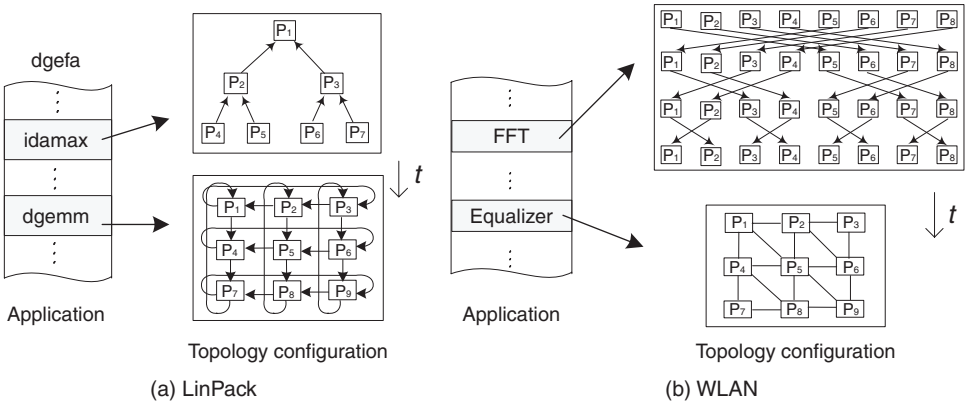

Figure 3. Applications and topologies.

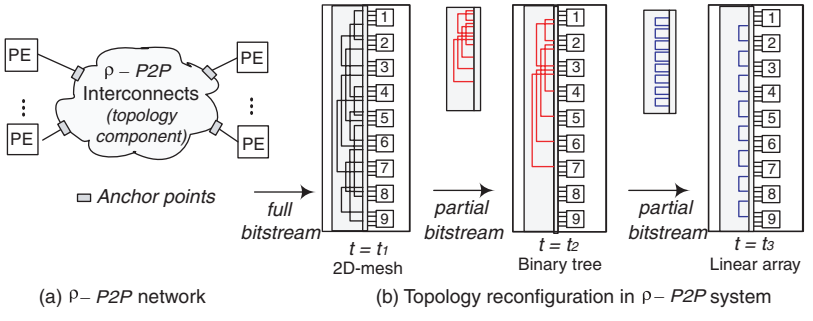

Figure 4. The ρ-P2P interconnects.

subroutine in *dgesl* routine. In this case, possible topologies for these subroutines can be a binary tree (for *idamax*) and a 2D-torus (for *dgemm*), as depicted in Figure 3(a). Another case can be found in telecommunications (or multimedia) applications. As an example, wireless LAN receiver (such as 802.11) application contains compute-intensive subroutines such as FFT and equalisation. In this case, possible topologies for these subroutines can be a butterfly (for FFT) and a 2D-mesh with diagonal (for equalisation), as depicted in Figure 3(b).

The proposed ρ-P2P network directly interconnects PEs. The network topology is implemented as a partially and dynamically reconfigurable component of the system. PEs are located in a static region and interconnects are located in the reconfigurable region. We utilise pre-fabricated native wire segments to construct the topology. The topology component is modular and can be replaced by other topologies. Anchor points, depicted in Figure 4(a), reserve wires to interconnect the reconfigurable components and the static components. Figure 4(b) depicts the exemplified reconfiguration steps where the PEs are initially interconnected in a 2D-mesh. Subsequently, the on-demand topology is reconfigured by updating the partial bitstreams only for the reconfigurable topology component during the respective times $t_2 - t_1, t_3 - t_2$. When the required communication patterns change, the physical interconnects can be quickly adapted. The reconfiguration latencies are directly proportional to corresponding partial bitstream sizes. The bitstream size is

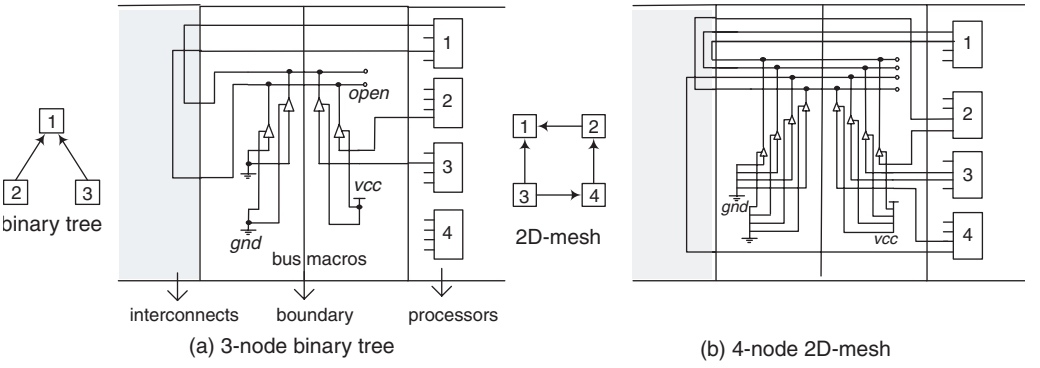

Figure 5. The topology implementation using TBUF-based bus macro array.

