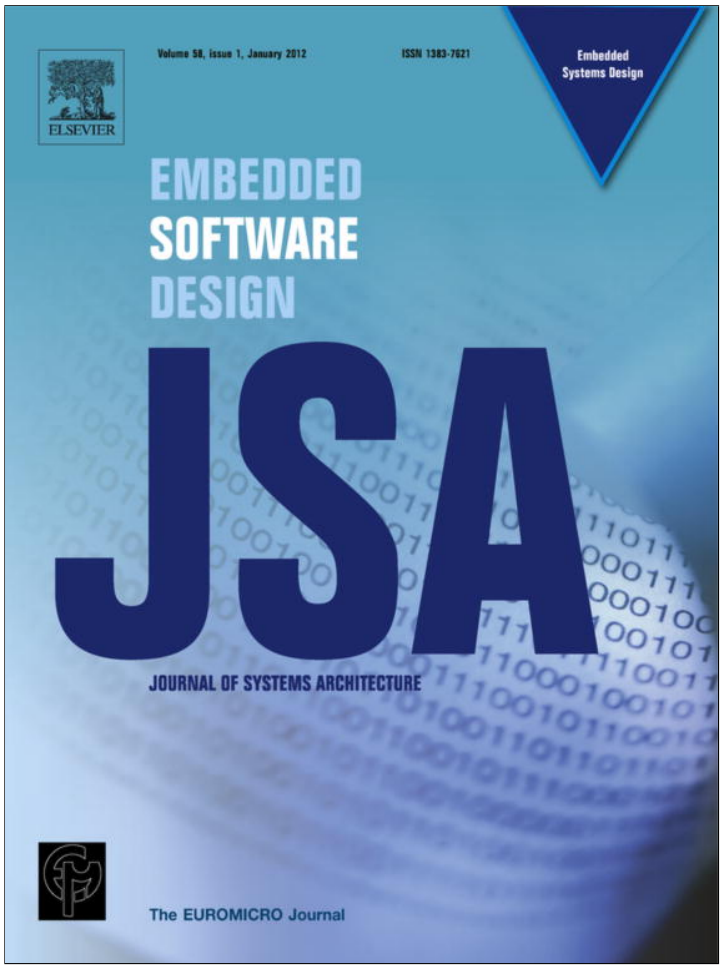


(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

Journal of Systems Architecture 58 (2012) 140–159

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A dynamically reconfigurable communication architecture for multicore embedded systems ☆

Salih Bayar *, Arda Yurdakul

Computer Engineering, Boğaziçi University, P.K. 2 TR-34342 Bebek, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 13 March 2011
Received in revised form 10 February 2012
Accepted 13 February 2012
Available online 23 February 2012

Keywords:
FPGA
Runtime reconfiguration
Partial reconfiguration
Dynamic reconfiguration
Bitstream compression
Multiprocessor
NoC
Interconnect

ABSTRACT

To deal with the communication bottleneck of multiprocessor systems, several communication architectures have been proposed in the last decade. Yet, none of them has demonstrated the performance of the direct connections between two communicating units. In this paper, we propose dynamically reconfigurable point-to-point (DRP2P) interconnects for setting up direct connection between two communicating units before the communication starts. DRP2P is neither point-to-point (P2P) nor Network-on-Chip (NoC); it stands between these two on-chip communication architectures. It is as fast as P2P and as scalable as NoC. Instead of using routers like in NoC, we utilize partial reconfiguration ability of FPGAs for routing data packets. Furthermore, DRP2P can work both on regular and irregular topologies. The only drawback of our approach is the reconfiguration latency. This drawback is completely hidden when the reconfiguration of the communication links is achieved during the computation times of the cores. DRP2P solves the scalability issue of P2P by setting up on-demand communication-specific links between cores. So, the occupied area and the total power consumption of communication architecture can be reduced significantly. We designed an on-chip self-reconfiguration core, c^2PCAP so as to achieve DRP2P interconnects as fast as possible. The c^2PCAP core is designed for Xilinx FPGAs and can partially reconfigure the FPGA at the highest rate proposed by the manufacturer (e.g. up to 400 MB/s for Virtex-4).

1. Introduction

The fastest communication architecture between two cores is known as the point-to-point (P2P) connections where two cores are directly connected. As the number of cores increases, P2P turns out to be a power- and area-hungry solution. In the last decade, the Network-on-Chip (NoC) has been proposed as an alternative to reduce power consumption and has been widely adopted by the SoC community.

Conventionally, two types of NoCs have been proposed in the literature: packet-switched and circuit-switched. The data transfer time is much shorter in circuit-switched NoCs than it is in packet-switched NoCs, because a dedicated path between two nodes is established before communication takes place. However, the circuit set-up time introduces additional latency on the overall communication. Besides, the dedicated communication path might make other nodes wait for a communication channel. The packet-switched NoC solves this problem by limiting the size of data traveling in the NoC to the size of a packet. Yet, this type of NoCs suffer from not only process time for packetization of data, header processing or buffering but also communication time overhead due to congestion control [1]. Hence, conventional NoCs suffer from communication latency, area overhead and power consumption introduced to the system by the routers [2].

Based on this observation, adaptive, programmable, reconfigurable NoC architectures have been proposed in the literature. Recently, methods that by-pass some of the routers are introduced so as to reduce the communication latency due to routers. In [9] these wires are static. In VIP [2], Skip-links [5] and RecoNoC [7], it is shown that these links can also be introduced dynamically to the system. In Reconfig-Net [6], the wires to and from the routers are dynamically added or removed. The number of routers is reduced by static analysis. In [10], three different communication architectures are utilized on the same SoC and one of these architectures is selected during run-time. This is accomplished by switches and an adaptive look-up table at each router. Topology switches are first introduced in ReNoC [4] where hybrid topologies can be setup by taking application specifications into account. Adaptive look-up tables are proposed in PNoC [1]. In RecoNoC

☆ This work is fully supported by The Scientific and Technological Research Council of Turkey, TÜBİTAK (Project No.: 104E038), Boğaziçi University Scientific Research Projects (Project No.: 5582 and 6346) and State Planning Organization of Turkey, (DPT) under the TAM Project, Grant No. 2007K120610.

* Corresponding author. Tel.: +90 212 359 7780; fax: +90 212 287 2461.
E-mail addresses: salih.bayar@boun.edu.tr (S. Bayar), yurdakul@boun.edu.tr (A. Yurdakul).
URLs: http://www.cmpe.boun.edu.tr/bayar (S. Bayar), http://www.cmpe.boun.edu.tr/yurdakul (A. Yurdakul).

1383-7621/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2012.02.003

[7], parameterized look-up tables are utilized to reduce reconfiguration time. In PNoC, modules are placed and removed dynamically as it is the case in DyNoC [3] where the unused routers are reused for computing purposes. In [8], a 2-D Reconfigurable Mesh NoC is developed. It is a new hybrid architecture, which uses not only routers but also configuration switches for packet routing. Some of these studies remain at simulation level [5], but there exist implementations either as an ASIC [2,4,8] or on a Xilinx FPGA [9,6,10,7,1,3]. The latest NoC approaches and our proposed architecture are summarized in Table 1.

Routers have a major effect on the occupied area and system power consumption. To avoid the drawbacks of routers, we tried to implement a router-less NoC-like system. In our case, routing is done by utilization of partial reconfiguration ability of FPGAs. From Table 1, it is obvious that almost all studies try to suppress the drawbacks of routers used in both packet and circuit switched networks. To achieve this, either the number of routers is reduced or routing through routers is minimized. This is done by either developing simpler router architectures or adding additional configuration switches or using dedicated P2P links for neighbor/long distant PEs and so on.

Xilinx FPGAs are well-known for the dynamic partial reconfiguration availability for almost every unit in the FPGA [11]. However, very few of the NoC implementations on the FPGAs utilize this property [6,1,3]. In these studies, power consumption and latency due to reconfiguration have not been reported. It is also ambiguous how reconfiguration is triggered and carried out in these studies. In the literature, there are agents and methods for dynamic partial reconfiguration, but these studies do not explain which one is utilized [12–21]. Other run-time adaptive approaches propose new reconfiguration or adaptation methods based on run-time monitoring of the application [2,3,8]. However, they do not explicitly mention how long their proposed reconfiguration/adaptation/monitoring scheme will take or how much power it will consume. The numerical time values for the proposed reconfiguration methods are not very promising. For example, TMAP [22] method adopted in RecoNoC requires 215 ms to update 8-bit coefficients of a 32-tap FIR. This is a quite long time for setting up communication architecture.

In this work, we propose a new communication architecture, namely dynamically reconfigurable point-to-point (DRP2P) interconnects. In DRP2P, routers need not exist. As a result, the area of the implementation is reduced. This directly reduces power consumption. Direct communication paths are setup by dynamic partial self-reconfiguration. This is done by exploiting the dynamic partial reconfiguration property of the Xilinx FPGAs with c^2PCAP, a dedicated on-chip engine developed for this purpose. This core adds a small overhead in the area and power consumption. However, the experimental results show that power consumption of DRP2P is much lower than that of a conventional two-dimensional NoC.

DRP2P is neither P2P nor NoC; it stands between these two on-chip communication architectures. It is as fast as P2P and as scalable as NoC. In NoC, the communication flows over the selected lines through the routers. In DRP2P, the communication flows over the lines activated by c^2PCAP. In other words, instead of using routers like in NoC, we utilize partial reconfiguration ability of FPGAs for routing data packets. Furthermore, DRP2P can work both on regular and irregular topologies. DRP2P solves the scalability issue of P2P by setting up on-demand communication-specific links between cores. So, the occupied area and the total power consumption of communication architecture can be reduced significantly.

In DRP2P, communication scenarios are downloaded to the FPGA one-by-one during the computation time of the cores. It is critical to understand that reconfiguration time plays a major role on assigning tasks to the cores and determining the number of communication scenarios: In Fig. 1b, the communication architecture is established during the computation. Hence, reconfiguration time is not noticed at all. However, in Fig. 1a, the computation time is too short to set up the communication channel. In this case, the reconfiguration time introduces an overhead to the communication latency, hence degrades the efficiency of DRP2P. This phenomenon dictates the design constraints on the self-reconfiguration engine:

- The reconfiguration engine should be as fast as possible so that the computing processors will not wait for the establishment of the communication channels

Table 1
Comparison of latest NoC approaches with DRP2P.

Network	Year	Switching	Device	Topology	Routing Alg.	Goal	Novelty	Application
PNoC [1]	2005	Circuit	FPGA	Custom	Deterministic	Reduce No. of Routers compared to regular NoCs	Consisting of a series of subnets. Place modules that communicate frequently in the same subnet	Image binarization
DyNoC [3]	2005	Packet	FPGA	2D-Mesh	Adapted XY (S-XY, SV-XY, SH-XY)	Direct comm. paths btw. neighbor PEs	Use routers as reusable elements, i.e. also for logic	Color generator and traffic Light Controller
ReNoC [4]	2008	Packet + Circuit	ASIC	Custom	Topology Switch + Router	Reduce No. of Routers	Wrapping routers with topology switches	VOPD
Skip-links [5]	2010	Packet	–	2D-Mesh	Adaptive	Jumping or skipping over the intermediate router	Connecting skip links together: skip chains	–
Reconfig-Net [6]	2010	-	FPGA	Custom	–	Reduce No. of Routers	Reconfigurable wires btw. routers	VOPD
VIP [2]	2010	Packet	ASIC	2D-Mesh	Wormhole switching	Bypass the router pipeline stages with VCs	Connect the source and destination of with dedicated P2P links	VOPD, MWD,H.263 enc., GSM, MP3 enc., MP3 dec.
RecoNoC [7]	2011	Packet + P2P Simplex Links	FPGA	2D-Mesh	Adaptive Wormhole	Bypass routers using shortcuts	Create shortcuts btw. distant nodes	–
AppAw [8]	2011	Packet + Circuit	ASIC	2D-Mesh	Wormhole switching	Bypass the routers all the way, use configuration switches to the utmost	Using simple switch boxes (configuration switches) between routers	VOPD, MWD, MP3 enc., MP3 dec., H.263 enc., H.263 dec.
DRP2P [This Work]	–	Circuit	FPGA	Any	No routing Algorithm required	No Routers. Instead of routers, utilize physical runtime reconfig. of FPGAs	No routers, totally reconfigurable wires	MP3 enc., MP3 dec., H.263 enc., H.263 dec., Multi Target Tracking (MTT), N-body

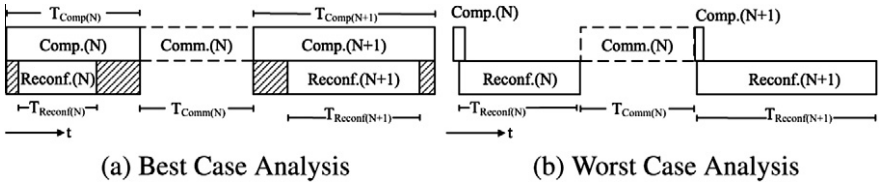


Fig. 1. Computation, reconfiguration and communication periods in partially DRP2P interconnects.

- It should have an on-chip cache to access the reconfiguration bitstreams quickly. Since on-chip memory is limited, the configuration bitstreams of communication scenarios should be small in size so as to increase the number of bitstreams on-chip.
- It must be dedicated and on-chip.
- It should be small to reduce area overhead.

Most of the applications on multi-core SoCs have non-uniform communication traffic patterns and they can be predicted statically [8]. In addition to this, most of the multi-core SoC applications do not have many different communication flows (number of edges in task graphs) and each of these cores mostly communicates with a few of other cores. Usually, the traffic flow of these applications is already known beforehand [2]. As a result of these, when we apply DRP2P to such systems, we can overlap the computation and reconfiguration latencies. Since communication traffic patterns can be predicted statically, it is possible to establish the next communication pattern while the current computation is being done.

DRP2P architecture can be used not only for the intra-switching purposes within an application; it can also be utilized for the inter-switching among different applications. Since the switching time between most use-cases in a SoC is at the order of few milliseconds [23], the time required to reconfigure the FPGA partially can be covered up safely.

This is the first paper that introduces DRP2P communication architecture. In Section 2, we will concentrate on the basic properties of this architecture, compare it with conventional NoC and $k \times k$ crossbar, discuss limitations of the architecture. Then, in Section 3, we will explain how DRP2Ps are obtained. This is the section where the self reconfiguration engine, c^2PCAP is also introduced with its applications on not only on low-cost FPGAs like Spartan 3, 6, but also high-end FPGAs like Virtex 4. We will show that it will be possible to setup DRP2Ps even on low-cost Xilinx FPGAs with our engine. In Section 4, we present results of basic tests and the real-life case studies. The final section concludes the work and sets directions for future research.

