A Light–Weight Network–on–Chip Architecture for Dynamically Reconfigurable Systems

Simone Corbetta, Vincenzo Rana, Marco Domenico Santambrogio and Donatella Sciuto
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Milan, Italy
Email: simone.corbetta@dresd.org, {rana, santambr, sciuto}@elet.polimi.it

***Abstract*—On–chip communication design is a complex task, since the communication requirements and the complexity of the target application are high. With the introduction of dynamic reconfiguration (a feature than can be found, for instance, in recent Field Programmable Gate Arrays), the design of a reconfigurable communication infrastructure becomes a suitable approach to increase both the flexibility and the adaptability of such a system. These are two of the key features of a communication infrastructure for reconfigurable systems, since usually the designer is not aware of which will be the executing modules and the communication requirements at run–time.**

This paper introduces and describes a novel reconfigurable communication infrastructure for dynamically reconfigurable architectures. The proposed approach is a tile–based Network–on–Chip in which the communication layer is completely decoupled from the computational one. The proposed approach is designed to support dynamic reconfiguration at the communication fabrics level.

I. Introduction

Dynamic reconfiguration makes it possible for a device to change its configuration during the normal system execution. The reconfiguration process writes the new system description to the desired portion of the device[1].

Thanks to this capability, new components can be loaded or existing components can be replaced at run–time without affecting the execution of the remainder portion of the system. Nevertheless, dynamic reconfiguration introduces another level of complexity in the development phase, since specific design methodologies have to be adopted to effectively realize such a system [1], [2]. Also, due to the high communication requirements in modern embedded–systems applications [3], the design is even more complex.

This paper presents a *novel* reconfigurable communication infrastructure tailored for dynamically reconfigurable systems. The rationale is based on the fact that an ad–hoc flexible communication infrastructure is a suitable approach to cope with the issues given by dynamically changing application parameters or requirements. Furthermore, the benefits of (partial) dynamic reconfiguration [4] can be addressed in order to define a flexible and adaptive communication fabrics, that changes at run–time according to the (current) application needs. This paper provides an overview of a novel design, taking advantage from several distinctive features from different communication scheme approaches.

The remainder of this paper is organized as follows: Section II will describe the problem to contextualize the proposed approach; previous works on the area of communication infrastructure for FPGA–based systems are then explored in Section III. In Section IV we will describe the proposed communication architecture, being dynamically reconfigurable with respect to topology, routing paths and network elements; preliminary experimental results will be given in Section V, while conclusions and future works can be found in Section VI.

II. Problem Description

The arrival of the Virtex–II and Virtex–II Pro family FPGAs made it possible for a system to be partially reconfigured at run–time; actually, with these devices dynamic reconfiguration is only *column–wise* [1], where the reconfigurable area is a rectangular shape with user–defined width but spanning the entire height of the device. More recent Virtex–4 devices [5] overcome the limitations imposed by column–wise reconfiguration by extending the device capabilities to support 2D–shaped modules, spanning a predefined height of the chip. Dynamic reconfiguration capabilities have to be supported by precise design methodologies [1], [2], in order to realize the desired reconfigurable architecture.

Figure 1 presents the differences between a 2D placement constraint and a 2D reconfiguration scenario. In particular, Figure 1(a) shows a 1D placement with a 1D reconfiguration scenario. Figure 1(a)(i) satisfies the 1D–constraint, while Figure 1(a)(ii) does not. The reasons why Figure 1(a)(i) is an unfeasible constraint are motivated by the fact that the regions used to define the placement constraints for the two functionalities overlap. Figure 1(b) shows a 2D placement constraints with a 1D reconfiguration, that means that even if 2D placement constraints are allowed it is not possible to reconfigure something that has an height different from the height of the entire device. Figure 1(b)(i) shows a feasible constraint while Figure 1(b)(ii) an unfeasible one. In this scenario, as we can see, we do not overlap the area where the two functionalities have to be mapped, their placement constraints, but we are overlapping the two reconfigurable areas, which is not allowed in a 1D reconfiguration scenario.

[1]In case only an arbitrary part of the device is reconfigured, we speak of *partial reconfiguration*.