determined by the required on-chip resources. We exploit the partial reconfiguration technique in modern FPGAs, using which we can implement our topology reconfiguration approach as a prototype of concept. Our approach can be applied to modern LUT-based reconfigurable hardware, such as partially reconfigurable Xilinx Virtex series devices. The layout of the static region can be identical for each system configuration and remains unchanged during the interconnects reconfiguration. The small sized topology components can be reconfigured while PEs are in operation within the static region. The design flow allows the PEs in the static region to be reconfigured in the same way as the reconfigurable region is configured. These static region can be composed of either application-specific PEs or general-purpose processors. It can be noted that the design flow allows the reconfiguration of these processors. In other words, possibly new functionality can be dynamically placed onto these PEs, though we consider these PEs are static in this work. In order to conduct a source-to-destination communication, N_l elements require approximately N_l cycles. Faster communication can be achieved compared to the PSN. This is due to the fact that the data is communicated in a point-to-point fashion without packetisation and multi-hop routing overheads. Compared to the PSN, the area is also significantly reduced. This is due to the fact that the interconnection network can be realised only with firm bus-macros and native on-chip wires.

Topology implementations: We target the Virtex-II Pro device, in which the configuration frame spans full vertical height. We use bus macro to implement the anchor points in Figure 4(a). The bus macro can be implemented using symmetrical tri-state buffer arrays, as presented in Hur et al. (2007). Figure 5 depicts the topology implementation using TBUF-based bus macro as an example. Two tri-state buffers constitute a single-bit wire line. The reconfigurable region and bus macros constitute the topology component. In this work, we implement LUT arrays to construct the topologies. The reconfigurable region and bus macro array constitute the modular topology component, similarly to the TBUF-based implementation. The interconnects are enabled or disabled by controlling the LUT input signals. Different network topologies are implemented between the bus macros. Figure 6 depicts the design of a topology component using bus macros as an example. The interconnects do not require logic resources such as slices, except the power and ground signals. The LUT-based bus macro is advantageous over traditional TBUFs in the following ways. First, a LUT-based implementation provides higher density. Second, the TBUF is connected to native long lines, which obviously incur higher capacitance

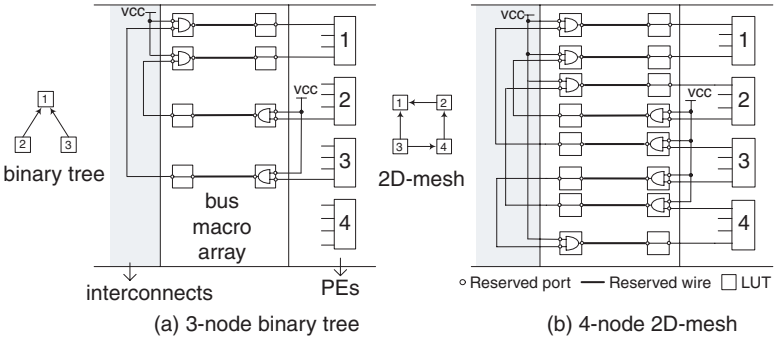

Figure 6. The topology implementation LUT-based using bus macro array.

compared to short wires, such as double, hex, direct lines. Third, these long lines can be driven by multiple locations, which often incur unstable behaviour of TBUFs.

Limitations and scalability issue: Modern FPGA interconnects can be viewed as electrically programmable physical circuit switched networks. These pre-fabricated interconnects inherently constitute reconfigurable point-to-point networks. LUTs and wires are regularly structured in an island style. Even though the wire reconfigurability is a valuable asset, FPGA interconnects have following limitations. First, underlying fabric is still bit-level. Subsequently, a designer finds difficulty to meet the timing requirements due the net delay skew. Second, logical topology and physical geographic topology are in most cases different. As an example, when coarse-grained logical direct link is mapped onto FPGAs, physical wires are spaghetti-like bit-level wires after the routing stage. Subsequently, increased wire length and delays incur reduced performance, due to the discrepancy between logical and physical networks. Third, the frame-based configuration scheme is inefficient, in that the configuration frames contain mingled bitstream for unnecessary intra-module interconnects, inter-module interconnects and logic.

