A Reconfigurable Network-on-Chip Architecture for Optimal Multi-Processor SoC Communication

Vincenzo Rana*, David Atienza†‡, Marco Domenico Santambrogio*, Donatella Sciuto*, and Giovanni De Micheli‡
* Dipartimento di Elettronica e Informazione (DEI) - Politecnico di Milano,
Via Ponzio 34/5, 20133 - Milano, Italy. E-mail: {rana, santambr, sciuto}@elet.polimi.it
† Dept. Computer Architecture and Automation (DACYA) - Universidad Complutense de Madrid (UCM),
Avda. Complutense S/N, 28040-Madrid, Spain. E-mail: datienza@dacya.ucm.es
‡ Integrated Systems Laboratory (LSI) - Ecole Polytechnique Fédérale de Lausanne (EPFL),
Station 14, 1015-Lausanne, Switzerland. E-mail: {david.atienza, giovanni.demicheli}@epfl.ch

***Abstract*— Network-on-Chip (NoC) has emerged as a very promising paradigm for designing scalable communication architecture for Systems on Chips (SoCs). However, NoCs designed to fulfill the bandwidth requirements between the cores of an SoC for a certain set of running applications may be highly sub-optimal for another set of applications. In this context, methods that can lead to versatility enhancements of initial NoC designs to changing working conditions, imposed by variable sets of executed real-life applications at each moment in time, are very important for designing competitive NoCs in industrial SoCs.**

In this work, we present a run-time reconfigurable NoC framework based on the partial dynamic reconfiguration capabilities of Field-Programmable Gate Arrays (FPGAs). This new NoC framework can dynamically create/delete express lines between SoC components (implementing dynamically circuit-switching channels) and perform run-time NoC topology and routing-table reconfigurations to handle interconnection congestion, with a very limited performance overhead. Moreover, we show in our experimental results that the addition of these dynamic reconfiguration capabilities into basic NoCs using our framework only implies a very limited area overhead (around 10% on average) with respect to the initial NoC designs; thus, it can bring great benefits when compared to traditional non-reconfigurable NoC design approaches for worst-case bandwidth requirements in SoCs with many possible sets of running applications.

I. Introduction and Problem Description

Latest applications ported to embedded systems (e.g., scalable video rendering, communication protocols) demand a large computation power, while must respect other critical embedded design constraints, such as, short time-to-market, low energy consumption or reduced implementation size.

Thus, embedded systems are complex *Systems-on-Chip (SoCs)* that consist of a large number of components, such as, processing elements, storage devices and even reconfigurable devices, such as *Field-Programmable Gate Arrays (FPGAs)*, to enhance the flexibility of final SoCs to be used in different environments [15], [5]. Nevertheless, one of the most critical areas of MPSoC design is the definition of the suitable interconnect subsystem for all these SoC components, due to architectural and physical scalability concerns [3]. In fact, traditional shared bus interconnects are relatively easy to design, but do not scale well for latest and forthcoming SoC consumer platforms.

In order to cope with the large communication demands of such SoCs, the use of modular and scalable *Networks-on-Chips (NoCs)* has been proposed [3]. Then, designing custom-tailored NoC interconnects that satisfy the performance and design constraints of the SoC for all the different combinations of possible executed applications is a key goal to achieve optimal commercial products [13], [2]. However, as general-purpose processor cores are used to run software tasks of different applications in SoCs, the communication between the cores cannot be precharacterized and fully optimized, since the application processes can be mapped differently to the cores, typically with the support of the compiler. Thus, to provide predictable performance of the NoC, the bandwidth capacity of the different links must be sufficient to support the peak rate of traffic on the links of the possible different mappings of the tasks onto the final SoC. Otherwise, the network might experience traffic congestion and the latency for the traffic streams and, hence, the interconnect performance will become unacceptable, which needs to be avoided to provide appropriate consumer devices. As a result, NoCs designs that guarantee worst-case bandwidth conditions of SoC operation with multiple concurrent application often leads to over-sized topologies and links on regular operation of the SoC. In this context, the development of new methods and frameworks that increase the run-time versatility of initial static NoC designs to adapt to different working conditions, originated by the diversity of sets of applications at each moment, is an important research area in the NoC domain.

