# Systematic Customization of On-Chip Crossbar Interconnects

Jae Young Hur[1], Todor Stefanov[2], Stephan Wong[1], and Stamatis Vassiliadis[1]

[1] Computer Engineering Lab., TU Delft, The Netherlands
http://ce.et.tudelft.nl
[2] Leiden Embedded Research Center, Leiden University, The Netherlands
http://www.liacs.nl

**Abstract.** In this paper, we present a systematic design and implementation of reconfigurable interconnects on demand. The proposed on-chip interconnection network provides identical physical topologies to logical topologies for given applications. The network has been implemented with parameterized switches, dynamically multiplexed by a traffic controller. Considering practical media applications, a multiprocessor system combined with the presented network has been integrated and prototyped in Virtex-II Pro FPGA using the ESPAM design environment. The experiment shows that the network realizes on-demand traffic patterns, occupies on average 59% less area, and maintains performance comparable with a conventional crossbar.

## 1 Introduction

A crossbar is widely used as an internet switch due to its non-blocking dedicated nature of communication and its simplicity, providing minimum network latency and minimum network congestion. It accommodates all possible connections, since traffic patterns are in most cases unknown. It is also widely used for networks-on-chip (NoC) as a basic building block [1]. Nevertheless, a major bottleneck of a conventional crossbar is the increasing amount of wires, as the number of ports grows. Figure 1 depicts the area of a conventional crossbar and a typical implementation. A crossbar consists of a traffic controller and a switch module. As the number of ports increases, the area of the switch module increases in a more unscalable way than the traffic controller. This work alleviates the scalability problem of a crossbar in a NoC-based reconfigurable platform. In general, communication patterns of different applications represent different logical topologies. The application performs best when the underlying physical interconnects are identical to the communication behavior of the parallel application. In modern NoC platforms, the logical topology information is available from the parallel application specification and the applications in most cases require only a small portion of all-to-all communications. Figure 2 depicts realistic applications [2], indicating that the required topologies are application-specific, much simpler than all-to-all topologies. Moreover, Figure 2 depicts that a single application can be specified differently (see MJPEG specifications).

P.C. Diniz et al. (Eds.): ARC 2007, LNCS 4419, pp. 61–72, 2007.

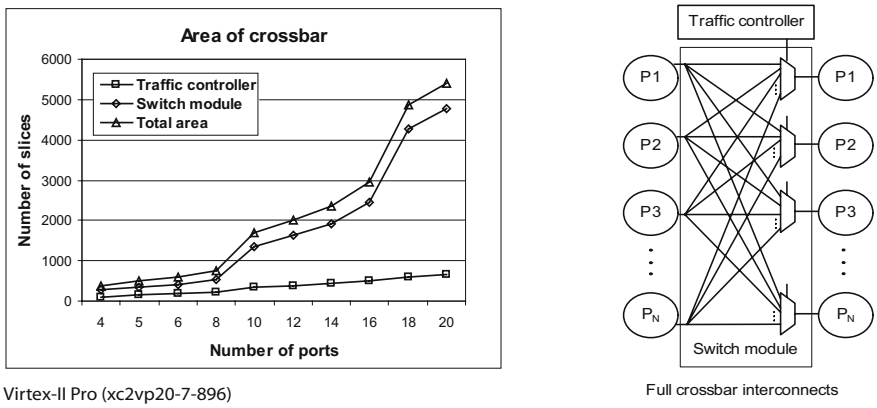



**Fig. 1.** A motivational example

Modern reconfigurable hardware can realize on-demand reconfigurability. Therefore, we designed a partial crossbar network, which dynamically adapts itself to traffic patterns, while using only the necessary wiring resources. In this work, we present a systematic design of fully customized crossbar interconnects. The presented network combines the high performance of a conventional crossbar and the reduced area of fully customized point-to-point interconnects for raw data communications. The main contributions of this work are:

- The developed network provides identical physical topologies to arbitrary logical topologies for an application.
- Experiments on realistic media applications show that on average 59% of area reduction is obtained compared to a full crossbar.
- The network component has been integrated in the ESPAM design environment [3][4].