In fact, a P2P network potentially has drawback in terms of wire delay, as the network size increases. However, this scalability problem also occurs for other logical networks, such as router-based network or shared bus. Though the implemented physical network behaves as a router-based network, physical wires are again bit-level wires which interconnect fine-grained LUTs. In FPGA, a logical network must be mapped onto a physical one. Based on the fact that inter-module communications are mostly coarse-grained, the degradation of the wire clock length (for both P2P network and router-based network) is inevitable. Though PSN provides a pipelined communication, a packet stays in each router for typically $3 \sim 6$ cycles throughout the communication. In order to alleviate these wire delay problems in the P2P networks, register(s) can be instantiated between long-distance routing points. By inserting register(s), the communication clock length can be reduced. In our experiment, the clock frequency of P2P network is comparable to the router-based network, as described in § 4.

3.3 *Configuration examples*

In order to compare the presented network and the reference network, we consider three workloads. Parallel merge sort (burst traffic), Cannon's matrix multiplication (element-wise traffic), and wavelet transform (burst traffic) have been chosen as

network traffic. Those algorithms are common and the implemented PEs are small in area such that larger networks can be realised in a single chip.

Parallel merge sort: The logical traffic pattern of the parallel merge sort is as follows. First, sequential sort is performed at the bottom-most nodes. Second, 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. Communication requires $O(n)$ steps for the problem size n. Sequential PEs are identical for ρ-P2P and PSN systems. Consider p PEs performing the parallel merge sort of N integers. System cycles are derived using Equations (2a), (2b), where MS_PSN and MS_P2P refer to the number of cycles when the application is operated in the PSN and P2P, respectively. In PSN, the system cycles are calculated by adding (computation cycles) + (communication cycles). Consider a complete binary tree with p PEs, $\alpha_q(2N/(p+1))\log(2N/(p+1))$ cycles are required for a sequential sort.

In P2P, $\alpha_p N$ cycles are required for the parallel computation for N total elements. For each element, α_p cycles are required to perform load, compare, forward operations. Also, $\beta_p N$ are required for communications, for each element. In case α_p and β_p are the same, all the communication cycles $\beta_p N$ are hidden by the computation, since communication and computation are overlapped.

The PSN is operated as follows. When the sequential PE points to a remote memory address and commands a packet transfer, the network interface in each bottom node generates a packet containing $2N/(p+1)$ elements. Subsequently, each packet requires $(\beta_s(2N/(p+1)) + \#hop \cdot L_h + P_{overhead})$ cycles to move up to the upper nodes. When each packet arrives in the upper nodes, the network interface assembles the packets and stores the elements in local memory. Upper nodes perform a parallel sort, in which each element requires α_s cycle(s) for the parallel computation. In Equation (2a), $(\log_2(p+1)-1)$ corresponds to the level of binary tree.

Figure 7 depicts the required steps and corresponding cycles for a 7-node 2D-mesh PSN. The sequential PE requires $\alpha_q(N/4)\log(N/4)$ cycles in bottom-most nodes $4 \sim 7$. For the sequential computation, each element requires α_q cycles. When the sequential PE points to a remote address and commands a packet transfer, the network interface generates a packet containing $N/4$ elements. After that, each packet requires $(\beta_s(N/4) + \#hop \cdot L_h + P_{overhead})$ cycles to move up to upper node 2 and node 3. When each packet arrives at node 2 and node 3, the network interface assembles the packets and

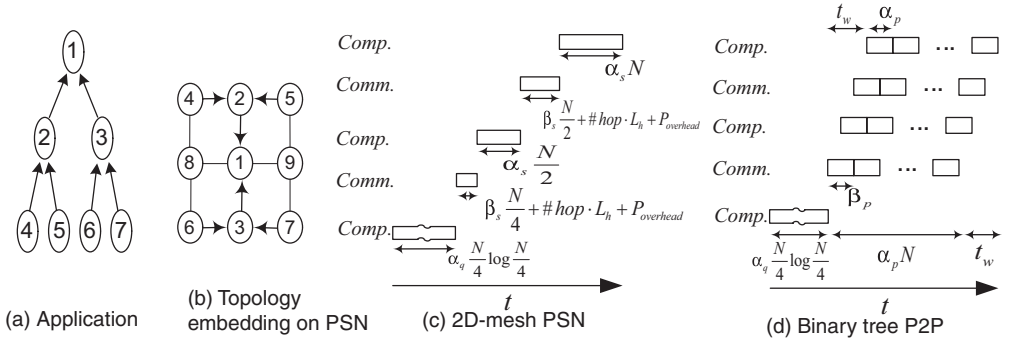

Figure 7. Way of computation and communication for parallel merge sort.

stores the elements in each local memory. Nodes 2 and 3 perform parallel sort for $\alpha_p(N/2)$ cycles. For the parallel computation, each element requires α_p cycle(s). These steps are repeated until the root node finishes the task. It can be noted that a waiting latency t_w in P2P can be negligible, since it is only 5 cycles in our implementation

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

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

Cannon's matrix multiplication: The logical 2D-torus traffic pattern of the Cannon's matrix multiplication is as follows. First, a scalar multiplication is performed at each node. Second, the intermediate result is transferred to the left/downward node. These two steps are repeated until the task is finished. A sequential computation requires $O(m^3)$ steps for the matrix of size $m \times m$. For the parallel computation, each PE is assumed to contain a single multiplier and an adder. Consider $\sqrt{p} \times \sqrt{p}$ PEs and 2 symmetric matrices with size $M \times M$. System cycles are derived using Equations (3a) and (3b), where MM_PSN, MM_P2P refer to the number of cycles when the matrix multiplication is operated in the PSN and ρ-P2P, respectively.