978-1-4244-1985-2/08/$25.00 ©2008 IEEE

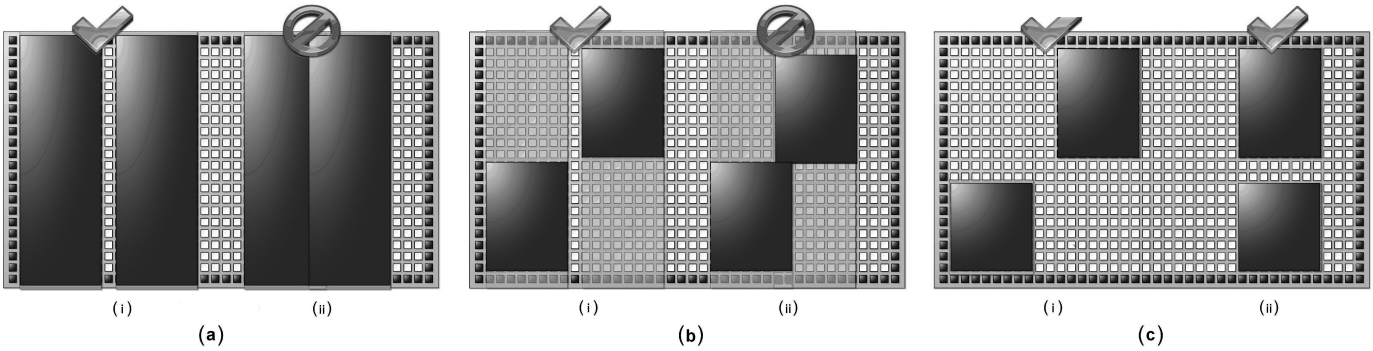

Fig. 1. 1D/2D placement constraints combined with 1D/2D partial reconfiguration

Finally, Figure 1(c) presents a 2D placement constraints with a 2D reconfiguration. In such a context both the cases, Figure 1(c)(i) and Figure 1(c)(ii), can be considered valid.

As previously described, with the newest Xilinx devices such as Virtex-4 and Virtex-5 FPGAs it is possible to configure rectangular regions providing the designers with great flexibility. Moreover the dynamic nature of the applications and of the actual modules loaded onto the device affect the definition of the *correct* communication infrastructure with respect to the design constraints and required performances. During the reconfiguration task, the communication infrastructure must guarantee that all the interconnections along the boundaries of the reconfigurable regions are preserved, otherwise the system will not be able to exploit the desired functionality.

2D reconfiguration can be exploited thanks to the *Early Access Partial Reconfiguration* (EAPR) design flow, for the design and realization of dynamically reconfigurable systems. EAPR [2] provides an extension to the previously widely–used module–based approach to dynamic partial reconfiguration [1]. The EAPR flow expedites the definition and the development of truly 2D–reconfigurable systems that consist of a static and a reconfigurable part, but not without a drawback: the loss of the relocation capability. As we will see, the definition of an ad–hoc communication infrastructure can solve this problem.

III. Related Works

This section presents previous works made on communication infrastructures for dynamically reconfigurable architectures. The approaches can be grouped in two classes: bus–based and network–based. The first class includes all those systems in which a bus is shared and used as the main communication medium. A network–based approach, on the contrary, is based on a distributed set of network elements responsible of forwarding the data; the interconnections among these components can be customized to realize the desired topology. The use of a bus–based system or a network–based infrastructure is driven by the application requirements and by the required performances and Quality–of–Service levels that have to be ensured.

In literature, the most relevant attempts to realize a bus–based system for reconfigurable architectures are RMBoC [6], [7] and BUS–COM [8]; the former (*Reconfigurable Multiple Bus–on–Chip*) has bus segments connecting hop–by–hop the modules interface, while the latter uses a centralized set of buses with an arbiter granting access to the communication medium in time–division multiplexing. The two approaches are reported in Figure 2. The single bus in BUS–COM (Figure

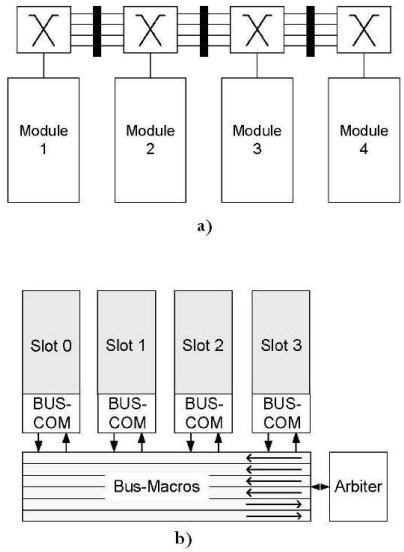

Fig. 2. Bus–based infrastructures: (a) RMBoC, and (b) BUS–COM

2 (b)) is divided into a set of different, parallel channels, and the access to them is given by the arbiter, according to a static or dynamic slot assignment. A static slot assignment provides a simple and deterministic way to manage concurrent accesses to the shared resource, while a dynamic slot assignment allows for a greater flexibility, granting access based on a predefined priority level. In RMBoC (Figure 2 (a)), meanwhile, the level of contention increases as the number of cores (called *modules*) is high.