2. Proposed DRP2P architecture

This architecture is inspired from the fact stated at the first statement of this paper: P2P is the fastest communication way. In [24], a study has been carried out to specify application-specific point-to-point (ASP2P) interconnects between pairs of cores during design time. These interconnects remain static during run-time. Our method improves this approach by introducing a different set of P2P between pairs of cores to the SoC in a time-multiplexed manner. We call the set of P2P interconnects in one time-slot as the "communication scenario". Each communication scenario must fit at the related communication channel. During design time, the application is profiled and analyzed so as to determine the communication scenarios at "each time-slot". The duration of the time-slot can be either dictated by the designer or computed by the static analysis. The channels are decided by analyzing and synthesizing the scenarios. In other words, to replace the current communication scenario with another one, dynamic reconfiguration engine c^2PCAP firstly erases the communication channel (tear-down), and then reconfigures the wires of the new scenario (set-up).

Note that infinitely many communication scenarios cannot be realized with DRP2P. The memory reserved for compressed bitstream storage, compression ratio, size of reconfigurable area, device type and size determine the number of communication scenarios. Fig. 2 gives an idea about number of partial bitstreams P that can be stored in the BRAM on Virtex-4 FPGA family. These values are application specific and may vary from application to application.

In addition to the c^2PCAP core, we also have a monitoring system in our design. The main task of monitoring system is to trigger the c^2PCAP core whenever a valid communication request comes from a module. A communication request can be valid, when it is already predefined and approved by also other modules, otherwise the request is kept waiting until it is approved by the others. That is, when a module is finished with its computation job, it should wait its next communicating partner to finish its computational task. Moreover, a communication grant signal is assigned to each module; these signals are altered only by the monitoring system.

In DRP2P, we reconfigure wires. We do not go in detail which ones are control signals to control the flow. These are determined during design time, during this time the analysis of the application is done. Different clock speeds are also supported by DRP2P. If two cores need to communicate at different clock frequencies, either one of the cores can be replaced with a dual port RAM or buffers can be placed in the communication channel. Obviously, cores at different communication speeds would require buffers to prevent data loss. However, we did not experiment this scenario because it is easily do-able with our approach: it should be noted that we are dynamically reconfiguring a region, i.e. the communication channel, not solely wires. Therefore different communication architectures can be configured, regardless of topology. The only constraint is the area of the communication channel. We carried out such an implementation in our first case study MTT, where the communication scenarios are not solely p2p but there might also exist architectures like crossbar or broadcast. Hence, if desired, buffers can also be placed equally likely.

DRP2P is designed for nonreactive systems. Start and end times of communication scenarios are not simultaneous. The longest communication time determines the duration of each scenario. However, profiling and analysis of the application has to be carried out before deciding on the communication scenarios.

2.1. Theoretical latency analysis of DRP2P

Assume that there exist M modules communicating each other and there are N different communication scenarios. Half of these modules $(M/2)$ are located on the left side and the other half resides on the right side. The modules in each half are placed one above the other and vertically. Suppose that data flow occurs from left to right and each module on the left half has a directional n-bit connection with its opposite partner on the right half. Hence, the

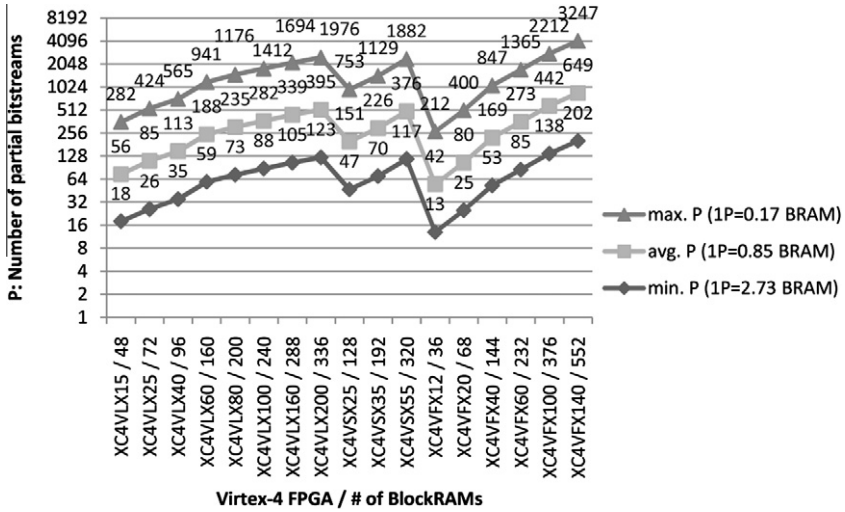


Fig. 2. Different P (# of partial bitstreams) values that can be stored in the BRAM on Virtex-4 FPGAs.

channel width W_C between modules (the total number of single wires between modules on both sides) is $M/2 \times n$. The duration of one communication scenario is given by the longest data transfer time between two communicating modules in that scenario. The data transfer time can be defined as follows: Assume that the data is transferred in k packets and let the size of each packet be s. For the maximum utilization of the wires between these modules, a different portion of data has to be sent at each cycle of the system clock, T_{clk}. Consequently, there should be no idle time until the data are completely transferred to the receiving module. Then, the data transfer time between two communicating modules is given as:

$$T_{comm} = ((k \times s) \div n) \times T_{clk} \tag{1}$$

Note that some of these scenarios can be repeated during the execution of the multiprocessor system. Therefore, there can be L ($\geqslant N$) scenarios in an application. Again, for illustrative purposes, we can assume that $L = N$ and each scenario runs only once. Assuming that the data transfer time in each communication scenario is equal, the total communication time is given as:

$$T_{comm\_tot} = N \times (((k \times s) \div n) \times T_{clk}) \tag{2}$$

If we assume that the initial communication scenario is downloaded with the complete bitstream, we can say that the same system requires $N - 1$ dynamic reconfigurations during the runtime of the multiprocessor system. Let RRR (Reconfiguration Repetition Rate) define the number of dynamic reconfigurations. There can be multiple locations for DRP2P interconnects and the required area has to be reserved during design time. Therefore the dynamic reconfiguration time, T_{reconf} for each reserved area can be easily determined. If we assume that there is only one reserved area for DRP2P, then we can calculate the total reconfiguration time given as:

$$T_{reconf\_tot} = RRR \times T_{reconf} \tag{3}$$

Now, referring to Fig. 1 and Eq. (1), we can write the best case and worst case conditions for DRP2P interconnects as follows:

$$T_{best} = T_{comm} = ((k \times s) \div n) \times T_{clk} \tag{4}$$

$$T_{worst} = T_{comm} + T_{reconf} = ((k \times s) \div n) \times T_{clk} + T_{reconf} \tag{5}$$

Similarly, the best and worst case conditions for the total data transfer time can be calculated by using Eqs. (2) and (3):

$$T_{best\_tot} = T_{comm\_tot} = N \times (((k \times s) \div n) \times T_{clk}) \tag{6}$$

$$T_{worst\_tot} = T_{comm\_tot} + T_{reconf\_tot} = N \times (((k \times s) \div n) \times T_{clk}) + RRR \times T_{reconf} \tag{7}$$

The data transfer size S_D in each scenario and the total size of transferred data $S_{D_{tot}}$ in the entire system are given as:

$$S_D = (M \div 2) \times k \times s \tag{8}$$

$$S_{D\_tot} = N \times S_D = N \times ((M \div 2) \times k \times s) \tag{9}$$

Based on the derived equations on a sample instance, we can numerically demonstrate the latency due to DRP2P interconnects. An implementation of a 32-bit width ($n = 32$), 8 modules ($M = 8$) communication structure ($W_C = 4 * 32$) is illustrated in Fig. 3. As we have four different communication scenarios ($N = 4$) in Fig. 3, there exists three passes between scenarios. Hence, RRR is 3 for this example. Assume that packet size s is 4-bytes (=32-bits).

The target FPGA is Virtex-4SX35 and the system clock is set to 100 MHz ($T_{clk} = 10ns$). For the reconfiguration, the configuration interface ICAP in 8-bit mode (a byte is sent to ICAP in each configuration clock cycle) is selected and the configuration clock is set to 100 MHz ($T_{cclk} = 10ns$); the reconfiguration speed is therefore 100 MB/s. Each partial bitstream size is 6524 Bytes. Hence, the reconfiguration time for each pass between different scenarios $T_{reconf} = 6524 * 10\ \text{ns} = 65.24\ \mu\text{s}$. For example, according to the Eq. (7), for k = 5, the total data transfer time for $S_{D\_tot} = 320$ Bytes in best and worst case (see Fig. 4) are calculated as follows:

$$T_{best\_tot} = 4 \times (((5 \times 32) \div 32) \times 10\ \text{ns}) = 0.0002\ \text{ms}$$

$$T_{worst\_tot} = 4 \times (((5 \times 32) \div 32) \times 10\ \text{ns}) + 3 \times 0.06524\ \text{ms} = 0.196\ \text{ms}$$

2.2. Comparison of DRP2P with 2-D Mesh NoC

For the comparison of DPRP2P with NoC architecture, we have preferred to use the Network on Chip emulator (NoCem) with mesh topology because of its availability. NoCem is available for free download by OpenCores [25]. The NoCem is an open source (protected under GNU General Public License), on Chip Network Emulation Tool and a body of VHDL code configurable by a top-level package file that can create a variety of Network on Chips on parameters of data-width, virtual channel implementations, topology, and in-network buffering lengths. Once parameterized, the resulting NoC is generated automatically with the use of VHDL

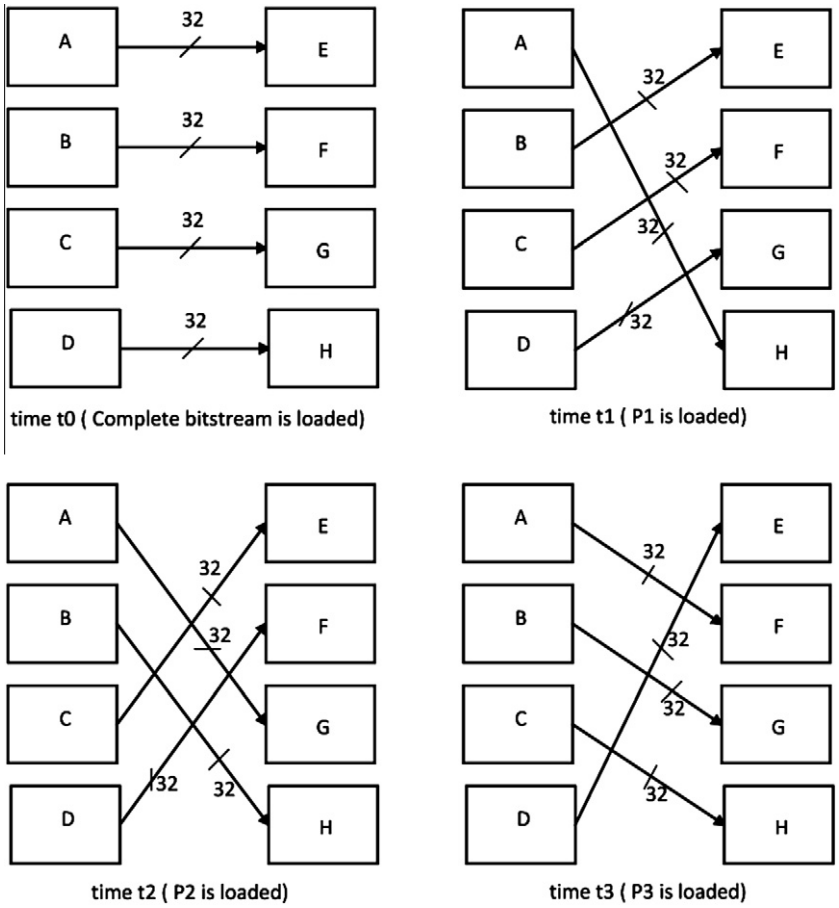


Fig. 3. Four different communication scenarios.

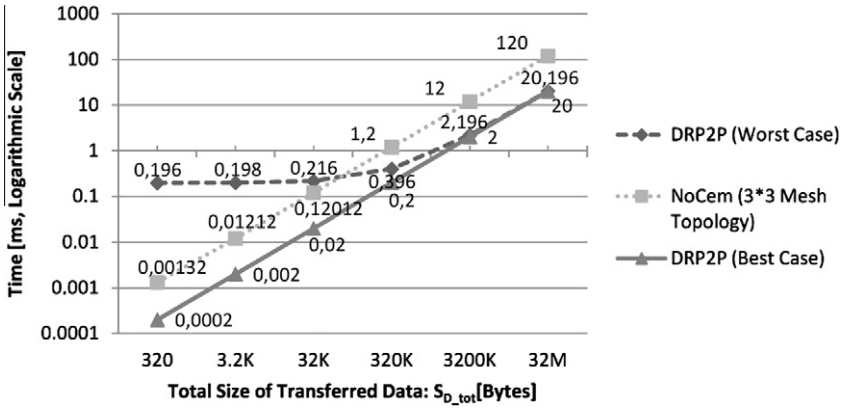


Fig. 4. Comparison of DRP2P and NoCem with $3 * 3$ mesh topology for different S_D values.

generics and generate statements. It supports three different NoC topologies: mesh, torus and double torus. For a $4 * 4$ mesh topology with 16-bit data width, it occupies 78% of Virtex-II FPGA (xc2vp30) [26].

To utilize DRP2P communication architecture efficiently, the designer should be aware of the fact that the computation time must be longer than reconfiguration latency ($T_{comp} \geqslant T_{reconf}$). For this example the reconfiguration latency is approximately about 0,196 ms (at 100 MHz, 8-bit configuration interface). Here it is obvious that the reconfiguration latency may be decreased to half (98 μs) or one fourth (49 μs) of the original one by increasing the reconfiguration interface width to 16-bit or 32-bit.

By changing values of S_{D_tot} for this example, we can obtain a comparison of DRP2P with NoCem architecture which is shown in Fig. 4. We have configured to NoCem as $3 * 3$ mesh topology with simple packet type; with no virtual channels. We set some of NoCem configuration parameters as given in Table 2. Here, in the best case, the partial reconfiguration always and totally overlaps with the computation. If the duration of a computation is greater than the partial reconfiguration time of the next communication pattern, then DRP2P interconnects is the best, regardless of the type and size of the NoC. So, each pattern will be configured during computation, that is, prior to the related communication and hence there will be no reconfiguration overhead in terms of

Table 2
NoCem configuration parameters related to VC and FIFO.

NoCem Parameters	Comment
constant NOCEM_TYPE: integer := NOCEM_SIMPLE_PKT_TYPE;	- - No VCs
constant NOCEM_CHFIFO_NOVC_TYPE: integer := 2;	- - generate FIFOs for No VC type
constant NOCEM_CHFIFO_TYPE: integer := NOCEM_CHFIFO_NOVC_TYPE;	- - Set to 2
constant NOCEM_FIFO_LUT_TYPE: integer := 0;	- - use LUTs for FIFOs, i.e. not BRAMs
constant NOCEM_TOPOLOGY_TYPE: integer := NOCEM_TOPOLOGY_MESH;	- - topology: mesh
constant NOCEM_FIFO_IMPLEMENTATION: integer := NOCEM_FIFO_LUT_TYPE;	- - use LUTs for FIFOs
constant NOCEM_CHFIFO_DEPTH: integer := 4;	- - set FIFO depth to 4

time. However, in the worst case, the computation takes very little time; the time interval of computation never overlaps with the partial reconfiguration.