Figure 8 depicts the required number of cycles for the P2P and the PSN. In P2P, PEs require α_p cycles to perform multiply, add, and transfer operations for each element. There are $\sqrt{p}$ phases of computations and $M^3/p\sqrt{p}$ computation cycles are required for each phase. The communication is performed between directly connected neighboring PEs, i.e., #hop is 1. In total, $\alpha_p(M^3/p)$ cycles are required for the system cycles.

In PSN, each phase requires a packet transmission for M^2/p elements with the overheads of L_h, $P_{overhead}$. Additionally, the communication is directly performed between distributed memories

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

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

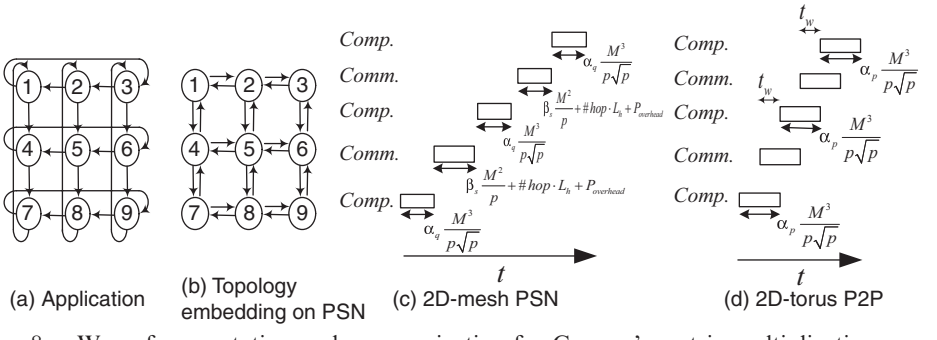

Figure 8. Way of computation and communication for Cannon's matrix multiplication.

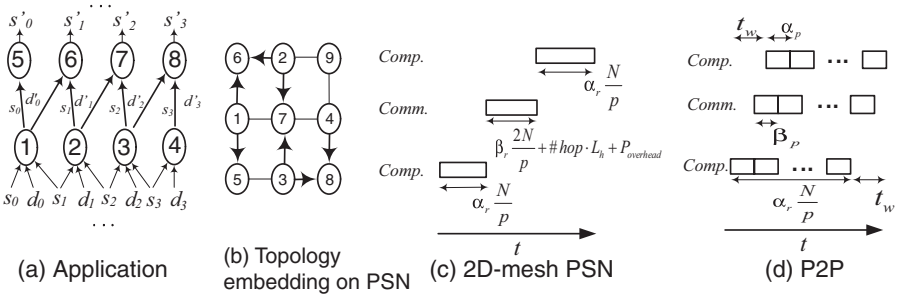

Figure 9. Way of computation and communication for lifting-based wavelet.

Wavelet transform: We consider lifting-based 2D discrete wavelet transform (DWT) application, using the (5,3) filter bank. The lifting operation consists of two stages, namely prediction stage (for high pass filter) and update stage (for low pass filter). The logical traffic pattern of the wavelet application is depicted in Figure 9(a) for 8 PEs. First, high pass filter coefficients d' are calculated by $(d_i - \lfloor (s_i + s_{i+1})/2 \rfloor)$ in nodes $1 \sim 4$ depicted in Figure 9(a). Second, low pass filter coefficients s' are calculated by $(s_i + \lfloor (2 + d'_{i-1} + d'_i)/4 \rfloor)$ in nodes $5 \sim 8$ depicted in Figure 9(a). These two steps are repeated until the task is finished. A sequential computation requires $O(n)$ steps for the image size of n pixels. Consider p PEs and problem size N. System cycles are derived using Equations (4a), (4b), where MV_PSN, MV_P2P refer to the number of cycles when the application is operated in the PSN, P2P, respectively.

In P2P, each PE requires α_r cycles to perform shifting and addition / substraction operations. The communication is performed between directly connected neighboring PEs. As a result, N/p pixels are concurrently computed and communicated in a pipelined manner. A waiting latency t_w in P2P can be also negligible, since it is only 4 cycles in our implementation.