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

## 2 Related Work

The concept and our general approach of on-demand reconfigurable networks are described in [5]. In this paper, partial crossbar interconnects in an FPGA are proposed to implement an on-demand network. Our design proposed in this paper is integrated in and verified using the ESPAM tool chain [3][4]. In [3][4], a design methodology from sequential application specifications to multiprocessor system implementations prototyped on an FPGA board is presented and considers a full crossbar interconnection network. Since a full crossbar is not scalable in terms of area, we present a customized partial interconnection network.

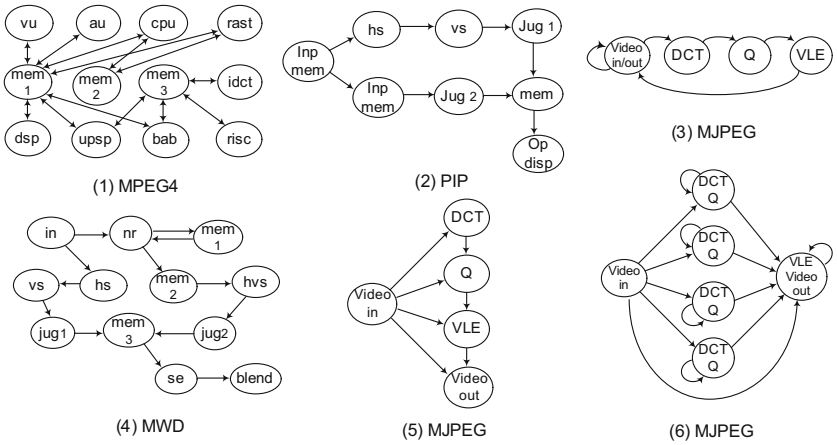



**Fig. 2.** Parallel specifications of practical applications

Numerous NoCs targeting ASICs (surveyed in [1]) employ rigid underlying networks and typically packet routers constitute tiled NoC architectures. In a typical tiled organization, one router is connected to a single processor (or IP core). Those packet routers accommodate a buffer at each port and internal full crossbar fabrics. NoCs targeting FPGAs [6]∼[11] employ fixed topologies defined at design-time. In these related work, the topology is defined by the interconnections between routers or switches, while the topology of our work is defined by the direct interconnection between processors. Figure 3 summarizes related work in terms of system sizes, clock frequencies, target devices, data widths, buffer depths, number of I/O ports, occupied network areas per switch (or router) and design characteristics of [6]∼[11]. Each of them entails specific design goals and different characteristics. As an example, [7] is designed for a partially reconfigurable module supported by the OS. [6]∼[8] adopt the packet switching and each router contains a full crossbar fabric. In [10], 10 switches are connected with 8 processors. In [11], 4 different topologies are presented and each circuit router is connected with multiple nodes. [9] presents a topology adaptive parameterized network component, which is a close approach to our work. While the physical topology in [9] is constructed between multi-ported packet routers, our adaptive topology is constructed between processors using partial crossbar interconnects. Our centralized partial crossbar constitutes a system network.

Recently, a couple of application-specific NoC designs are proposed. [12] presents a multi-hop router based network customization, while our network is single-hop based. [13][14] present an internal STbus crossbar customization and verified it through simulation. Our work is similar to [13][14] in that a crossbar is customized. In [13][14], each arbiter of the bus-based partial crossbar is connected to all processors, while in our work, a point-to-point direct link is established

and different sized multiplexors are utilized. Additionally, our work is verified by actual prototyping. Finally, [15][16] present dynamic configurations of bus segments, while our work presents a full customization of NoC topologies.