In Fig. 4, we have shown the timing results of DRP2P and NoCem for different SD_{tot} values. To find average data transfer per edge, we should consider each edge in different communication scenarios. Since each node communicates with a different node in each scenario, the average communication volume per edge $CV_{edge,avg}$ can be formulated as:

$$CV_{edge,avg} = S_{D_{tot}} \div N \div (M \div 2) \div t \quad (10)$$

For example, in Fig. 3, node A communicates with node E only in the first scenario. So, the average communication volume per edge $CV_{AE,avg}$ for SD_{tot} values of 3200 KB, 32 M and corresponding communication times t = 1.2 ms and t = 12 ms is calculated as:

$$CV_{AE,avg} = 3200\ \text{KB} \div 4 \div (8 \div 2) \div 12\ \text{ms} \approx 266667\ \text{KB/s}$$
$$CV_{AE,avg} = 32\ \text{MB} \div 4 \div (8 \div 2) \div 120\ \text{ms} \approx 266667\ \text{KB/s}$$

The applications (MP3-Encoder, H263-Decoder, etc.) in MMS suite (see Fig. 7) have communication flows such as 116873, 75025, 38016, etc. These communication volumes have a unit of 10 KB, which represent the average communication volume between different tasks in 1 s. So, the communication flows 116873, 75025, 38016 represent the communication volumes of 1168730 KB/s, 750250 KB/s, and 380160 KB/s, respectively. As it is obvious from these results, the SD_{tot} values above 320 K seem to be in the range of Multi-Media System application average communication flows. Since, we already target such computation and communication intensive applications, the sample data sizes that we show seem to be very realistic.

We did not carry out a specific study on throughput because we transfer data at each clock period in DRP2P. Since there is no buffer or logic between two communicating nodes in DRP2P, the throughput in both cases of is identical to the throughput of a wire.

As it is obvious from the chart, in the worst case analysis for the small S_{D_tot} values (up to 320 K), the NoC outperforms DRP2P. However, as the S_{D_tot} value increases (from 320 K), DRP2P gives better results than NoC architecture. Hence, it can be easily claimed that DRP2P architecture is more suitable than NoC approach for large data transfer where the possible number of different communication architecture patterns (N) between nodes are limited in terms of data storage. However, in the best case, DRP2P always outperforms NoC.

Table 3 presents the power consumption results of ML402 board and occupied area of the design in Fig. 3 on Virtex-4SX35 and Spartan-6 XC6SLX45, respectively. We have implemented this design with DRP2P and $3 * 3$ mesh topology with simple packet type of NoCem. Here, the modules M1-M4 are $8 * 8$, while M5-M8 are $16 * 16$ unsigned multipliers for both DRP2P and NoCem designs. The occupied area also includes these LUT-based multiplier blocks. The difference between design types "DRP2P without reconfiguration" and "DRP2P with reconfiguration" is the case, where we have cut the clock input of the ICAP module for the design without reconfiguration. The rest of the designs are identical.

Table 3
Power consumption of ML402 board and occupied area on Virtex-4SX35 and Spartan-6 XC6SLX45, 8-Modules (W_C:128).

	ML402 Board power [Watt]	Virtex-4SX35 occupied area		Spartan-6 XC6SLX45 occupied area	
		Slices	BRAM	Slices	BRAM
Available on FPGA		10086	192	6822	116
Empty design	3195	–	–	–	–
DRP2P without Reconfiguration	3390	2011 (13%)	8 (4.1%)	603 (8%)	8 (6.8%)
DRP2P with Reconfiguration	3551	2011 (13%)	8 (4.1%)	603 (8%)	8 (6.8%)
3*3 Mesh of NoCem	3726	10086 (65%)	–	3868 (56%)	–

The aim is to observe the pure power consumption due to partial reconfiguration.

As it is obvious from the timing, power and area results of DRP2P and NoCem designs, DRP2P approach gives better results than NoC architecture when the packets being sent are getting larger. In addition to timing performance of DRP2P, it is also more power and area efficient than the design with 2D-Mesh implementation of NoCem on both Virtex-4 and Spartan-6 FPGAs. When the difference of DRP2P and NoCem with empty design board power is considered, NoCem consumes 531mw, for the same design DRP2P consumes only 356mW; power gain in DRP2P is about 33% compared to NoCem. Therefore, it makes more sense to use DRP2P for such communication infrastructures like in Fig. 3.

2.3. Comparison of DRP2P with a 2-D Reconfigurable Mesh NoC [8]

We have implemented the communication infrastructure of the audio video benchmark [27] by utilizing runtime partial reconfiguration property of Spartan-6 FPGA. The audio video benchmark includes H.263 decoder, H.263 encoder, MP3 encoder, and MP3 decoder applications. In [27], the audio video benchmark is implemented on 16 different cores. Although the communication flow among the cores is different for each application, each application uses the similar set of IP-cores. Taking advantage of this property, the number of used cores for audio video benchmark are reduced to 12 cores in [8]. As in [8], we have also used the same task mapping procedure for audio video benchmark, i.e. Multi-Media System (MMS). The task-graphs of different MMS applications are combined in Fig. 5 [8]. In this graph, nodes represent the cores and edges are the communication flows. Edge weights are the average communication volumes among different cores that have unit of 10 KB/s [27]. In Fig. 6, the task graph of each application in MMS suite is given separately.

In [8], communication infrastructure of MMS is implemented on the reconfigurable NoC, which uses not only routers but also

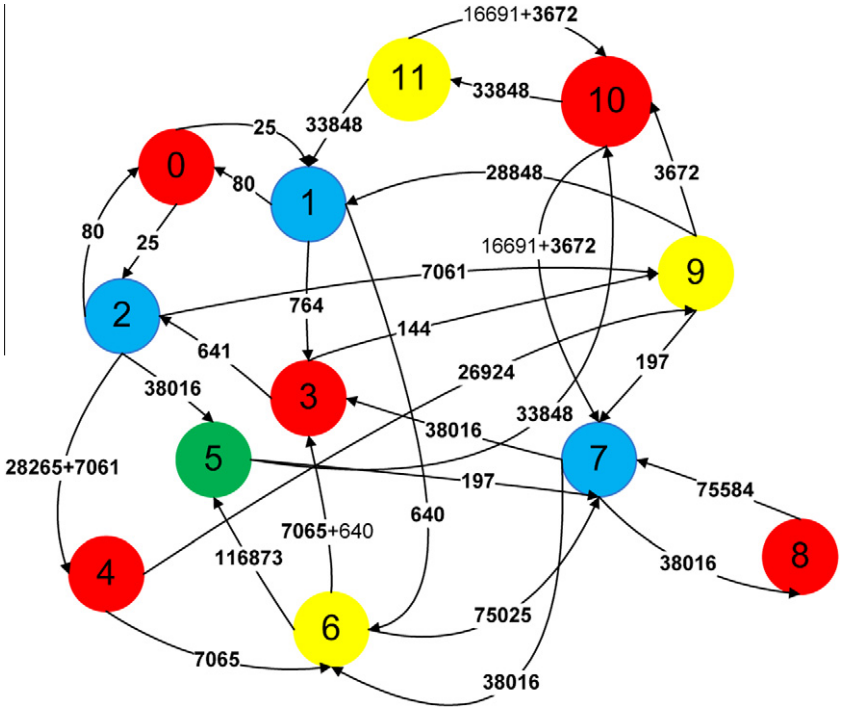


Fig. 5. Multi-Media System (MMS) complete task graph on 12 cores.

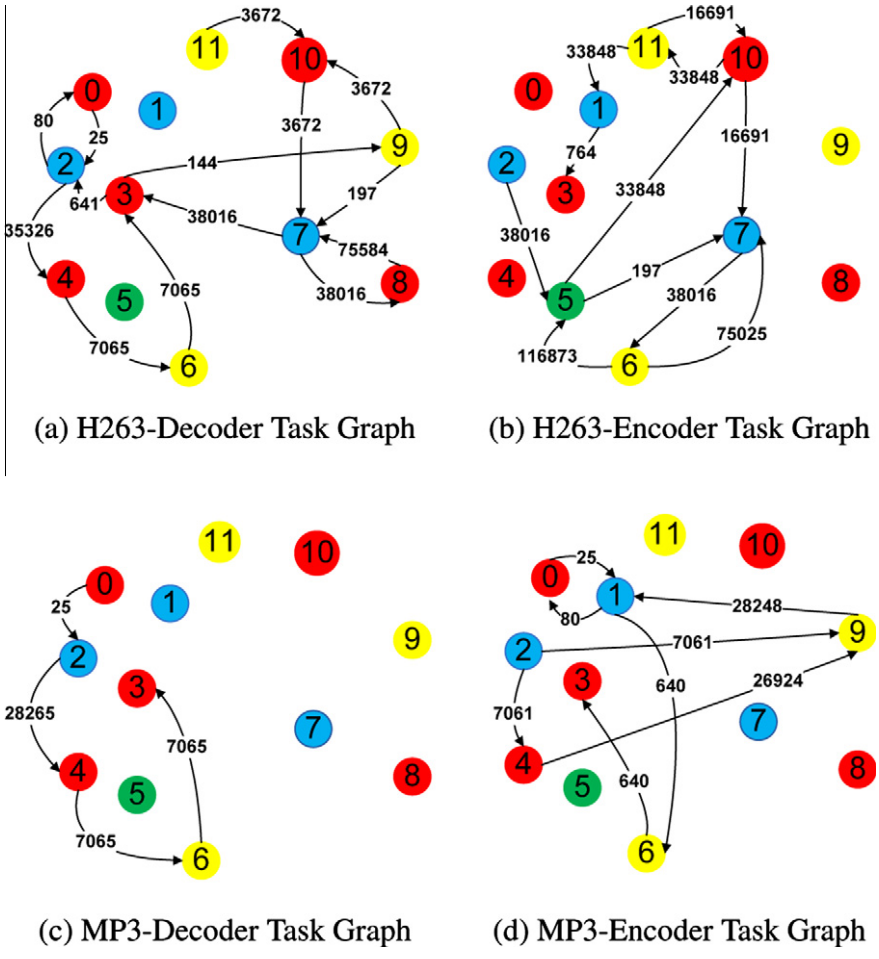


Fig. 6. Task graphs of applications in MMS suite.

Table 4
Area overhead of DRP2P for MMS communication architecture.

Resource Type	Spartan-6SLX45 device utilization summary [used/available]					
	c^2PCAP	PB Repository	RAs with BMs	Muxes and Demuxes	Total	Percentage
Slices	82/6822	–	768/6822	104/6822	954/6822	13.98
BRAM	–	52/116	–	–	52/116	44.83

configuration switches for packet routing. Since each application in MMS is not to work at the same time, they can run at different time slots by sharing the same hardware resources, i.e. task cores. Therefore by removing the previous application, the next application can be loaded to the hardware on demand. According to that work, the switching operation between applications in MMS can be done either by a configuration manager in reconfiguration process or by storing configuration data in configuration switches and routers. In [8], with the term *reconfiguration*, the authors refer to the updating of the parameters of configuration switches. However, in our implementation, the switching operation between these applications is done through runtime partial reconfiguration of Spartan-6 FPGA, which is an actual FPGA fabric reconfiguration. Due to the fact that the switching time between most use-cases in a SoC is at the order of few milli-seconds [23], the time required to partially reconfigure the FPGA can be covered up safely.

Our objective is to minimize the switching time between applications. This is achieved by minimizing the reconfigurable area on a partially reconfigurable FPGA. To minimize the reconfigurable area, as an initial step, we applied WelshPowell heuristic graph coloring algorithm [28] to the MMS complete graph (see Fig. 5) for the physical location of cores. This algorithm gives the same colors to the non-adjacent nodes in a given graph. The chromatic number of the MMS complete graph is 4 meaning that there should be 4 different sets of cores communicating each other (Set-1 (blue): Nodes 1, 2, 7; Set-2 (red): Nodes 0, 3, 4, 8, 10; Set-3 (yellow): 6, 9, 11; Set-4 (green): Node 5). Since the atomic unit of reconfigurable area consists of frames that covers 16 CLBs in the height for Spartan-6 FPGAs [29], the cores should be placed in a way to use as few frames as possible. In addition to this, it is not essential to locate the communicating cores in different sets, because the communicating nodes within the same set can also be connected through the bus macros that are located in the same column. This can be achieved in average by keeping the number of nodes same in each set. In this way, area reserved for reconfigurable will be minimized. The nodes that communicate most with the already included nodes in target set are selected: the node 8 communicates only with the node 7, therefore we shifted this node from Set-2 to Set-1. Thus, we achieved 2 sets with the same size, i.e. each has 4 nodes. The node 5 communicates most with node 6. Hence we moved this node to Set-3. As a result, the new sets are given in the following:

- Set 1 has the nodes 1, 2, 7 and 8.
- Set 2 has the nodes 0, 3, 4 and 10.
- Set 3 has the nodes 5, 6, 9 and 11.

Since Set 2 has the heaviest communication flow in average, we have located this in the middle of Set-1 and Set-3. Apart from this, we have tried to locate most communicating node pairs in a mutual manner in each set. Additionally, for the communicating node pairs which are located Set-1 and Set-3, we have created additional channels in the area of Set-2. Our architecture is shown in detail in Fig. 7. Here note that, all steps for locating nodes on the FPGA is done manually by hand design. An efficient algorithm for the node to core mapping can be developed for this problem. In our approach, there are dedicated P2P 32-bit links for each communication flow in MMS task graph.

At a time, there is only one of these configurations is available on the Spartan-6 FPGA. The application switching is done by reconfiguring the two reconfigurable areas at the same time. Since configuration switching occurred infrequently (on-demand application loading), runtime reconfiguration latency and power dissipation for configuration switching can be ignored [8].

To meet the timing requirements of the application, we have set computation clock to 300 MHz (according to heaviest data flow in the task graph) and configuration clock to 100 MHz for the 16-bit ICAP. Here, it is worth talking into consideration that long links in the system can decrease the clock frequency. To avoid this, long links can be partitioned by putting additional FFs on them, i.e. pipelining fashion. The long links in our system mostly have bus macros at their heads and tails. Furthermore, bus macros are composed of FFs naturally. Hence, we do not need any additional FFs to pipeline the system.