In PSN, there are three phases of operations. First, d' is computed for N/p elements in bottom PEs. Second, burst communication of $2N/p$ elements (for d' and s) is concurrently conducted. Finally, s' is computed for N/p elements in upper PEs. In addition the communication is directly performed between distributed memories.

$$WV\_PSN = 2\alpha_r \frac{N}{p} + \left(\beta_r \frac{2N}{p} + \#hop \cdot L_h + P_{overhead}\right) \tag{4a}$$

$$WV\_P2P = \alpha_r \frac{N}{p} \tag{4b}$$

4. Experimental results

In this work, 4 experiments were conducted. First, the system cycles are analytically derived from Equations (2), (3), (4) to measure the performance gain of ρ-P2P over PSN in the FPGA. The coefficients are obtained from the customised hardware implementations. For the parallel merge sort, $\alpha_q = 2.6$, $\alpha_s = 2.3$, $\beta_s = 1$ $L_h = 6$, $\alpha_p = 2$, $\beta_p = 2$ were obtained. For the matrix multiplication, $\alpha_q = 2$, $\alpha_p = 1$, $\beta_s = 1$ were obtained. The α_q is 2, since the PE requires 2 cycles to access a local memory, while α_p is 1, since the data is

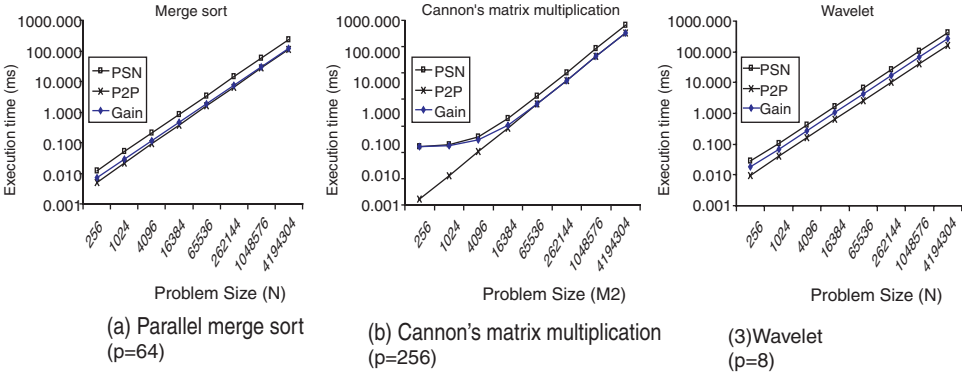

Figure 10. Execution time and performance gains of ρ-P2P and PSN.

communicated directly between PEs. For the wavelet transform, $\alpha_r = 3.7$, $\alpha_p = 1$, $\beta_r = 1$, $\beta_p = 1$ were obtained. We can fairly compare P2P and PSN, since the computational latency is the same for both cases. Figure 10 depicts the system execution time for a fixed network size and different problem sizes. The clock frequencies were obtained from the experiments. As can be observed, the P2P network performs on average 2× better (Note: graphs have a log scale). In our experiment, the clock period of P2P is comparable to the clock period of the PSN. This is due to the fact that the 2D-mesh based network occupies more area than the P2P based network, by more than factor of 3. Figure 10 also shows the performance gain of P2P over PSN in terms of the execution time. The performance gain is obtained by (PSN2 system cycles - P2P system cycles)×(clock period). For a given network size, the performance gain also increases, as the problem size increases. As Figure 10 depicts, the performance gain that can grow up to 120 ms (for parallel merge sort), 334 ms (for matrix multiplication), and 27 ms (for wavelet). It can be noted that the Virtex-II Pro xc2vp30 device requires 29 ms as an entire chip reconfiguration time. Consequently, we can note that for large problem sizes in absolute time the performance gain can grow larger than the actual reconfiguration time. This means that it can be beneficial to dynamically switch networks on demand.

Second, PSN and P2P were 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 $N = 512$ (for parallel merge sort), $M = 16$ (for matrix multiplication), and $N = 512$ (for wavelet transform). 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 11(a), 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 the 2D-mesh PSN, the binary tree is embedded in a 2D-mesh using the algorithm of Lee and Choi (1996). The binary tree P2P reduces on average 67% area and 37% of execution time compared to PSN. In Figure 11(a), PSN and P2P for the same network size can be fairly compared, since an actual number of PEs that perform the computations is the same. Figure 11(b) shows the comparison of PSN and ρ-P2P for the matrix multiplication. The sequential matrix multiplication requires 12546 ($\approx 3 \times 16^3$) cycles. An embedded

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	

(a) 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	

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

Network	Topology	#nodes	#cycles	Area		Max.	Execution time	
				#slices	reduction(%)	Freq.(MHz)	[us]	reduction(%)
Sequential	-	1	1921	279	-	119.0	16.1	-
PSN	2D-mesh	9	617	5320	93.7	115.0	5.4	60.1
P2P	Custom	8	257	333		120.0	2.1	