| NoC | Year | System #cores | Freq. MHz | Chip | Switch | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | DataWidth | BufferSize | I/O | Area (#slices, #BRAMs) | Characteristic |
| [6] | 2004 | 2X2 | 25 | V2-1000-4 | 8-bit | 8 flits | 5 | 316 | worm-hole |
| [7] | 2004 | 3X3 | 33 | V2-6000 | 16-bit | BRAM | 5 | 446, 5 BRAMs | virtual cut-through |
| [8] | 2005 | 3X3 | 33 | V2Pro30 | 8-bit | BRAM | 5 | 352, 10 BRAMs | parallel routing |
| [9] | 2005 | 3X3 | 50 | V2Pro40 | 16-bit | BRAM | 5 | 552, 5 BRAMs | flexible topology |
| [10] | 2006 | 8 | 166 | V2-6000-4 | 32-bit | no buffer | 4 | 732 or 1464 | time-multiplexed |
| [11] | 2006 | 8 | 134 | V2Pro30-7 | 16-bit | no buffer | 8 | 1223, 1 BRAM | circuit switching |

**Fig. 3.** Summary of related work

## 3 Design and Implementation

As mentioned earlier, our goal is to design reconfigurable interconnection networks, in which the physical topology is identical to the logical topology specified by the application partitioning. The physical interconnects are also required to be instantly switched to adaptively meet the dynamic traffic patterns. The logical topology is represented by a point-to-point graph, in which each node has possibly a different number of input and output links. In this work, a parameterized multiplexor array has been implemented for switch modules as a design technique. Topology-specific and different sized multiplexors ensure that demanding interconnects are established. Additionally, routing paths are selected by a crossbar traffic controller. The switch module is generic in terms of data widths, number of processors, and custom topologies. In this way, the network interconnects can be adapted to an arbitrarily specified logical topology and arbitrary number of processors.

### 3.1 Design Environment

The parameterized switch module has been integrated as a modular communication component in the ESPAM tool chain as depicted in Figure 4, in which the MJPEG data flow specification in Figure 2(3) is considered as an example. Details of the ESPAM design methodology can be found in [3][4]. In ESPAM, 3 input specifications are required, namely application / mapping / platform specification in XML. An application is specified as a Kahn Process Network (KPN). A KPN is a network of concurrent processes that communicate over unbounded FIFO channels and synchronize by a blocking read on an empty FIFO. The KPN is a suitable model of computation on top of the presented crossbar-based interconnection network, due to its point-to-point nature of communication and its simple synchronization scheme. However, the design technique of the presented network can be used in other systems. A KPN specification is automatically generated from a sequential Matlab program using the COMPAAN tool [17]. From the KPN specification, a task dependency graph is manually extracted.

Each process is assigned to a specific processor in the mapping specification. The number of processors, type of network and a port mapping information are specified in the platform specification. Figure 4 depicts how the customized interconnects can be implemented from the specified application. In the platform specification, four processors are port-mapped on a crossbar. From the mapping and platform specifications, port-mapped logical network topology is extracted as a static parameter and passed to ESPAM. Subsequently, ESPAM refines the abstract platform model to an elaborate parameterized RTL (hardware) and C/C++ (software) models, which are inputs of the commercial Xilinx Platform Studio (XPS) tool. Finally, the XPS tool generates the netlist with regard to the parameters passed from the input specifications and generates a bitstream for the FPGA prototype board to check the functionality and performance.