In this paper we introduce a novel run-time reconfigurable NoC framework, which exploits the partial dynamic reconfiguration capabilities of *FPGAs* to adapt at run-time the implemented NoC interconnect to the specific working requirements of the final SoC at each moment in time. In particular, the proposed NoC framework is able to reduce the latency of interconnecting the included SoC components by dynamically establishing or deleting a number of dedicated point-to-point connections between them (or express lines in the NoC literature [3]), which is particularly suited for video and audio streaming. Thus, circuit-switching communication can be dynamically configured in the SoC. In addition, our framework enables a fast dynamic reconfiguration of routing tables (few cycles) and overall NoC topologies (few milliseconds), which provides new promising means to overcome congestion and consequently provide more reliable and high-performance NoC designs. Furthermore, our experimental results show that the addition of the dynamic reconfiguration capabilities in basic NoCs using our framework only involves limited area overheads (around 10% on average) with respect to initial NoC designs without reconfiguration capabilities. Hence, the proposed reconfigurable NoC framework is viable to be considered in commercial designs of SoCs.

The rest of the paper is organized as follows. In Section II we overview previous work in the field on reconfigurable NoCs. Then, in Section III we introduce our reconfigurable NoC architecture, spanning from the included basic NoC architecture to the additional components to enable the NoC Next, in Section IV we discuss the major reconfiguration capabilities and methods to implement them in the proposed adaptive NoC framework. Later, in Section V we present the area, performance and latency evaluation of the reconfiguration capabilities of our framework in a real implementation on a large commercial FPGA implementing a multi-processor SoC. Finally, in Section VI we draw the main conclusions of this work.

II. State of the art

In recent years, several works focused their attention into the definition of a reconfigurable communication infrastructure for reconfigurable Systems-on-chip. For instance, in [11] the authors present a methodology for developing dynamic network reconfiguration processes, but they define a reconfiguration just as the change from one routing function to another while the network is up and running. For this reason they present a theoretical work based on the limiting assumption (not valid in the approach presented in this paper) that the network topology can be considered fixed (like in [7]).

The work presented in [6] describes an integrated modeling, simulation and implementation tool for reconfigurable NoCs. The work is based on the optimization of a single given application and no details are given about the reconfigurable architecture of the NoC and about reconfiguration mechanisms. In addition to this, flow control is not supported and the proposed NoCs are quite expensive in terms of area usage (2733 slices, around 30% of the total slices of a Xilinx Virtex II Pro XC2VP20 device), for a 2x2 torus running at 85 MHz, while the proposed approach, with a lower area usage, allows the creation of a 3x3 mesh running at 170 MHz.

In [1], a dynamically reconfigurable NoC architecture is presented. This NoC can be dynamically configured with respect to routing, switching and data packet size, but all the required resources have to be allocated at design time, since at run-time it is only possible to dynamically change a limited number of parameters. A similar approach can be found in [10], where the proposed NoC can be configured at run-time, but only with respect to memories content, resources addressing and control parameters, while topology, buffers size and port connections have to be determined at design time.

In [9], a scalable dynamic NoC for dynamically reconfigurable FPGAs (CuNoC) is presented. The main idea behind CuNoC approach is to fill the whole reconfigurable devices with very small communication units called CUs, that can establish a communication channel between two different cores. The main drawback of this approach is, in addition to the huge power consumption, the high latency for each communication. In fact, the number of hops required for a communication is very high on average, as each packet has to pass through a high number of CUs (each one having a latency of 2 clock cycles), since it is not possible neither to define a custom topology nor to configure express lines between CUs. Furthermore, if an obstacle is present between two cores that need to communicate, it is necessary to go around it, which increases the number of hops of each packet.

In [14] the authors present CoNoChi, that is an adaptable NoC for dynamically reconfigurable hardware design. The reconfigurable device is divided in a matrix, and each cell of this matrix can hold either a computational module or a communication element (a switch or point-to-point interconnect). Since it is not possible to pass through a computational module, each communication channel has to go around all the computational elements placed on the reconfigurable device; thus, express lines cannot be configured at run-time. In addition to this, the area requirement for a single switch is very high, as it varies from 463 to 493 slices (around 5% of a XC2VP20 device). Then, the working frequency is quite low (it ranges from 66 to 73 MHz) and the actual latency of each switch is 5 clock cycles, which can be a significant penalty for interconnection mechanisms nowadays.