(c) Wavelet (N=512, p=8)

Figure 11. Implementation results.

hardwired 18 × 18 bit multiplier, an adder, and a simple control unit are implemented for each PE. These PEs are identical for ρ-P2P and 2D-mesh PSN systems. In P2P, each PE performs an integer multiplication in a single cycle. In ρ-P2P, 94.6% of an execution time is reduced, when compared to 2D-mesh PSN. This is due to the fact that the number of communication hops in P2P are significantly less than the number of hops in PSN. In PSN with 16×16 PEs, 16 hops are required for each packet, while P2P requires only a single hop. Additionally, 82% of area is reduced, since complex router modules are eliminated. Figure 11(c) shows the comparison of PSN and P2P for the wavelet transform. The sequential PE requires 1921($\approx$3.75 × 512) cycles. Each PE is implemented with simple shifter/adder units, which is identical for ρ-P2P and PSN. The P2P with on-demand topology in Figure 9(a) performs the operation with a single-hop communication, in a pipelined manner. In P2P, 60% of an execution time is reduced, when compared to 2D-mesh PSN. In 3 × 3 2D-mesh PSN, 2 hops are required to transfer each packet.

Third, we experimented on mapping the parallel merge sort traffic onto the PSN with on-demand topology, which is a similar approach to Bartic et al. (2005). Figure 12 depicts the experimental results for fixed problem size and different network sizes. As expected, the PSN with the application-specific topology performs better than the 2D-mesh PSN. The PSN with the binary tree topology performs on average 6% better, compared to the 2D-mesh PSN. The performance improvement is relatively small, since the binary tree topology can be mapped onto 2D-mesh without congestions

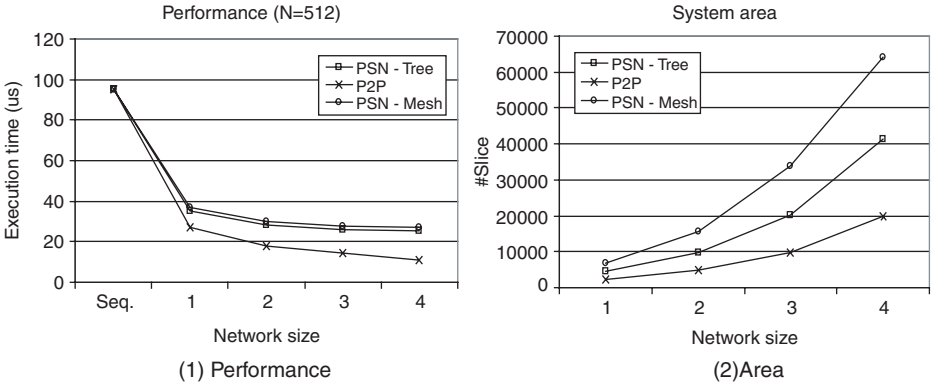

Figure 12. Mapping on different networks for parallel merge sort traffic.

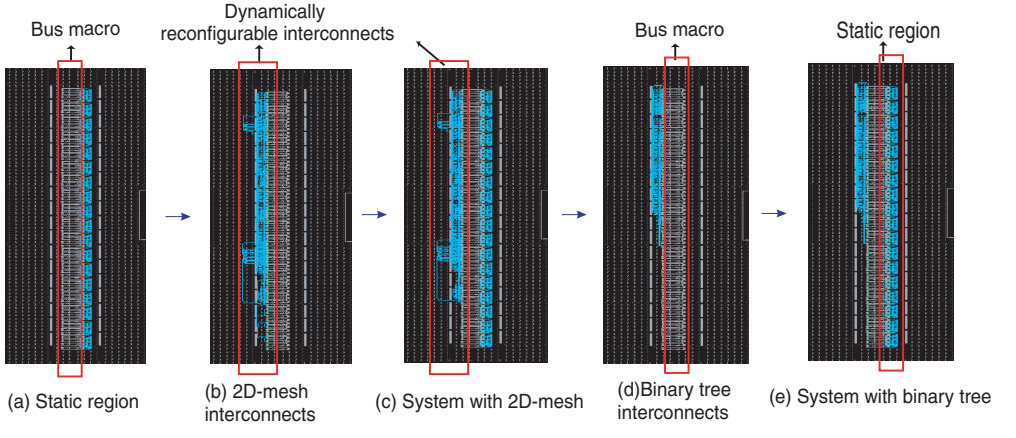

Figure 13. Partial run-time reconfiguration.

(Lee and Choi 1996). The PSN with the binary tree topology reduces 36% of area, compared to the 2D-mesh PSN. This is due to the fact that the binary tree router is simpler than the 2D-mesh router. On the other hand, our P2P network performs 39% better and requires 50% less area, compared to PSN with the binary tree. This is because of the fact that the P2P eliminates the cost as well as latency related to the router.