The height of each reconfigurable area is determined by the number of used hard bus macros (BM). Each BM is 32-bits and occupies 4-CLB height. Since there are 15 CLBs on a side at most (the most left BM column in Fig. 7), each reconfigurable area occupies 60-CLB height, i.e. about four ($\lceil 60/16 \rceil$) Clock Regions (CR) are included to the each of reconfigurable areas (RA). In addition to this, the width of reconfigurable area is about 3-CLB by counting left and right BMs. So the total reconfigurable area (RA_{tot}) can be found in the following formula:

$$RA_{tot} = \#RAs \times \#CRs \times (16CLBs) \times (3CLBs) \quad (11)$$

In our case, we have two reconfigurable areas and there are four CRs included to the partial bitstream for each of RAs. According to the Eq. (11), the total RAs occupy 384 CLBs, i.e. about 9% of the Spartan-6 FPGA (SLX45). Total partial bitstream size of two RAs is about 100 KB, which results in about 0.5 ms of the reconfiguration time. In Table 4, the occupied area of the MMS suite with DRP2P on Spartan-6SLX45 FPGA is given. The communication architecture of MMS suite with DRP2P occupies about 14% of the Spartan-6SLX45 FPGA. In addition to this, for the storage of PBs, almost 45% of the on-chip available BRAMs are used. If the cache method is preferred, the usage of BRAMs can be reduced to one quarter (i.e. 12%) of the non-cache method (i.e. 45%). In the cache method, only the current PB is stored on BRAMs, the rest are stored on the external memory.

Here, it is worth mentioning that each application uses a subset of cores in the MMS suite. Therefore, unused cores (nodes that have no connections to any other nodes in each task graph) for each application can be switched off to reduce the power consumption. This can be done by pruning the clock input of unused core at each scenario change, i.e. application switching. The pruning process of clock input, which is named *clock gating* method, for unused core can be done by utilizing partial runtime reconfiguration. We have already used the similar technique *clock scaling* for the purpose of power reduction and reconfiguration testing in different target devices [29–31].

The work in [8] is implemented on ASIC with 65-nm technology. However, our work is implemented on Spartan-XC6SLX45 FPGA with 45-nm technology. Hence, we did not compare area

Table 5
Comparison of message (8-flit packet) latencies in clock cycles for conventional, Reconfigurable Mesh NoC [8] and DRP2P.

Benchmark	Application	8-flit packet latency		
		Conventional $3*4$ NoC	Reconfigurable $3*4$ NoC [8]	DRP2P
MMS	H.263 Decoder	23.308	19.332	17.780
	H.263 Encoder	20.555	18.341	14.400
	MP3 Decoder	25.009	20.701	8.000
	MP3 Encoder	28.879	20.807	14.670

and power results for DRP2P and [8]. In Table 5, message latencies of three different implementations of MMS communication architecture are given. From these results, it is obvious that DRP2P gives the best performance in terms of speed.

2.4. Comparison of DRP2P with other communication architectures

In Fig. 8, an n-bit width $256*256$ cross point communication architecture and its possible implementations are illustrated. The first and the most common implementation of this architecture is multiplexer based approach, shown in Fig. 8a. It cannot be denied that this approach is the most speed efficient. Switching between different communication patterns occur in a few clock cycles. However, the occupied area by this approach is infeasible. Even for 1-bit width of $256*256$ cross point architecture, the required number of slices is 19476/20480 of a Spartan-XC3S2000 (95% of the whole chip). Hence, the power consumption of such a huge circuit will be very high. Therefore this approach will not be practicable for small FPGAs (<2 M gates).

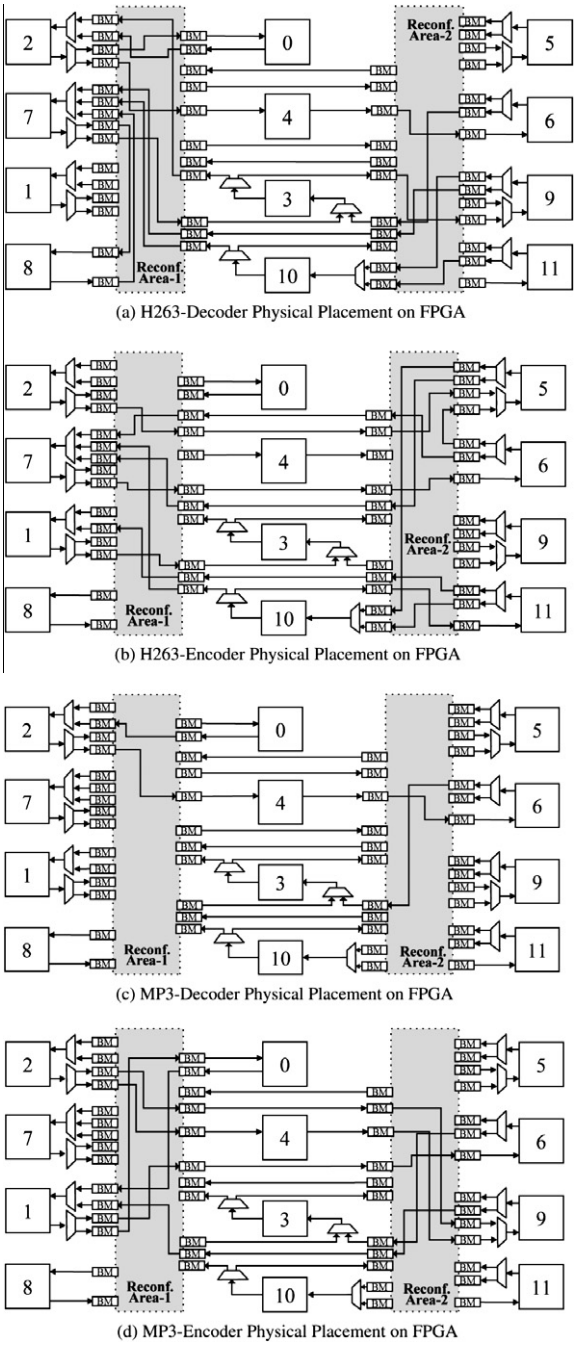
The second possible way to implement this architecture may be using a ring Network-on-Chip (Fig. 8b). However, it is obvious that communication from one node to other mostly requires multihops (e.g. M1 to M512 requires 256 hops) which actually results in increased latency due to packetization, routing and switching through multiple routers. Apart from having increased latency, this approach seems to be infeasible in terms of area and power.

Alternatively, a mesh NoC, Fig. 8c, can be a competitor to other approaches. Like the multiplexer based approach, the mesh topology is also more time efficient then ring topology NoC. The worst case time cost of communication from any node to any other node (e.g. from M1 to M512) is traveling about 46 routers (vertically 15 hops, horizontally 31 hops). The number of hops can be drastically reduced by adding some new links to the mesh topology. However, because of very complex routing between routers the occupied area by this method is unpractical to be implemented on a small FPGA. Even a $4*4$ 16-bit width 2D-mesh grid topology occupies about 78% of Virtex-II FPGA (xc2vp30) [26].

All above mentioned approaches suffer from either speed or area (likewise power). A balanced solution in terms of speed and area may be DRP2P communication architecture, represented in Fig. 8d. Such a configuration architecture can fit into 192/5120 CLBs of a Spartan-XC3S2000 (3.75% of the whole chip). To pass 256 signals safely, we have used 32 bus macros, which are located one below the other. In addition to the occupied area of slice based bus macros, 2-CLB-width area is reserved for signal flow from F2R bus macros to R2F bus macros. So, totally 6-CLB-width area is occupied for glitch-free signal flow between modules. As a result, to pass 256 signals $32 \times 6 = 192$ CLBs are occupied. The information of each communication scenario must be stored as a partial bitstream for this approach.

2.5. Limitations of DRP2P architecture

Since the reconfiguration time plays major role in DRP2P architecture, we would like to give limitations of reconfiguration on



(a) H263-Decoder Physical Placement on FPGA

(b) H263-Encoder Physical Placement on FPGA

(c) MP3-Decoder Physical Placement on FPGA

(d) MP3-Encoder Physical Placement on FPGA

Fig. 7. A simplified view of MMS suite physical placement on Spartan-6 FPGA.

Xilinx FPGAs at first. In the following paragraphs, possible shortest reconfiguration times for Spartan-6 and Virtex-6 FPGAs from XILINX are given.

Spartan-6 and Virtex-6 FPGAs offer the partial reconfiguration in two-dimensional manner. The atomic unit that can be reconfigured is a single frame within a clock region (in terms of CLB configuration) [29]. The frame size in Virtex-6/ Spartan-6 family is 162 Bytes [11]. Spartan-6 supports 16-bit ICAP for reconfiguration at most 100 MHz, which results in 200 MB/s reconfiguration speed. However, it is possible to use 32-bit ICAP mode for Virtex-6 at most 100 MHz, thus the reconfiguration speed of 400 MB/s can be achieved. So, the reconfiguration times of possible smallest reconfigurable area for Spartan-6 and Virtex-6 FPGAs are as in the following:

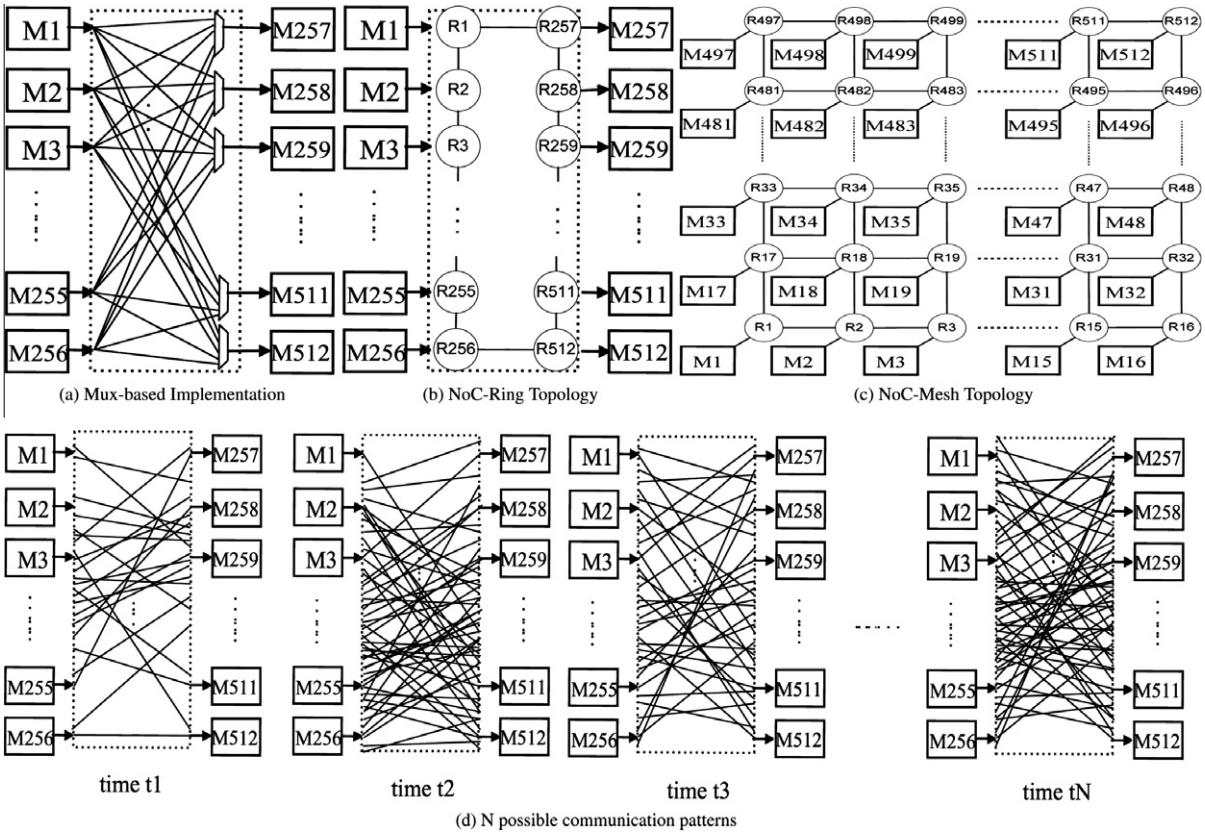


Fig. 8. n-bit width k∗k ($k = 256$) point communication architecture and possible implementations.

- Spartan-6: $T_{min\_reconf} = 162B \div (200MB/s) = 0.81\ \mu s$.
- Virtex-6: $T_{min\_reconf} = 162B \div (400MB/s) = 0.405\ \mu s$.

As a result, there is no any design that can be reconfigured dynamically shorter than 0.81 μs on Spartan-6 and 0.405 μs on Virtex-6 FPGAs. In the following, limitations of DRP2P architecture are given:

- If the switching operation between different communication scenarios occurs very frequently (e.g. order of a few microseconds or faster), the time required to reconfigure the FPGA partially cannot be covered. Therefore, reconfiguration time overhead must be included in the execution time of the application, i.e. the worst case occurs.
- The number of different scenarios is also limited for DRP2P architecture. For each possible scenario, there must be a partial bit-stream. The total size of these partial bit-streams should not exceed the available internal or external memory. The size and number of partial bit-streams are dependent on the reconfigurable area and possible different scenarios.
- If the application execution time cannot be predictable (e.g. in dynamic systems) according to the current input, the start point of runtime reconfiguration cannot be forecasted. This comes up with the worst case execution time of reconfiguration process.

3. Design flow for DRP2P

The design flow for DRP2P is given in Fig. 9. In this Figure, numbers over the boxes represent the appropriate section for each process.

3.1. The application is being profiled...

Initially, a design on FPGA is being profiled and its communication infrastructure between computing elements is being extracted manually at design time. According to timing and area requirements of the design, the dynamic communication structure of nodes is disclosed with simulation tools. This process gives us the task mappings, communication scenarios and communication channels.

3.2. Partial bitstream generation for all communication scenarios

After determining communication scenarios, a partial bitstream is generated for each of them. Partial bitstream generation steps are given in the following paragraphs:

3.2.1. Dynamic partial self-reconfiguration flow

The c^2PCAP core behaves as if it is a mirror of SelectMAP port, as same in ICAP. The configuration control flow in this work is very similar to "SelectMAP configuration Flow Diagram" in [11] except that PROG, INIT, DONE pins are not taken into account in our study. This control flow has been already illustrated in detail in Fig. 5 of our cPCAP core study [30].