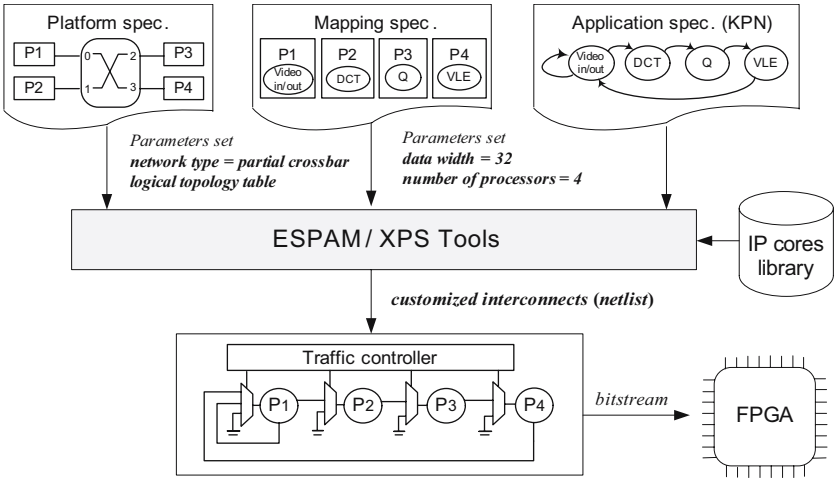



**Fig. 4.** An integration in the ESPAM design framework

### 3.2 Implementation of the Partial Crossbar

The switch module is customized with parameterized multiplexor arrays using a generic VHDL function. Our network has been implemented with the following steps:

1. Port mapping: Given an application and a mapping specification, topologically sorted processors are associated with crossbar ports in the platform specification.
2. Topology extracting: The topology information is extracted from the three input specifications.
3. Parameter passing: The extracted topology tables are passed as static parameters to the switch module. Multiplexor select signals are generated in the traffic controller and also passed as a dynamic parameter to the switch module.

Figure 5 depicts how actual interconnects are customized for the MJPEG application specification in Figure 2 (3). First, priorities are given to task-assigned processors based on the topological sort in the task graph. Figure 5(1) depicts the topological sort and linear ordering of processors. The dotted lines represent the levels of sorting. Considering *Video_in* node P1 as a root in the first level, a next priority is given to the directly connected processor in the second level. Following the precedence graph, these steps are repeated until the priority is given to all candidates. Consequently, the processors are linearly ordered in the following sequence: P1,P2,P3,P4. After that, all of these processors are associated with the crossbar ports. The round-robin scheduler in the traffic controller performs an arbitration of the requests during run time using the topology sort. Note that the processor without incoming links in the task graph does not request for data and the scheduler excludes those processors in the scheduling candidate lists. As an example, $Video\_in$ nodes in Figure 2(5) and (6) are not considered for the scheduling in the traffic controller.

Second, the graph topology is extracted from the three input specifications. Each processor port has a set of incoming and outgoing links, as depicted in Figure 5(3). Table 1 shows the number of incoming links and a list of ports from which the links are originated (for the task graph in Figure 2(3)). As an example, port P1 has two incoming links from ports P1 and P4, indicating that processor P1 reads the data located in the FIFOs connected to either P1 or P4. Table 2 shows the number of outgoing links and a list of ports to which the links are directed. As an example, P1 port has two outgoing links to ports P1 and P2, indicating that the data in FIFOs connected to P1 is transferred to either processor P1 or processor P2. The topology of the physical interconnection network can be efficiently constructed using customized multiplexor arrays. Table 1 and 2 are used to systematically implement customized multiplexor arrays instead of full multiplexors. Note that an $N$-port full crossbar contains $N$-way multiplexors per port, while our network contains variable-way multiplexors, depending on the graph topology. There are two types of multiplexors, namely processor-side multiplexors and FIFO-side multiplexors, as depicted in Figure 5(3). Table 1 is used to implement processor-side multiplexors controlled by $CTRL\_FIFO$ signals and Table 2 is used to implement FIFO-side multiplexors controlled by $CTRL\_PROC$ signals. A state diagram of the traffic controller which controls signals $CTRL\_FIFO$ and $CTRL\_PROC$ is depicted in Figure 5(2). The traffic controller deals with a handshaking protocol (i.e., data request, acknowledgement, data transfer) and a round-robin arbitration. The arrows indicate state transition conditions and the bold words indicate the actions in each state. The traffic controller checks whether there is a request using a circular round robin policy. The request signal contains a target port and a target FIFO index. In case there is a request, the request is registered and the traffic controller checks whether the target port is busy or idle. If the target port is idle and the designated FIFO contains data, $CTRL\_FIFO$ and $CTRL\_PROC$ signals are generated and multiplexor input links are selected by those two signals.