III. The proposed reconfigurable architecture

A. Reconfiguration support

In order to configure an FPGA with the desired functionality, we need to use one or more bitstreams. A bitstream is a binary file in which configuration information for a particular Xilinx device is stored, that is where all the data to be copied on to the configuration SRAM cells, the configuration memory, are stored, along with the proper commands for controlling the chip functionalities. Therefore Virtex devices, such as Virtex II Pro and Virtex 4, are configured by loading application specific data into their configuration memory, as shown in Figure 1. On the Virtex FPGAs the configuration memory

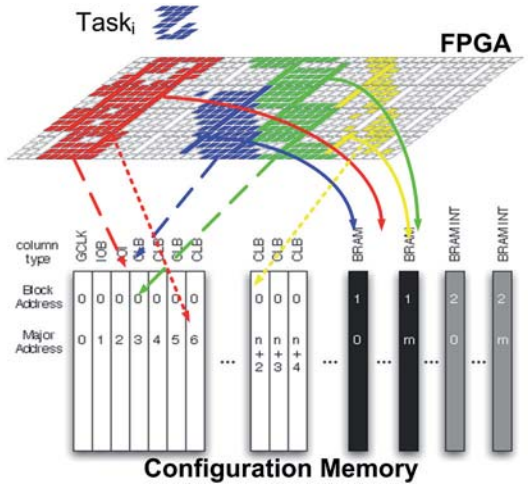

Fig. 1. Configuration memory setup

is segmented into frames. Virtex devices are partially reconfigurable and a frame is the smallest unit of reconfiguration. According to the device, this element can span the entire length of the FPGA, such as in the Virtex II Pro context, or just part of it, as in Virtex 4 devices. The number of frames and the bits per frame are specific for each device family. The number of frames is proportional to CLB width. Bitstreams can be either partial or full. A full bitstream configures the whole configuration memory and is used for static design or at the beginning of the execution of a dynamic reconfiguration system, to define the initial state of SRAM cells. Partial bitstreams configure only a portion of the device and are one of the end products of any partial reconfiguration flow.

FPGAs provide different means for configuration, under the form of different interfaces to the configuration logic on the chip. There are several modes and interfaces to configure a specific FPGA family, among them the the IEEE 1149.1 *Joint Test Action Group (JTAG)* download cable (the one used in this work), the SelectMAP interface, for daisy-chaining the configuration process of multiple FPGAs, configuration loading from PROMs or compact flash cards, microcontroller-based configuration, an *Internal Configuration Access Port (ICAP)* and so on, depending on the specific family. The ICAP provides an interface which can be used by internal logic to reconfigure and read back the configuration memory. In every FPGA a configuration logic is built on the chip, with the purpose of implementing the different interfaces for exchanging configuration data and to interpret the bitstream to configure the device. A set of configuration registers defines the state of this configuration logic at a given moment in time. Configuration registers are the memory where the bitstream file has direct access. Actual configuration data is first written by the bitstream into these registers and then copied by the configuration logic on the configuration SRAMs.

B. Architecture description

As previously hinted, the communication infrastructure of the proposed architecture is based on the Network-on-Chip (NoC) paradigm.

Furthermore, in order to exploit a *2-layered* approach, in which the *computational layer* is completely decoupled from the *communication layer*, the proposed reconfigurable architecture mainly consists of two different parts: a *static part* and a *reconfigurable part*.

The **static part** consists of all the computational elements and the network interfaces. On the one hand, computational elements can be further divided into two categories. The first one consists of *masters*, that are the active components of the system, such as microprocessors (either a soft-core, as a MicroBlaze, or a hard-core as a PowerPC), that can initialize new transactions on the network (deployed in the communication layer); these components are connected to the communication infrastructure through *NI initiators* (see Figure 2). The second one consists of *slaves*, such as memories, that represent the components that act in a passive mode, by receiving and answering transaction coming from active elements; these components are connected to the communication infrastructure through *NI targets* (see Figure 2).

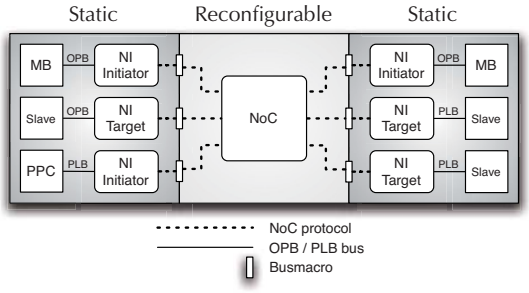

Fig. 2. Interfaces between static and reconfigurable parts

The **reconfigurable part** is composed by all the reconfigurable elements, used to adapt at run-time the structure of the system implemented on the FPGA. These elements can be either computational components or elements used to update the communication infrastructure. Network interfaces toward the communication infrastructure can implement bridges between On-chip Peripheral Bus (OPB), Processor Local Bus (PLB) or Open Core Protocol (OCP) and the network protocol, as shown in Figure 2. The only part of network interfaces (both initiator and target network interfaces) that has to be modified at run-time are routing tables, that are used to dynamically change the routing of packets on the network. Thus, all the network interfaces have been placed into the static part of the system and routing tables have been deployed on BRAM blocks. In this way it is possible to dynamically modify routing tables by changing the content of BRAM blocks at run-time, as described in Section IV-A.