Before this, communication infrastructure of the design is located an place, where it communicates through only hard bus macros. The details of generation of hard bus macros are given:

3.2.2. Slice based bus macros

For a lossless data communication between reconfigurable and fixed area in a reconfigurable design, the designer must use a type of bus macro, which guarantees a safe data flow in both directions at the time of reconfiguration process. Such a lossless communica-

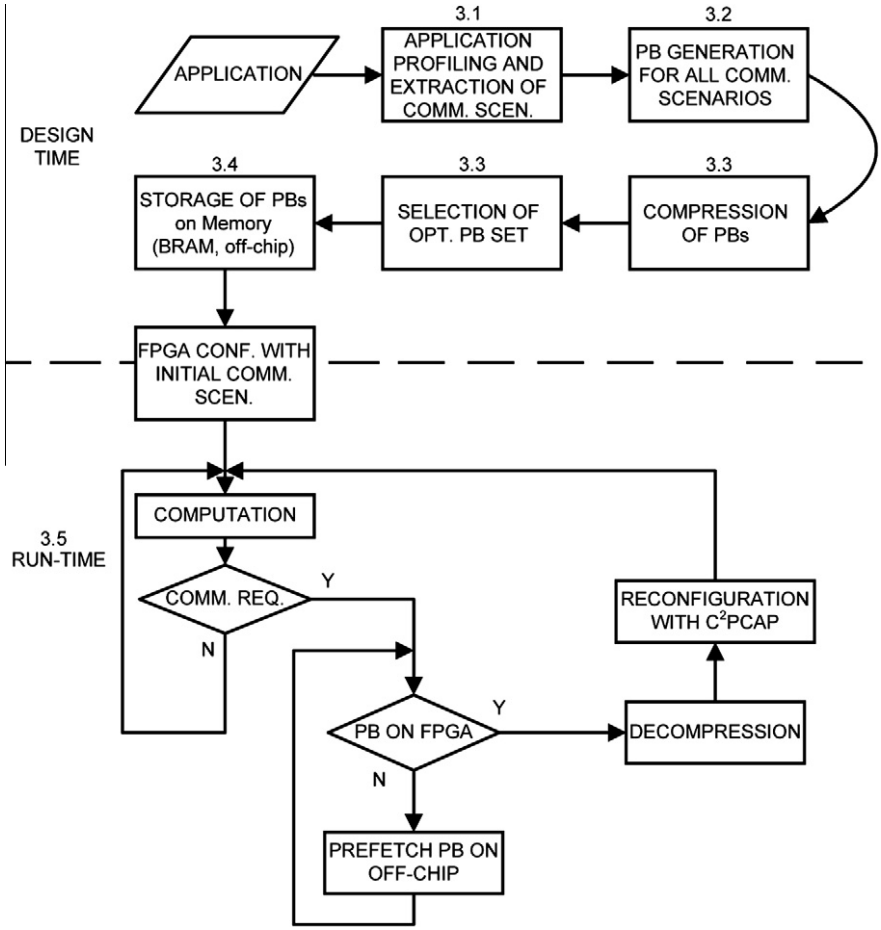


Fig. 9. Design flow for DRP2P.

tion can be implemented by tri-state buffers offered by Xilinx Inc. However, some FPGA series from Xilinx such as Virtex-4 and pure Spartan-3 families have no tri-state buffers. Hence, for these device families a new type slice-based bus-macro, which acts as a tri-state buffer, should be developed. There has been some papers published related to this subject such as [32] and [33] using LUT based multiplexer and and-gate approaches, respectively. The similar slice based bus macros are used in this work as in Fig. 10. Contrary to and-gate implementation, transparent latches keep the last data before reconfiguration starts, thus there is no data loss during reconfiguration. To guarantee a glitch-free signal flow from one side to other side two kinds of bus macros (from fixed to reconfigurable area: F2R and from reconfigurable to fixed area: R2F) are used. Since, each bus macro (narrow mode) has a width of 2-CLB, totaly 4-CLB-width area is occupied for F2R and R2F slice based bus macros to pass 8-bit signal from one side to other side safely.

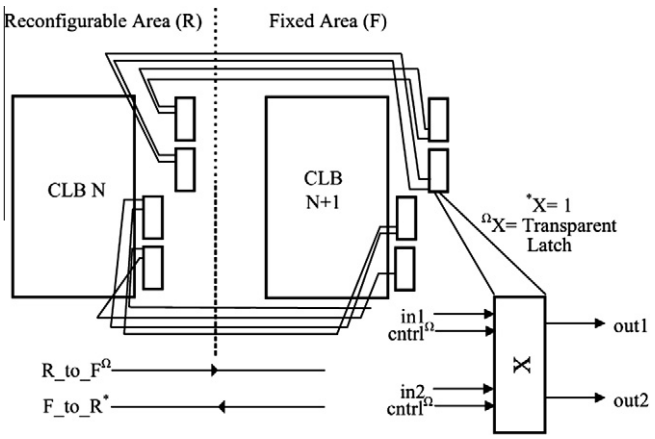


Fig. 10. General structure of slice based bus macros.

Actually, there are no available slice based hard bus macros offered by Xilinx for Spartan-6 FPGAs. However, there are some bus macros which are directly used for some target FPGA families (e.g. Virtex-5, Virtex-6) from Xilinx. Instead of designing a new bus macro we manipulated and adapted Virtex-5 bus macros to the Spartan-6 bus macros in our recently published study [29]. The 4-bitwidth synchronous single slice hard bus macro component was illustrated in Fig. 3 and 4 in [29].

3.3. Partial bitstream manipulation

"Compression of Partial Bitstreams" and "Selection of Optimum Partial Bitstream Set" is explained in this subsection. Our proposed c^2PCAP is designed for decompressing the compressed bitstreams stored in the on-chip BlockRAM of the Xilinx FPGAs.

3.3.1. Previous studies on configuration compression techniques

In the literature, there are studies which are directed for reducing the configuration size by proposing new mapping, placement, routing algorithms [34–39]. This step is done at the time of bitstream generation or before it. However our method is bit-stream compression which is independent from mapping, placement,

routing. Our technique is applied only to partial bitstreams generated by vendor tools such as bitgen from XILINX ISE.

There are various studies in the field of configuration compression and decompression techniques [40–45]. Common approaches are Run-length encoding, Huffman coding, Arithmetic coding, LZ based coding and their versions that are improved, extended or manipulated. Our study and some of example studies are summarized in Table 6.

With our proposed algorithm, we achieved up to 99% space savings as we have concentrated on compression of interconnects solely. The following subsections explain how the partial bitstreams are compressed by software and how they are decompressed by hardware, i.e. c^2PCAP core.

3.3.2. Compression

- Similarity extraction

Let p_i and p_j denote two partial bitstreams that successively reconfigures a region on the FPGA for DRP2P interconnects. Since this region has to be determined during design time, the length of both partial bitstreams are the same. Then we can extract similarities between two partial bitstreams by a simple XOR operation.

$$p_{ij} = p_{ji} = p_i \oplus p_j \tag{12}$$

Let p_{ij} be named as the joint bitstream of p_i and p_j. Obviously, as the similarities between p_i and p_j increases, the number of zero entries in p_{ij} increases. If there are N different communication scenarios in an application, then there are $C(N,2)$ joint bitstreams. Note that XOR operation also helps in generating partial bitstream from another one by using the joint bitstream:

$$p_j = p_i \oplus p_{ij} \tag{13}$$

- Compression

We introduce a zero run length coding technique to compress both types of bitstreams. In this technique, each byte in the bitstream is checked whether it is 0 or not. If it is not zero, it is directly written to the BRAM file. Otherwise, firstly the number of zero bytes is determined. Then the byte count is written to the BRAM file and the parity bit of the related memory location is set. Let l_i and l_{ij} represent the lengths of compressed partial and joint bitstreams, namely p_i^c and p_{ij}^c, respectively. Note that the length of compressed bitstreams is usually smaller than the related partial and joint bitstreams.

- Selection of optimal compressed bitstream set

There are $C(N,2)+N$ compressed bitstreams for N communication scenarios. We define I^s as the optimal set with N compressed bitstreams and this set can be used in the generation of all partial bitstreams by using Eq. (13) after decompression. Therefore it is obvious that I^s must contain at least one compressed partial bitstream. The other members of I^s can be either compressed partial or compressed joint bitstreams. Hence, our aim is to obtain I^s such that the sum of the lengths of the selected compressed bitstreams is minimum so as to use as small BRAM area as possible. In this case, our problem turns out to be a subset sum problem which is known to be NP-complete. The fastest known exact algorithms utilizing dynamic programming whose worst case execution time would be around $O\left(K^2 2^K\right)$, where $K = C(N,2)+N$ if we had used it for the solution of our problem. However we propose a faster exact algorithm that has the time complexity of $O\left(N^2 2^N\right)$, because the Algorithm 1 is designed to omit infeasible solutions in the design exploration space. It operates on the indices of compressed bitstreams and initially assumes that the set of compressed partial bitstreams is the best solution (lines 1 and 2). Then it exhaustively searches all possible solutions. In each iteration, there are $m \geqslant 1$ compressed partial bitstreams (lines 3–6). The remaining members are compressed joint bitstreams which are generated from G_x (line 6) and R (line 7). G_x is the subset of I^s and it contains partial bitstreams p_i that have to be stored in BRAM. R contains the indices that are in I^s but not in G_x. Compressed joint bitstreams p_{ij} will be generated from the compressed bitstreams whose first index is from G_x and second index is from R. Initially the cost of p_i's ($i \in G_x$) is calculated (line 8). Then the costs of joint bitstreams which use $i \in G_x$ as the first index are evaluated one by one.The index pair (i, j) for compressed joint bitstreams is selected in a way to minimize the storage cost in BRAM (lines 8–14).

Algorithm 1. Picking the smallest partial bitstream set

```
   /* Pick the smallest partial bitstream set                    */
1  I^s = I = {1, 2, ..., N − 1, N}
2  cost=∑_{i∈N} l_i;
3  m=1;
4  while (m < N) do
5      K_m={the set of all subsets with m members from I}
6      foreach G_x ∈ K_m, 1 ≤ x ≤ C(N, m) do
7          R = I − G_x;
8          sum = ∑_{i∈G_x} l_i;
9          foreach j ∈ R do
10             cost_j = min_{i∈G_x} l_ij
11             i* = {the i ∈ G_x with cost_j};
12             if sum + cost_j ≥ cost then
13                 break;
14             end
15             sum+ = cost_j;
16             S = S ∪ {i* j};
17         end
18         if sum < cost then
19             cost = sum;
20             I^s = G_x ∪ S;
21         end
22     end
23     m++;
24 end
```

A sample run of our algorithm is available in Fig. 11. The sizes of compressed partial and joint bitstreams are shown in tables. The rule for filling the tables is defined with the following equations:

$$\begin{aligned} Table(p_i, p_i) &= l_i \\ Table(p_i, p_j) &= Table(p_j, p_i) = l_{ij} \end{aligned} \tag{14}$$

There are four different communication scenarios in this example. Therefore, there are four partial, 10 joint bitstreams. Each row includes sizes of one compressed partial bitstream and $N-1$ compressed joint bitstreams that can be used in the generation of all other partial bitstreams with the original bitstream in that row. For example, $Table(p_2, p_2)$ shows the bit-length of p_2^c and $Table(p_2, p_1)$, $Table(p_2, p_3)$ and $Table(p_2, p_4)$ show the bit-lengths of $p_{21}^c (= p_{12}^c)$, p_{23}^c and p_{24}^c, respectively. Hence, p_1^c, p_3^c and p_4^c can be generated as follows:

Table 6
A summary of some previous studies on configuration compression techniques.

Study	Technique	Approach	Decompression	Space savings (up to)	Target device
[40]	Huffman, arithmetic and LZ coding	Wildcard	Complex	80%	Virtex
[41]	Adaptive LZW	Intra-frame, Inter-frame and Inter-bitstream regularities	Complex	45% for partial, 38% for complete	Virtex
[42]	RL encoding	Golomb code	Simple	78%	N/A
[45]	LZ	Dictionary	Complex	41%	N/A
Our study	RLL	XOR method	Simple	99% for partial	All Xilinx FPGAs

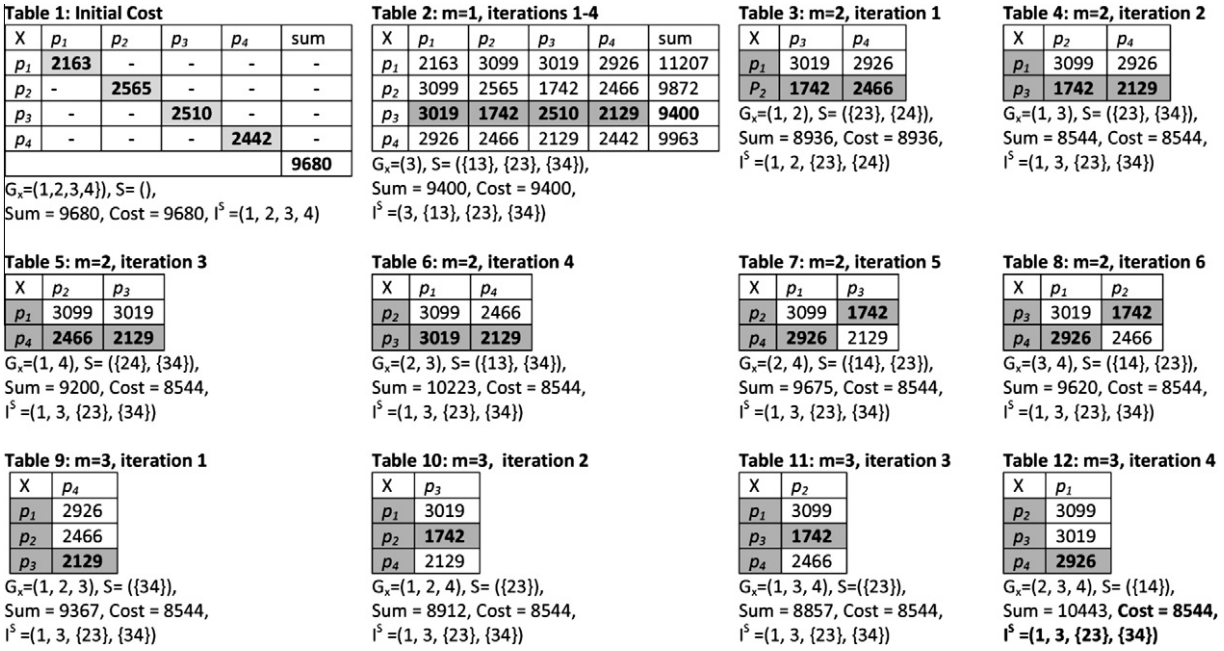


Fig. 11. Steps of the Algorithm 1 for an eight-core implementation on a Virtex-4 FPGA. (X = XOR, if $i \neq j$; X = AND, if $i = j$).

$p_1 = d(p_2^c) \oplus d(p_{12}^c)$, $p_3 = d(p_2^c) \oplus d(p_{23}^c)$and $p_4 = d(p_2^c) \oplus d(p_{24}^c)$. Here, $d(.)$ is the decompression function explained in Section 3.6.1. The best intermediate solutions are identified with shaded cells and bold entries in each table. At first, the initial cost (line 2 in Algorithm 1) is calculated as in the first table in Fig. 11. The second table in Fig. 11 calculates the costs of partial bitstreams set for $m = 1$ (the first iteration of "while" loop), where G_x contains only a single partial bitstream. In Tables 3 to 8 in Fig. 11, where G_x contains two different original bitstreams, the costs of partial bitstreams set for $m = 2$ (the second iteration of "while" loop) are calculated. In Tables 9 to 12 in Fig. 11, where G_x contains three different original bitstreams, the costs of partial bitstreams set for $m = 3$ (the third and last iteration of "while" loop) are calculated. In this example, the solution ($I^s = 1, 3, \{23\}, \{34\}$) is found in the fourth table ($m = 2, Iteration = 2$). Note that, the inner "foreach" loop (lines 9–17) of Algorithm 1 finds the minimum value in each column. The iterations for each m value are relevant to outer "foreach" loop (lines 6–22) of Algorithm 1.