Fourth, intended as a proof-of-concept, the run-time reconfiguration of the interconnects was realised in the Virtex-II Pro xc2vp30 on the Digilent XUP-V2P prototyping board. Figure 13 demonstrates the example procedure of the partial run-time reconfiguration. Each sub-module was implemented in VHDL using the ISE 8.2 tool. Actual network interconnects in Figures 13(b) and 13(d) correspond to the topology components, as depicted in Figure 4. As an example, we reconfigure the binary tree interconnects by updating partial bitstream (Figure 13(d)). The layout of the static region (Figure 13(a)) is identical for each system configuration and remains unchanged during the interconnects reconfiguration. Virtex-II Pro xc2vp30 device contains 1756 frames. A single frame contains 206 words and each word is 32-bit wide. The internal configuration

	#Nets (with LUT-LUT wire delay (ns))					Area		Bitstream utilization				Configuration Time (us)
	d < 1	d < 2	d < 3	d < 4	d < 5	#LUTs	%	#Frames	%	#Set bits	%	
3 x 3 mesh PSN	18107	10161	2208	498	43	5850	21.4	1756	100	224064	1.9	28974
7 node Tree - P2P	713	183	16	32	16	97	0.4	26	1.5	3620	0.03	670
9 node mesh - P2P	870	170	0	16	0	193	0.7	48	2.7	6439	0.06	1011

Figure 14. Routing analysis.

access port (ICAP) controller configures the bitstream at a rate of 400 Mbps (= 8-bit interface × 50 MHz). Therefore, the configuration latency for an entire chip can be derived by (1756 × 206 × 32 × (1/(8 × 50 MHz)) + overhead) or 29 ms. The overhead includes the initialization latency and CRC check latency. In other words, each frame requires approximately 16.5 us of a configuration latency. The reconfiguration latency can be derived by (number of required frames × configuration latency per frame). Figure 14 depicts the routing analysis, which shows the number of utilised nets with delays, area, bitstream utilisation, and configuration time. The partial bitstream size in number of frames is only 26 (for 7-node binary tree) and 48 (for 9-node mesh) out of 1756. The required reconfiguration latencies to change topology are 670 us (for binary tree) and 1011 us (for mesh). These partial reconfiguration latencies includes overheads, such as initialization, padding frames, and CRC checks. Therefore, the reconfiguration latency can be significantly reduced by utilising these partial bitstream. Additionally, 97 LUTs for binary tree and 193 LUTs for mesh (out of 27392) of logic resources were used for reconfigurable region. When TBUFs are utilised, 79 frames (or 1.6 ms, including configuration overheads) are required to reconfigure into the binary tree topology. Therefore, 58% of reduced reconfiguration latency is achieved compared to TBUF-based implementation. Similarly, 44% of reduced reconfiguration latency is required to reconfigure into 2D-mesh topology. We have also counted the actual *set* bits in the bitstream. As depicted in Figure 14, the number of set bits is less than 2% of the entire bitstream size. In addition, the set bits for the partial bitstream is less than 0.1%. This means the actually necessary on-chip resources are extremely small. It can be noted that our approach can be better applied to LUT-based modern FPGA devices. As an example, Virtex-4 devices are configured in a reduced (16 CLBs high) height for a frame, while the Virtex-II Pro is configured in a full-height frame by frame. Moreover, Lysaght et al. (2006) reports that Virtex-4 supports 32-bit configuration interface with 100 MHz. Therefore reduced reconfiguration latency can be obtained in modern devices, compared to Virtex-II Pro.

5. Conclusions

In this work, we presented partially reconfigurable FPGA interconnects to implement on-demand network topologies. Our P2P networks have been evaluated by comparing with the 2D-mesh packet switched networks. For the considered applications, we obtained 2× better performance and 70% less area compared to the reference network (a packet switched mesh network), by directly interconnecting processors and by avoiding the use of complex routers. Moreover, we showed that the topology reconfiguration latency can be significantly reduced using a partial reconfiguration technique. Therefore, systems facilitating processors directly interconnected with the proposed reconfigurable interconnects can be suitable for our general approach.

References

Bartic, T.A., Mignolet, J.-Y., Nollet, V., Marescaux, T., Verkest, D., Vernalde, S., and Lauwereins, R. (2005), ''Topology Adaptive Network-on-chip Design and Implementation,'' *IEE Proceedings of Computers & Digital Techniques*, 152, 467–472.

Bertozzi, D., Jalabert, A., Murali, S., Tamhankar, R., Stergiou, S., Benini, L., and Micheli, G.D. (2005), ''NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip,'' *IEEE Transactions on Parallel and Distributed Systems*, 16, 113–129

Bjerregaard, T., and Mahadevan, S. (2006), ''A Survey of Research and Practices of Network-on-chip,'' *ACM Computing Surveys*, 38, 1–51.