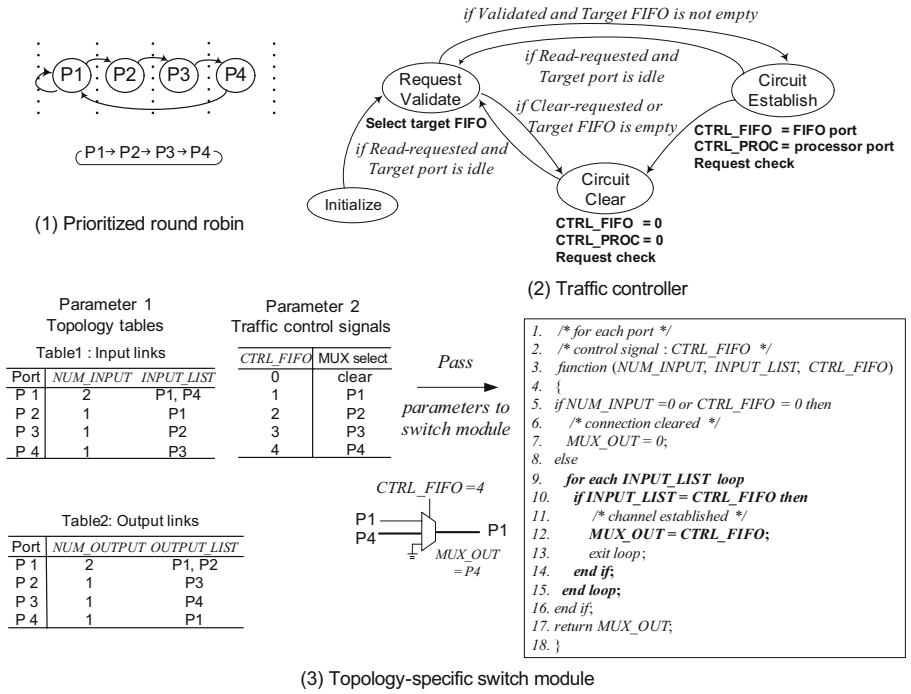



**Fig. 5.** Parameterized switch module and traffic controller

Third, a VHDL function has been implemented to generate multiplexor arrays. The two parameters described above are passed to a VHDL function to actually establish a circuit link. As an example, processor P1 reads a data located in a FIFO connected to P4, with $CTRL\_FIFO = 4$ generated by the traffic controller, as depicted in Figure 5(3). The function to generate processor-side multiplexors has been implemented with a simple priority encoder as described in lines 9∼15 in Figure 5(3). The function for FIFO-side multiplexors can be implemented in the same way. Once the request is given the priority, 2 cycles are required to establish a circuit link to the designated target port. Once a link is established, a remote memory behaves as a local memory until the link is cleared.

The communication controller in [3][4] is also used in this work as a common network interface in order to integrate the presented network component in ES-PAM. Figure 6(1) depicts that P1 connected to the crossbar port 0 reads from a remote memory connected to crossbar port 3 (represented by the bold line). The switch module requires three multiplexors per crossbar port as depicted in Figure 6(2). Multiplexors for other ports are not depicted for the sake of clarity. P1 sends a read-request to the traffic controller and the traffic controller sends the designated FIFO index to the communication controller connected