This architectural solution enables connecting the static parts to the reconfigurable ones by using network interfaces that are considerably thinner of the ones used within the static part of the system. Regarding this static part, the used interconnect can be either OPB and PLB buses, or on an ad-hoc point-to-point communication infrastructure, as shown in Figure 2.

IV. Reconfiguration features

Each reconfigurable part of the system can be dynamically reconfigured at run-time to modify either a part or the whole underlying communication infrastructure. This reconfiguration can be done by the reconfiguration controller (see Figure 3), which is a master component present on the static part, trough partial reconfiguration operations.

The reconfiguration controller is connected both to the external dynamic memory (DDR) interface and to the ICAP interface through

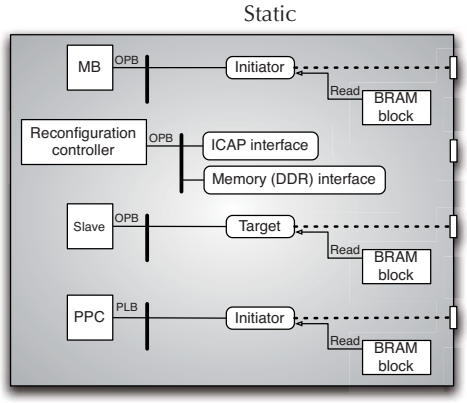

Fig. 3. A static part with a reconfiguration controller

the OPB bus. The DDR memory is used to store partial bitstreams that can be used to reconfigure at run-time the reconfigurable device. In order to perform a reconfiguration process, the reconfiguration controller has to read the desired bitstream from the memory and to pass it to the ICAP interface, connected to the ICAP component, that will take care of the physical reconfiguration process. The reconfiguration controller is aware of both the current configuration of the reconfigurable NoC (routing tables, topology and express lines) and the current communication requirements, such as the cores that have to communicate and the required bandwidth and latency. In this way, the controller is able to adapt the underlying communication infrastructure in order to satisfy communication requirements, even when they vary at run-time.

As previously hinted, the proposed reconfigurable NoC can be dynamically adapted to the current operating scenario by modifying network interfaces routing tables at run-time, as described in Section IV-A. Furthermore, a dynamic change into the proposed reconfigurable NoC can also involve either the connection among switches (by inserting or removing express lines) or the whole topology, as described in Section IV-B.

A. Path reconfiguration

Storing routing tables in BRAM blocks allows to dynamically change them at run-time in two different ways. The first solution is to write the new routing table with a simple *write operation* on the selected BRAM block. This write operation can be performed by the reconfiguration controller, that has to manage both the physical reconfiguration and the modification of BRAMs content, since routing tables have to be always consistent with respect to the current topology of the network. Using the reconfiguration controller for writing on BRAM blocks makes it necessary to directly connect it to each BRAM block, increasing the complexity and the area usage of the reconfigurable system.

A second solution consists of performing a partial dynamic reconfiguration of BRAM blocks, as described in [12]. This reconfiguration has to be performed by the reconfiguration controller, but in this case there is no need to directly connect it to each BRAM block, since these elements are updated by the controller using the configuration memory; thus, no area overhead is introduced. Performing this kind of reconfiguration enables dynamically changing BRAM blocks content (routing tables), in order to change the functionality of the network interfaces at run-time, while leaving unaltered all the logic implementing the functionality of the system; this allows a complete decoupling between routing tables and the logic that implements both the static and the reconfigurable components. The main drawback of

this solution is the increment of the time overhead of the network reconfigurations, as stated in Section V.

Finally, the proposed solutions can also be exploited in parallel in the same system, in order to considerably increase its flexibility, even if its complexity will be increased too.