Braun, L., Hbner, M., Becker, J., Perschke, T., Schatz, V., and Bach, S. (2007), ''Circuit Switched Run-Time Adaptive Network-on-Chip for Image Processing Applications,'' *Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL'07)*, 688–691.

Brebner, G., and Levi, D. (2003), ''Networking on Chip with Platform FPGAs,'' *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT'03)*, 13–20.

Chang, C., Wawrzynek, J., and Brodersen, R.W. (2005), ''BEE2: a High-end Reconfigurable Computing System,'' *IEEE Design & Test of Computers*, 22, 114–125.

Dally, W.J., and Brian, T. (2001), ''Route Packets, Not Wires: On-Chip Interconnection Networks,'' *Proceedings of 38th International Conference on Design Automation Conference (DAC'01)*, 684–689.

Horowitz, E., and Zorat, A. (1981), ''The Binary Tree as Interconnection Network: Applications to Multiprocessor Systems and VLSI,'' *IEEE Transactions on Computers*, 30, 247–253.

Huebner, M., Becker, T., and Becker, J. (2004), ''Real-time LUT-based Network Topologies for Dynamic and Partial FPGA Self-reconfiguration,'' *Proceedings of the 17th symposium on Integrated Circuits and System Design (SBCCI'03)*, 28–32.

Hur, J.Y., Wong, S., and Vassiliadis, S. (2007), ''Partially Reconfigurable Point-to-Point Interconnects in Virtex-II Pro FPGAs,'' *Proceedings of International Workshop on Applied Reconfigurable Computing (ARC'07)*, 49–60.

Kapre, N., Mehta, N., deLorimier, M., Rubin, R., Barnor, H., Wilson, M.J., Wrighton, M. and DeHon, A. (2006), ''Packet Switched vs Time Multiplexed FPGA Overlay Networks,'' *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06)*, 205–216.

Lee, S.-K., and Choi, H.-A. (1996), ''Embedding of Complete Binary Tree into Meshes with Row-Column Routing,'' *IEEE Transactions on Parallel and Distributed Systems*, 7, 493–497.

Leighton, F.T., (1992), *Introduction to Parallel Algorithms and Architectures: Arrays–Trees–Hypercubes*. Morgan Kaufmann Publishers, Inc.

Lysaght, P., Blodget, B., Mason, J., Young, J., and Bridgeford, B. (2006), ''Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration on XILINX FPGAs,'' *Proceedings of 16th International Conference on Field Programmable Logic and Applications (FPL'06)*, 1–6.

Marescaux, T., Nollet, V., Mignolet, J.-Y., Moffat, A.B.W., Avasare, P., Coene, P., Verkest, D., Vernalde, S., and Lauwereins, R. (2004), ''Run-time Support for Heterogeneous Multitasking on Reconfigurable SoCs,'' *Integration, The VLSI Journal*, 38, 107–130.

Mello, A., Tedesco, L., Calazans, N., and Moraes, F. (2005), ''Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC,'' *Proceedings of 18th Symposium on Integrated Circuits and Systems Design (SBCCI'05)*, 178–183.

Moraes, F., Calazans, N., Mello, A., Möller, L., and Ost, L. (2004), ''HERMES: an Infrastructure for Low Area Overhead Packet-switching Netwoks on Chip,'' *Integration, the VLSI Journal*, 38, 69–93.

Nikolov, H.N., Stefanov, T.P., and Deprettere, E.F. (2006), ''Efficient Automated Synthesis, Programming, and Implementation of Multi-processor Platforms on FPGA Chips,'' *Proceedings of 16th International Conference on Field Programmable Logic and Applications (FPL'06)*, 323–328.

Patel, A., Madill, C.A., Saldana, M., Comis, C., Pomes, R., and Chow, P. (2006), ''A Scalable FPGA-based Multiprocessor,'' *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06)*, 111–120.

Raaijmakers, S., and Wong, S. (2007), ''Run-Time Partial Reconfiguration for Removal, Placement and Routing on the Virtex-II-Pro,'' *Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL'07)*, 679–683.

Vassiliadis, S., and Sourdis, I. (2006), ''FLUX Networks: Interconnects on Demand,'' *Proceedings of International Conference on Computer Systems Architectures Modelling and Simulation (IC-SAMOS'06)*, 160–167.

Xilinx Application Note 290 (2004), ''Two Flows for Partial Reconfiguration: Module Based or Difference Based,'' Available online at: http://www.xilinx.com (accessed 9 September 2004).

Zeferino, C., and Susin, A. (2003), ''SoCIN: A Parameteric and Scalable Network-on-Chip,'' *Proceedings of 16th Symposium on Integrated Circuits and Systems Design (SBCCI'03)*, 169–174.