3.4. Storage of partial bitstreams on memory

After compression and selection of optimum set processes, partial bitstream of each communication scenario is stored as initial values for BlockRAM or off-chip RAM.

3.5. c²PCAP architecture

All processes including "Decompression", which are being executed at run-time, are done under the control of c^2PCAP core.

An FPGA from XILINX configures itself through its SelectMAP port under the control of c^2PCAP. Through the parallel SelectMAP interface, the partial reconfiguration information is accepted and the reconfiguration process is executed again by the same target FPGA, where this unique FPGA acts as not only a *slave* but also a

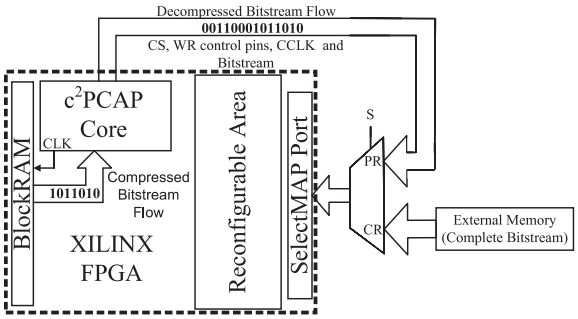


Fig. 12. Hardware architecture of the system over external configuration port.

master at the time of reconfiguration as shown in Fig. 12. Note that the SelectMAP interface is not only dedicated for partial reconfiguration (PR), it might also be used in other designs for external configuration memory (CR:Complete (Re) Configuration, S is used for switching between PR and CR).

The similar structure is also used for devices, which have ICAP modules like Spartan-6, Virtex-4 FPGA. So, through its ICAP interface, the Xilinx FPGA configures itself under the control of c^2PCAP core. As shown in Fig. 13, it is noted that the supported bit-width of ICAP for configuration varies considerably from architecture to architecture: 8-bit for Virtex-II (Pro), 16-bit for Spartan-6, 8/32-bit for Virtex-4, 8/16/32-bit for both Virtex-5 and Virtex-6 [11].

As shown in Fig. 14, the configuration clock (CCLK) is internally generated by DCM. Since FPGA acts as slave during reconfiguration, the source of CCLK is not important for the FPGA. The c^2PCAP core reads a byte from the BRAM at each clock cycle. Under the control of CS/CE, WRITE, CCLK signals, this byte is sent to SelectMAP for Spartan-3 and to ICAP interface for Virtex-4.

The bitstream information, which is accepted from BRAM, is compressed. Since the decompression of bitstream information is achieved at the time of reconfiguration, there is no need any additional time for decompression. Therefore when the CCLK speed is set to 75 MHz, the reconfiguration speed is 75 MB/s. Note that the reconfiguration speed is independent from the size of partial bitstream. The BUSY signal is only used if the configuration clock frequency exceeds 50 MHz. In this study, the CCLK for c^2PCAP core can be configured to operate up to 75 MHz for Spartan-3 and 100 MHz for Spartan-6, Virtex-4. As a result we reached 75 MB/s (75 MHz, 8-bit SelectMAP) on Spartan-3, 200 MB/s (100 MHz, 16-bit ICAP) on Spartan-6, 300 MB/s (75 MHz, 32-bit ICAP) for compressed and 400 MB/s (100 MHz, 32-bit ICAP) for uncompressed partial bitstreams on Virtex-4.

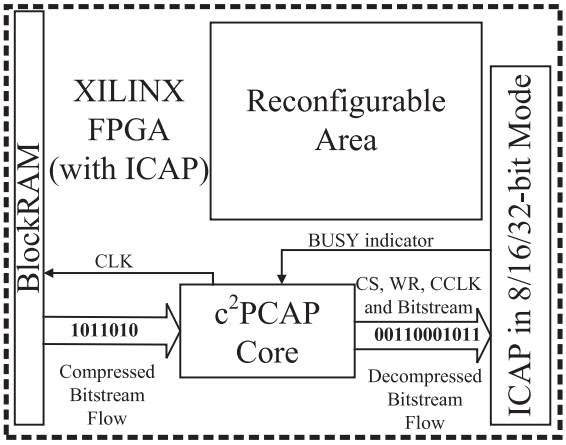


Fig. 13. Hardware architecture of the system over internal configuration port.

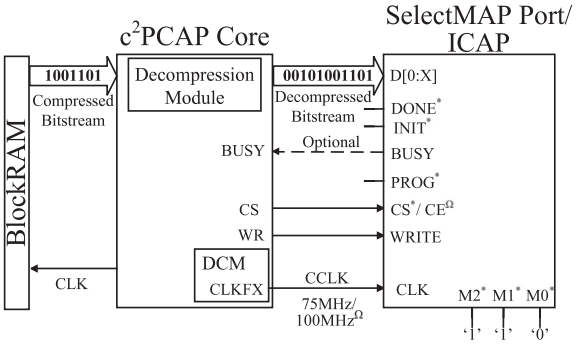


Fig. 14. c^2PCAP core and SelectMAP/ICAP interfaces (*X*: 7/15/31, ∗: only for SelectMAP, Ω: only for ICAP).

3.6. Previous works on self-reconfigurable systems

A lot of studies related to dynamic partial reconfiguration property of reconfigurable architectures have been done in the literature. Especially, partial reconfiguration of FPGAs has an important role in such design techniques. There have been different works to utilize the partial reconfiguration ability of XILINX FPGAs [46,47] and [12–21]. We have summarized the most of these studies in Table 7.

Although we have also used 11 external wires and the SelectMAP port as a reconfiguration interface, we have used BRAM to store partial bitstream, c^2PCAP core to control the reconfiguration flow. However, our c^2PCAP core is very small, which is 261 slices and only 13% of a Spartan-3S200. Thus, we have accomplished a processor-independent run-time reconfigurable system and presented a new approach for storing compressed partial bitstreams on BRAM within the FPGA.

Most of these studies process uncompressed partial bitstreams that are stored in external memory. However, c^2PCAP can process compressed partial bitstreams that are stored in BRAMs. This has two main advantages: (1) One BRAM can hold more compressed partial bitstreams than regular uncompressed partial bitstreams (2) Access time to a partial bitstream in a BRAM is much shorter than the access time to a partial bitstream in an external memory. Decompression is realized in c^2PCAP during reconfiguration time without sacrificing from reconfiguration speed. In addition to the configuration bitstream's storage type, none of above-mentioned studies offers a processor-independent platform. All studies use MicroBlaze or PowerPC as their reconfiguration manager which results in large design sizes. However, our c^2PCAP core is very small, which is 261 slices. Another disadvantage of using MicroBlaze/PowerPC as a reconfiguration flow controller is that they have to postpone their computational jobs while they are busy with reconfiguration. This will slow down the operations which are executed by them. Therefore it has more sense to use a stand-alone core which is dedicated only for control of reconfiguration flow. Apart from these, c^2PCAP core use either parallel SelectMAP [11] or parallel ICAP [11] method in 8/16/32-bit modes, hence we send 8/16/32 bits at each configuration clock cycle.

Moreover, above mentioned designs either suffer from speed or do not offer a processor-independent self-reconfiguration platform for low-cost state-of-the art FPGAs. In addition to these, overclocking of ICAP module of an FPGA is not recommended by the device vendor. To run safely, the speed limit of ICAP is 100 MHz. Therefore, the maximum throughput that can be achieved is 400 MB/s (32-bit mode, not supported in Spartan-6) theoretically. Since Spartan-6 supports reconfiguration through ICAP only in 16-bit interface, the maximum theoretical reconfiguration speed through ICAP at 100 MHz is 200 MB/s. In our case, we propose a processor-independent self-reconfiguration platform for low-cost Spartan-6 FPGA with a safe throughput of 200 MB/s (Spartan-XC6SLX45 ICAP, 16-bit mode, 100 MHz).

3.6.1. Decompression

It is done by hardware and consists of three components as explained in the following subsections.

- Look up table

It is a BRAM unit and contains the indices of partial bitstreams which are used in the generation of *N* communication scenarios.

Table 7
A summary of some previous works on self-reconfigurable systems.

Study	Control engine	Partial bitstream (PB) repository	Interface	Speed	Target device
[12]	MicroBlaze	Off-chip	JTAG	2Mbits/s	Spartan-3
[13]	MicroBlaze	Off-chip	SelectMAP	66 MB/s	Spartan-3
[14]	MicroBlaze	Off-chip	JTAG	0.24 MB/s	Spartan-3
[15]	PowerPC	Off-chip	ICAP	417 KB/s	Virtex-II Pro
[16]	PLB ICAP	Off-chip	ICAP	295.4 MB/s	Virtex-II Pro and Virtex-4
[17]	host PC UART interface	Off-chip	ICAP	200 MB/s	Spartan-6
[18]	MicroBlaze	Off-chip	ICAP	0.54 MB/s	Virtex-4
[19]	MicroBlaze PowerPC	Off-chip	ICAP FSL-ICAP XPS-ICAP	295.4 MB/s	Virtex-4
[20]	MicroBlaze	Off-chip	overclocked ICAP *xps_hwicap* *xps_hwicap*with DMA	800 MB/s 128 MB/s 174 MB/s	Virtex-5
[21]	MicroBlaze	Off- Chip	*dma_hwicap* *mst_hwicap* *bram_hwicap*	82.6 MB/s 253.2 MB/s 371.4 MB/s	Virtex-4
Our study	c^2PCAP	On-chip	SelectMAP or ICAP	300 MB/s 400 MB/s	All Xilinx FPGAs

Table 8
A sample look up table (I^s = {1,3,{23},{34}}, ADR_{p_1}: address of p_1^c, $ADR_{p_{23}}$: address of p_{23}^c).

Communication patterns	Field I	Field II
1	ADR_{p_1}	0
2	ADR_{p_3}	$ADR_{p_{23}}$
3	ADR_{p_3}	0
4	ADR_{p_3}	$ADR_{p_{34}}$

Each communication instance is represented by two address fields in the table. Fields I and II represent the address entries for the compressed partial and the compressed joint bitstreams, respectively. If a communication instance can be implemented solely by a partial bitstream, then its corresponding joint bitstream entry is null. An example is shown in Table 8 for $N = 4$. As an example, we may use the optimum set, p_1^c, p_3^c, p_{23}^c, p_{34}^c, which is found in Fig. 11. For this example, in the look up table, we have address entry values of p_1^c, p_3^c, p_3^c, p_3^c in Field I and 0, p_{23}^c, 0, p_{34}^c in Field II, respectively.

3.6.2. Decompressor

If the parity of the byte in the compressed bitstream in BRAM is 1, then as many zeroes are generated as the byte value. For example, p_{34}^c becomes p_{34}.

3.6.3. Extractor

In a communication instance, in the look up table, if entry II is null, then the partial bitstream is automatically downloaded to the FPGA after the decompression process. Otherwise an XOR operation is realized after the decompression between fields I and II as indicated in Eq. (13). For example, for the 4th communication pattern we have to obtain the p_4, 4th row in Table 8. p_4 is extracted by applying an XOR operation to p_2 and p_{24}: $p_4 = p_2 \oplus p_{24}$.

4. Test and results

The soft c^2PCAP core is developed in VHDL. It has been synthesized on Spartan-3S1000 Starter Kit Board, Atlys Spartan-6 FPGA Development Board and also on ML402 (Virtex-4) Evaluation Platform for experimental purposes.

The terms "*Compression Ratio (CR)*" and "*Space Savings (SS)*" used in tables can be formulated in Eqs. (15) and (16), respectively (Compressed Size:CS, Original Size:OS).

$$CR = (OS \div CS) : 1 \tag{15}$$

$$SS = ((1 - (CS \div OS)) \times 100)\% \tag{16}$$

Here it is noted that the "*CS*" and "*SS*" are actually based on the structure and size of the partial bitstream.

Different communication patterns have been designated between a number of computing cores on Spartan-3S1000 and Virtex-4SX35. There are four different communication scenarios in all examples and a partial bitstream is generated for each one; i.e. RRR = 4. Note that, the communication scenarios change in round robin manner; i.e. $p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_4 \rightarrow p_1 \rightarrow p_2 \ldots$.

The implementation cost of the communication structure is summarized in Tables 9 and 10. In these tables, various designs with different number of modules (8–48) and W_C values (16–384) are shown. The reconfigurable area is fixed to 2 CLB columns for every design. In Table 9, the size of each original partial bitstream and of all other possible joint bitstreams for three different designs are available. The reconfigurable area for each design is the same and has the same number of logic components. Hence, as the number of modules and also wiring between modules increases, the compression ratio decreases. The reason for this phenomenon is that the number of consecutive zeroes in the partial bitstreams decreases. The communication pattern between modules for each design is completely random and different from each other. The results for the designs with regular connection patterns would be better than our results.

As it is obvious from the tables, by increasing the number of modules, bigger compressed bitstreams are obtained. In Table 9, while p_4^c is 744 Bytes for the design with 8-modules (W_C:16) and it is 9742 Bytes for the design with 48-modules (W_C:192). The choice of smallest partial bitstream sets is summarized in Table 10. After generating all possible partial and joint bitstreams for three different designs, the optimal compressed bitstream sets are selected according to Algorithm 1. For example, for the design with 8-modules (W_C:16), the bitstream set p_2^c, p_{12}^c, p_{23}^c, p_{24}^c is found as an optimum solution and the average SS is 98.57% for the complete bitstream set. Space savings results for this optimum solution are indicated in bold in Tables 9 and 10 respectively.

The average number of bitstreams per on-chip BRAM differs from PCAP to cPCAP and c^2PCAP cores. This is illustrated in

Table 9
Partial bitstream size and their compression ratios for communication reconfiguration on Spartan-3 1000.

Partial bitstream	# of Modules	Bit width	Original size (bytes)	Compressed size (bytes)	Compression ratio	Space savings (%)	# of BRAMs (original)	# of BRAMs (compressed)
p_1	8	16	24060	750	32.08:1	96.88	11.74	0.366
	16	32		1050	22.91:1	95.64		0.512
	48	384		7523	3.32:1	69.85		3.673
p_2	8	16	24060	719	33.46:1	**97.01**	11.74	0.351
	16	32		1070	22.49:1	95.55		0.522
	48	384		7575	3.18:1	68.52		3.699
p_3	8	16	24060	743	32.38:1	96.91	11.74	0.362
	16	32		1117	21.54:1	95.36		0.545
	48	384		8417	2.86:1	65.02		4.110
p_4	8	16	24060	744	32.34:1	96.91	11.74	0.362
	16	32		1098	21.91:1	95.44		0.536
	48	384		9742	2.47:1	59.51		4.757
p_{12}	8	16	24060	199	120.09:1	**99.17**	11.74	0.097
	16	32		428	56.21:1	98.22		0.208
	48	384		4045	5.95:1	83.19		1.975
p_{13}	8	16	24060	259	92.9:1	98.92	11.74	0.126
	16	32		487	49.4:1	97.98		0.237
	48	384		3937	6.11:1	83.64		1.922
p_{14}	8	16	24060	206	116.8:1	99.14	11.74	0.100
	16	32		468	51.41:1	98.05		0.228
	48	384		6824	3.53:1	71.64		3.332
p_{23}	8	16	24060	236	101.95:1	**99.02**	11.74	0.115
	16	32		488	49.3:1	97.97		0.238
	48	384		5424	4.44:1	77.46		2.648
p_{24}	8	16	24060	222	108.38:1	**99.08**	11.74	0.108
	16	32		420	57.29:1	98.25		0.205
	48	384		6805	3.54:1	71.72		3.323
p_{34}	8	16	24060	238	101.09:1	99.01	11.74	0.116
	16	32		375	64.16:1	98.44		0.183
	48	384		6739	3.57:1	71.99		3.290

Table 10
Partial bitstream storage cost of communication reconfiguration for PCAP [48], cPCAP [30] and c^2PCAP cores.