B. Express lines and topology reconfiguration

In order to exploit express lines reconfiguration, it is necessary to define a reconfigurable architecture that consists of several reconfigurable parts, in which it is possible to deploy the switches of the NoC. This can be done by means of the Early Access Partial Reconfiguration design flow [8] defined by Xilinx. This flow allows to implement a reconfigurable architecture containing an arbitrary set of reconfigurable regions (which shape is a rectangle spanning the whole height of the reconfigurable device, for FPGA of Virtex, Virtex II and Virtex II Pro families, or an arbitrary rectangle for FPGA of Virtex IV and Virtex V families). Both the static architecture and each reconfigurable module, which need to be placed in a single reconfigurable region, can be configured on the target device by using a specific bitstream, namely, a complete bitstream for the static part and a partial bitstreams for the reconfigurable modules. All the bitstreams generated by this flow are the ones used by the previously described reconfiguration controller to change the current configuration of the system; in other words, the reconfiguration controller is able to select a partial bitstream to be configured on the device in order to change the underlying communication infrastructure. In particular, if an express line has to be placed between to switches that belong to the same reconfigurable region, the reconfiguration controller has to configure a new version of the reconfigurable region in which the two switches are directly connected (through a new connection). A similar procedure can be applied to completely change the topology of the NoC. In this case, a deeper modification of the selected reconfigurable part is needed, in order to make it possible to change the number and the kind of the switches of the same reconfigurable part (and thus of the whole NoC).

The number of express lines that can be established between two reconfigurable regions has to be decided at design-time, since each bus-macro (which enables to establish a single reliable communication channel among different regions) has to be placed during the place and route phase of the architecture. Furthermore, the maximum number of express lines, which is always in the order of tens for FPGA of Virtex II, Virtex II Pro, Virtex IV and Virtex V families, is limited by the amount of available resources along the edge among reconfigurable and static regions; hence, it strictly depends both on the target reconfigurable device and on the shape of each reconfigurable or static region.

V. Experimental results

This section presents a set of experimental results that validate the performance of the proposed reconfigurable architecture. These results have been achieved by implementing the proposed reconfigurable architectures on a Xilinx Virtex II Pro (XC2VP20) device. However, the same approach can be easily adapted to another device, even in a different family, such as Virtex IV and Virtex V.

A. Routing tables reconfiguration analysis

Regarding **routing tables reconfiguration**, it can be performed in a few clock cycles if it is performed with a simple write operation. In particular, if routing tables reconfiguration is performed directly by the reconfiguration controller, the latency of the reconfiguration is only 2 clock cycles (0.02 μs at 100 MHz). On the other hand, by performing a partial dynamic configuration of BRAMs, even if both the area and the complexity overheads are not increased, the latency of the reconfiguration is considerably higher (2.24 ms at 100 MHz).

B. Express lines and topology reconfiguration analysis

Even if **express lines reconfiguration** and **topology reconfiguration** can be used in order to achieve different modifications of the underlying network, from the timing overhead point of view, they are characterized by the same values, because the time required to reconfigure a reconfigurable region is exactly the same in both cases. Since the reconfiguration on Xilinx Virtex II Pro devices can only be performed with a 1D approach, the reconfiguration latency is directly related to the width of the reconfigurable region that has to be reconfigured. For instance the reconfiguration latency for a 4 slices width region, which can be filled with up to two switches, is around 21 ms, while a 20 slices width region, which can include up to ten switches, requires around 104 ms, as shown in Table I,

TABLE I
EXPRESS LINES AND TOPOLOGY RECONFIGURATION RESULTS

Width of the reconfigurable slot (slices)	Reconfiguration latency (*ms*)	Bitstream size (Kb)
4	21	32
6	30	46
8	41	62
10	52	78
12	63	94
14	75	112
16	80	120
18	93	140
20	104	156

In particular, for what concerns **express lines reconfiguration**, it can be exploited both to reduce the traffic on a part of the NoC and to decrease the latency between two switches. In order to better explain how it is possible to dynamically configure express lines on the proposed reconfigurable architecture, let us consider a simple 3x3 mesh network, similar to the one presented in Figure 4 (A). Without any express line, if the MicroBlaze 0 (MB 0) has to communicate with Slave 4, 3 hops (a path to a destination on a network can be considered as a series of hops, through switches) are necessary in order to go from Switch 0 (to which MB 0 is connected through an initiator network interface) to Switch 5 (to which Slave 4 is connected through a target network interface). To this end, each packet has to pass, for instance, through Switch 1 and Switch 2, in order to reach its final destination. In a similar way, the communication between PPC 0 and Slave 3 requires at least 2 hops (between Switch 6 and Switch 8), since each packet has also to pass through Switch 7.