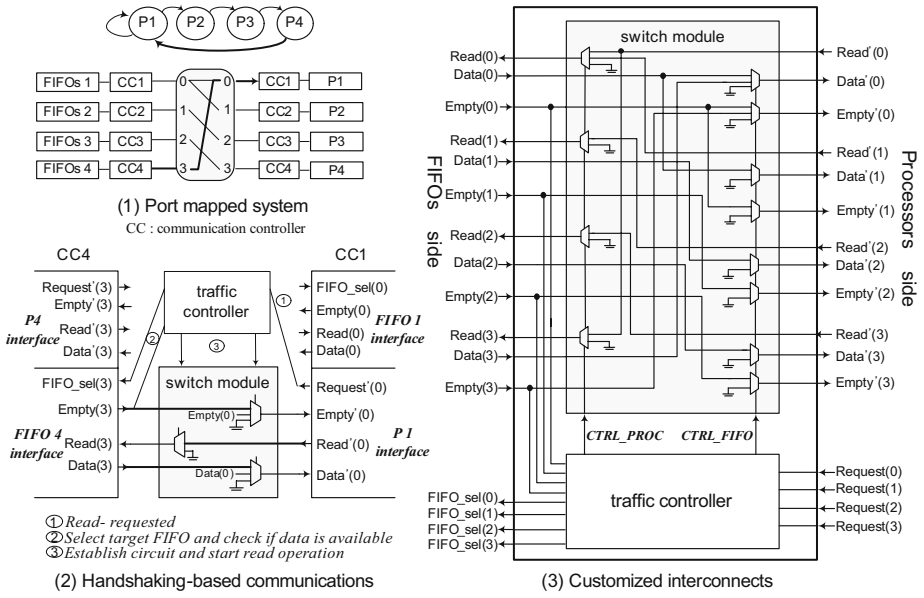



**Fig. 6.** Customized interconnection network

to P4. P4 responds to the traffic controller whether the FIFO is empty or not. If the FIFO is not empty, the traffic controller generates *CTRL_PROC* and *CTRL_FIFO* signals to establish a communication line. Figure 6(3) depicts the finally customized 4-port switch module, in which 12 multiplexors are instantiated. In general, a $N$-port switch module in our network contains $3 \times N$ variable-way multiplexors. In our network, the data is communicated in a point-to-point manner without a packetization, without multi-hop header processing overheads, and without intermediate buffering overheads. Therefore, a low latency and a high throughput communication is achieved. The occupied network area depends on the logical topology of an application. Only in case the application requires an all-to-all topology, the network is identical to a full crossbar. Additionally, our network efficiently utilizes the available bandwidth, since only required links are established and the links are fully utilized.

## 4 Experimental Results

In this work, two experiments have been conducted for realistic applications. First, a full crossbar and the presented partial crossbar have been compared in terms of area utilization in order to measure the area reduction. The application task graphs of MPEG4, PIP, MWD are taken from [2]. The task graphs of H.263 encoding, MP3 encoding, and MMS are taken from [18]. The task graphs of 802.11 MAC, TCP checksum, VOPD are taken from [15],[19],[20], respectively.

Assuming each node is associated with a single crossbar port, the full and partial crossbar networks are synthesized, placed and routed using the Xilinx ISE tool on Virtex-II Pro (xc2vp20-7-896) FPGA and the areas have been obtained. Figure 7 depicts topologies, number of nodes, number of required links, area of the traffic controller, area of the switch module, area of the crossbar, chip occupation and area reduction in percentage. The crossbar area is the summation of the area of the traffic controller and switch module. As Figure 7 shows, the area of the network is highly dependent on the number of nodes and links. The partial crossbar network requires on average 61% less area, compared to the full crossbar. The same traffic controller is used for both cases. The area of our network is not only dependent on the number of nodes that determine its size but also on the network topology. It is observed that the higher area reduction is obtained as the network size increases. This is due to the fact that the average number of incoming and outgoing links per node does not increase as the number of nodes increases.