Method	# of Modules	Bit width	Reference bitstream	Bitstreams	Total size (Bytes)	# of BlockRAMs	Average # of BlockRAMs per 1 bitsream	Space savings (%)
PCAP [48]	All	All	–	p_1, p_2, p_3, p_4	96240	46.99	11.75	0
cPCAP [30]	8	16	–	p_1^c, p_2^c, p_3^c, p_4^c	2956	1.44	0.36	96.92
	16	32			4335	2.12	0.53	95.49
	48	384			33257	16.24	4.06	65.44
c^2PCAP	8	16	p_2	p_2^c, p_{12}^c, p_{23}^c, p_{24}^c	1376	0.67	0.17	**98.57**
	16	32	p_4	p_4^c, p_{14}^c, p_{24}^c, p_{34}^c	2361	1.15	0.29	97.54
	48	384	p_1	p_1^c, p_{12}^c, p_{13}^c, p_{14}^c	22329	10.90	2.73	76.80

Fig. 15. In this chart, it is clear that while the value of W_C is getting larger, the average number of bitstreams per BRAM decreases for cPCAP and c^2PCAP cores. In addition to this, regardless of the design, it is evident that the c^2PCAP core is the most cost effective in terms of storage.

4.1. Case studies

4.1.1. Target tracking application in a customized multiprocessor architecture

In [49], an FPGA-based multiprocessor-system-on-chip (MPSoC) architecture, which is optimized for Multiple Target Tracking (MTT) in automotive applications, is represented. There are 23 Nios-II processors in this architecture and the communication architecture between them changes with time in a round-robin manner. As illustrated in Fig. 16, there are mainly five different scenarios that runs sequentially and repeatedly with time. In this architecture, the Pulse Repetition Time (PRT), which is the time interval between two successive radar scans, is 25 ms. Therefore, the total reconfiguration time for five scenarios must be smaller than PRT. In addition to this, the longest reconfiguration time (e.g. 2.88 ms for P2–P3 at 100 MHz with ICAP 8-bit mode) must be smaller than the shortest computation time (8 ms for Track Maintenance Block). Since $T_{comp} \geqslant T_{reconf}$, it is safe to say that the computational time overlaps with the reconfiguration time. Hence, the reconfigurable communication architecture is utilized in an optimum manner. It should be noted that, the communication architecture of this case study is not implemented as DRP2P interconnects. Each communication scenario (Fig. 16a–f) is thought as a communication circuitry and replaced with others through DRP2P.

We implemented the communication architecture of this study on a Virtex-4LX100. The reconfiguration latencies for each scenarios are summarized in Table 11a. Although each reconfiguration time in ICAP 8-bit mode at 50 MHz is shorter than the shortest computation time (8 ms for Track Maintenance Block), due to the

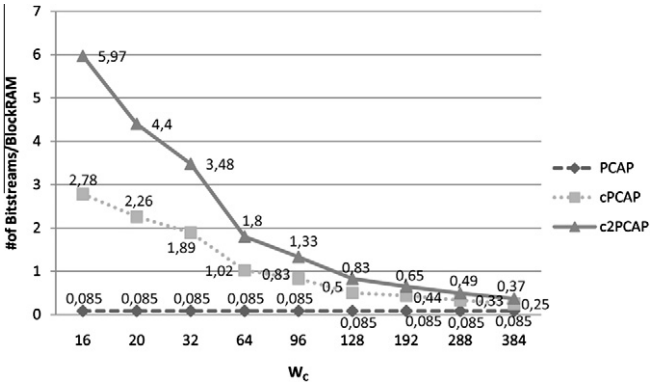


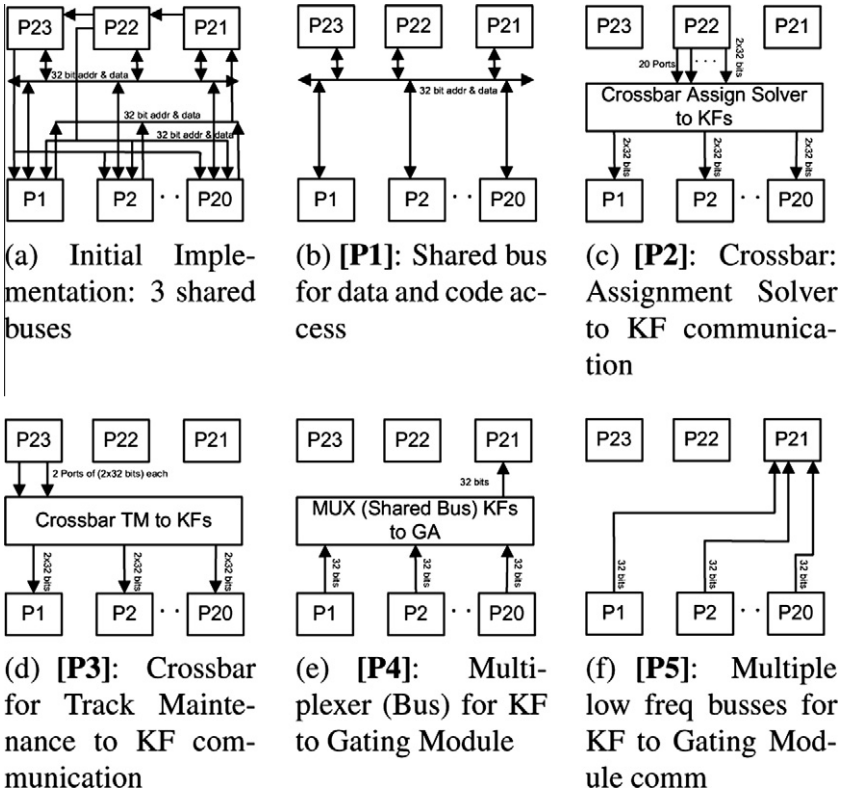
Fig. 15. # of Bitstreams per BRAMs vs. W_C for communication reconfiguration of PCAP [48], cPCAP [30] and c^2PCAP cores.

fact that the total reconfiguration time (27.17 ms) is greater than PRT, it seems to be unapplicable. However, the total reconfiguration latency (13.57 ms) using ICAP 8-bit mode at 100 MHz is shorter than PRT and each reconfiguration time is shorter than the shortest computation time. Therefore it is feasible to use c^2PCAP with ICAP 8-bit at 100 MHz for the best DRP2P interconnects.

Each uncompressed partial bitstream is approximately 260 KB for this study. By cPCAP, each partial bitstream is compressed to the 79 KB approximately with ~70% space savings (storage requires ~40 BRAM blocks). By c^2PCAP, each bitstream is compressed to the 58 KB approximately with ~78% space savings (storage requires ~28 BRAM blocks). As a result, the storage cost of design will be 311 KB and requires ~152/240 BRAM blocks on a Virtex-4LX100 (*e.g.* $p_1, p_3, p_{23}, p_{14}, p_{35}$). By using on-chip memory as a cache, which means storing only bitstream (s) for the next communication scenario and removing past bitstream (s) from BRAM, only ~60/240 BRAM blocks (25% of total) will be occupied. There are two different approaches to use on-chip memory as a cache, first approach keeps a single bitstream on BRAM while the second one keeps multiple bitstreams. Assume that there is only one partial bitstream and three joint bitstreams (e.g. $p_2, p_{12}, p_{23}, p_{24}$) in the optimal set for four different communication scenarios. In the first approach, only one partial bitstream (*e.g.* p_2) is stored on BRAM, the joint bitstreams are loaded when the corresponding bitstream is needed. With this approach, the storage cost for bitstreams can be reduced by factor $1 \div N$. If we apply this approach to this example (e.g. $p_1, p_3, p_{23}, p_{14}, p_{35}$), we should store at most two partial bitstreams on BRAM, the BRAM usage can be decreased from ~152/240 to ~60/240. While it is storage efficient, the main drawback of the first approach is that it may require multiple bitstream load at a time. For example, assume that only p_1 is available on the cache and p_5 must be downloaded. To achieve this, at first p_1 must be removed then p_3 and p_{35} must be loaded to cache. As a second approach, for the given example (e.g. $p_1, p_3, p_{23}, p_{14}, p_{35}$), p_1 and p_3 are stored on BRAM initially. They can be directly read from cache, when they will be used. To obtain p_2, only p_{23} is taken to cache. For p_4, p_{14} is also loaded to cache, in the same way to get p_5, after removing unused joint bitstreams p_{35} is loaded. So, instead of storing five partial bitstreams on BRAM, it is enough to store at most three of them for this example. So, we may reduce the BRAM usage from ~152/240 to ~91/240 for such a design by using cache method.

4.1.2. N-body problem

One of the computation intensive problems is the N-body problem [50], which can be applied to extensive applications from various domains in engineering and science. The N-body problem deals with N particles. Each particle interacts with the remaining ones in each time step. To reduce the solution time of this problem



(a) Initial Implementation: 3 shared buses

(b) **[P1]**: Shared bus for data and code access

(c) **[P2]**: Crossbar: Assignment Solver to KF communication

(d) **[P3]**: Crossbar for Track Maintenance to KF communication

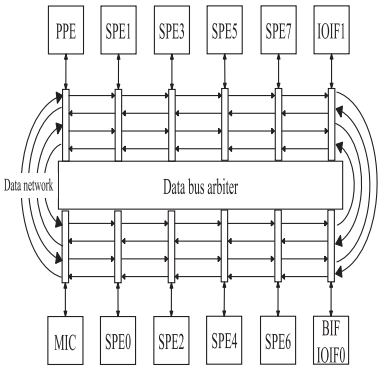
(e) **[P4]**: Multiplexer (Bus) for KF to Gating Module

(f) **[P5]**: Multiple low freq busses for KF to Gating Module comm

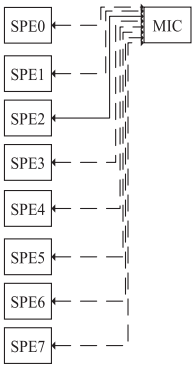
Fig. 16. Different scenarios of MPSoC architecture for MTT in a PRT.

Table 11
Reconfiguration latencies [ms] for case studies.

Freq. MHz	ICAP (8-Bit)		ICAP (32-Bit)	
PB	50	100	75	100
(a) MTT				
p_1	5.36	2.68	0.894	0.670
p_2	5.36	2.68	0.894	0.670
p_3	5.77	2.88	0.962	0.721
p_4	5.47	2.73	0.911	0.683
p_5	5.21	2.60	0.868	0.651
Total	27.17	13.57	4.529	3.395
(b) N-body				
p_1	0.217	0.109	0.036	0.027
p_2	0.217	0.109	0.036	0.027
p_3	0.226	0.113	0.038	0.028
p_4	0.240	0.120	0.040	0.030
p_5	0.226	0.113	0.038	0.028
p_6	0.226	0.113	0.038	0.028
p_7	0.226	0.113	0.038	0.028
p_8	0.324	0.162	0.054	0.040
Total	1.902	0.951	0.317	0.238



(a) Element interconnect bus (EIB). (BIF: Broadband interface, IOIF: I/O interface)[52]



(b) 128-bit-width DRP2P communication between SPEs and MIC (Only SPE2 is connected to MIC at this configuration)

Fig. 17. Element interconnect bus (EIB) and DRP2P communication between SPEs and MIC.

($O(N^2)$), the Barnes–Hut algorithm is applied to the N-body problem and Cell Broadband Engine Architecture is used [51].

In [52], the design of the Cell processors (The Cell Broadband Engine processor) on-chip network and its communication and synchronization protocols are proposed. Fig. 17a shows the Element Interconnect Bus (EIB), the main component of the Cell processor's communication architecture, which provides the communication between 8 Synergistic Processor Elements (SPEs), Power Processor Element (PPE), I/O Interface and Memory Interface Controller (MIC).

In this case study, we implemented DRP2P to connect the nodes (SPEs) with 128-bit width reconfigurable bus channel to the MIC. These connections change in round robin manner. Hence, there is only one 128-bit-width connection with MIC at a time. So, each SPE communicates (sends or receives, in both directions) with MIC sequentially. Fig. 17b shows this communication architecture. Here, only SPE2 is connected to MIC, all other SPEs are unconnected and process their computations at this time. While one SPE is communicating with the MIC, the others continue their computations without being interrupted. This architecture is implemented on the Virtex-4LX100 FPGA and the reconfiguration latency for each connection is summarized in Table 11b. Here p_1 is the partial bitstream, which has the communication information of SPE0 and MIC. In the same manner, the rest partial bitstreams have the communication information of SPE1-SPE7 and MIC, respectively.

Fig. 18 illustrates the communication latencies of EIB on Cell processor and DRP2P interconnects on Virtex-4LX100 FPGA. With a clock speed of 3.2 GHz, the Cell processor is a heterogeneous multi-core chip, which is capable of massive floating-point processing. Our target platform, Virtex-4LX100 FPGA from XILINX, is a reconfigurable, fully parallel platform with a clock speed of 100 MHz. As obvious from the illustration, DRP2P through ICAP in 32-bit mode at 100 MHz (DRP2P_ICAP_32b_100MHZ) has the best performance regardless of the problem's size. The configuration clock speed and the width of configuration interface is proportional to the power consumed during reconfiguration process. Hence, DRP2P_ICAP_8b_50MHZ can be regarded as the least power consuming method among DRP2P options. On the other hand, it performs worse than EIB on Cell processor when the number of particles fewer than 8192. DRP2P_ICAP_32b_75MHZ and DRP2P_ICAP_8b_100MHZ, are better than Cell processor in terms of communication latencies when the number of particles is greater than 2048 and 4096, respectively.