In the proposed reconfigurable architecture, it is possible to configure a direct connection between the port 3 of Switch 0 and the port 4 of Switch 5, and another one between the port 4 of Switch 6 and the port 3 of Switch 8. In this way, in addition to considerably reduce the congestion of Switches 1, 2 and 7, each communication between MB 0 and Slave 4 or PPC 0 and Slave 3 can be achieved with a single hop (from Switch 0 to Switch 4 and from Switch 6 and Switch 8), thus notably reducing the latency between these elements. The number of express lines that have to cross static parts has to be defined at design time (since the involved static parts have to be aware of them), while the number of express lines that lies within a single reconfigurable region only depends on the available resources of the selected region.

Since communication among the elements of the system can change at run-time in a non-predictable way, it is possible that the system reaches a status (for instance when the applications running on MB 0 and on PPC 0 change) in which MB 0 has to communicate with Slave 3 and PPC 0 has to communicate with Slave 4. With

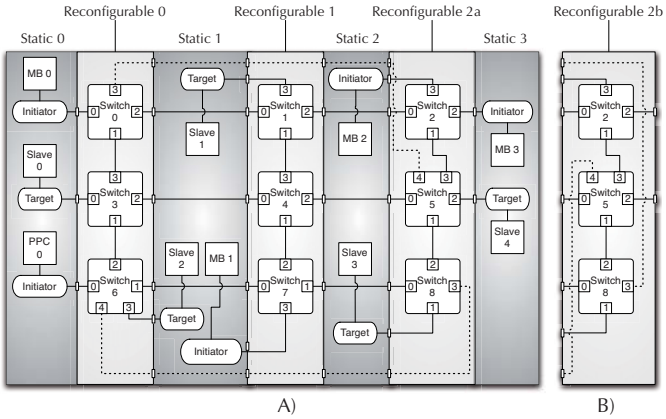

Fig. 4. Complete reconfigurable system schema, with an express line between Switch 0 and Switch 5 and another one between Switch 6 and Switch 8 (A), and the Reconfigurable region 2 with an express line between Switch 0 and Switch 8 and another one between Switch 6 and Switch 5 (B)

the configuration of Figure 4 (A), each master can reach the desired slave, by using both express lines, with 2 hops (from Switch 0 to Switch 8 and from Switch 6 to Switch 5). However, a problem that can arise is that these two paths share the link between the port 1 of Switch 5 and the port 2 of Switch 8, thus leading to a contention of the same resource. A possible solution is the partial dynamic reconfiguration of the reconfigurable region number 2, in order to achieve the configuration of the system shown in Figure 4 (B), which can be achieved by adapting the routing tables according to the new configuration of the system, as described in Section IV-A). In this way, not only the congestion of the link between Switch 5 and Switch 8 is completely resolved, but also the latency of the two communication paths decreases to a single hop, *i.e.*, providing a circuit-based switching connection.

An important consideration is that, while the partial reconfiguration of the reconfigurable region 2 is performed, the communication among other parts of the system does not need to be interrupted, as long as it does not affect the region that is reconfigured. For instance, if MB 1 has to communicate with Slave 0 or Slave 1, this communication can take place even during the reconfiguration of the reconfigurable region 2.

The physical implementation of the previously presented architecture is shown in Figure 5, where A indicates the static part, while B, C and D represent the three reconfigurable regions (which width is, respectively, 16, 20, and 14 slices - the 18%, 22% and 16% of a XC2VP20 device). All the reconfigurable regions have been filled with three switches each one, in order to implement the previously presented 3x3 mesh.

Table II shows the experimental results regarding area usage and reconfiguration latency of the proposed architecture on a XC2VP20 device. The bus-macro overhead consists of 288 slices, while the complete 3x3 mesh requires 2237 slices. Thus, the overhead introduced by the proposed approach represents the 10% (on average) of the initial NoC.

Furthermore, it is possible to configure at least two express lines in the implemented architecture, and since each express line of the presented design has a latency lower than 4 ns, it is possible to exploit each direct connection within a single clock cycle at 100 MHz (while the latency required by the connection passing through Switch 1, Switch 2 and Switch 5 is greater then 40 ns, *i.e.*, 4 clock cycles).