| Topologies | #nodes | #links | Type | Area (#slices) | | | Chip area(%) | Reduction(%) |
|---|---|---|---|---|---|---|---|---|
| | | | | Controller | Switch | Combined | | |
| TCP Checksum | 5 | 14 | Full | 148 | 340 | 488 | 5.3 | 40.4 |
| | | | Partial | 148 | 143 | 291 | 3.1 | |
| MP3 enc | 5 | 10 | Full | 148 | 340 | 488 | 5.3 | 47.7 |
| | | | Partial | 148 | 107 | 255 | 2.7 | |
| H.263 enc | 7 | 14 | Full | 196 | 476 | 672 | 7.2 | 49.7 |
| | | | Partial | 196 | 142 | 338 | 3.6 | |
| PIP | 8 | 8 | Full | 211 | 544 | 755 | 8.1 | 55.6 |
| | | | Partial | 211 | 124 | 335 | 3.6 | |
| 802.11 MAC | 9 | 20 | Full | 315 | 1224 | 1539 | 16.6 | 64.5 |
| | | | Partial | 315 | 231 | 546 | 5.9 | |
| MPEG4 | 12 | 26 | Full | 387 | 1632 | 2019 | 21.8 | 64.6 |
| | | | Partial | 387 | 328 | 715 | 7.7 | |
| MWD | 12 | 13 | Full | 387 | 1632 | 2019 | 21.8 | 70.9 |
| | | | Partial | 387 | 200 | 587 | 6.3 | |
| VOPD | 16 | 20 | Full | 493 | 2448 | 2941 | 31.7 | 73.3 |
| | | | Partial | 493 | 291 | 784 | 8.4 | |
| MMS | 25 | 47 | Full | 798 | 6800 | 7598 | 81.9 | 81.8 |
| | | | Partial | 798 | 587 | 1385 | 14.9 | |

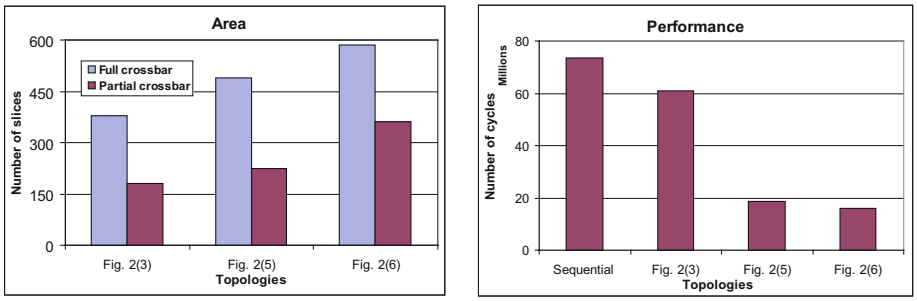
**Fig. 7.** Experiments on topologies of practical applications

Second, actual systems have been implemented in order to measure the performance and the area of the presented network on a prototype board. We considered an MJPEG encoder application that operates on a single image with size $128 \times 128$ pixels. We experimented with different MJPEG task graph topologies. We used the ESPAM tool chain and prototyped a multiprocessor MJPEG system onto the ADM-XPL FPGA board[21]. Figure 8 depicts the experimental results. We have experimented with the three alternative task graphs in Figure 2(3),(5),(6), where our network provided the on-demand topologies. The implemented system is homogeneous in that each node contains a MicroBlaze

processor. It can be observed that the topology plays an important role for the system performance. The partial crossbar network requires on average 48% less area, compared to the full crossbar. The system cycles decrease as the number of processors increase and the performance of the partial crossbar is comparable to the full crossbar. The operating clock frequency varies with the network configurations. As an example, the network component for Figure 2(3) is operated at 119MHz and the data width is 32-bit. Therefore, 3.8Gbps of bandwidth per link is available in the standalone network. The network component is not a bottleneck for the system performance, since the embedded block RAMs (used for FIFOs) operate at 100MHz.

| Topology | from Fig. 2 (3) | | | | from Fig. 2 (5) | | | | from Fig 2 (6) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #nodes | 4 | | | | 5 | | | | 6 | | | |
| #links | 5 | | | | 7 | | | | 14 | | | |
| Type | Performance | Area (#slices) | | | Performance | Area (#slices) | | | Performance | Area (#slices) | | |
| | System cycles | Controller | Switch | Combined | System cycles | Controller | Switch | Combined | System cycles | Controller | Switch | Combined |
| Full | 61104796 | 107 | 272 | 379 | 18580930 | 148 | 340 | 488 | 15940149 | 176 | 408 | 584 |
| Partial | 60862613 | 107 | 73 | 180 | 18580768 | 148 | 76 | 224 | 15940023 | 176 | 186 | 362 |

(1) MJPEG case study




(2) Area and performance

**Fig. 8.** Experiments on the prototype for MJPEG applications

## 5 Conclusions

In this paper, we presented an actual design and implementation of novel partial crossbar interconnects designed for reconfigurable hardware. We showed that on-demand interconnects can be implemented using parameterized multiplexor arrays. The network was integrated in the ESPAM tool and multiprocessors interconnected with our networks were implemented on a prototype board. We showed that the performance of our partial crossbar-based network is comparable to the performance of a conventional crossbar. The presented network efficiently utilizes the available bandwidth. Moreover, our network provides a single hop communication latency and the network area is significantly reduced compared to a full crossbar.