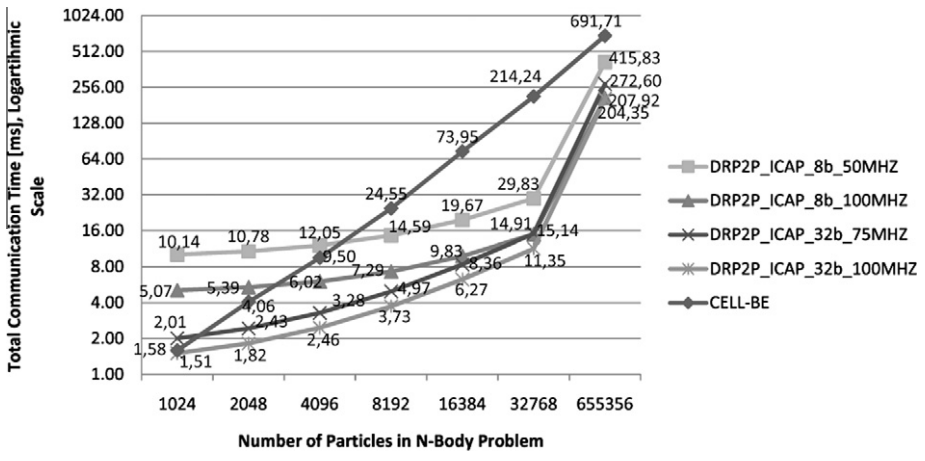


Fig. 18. Total time consumed for communication between SPEs in N-body problem with different number of particles.

5. Conclusion

In this paper we have discussed dynamic reconfigurable point-to-point interconnects in parallel multi-core architectures. We have showed that there will be no reconfiguration latency and the system will be working at its highest attainable speed as in direct connections if communication paths can be reconfigured while the processors are operating on the data in their local memories. This can be achieved when the partial reconfiguration totally overlaps with the computation time ($T_{comp} \geqslant T_{reconf}$).

We have mentioned that DRP2P architecture can be used not only for the intra-switching purposes within an application; it can also be utilized for the inter-switching among different applications. Since the switching time between most use-cases in a SoC is at the order of few milliseconds [23], the time required to reconfigure the FPGA partially can be covered up safely. Our first case study can be given as an example for switching among different applications or that is to say communication architectures by utilizing DRP2P approach. In this case study, e.g. we change the communication infrastructure of the system from shared bus to crossbar: this is done by removing the communication circuitry of first scenario (see Fig. 16b) from the FPGA and loading the communication circuitry of the second scenario (see Fig. 16c) to the FPGA. Instead of implementing a packet or circuit switched NoC, we preferred to use DRP2P for this case study. Because this system has 23 cores, it requires constructing a 2D $5 * 5$ mesh NoC (32-bit). Even for a $4 * 4$ mesh topology with 32-bit data width, the open core NoCem occupies 125% of Virtex-II FPGA (xc2vp30) [26]. As a result of this, it seems impossible to implement a $5 * 5$ mesh NoC (32-bit) even on a high-end FPGA from XILINX.

The self-reconfiguration of a Virtex-4, Spartan-6 and Spartan-3 FPGAs through ICAP and SelectMAP ports are examined in detail. In addition to this, storing different partial bitstreams on BRAM within the target FPGA is also discussed in this paper. The most important advantage of this study is to achieve a very fast partial reconfiguration compared to other studies in the literature.

Since there is no need to use an external intelligent agent, there is also no need to use an external interface between controller and the target reconfigurable device. As a result, neither an external controller nor an external wiring between these devices is necessary for c^2PCAP core. Therefore, the hardware cost can be reduced. As the supply voltages for driving external devices are higher than the supply voltage for the internal operations of the FPGA, the total power dissipation is reduced. Hence, storing compressed partial bitstreams on on-chip memory and under the control of an internal agent, c^2PCAP core, reduces hardware cost and power consumption simultaneously. Furthermore, with the capability of storing compressed partial bitstreams, different partial bitstreams can be stored on-chip memory at a glance.

The developed c^2PCAP core can be applied not only on a Virtex-4, Spartan-6 or on a pure Spartan-3 and also on other Xilinx FPGA series such as Virtex-II, Virtex-5, Virtex-6, Spartan-3A (N) and so on.

In addition, by occupying a small area on the FPGA, the portable c^2PCAP core can be located anywhere in any Xilinx FPGA.

Processes, such as communication scenario extraction and selection of most appropriate communication infrastructure are being done manually for DRP2P at the moment. For these processes, we are developing automation tool, which extracts the communication scenario and decides the most suitable interconnection network at a time for a given compute intensive application.

Acknowledgments

We would like to thank Ömer Çoğal for supplying numeric test values for NoC, Smail Niar and Betül Demiröz for their supports, suggestions and corrections in our case studies.

References

[1] C. Hilton, B. Nelson, Pnoc: a flexible circuit-switched NoC for fpga-based systems, IEE Proceedings on Computers and Digital Techniques 153 (3) (2006) 181–188.
[2] M. Modarressi, A. Tavakkol, H. Sarbazi-Azad, Virtual point-to-point connections for NoCs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 29 (6) (2010) 855–868.
[3] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, J. van der Veen. Dynoc: a dynamic infrastructure for communication in dynamically reconfigurable devices, in: International Conference on Field Programmable Logic and Applications, 2005, pp. 153–158.
[4] M.B. Stensgaard, J. Sparso. Renoc: a Network-on-Chip architecture with reconfigurable topology, in: Second ACM/IEEE International Symposium on Networks-on-Chip, 2008, NoCS 2008, pp. 55 –64.
[5] S.J. Hollis, C. Jackson, Skip the analysis: self-optimising networks-on-chip (invited paper), in: International Symposium on Electronic System Design (ISED), 2010, pp. 14–19.
[6] J.L. Ma, C. Wang, Y. Wen, T.Z. Chen, W. Hu, J. Chen. Dynamic reconfigurable networks in NoC for I/O supported parallel applications, in: International Conference on Computer and Information Technology, 2010, pp. 0:2768–0:2775.
[7] R. Vancayseele, B.A. Farisi, W. Heirman, K. Bruneel, D. Stroobandt. Reconoc: a reconfigurable Network-on-Chip, in: Sixth International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011, pp. 1–2.
[8] Mehdi Modarressi, Arash Tavakkol, Hamid Sarbazi-Azad, Application-aware topology reconfiguration for on-chip networks, IEEE Transactions on VLSI Systems 19 (11) (2011) 2010–2022.
[9] U.Y. Ogras, R. Marculescu, Application-specific Network-on-Chip architecture customization via long-range link insertion, in: IEEE/ACM International Conference on Computer-Aided Design, 2005, ICCAD-2005, pp. 246–253.
[10] H. Giefers, M. Platzner, A triple hybrid interconnect for many-cores: Reconfigurable Mesh, NoC and barrier, in: International Conference on Field Programmable Logic and Applications (FPL), 2010, pp. 223 –228.
[11] Available from: <http://www.xilinx.com/support/documentation>.
[12] K. Paulsson et al., Implementation of a Virtual Internal Configuration Access Port (JCAP) for enabling Partial Self-Reconfiguration on Xilinx Spartan-III FPGAs, FPL'07, August 2007.
[13] I. Gonzalez et al. Self-reconfigurable embedded systems on low-cost fpgas. IEEE Micro, July–Aug. 2007, pp. 49–57.
[14] K. Paulsson et al., Exploitation of the External JTAG Interface for Internally Controlled Configuration Readback and Self-Reconfiguration of Spartan 3 FPGAs, ISVLSI'08, April 2008.
[15] M.L. Silva, J.C. Ferreira, Support for partial run-time reconfiguration of platform FPGAs, Journal of Systems Architecture 52 (12) (2006) 709–726.
[16] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hübner, J. Becker, A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput. in: Proc. of the International Conference on Field Programmable Logic and Applications FPL 2008, 2008, pp. 535–538.
[17] Dirk Koch, Christian Beckhoff, Jim Torrison, Advanced Partial Run-time Reconfiguration on Spartan-6 FPGAs, FPT'10, December 2010.
[18] Sheetal U. Bhandari, Shaila Subbaraman Shashank Pujari, Rashmi Mahajan, Internal dynamic Partial reconfiguration for real time signal processing on FPGA, Indian Journal of Science and Technology 3 (4) (2010) 365–368.
[19] Michael Hübner, Diana Göhringer, Juanjo Noguera, Jürgen Becker, Fast dynamic and Partial Reconfiguration data path with low hardware overhead on Xilinx FPGAs, IPDPS'10, 2010.
[20] François Duhem, Fabrice Muller, Philippe Lorenzini, FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA, ARC'2011, 2011.
[21] Ming Liu, Wolfgang Kuehn, Zhonghai Lu, Axel Jantsch. Run-Time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration, in: Proc. of the International Conference on Field Programmable Logic and Applications, August 2009.
[22] Karel Bruneel, Fatma Abouelella, Dirk Stroobandt, Automatically mapping applications to a self-reconfiguring platform, in: Proceedings of the Conference on Design, Automation and Test in Europe, DATE '09, 2009, European Design and Automation Association, 3001 Leuven, Belgium, Belgium, pp. 964–969.
[23] Srinivasan Murali, Martijn Coenen, Andrei Radulescu, Kees Goossens, Giovanni De Micheli, A methodology for mapping multiple use-cases onto networks on chips, in: DATE, IEEE, 2006, pp. 118–123.
[24] J.Y. Hur et al., Partially reconfigurable point-to-point FPGA interconnects, International Journal of Electronics 95 (7) (2008) 725–742.
[25] Nocem, network on chip emulator @ONLINE, April 2007. Available from: <http://www.opencores.org/project.nocem>.

[26] Graham Schelle, Dirk Grunwald, Onchip interconnect exploration for multicore processors utilizing fpgas, Memory, 2006, pp. 1–4.
[27] Jingcao Hu, Umit Y. Ogras, Radu Marculescu, System-level buffer allocation for application-specific networks-on-chip router design, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25 (12) (2006) 2919–2933.
[28] D.J.A. Welsh, M.B. Powell, An upper bound for the chromatic number of a graph and its application to timetabling problems, The Computer Journal 10 (1) (1967) 85–86.
[29] Salih Bayar, Mehmet Tükel, Arda Yurdakul, A self-reconfigurable platform for general purpose image processing systems on low-cost spartan-6 fpgas, in: Proceedings of the Sixth International Workshop on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC 2011, IEEE, 2011, Montpellier, France, 20–22 June, pp. 1–9.
[30] Salih Bayar, Arda Yurdakul, Self-reconfiguration on spartan-iii fpgas with compressed partial bitstreams via a parallel configuration access port (cpcap). PRIME2008, June 2008.
[31] Katarina Paulsson, Michael Hübner, Salih Bayar, Jürgen Becker, Exploitation of Run-Time Partial Reconfiguration for Dynamic Power Management in Xilinx Spartan III-based Systems, ReCoSoc2007, Montpellier, France, June 2007.
[32] M. Hübner et al., Real-Time LUT-Based Network Topologies for Dynamic and Partial FPGA Self-Reconfiguration, SBCCI'04, September 2004.
[33] L. Möller et al., Infrastructure for Dynamic Reconfigurable Systems, Choices and Trade-offs, SBCCI06, August 6.
[34] K.P. Raghuraman, H. Wang, S. Tragoudas, A Novel Approach to Minimizing Reconfiguration Cost for LUT-Based FPGAs. VLSID05, Kolkata, India, January 3–7, 2005.
[35] M. Rullmann, R. Merker, A Reconfiguration Aware Circuit Mapper for FPGAs, IPDPS 2007.
[36] W. Chen et al., A New Placement Approach to Minimizing FPGA Reconfiguration Data, ICESS2008.
[37] F. Mehdipour et al., Reducing Reconfiguration Time of Reconfigurable Computing Systems in Integrated Temporal Partitioning and Physical Design Framework, pp. 1–8, Proceedings of 20th IPDPS, April 2006.
[38] P. Stepien, M. Vasilko, On Feasibility of FPGA Bitstream Compression During Placement and Routing, FPL'06, Madrid, Spain, August 28–30, 2006.
[39] B. Sellers et al., Bitstream Compression Through Frame Removal and Partial Reconfiguration, FPL'09, Czech Republic, August 31–September 2, 2009.
[40] Z. Li, S. Hauck, Configuration Compression for Virtex FPGAs, FCCM01, 29 April–2 May 2001.
[41] H. Gu, S. Chen, Partial Reconfiguration Bitstream Compression for Virtex FPGAs, CISP'08, May 27–30.
[42] D. Koch, J. Teich, Platform Independent Methodology for Partial Reconfiguration, CF'04, April 2004.
[43] R. Ştefan, S.D. Coţofană, Bitstream Compression Techniques For Virtex 4 FPGAS, FPL'08, September 2008.
[44] D. Koch, et al., Bitstream Decompression for High Speed FPGA Configuration from Slow Memories, ICFPT'07, December 2007.
[45] A. Dandalis, V.K. Prasanna, Configuration compression for FPGA-based embedded systems, IEEE TVLSI 13 (12) (2005) 1394–1398.
[46] M. Edwards, P. Green, Run-time support for dynamically reconfigurable computing systems, Journal of Systems Architecture 49 (4-6) (2003) 267–281.
[47] A. Ahmad et al., Efficient architectures for 3D HWT using dynamic partial reconfiguration, Journal of Systems Architecture 56 (2010) 305–316.
[48] Salih Bayar, Arda Yurdakul, Dynamic partial self-reconfiguration on spartan-iii fpgas via a parallel configuration access port (pcap), HIPEAC2008, January 2008.
[49] J. Khan et al., Trade-off Exploration for Target Tracking Application in a Customized Multiprocessor Architecture, EURASIP Journal on Embedded Systems (2009).
[50] K. Asanovic et al., The Landscape of Parallel Computing Research: A View from Berkeley, Technical Report UCB/EECS-2006-183, December 18, 2006.
[51] B. Demiroz, et al., Parallelizing Barnes–Hut Method on the Cell BE Architecture. MULTIPROG, January 24, 2010.
[52] M. Kistler et al., Cell multiprocessor communication network: built for speed, IEEE Micro 26 (2006) 10–23.

Salih Bayar received his B.Sc. degree in electronics and communication engineering from Yıldız Technical University, Istanbul, Turkey in 2003. He has received his M.Sc. degree in Electrical Engineering and Information Technology in the field of specialization Systems Engineering from University of Karlsruhe (TH), Karlsruhe, Germany, in 2007. Since 2007, he has been research assistant and Ph.D. student in Computer Engineering Department in Boğaziçi University, Istanbul, Turkey. His research interests are reconfigurable computing, dynamic and partial reconfiguration of Xilinx FPGAs, multi-processor and embedded multi-core architectures, System-on-Chip, Network-on-Chip and design automation.

Arda Yurdakul received the B.SC. degree in electrical and electronics engineering with honors from Boğaziçi University, Istanbul, Turkey in 1992. Her M.SC. and Ph.D. degrees are also from the same university in 1994 and 1999, respectively. She was with Kadir Has University from 1999 to 2001 as an Assistant Professor. Since 2001, she has been with Computer Engineering Department in Boğaziçi University where she is currently an Associate Professor. She was the IEEE Turkey section chair from January, 2010 to July, 2011. Her main research interests are embedded systems, reconfigurable computing, electronic design automation.