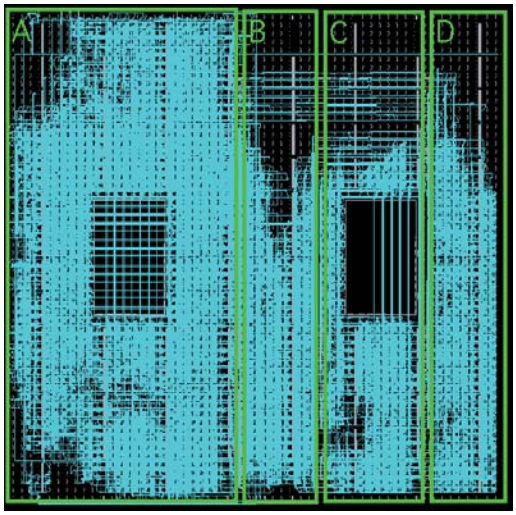

Fig. 5. Physical implementation of the reconfigurable 3x3 mesh

TABLE II
AREA USAGE AND RECONFIGURATION LATENCY RESULTS

	Area usage (slices)	**Area usage (%)**	**Reconfiguration latency (*ms*)**
Reconfigurable region B	800	8.6	80
Reconfigurable region C	637	6.9	104
Reconfigurable region D	800	8.6	75
Complete 3x3 mesh	2237	24.1	259
Bus-macro overhead	288	3.1	

On the other hand, a **topology reconfiguration** can be exploited on the same architecture in order to adopt a specific NoC for each application that has to be run on the system. In this case, we have validated the application with our approach with three different versions of real-life SoC benchmarks, namely, a video processing application of 32 cores (A), a Video Object Plane Decoder of 34 cores (B) and an image processing application of 23 cores (C). We refer the readers to [4] for the communication characteristics of these benchmarks. As shown in Figure 6, if these different applications have to deployed on the same system, it is possible to employ either a static network or three specific NoCs, each one designed ad-hoc for the particular application. The second choice can be adopted if the time interval that occurs between two consecutive applications is greater than the time overhead required by the reconfiguration process; thus, it is possible to transparently change the underlying NoC.

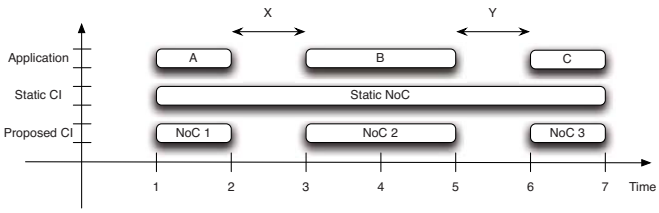

Fig. 6. Temporal evolution of a generic system

In order to test the application of our dynamically reconfigurable

framework in this context, we have developed a static NoC and three specific ones for each of the three aforementioned SoC benchmarks application. The static NoC consists of 6 switches (1 switch of 8x8, 2 switches of 9x9, 2 switches 10x10 and 1 switch of 11x11), while both NoC 1 (for application A) and NoC 3 (for application C) consists of 4 switches (3 switches of 10x10 and 1 switch of 11x11) and NoC 2 (for application B) consists of 4 switches (1 switch of 10x9, 2 switches of 10x10 and 1 switch of 10x11). As shown in Table III, the static NoC option is characterized by a higher area usage, a higher average power consumption (evaluated as proposed in [2]) and a higher average latency, with respect to the three ad-hoc NoCs specifically designed for each application. Using the specific NoCs, it can be reported reductions of 34% in latency and 24% in power consumption. Finally, the overall latency for the reconfiguration of the NoC to be used at run-time is very limited, making it applicable in real-life scenarios where applications are switched dynamically by users.

TABLE III
SPECIFIC NOCS EXPERIMENTAL RESULTS

NoC	Number of switches	Average latency (clock cycles)	Average power consumption (mW)
Static NoC	6	5.96	278.021
NoC 1	4	3.9	211.789
NoC 2	4	4	204.308
NoC 3	4	4.07	216.519

Finally, table IV presents a comparison among state-of-the-art solutions and our approach, which shows clear benefits regarding area overhead reduction, timing performance improvements and enhancements of reconfiguration features.

TABLE IV
AREA OVERHEAD, TIMING PERFORMANCE AND FEATURES COMPARISON AMONG STATE OF THE ART SOLUTIONS AND THE PROPOSED APPROACH

Approach	CuNoC ([9])	CoNoChi ([14])	Proposed work
Switch size (slices)	from 72 to 491	from 363 to 493	from 86 to 267
Communication infrastructure size (slices)	All the available resources	NA	2727 for a 3x3 mesh
Frequency (MHz)	from 272 to 336	from 66 to 111	170
Single switch latency (clock cycles)	2	5	1
Single switch latency (*ns*)	from 6 to 7.4	from 45 to 76	5.9
Flow control	NA	NA	Supported
Path reconfiguration	Not supported	NA	Supported
Express lines reconfiguration	Not supported	Not supported	Supported
Topology reconfiguration	Not supported	Supported	Supported