**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. 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.
2. D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G.D. Micheli, "NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip," IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 2, pp. 113–129, Feb 2005.
3. H. Nikolov, T. Stefanov, and E. Deprettere, "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), pp. 323–328, Aug 2006.
4. H. Nikolov, T. Stefanov, and E. Deprettere, "Multi-processor System Design with ESPAM," Proceedings of 4th IEEE/ACM/IFIP International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS'06), pp. 211-216, Oct 2006.
5. 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.
6. 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.
7. T. Marescaux, V. Nollet, J.Y. Mignolet, A.B.W. Moffat, P. Avasare, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, "Run-time Support for Heterogeneous Multitasking on Reconfigurable SoCs," Integration, the VLSI Journal, vol. 38, no. 1, pp. 107–130, Oct 2004.
8. B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri, "LiPaR: A Lightweight Parallel Router for FPGA-based Networks-on-Chip," Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI'05), pp. 452–457, Apr 2005.
9. T.A. Bartic, J.-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins, "Topology adaptive network-on-chip design and implementation," IEE Proceedings of Computers & Digital Techniques, vol. 152, no. 4, pp. 467–472, Jul 2005.
10. N. Kapre, 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.
11. C. Hilton and B. Nelson, "PNoC: A flexible circuit-switched NoC for FPGA-based systems," IEE Proceedings of Computers & Digital Techniques, vol. 153, no. 3, pp. 181–188, May 2006.
12. K. Srinivasan and K.S. Chatha, "A Low Complexity Heuristic for Design of Custom Network-on-chip Architectures," Proceedings of International Conference on Design, Automation and Test in Europe (DATE'06), pp. 130–135, Mar 2005.

13. S. Murali and G.D. Micheli, "An Application-specific Design Methodology for STbus Crossbar Generation," Proceedings of International Conference on Design, Automation and Test in Europe (DATE'05), pp. 1176–1181, Feb 2005.
14. M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon, "Analyzing On-Chip Communication in a MPSoC Environment," Proceedings of International Conference on Design, Automation and Test in Europe (DATE'04), pp. 752 – 757, Feb 2004.
15. K. Sekar, K. Lahiri, A. Raghunathan, and S. Dey, "FLEXBUS: A High-Performance System-on-Chip Communication Architecture with a Dynamically Configurable Topology," Proceedings of 42th International Conference on Design Automation Conference (DAC'05), pp. 571–574, Jun 2005.
16. 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.
17. B. Kienhuis, E. Rijpkema, and E. Deprettere, "Compaan: Deriving Process Networks from Matlab for Embedded Signal Processing Architectures," Proceedings of 8th International Workshop on Hardware/Software Codesign (CODES'2000), pp. 13–17, May 2000.
18. J. Hu and R. Marculescu, "Energy-Aware Mapping for Tile-based NoC Architectures Under Performance Constraints," Proceedings of the 8th Asia and South Pacific Design Automation Conference (ASP-DAC'03), pp. 233–239, Jan 2003.
19. K. Lahiri, A. Raghunathan, G. Lakshminarayana and S. Dey, "Design of High-Performance System-On-Chips Using Communication Architecture Tuners," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23, no. 5, pp. 620–636, May 2004.
20. S. Murali and G.D. Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," Proceedings of International Conference on Design, Automation and Test in Europe (DATE'04), pp. 896–901, Feb 2004.
21. Alpha Data Parallel Systems, Ltd., http://www.alpha-data.com/adm-xpl.html.