VI. CONCLUSIONS

NoCs have been proposed as a very promising scalable communication paradigm SoCs. However, methods that provide versatility enhancements of initial NoC designs to changing working conditions, imposed by variable sets of executed applications at run-time, are key to design competitive NoCs in industrial SoCs. In this work we have presented a novel NoC reconfigurable framework that can reconfigure the NoC topology at run-time, as well as enabling path reconfiguration and express lines creation/removal, while introducing an overhead on average of 10% of an initial static NoC design. Moreover, our experimental results have shown that in the proposed framework, on average, a reconfigurable switch only occupies 41% of the slices needed by a CoNoChi switch, the state-of-the-art reconfigurable NoC approach, whereas our reconfigurable NoC can run at almost double the frequency (170 MHz *vs.* 88.5 MHz) of CoNoChi. Finally, our approach introduces less than one tenth of the latency introduced by a CoNochi switch (respectively, 5.9 ms and 60.5 ms). Thus, it is a promising framework to be applied to commercial NoC-based SoC solutions.

ACKNOWLEDGMENTS

This work was partially supported by the HiPEAC network of excellence (www.hipeac.net), the Swiss NSF Research Grant 20021-109450/1 and Spanish Government Research Grants TIN2005-5619 and CSD00C-07-20811.

REFERENCES

[1] B. Ahmad, A.T. Erdogan, and S. Khawam. Architecture of a dynamically reconfigurable noc for adaptive reconfigurable mpsoc. *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on*, pages 405–411, 15-18 June 2006.

[2] F. Angiolini, P. Meloni, S. Carta, L. Benini, and L. Raffo. Contrasting a NoC and a traditional interconnect fabric with layout awareness. In *Proceedings of Design, Automation and Test in Europe Conference (DATE'06)*, pages 124–129, Munich, Germany, 2006.

[3] Luca Benini and Giovanni De Micheli, editors. *Networks on chips: Technology and Tools*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2006.

[4] Davide Bertozzi, Antoine Jalabert, Srinivasan Murali, Rutuparna Tamhankar, Stergios Stergiou, Luca Benini, and Giovanni De Micheli. Noc synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. Parallel Distrib. Syst.*, 16(2):113–129, 2005.

[5] Gordon Brebner and Delon Levi. Networking on chip with platform fpgas. In *Proceedings of the 2003 International Conference on Field-Programmable Technology (FPT)*, pages 13–20, December 2003.

[6] D. Ching, P. Schaumont, and I. Verbauwhede. Integrated modeling and generation of a reconfigurable network-on-chip. *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 139–, 26-30 April 2004.

[7] A. Hansson and K. Goossens. Trade-offs in the configuration of a network on chip for multiple use-cases. *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, pages 233–242, 7-9 May 2007.

[8] Xilinx Inc. *Early Access Partial Reconfiguration Guide*. Xilinx Inc., 2006.

[9] S. Jovanovic, C. Tanougast, S. Weber, and C. Bobda. Cunoc: A scalable dynamic noc for dynamically reconfigurable fpgas. *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 753–756, 27-29 Aug. 2007.

[10] A. Kumar, A. Hansson, J. Huisken, and H. Corporaal. An fpga design flow for reconfigurable network-based multi-processor systems on chip. *Design, Automation and Test in Europe Conference and Exhibition, 2007. DATE '07*, pages 1–6, 16-20 April 2007.

[11] Olav Lysne, Timothy Mark Pinkston, and Jose Duato. A methodology for developing dynamic network reconfiguration processes. *icpp*, 00:77, 2003.

[12] Alessio Montone, Vincenzo Rana, Marco Domenico Santambrogio, and Donatella Sciuto. Harpe: a harvard-based processing element tailored for partial dynamic reconfigurable architectures. *22nd IEEE International Parallel and Distributed Processing Symposium - 15th Reconfigurable Architectures Workshop*, April 2008.

[13] Srinivasan Murali, Martijn Coenen, Andrei Radulescu, Kees Goossens, and Giovanni De Micheli. Mapping and configuration methods for multi-use-case networks on chips. In *Proceedings of the 2006 conference on Asia South Pacific design automation (ASP-DAC)*, pages 146–151, New York, NY, USA, 2006. ACM Press.

[14] T. Pionteck, R. Koch, and C. Albrecht. Applying partial reconfiguration to networks-on-chips. *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1–6, 28-30 Aug. 2006.

[15] A. Vicentelli and G. Martin. A vision for embedded systems: Platform-based design and software. *IEEE Design and Test - Special Issue of Computers*, 18(6):23–33, November 2001.