Actually, both the approaches lack of flexibility and scalability: the presence of a (shared or segmented) bus limits the number of attacching modules: with the increasing number of cores, the level of contention for the shared resource increases and so does the waiting time for each module, affecting the end–user perspective of the application. In a system with an

high density of modules this would cause a performance loss or, in the worst case, a Denial–of–Service (DoS). Flexbility is further limited by the static topology of the communication infrastructure, since there is no possibility of changing the way modules are interconnected each other. Also, neither RMBoC nor BUS–COM can be applied to effective dynamic reconfiguration; the static architecture limits the adaptability of the communication scheme to the unpredictable changing scenarios, as we require.

On the other hand, network–based interconnects rely on the use of switching or routing elements to exchange information; this is done in a packet–switched nature, instead of hardwired circuit connections. The Network–on–Chip approach has been proposed in [9] as a novel design paradigm for System–on–Chip applications. The idea is to borrow concepts and techniques from well know data networks and apply them to the on–chip context.

XPipes has been proposed as the first real implementation of a NoC with the purpose of supporting on–chip multi–processors interconnections [10]. It is based on a stacked communication protocol: the Smart Stack. It has three layers. The *data–link layer* is used to achieve a predefined level of tolerance, assuming that the underlying communication medium has non–zero probability of error. The *network layer* is used as an end–to–end control for packet routing, while the information is encapsulated in the *transport layer*.

A single basic element, the *switch*, is used to forward the information to the required destination. The routing path is determined statically, and each destination element in the architecture has one predefined routing path, from each sender.

Since XPipes is intended to be adaptable (at synthesis time) to the application requirements, *XPipesCompiler* is used to automatically generate the HDL design files with an instance of the specific network, given the user–defined parameters. The XPipes Library (see Figure 3 for the XPipesCompiler

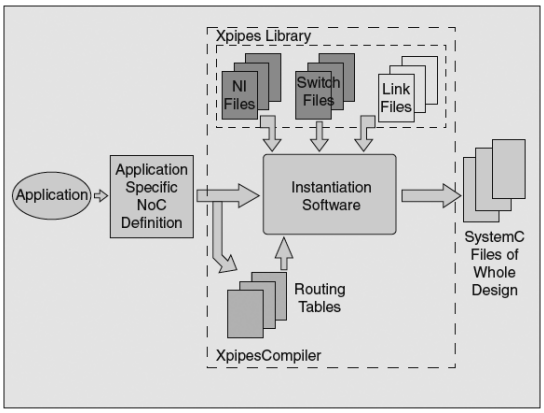

Fig. 3. XPipes synthesis framework [10]

framework) contains a set of switch and network–interface descriptions that can be (re)used. The instantiation software takes as input the user–defined parameters, the required soft–macros and the routing tables to generate a set of System–C design files that represent the network.

The limits imposed by XPipes not supporting dynamic reconfiguration have been overcome in CoNoChi (*Configurable Network–on–Chip*) [11]. CoNoChi is realized by directly applying partial dynamic reconfiguration to the communication level of the SoC. Aim of CoNoChi is to to be able to support dynamic changes to the communication topology, by adding (removing) new (existing) cores. As in XPipes, the basic element of CoNoChi is a switch. A single network element is then used to generate the desired topolgy; two or more modules can be connected directly or through a non–standard interface that allows for logical addressing mode of the cores. This last feature is very useful in cases we want to realize dynamically reconfigurable architectures: if a processing element is physically replaced in the device, that module would be reachable at the cost of a small computational overhead, due to the mapping of the logical address to the physical one.

Table I relates to a qualitative comparison among the approaches presented so far. The purpose of the comparison is to highlight the main advantages and drawbacks of the specific communication infrastructure, with respect to a predefined metric. The first column presents a list of metrics that are the basis of the comparison among the different approaches; for each of them, each of the four architectures has been analysed and the corresponding metric has been evaluated.

We can see from Table I the main advantages of the different approaches. We can group the four approaches in two groups, without loss of generality: bus versus network implementations. The main relevant entries are reported in the second half of the table. Flexibility is achieved in a Network–on–Chip due to its structural implementation using a single network element, the switch. In a bus infrastructure, indeed, the shared communication medium is statically allocated and the topology of the entire communication is inherently static, it cannot be changed. Thus, as well as with the increasing number of attaching modules the bus becomes a bottleneck, a bus–based infrastructure is not suitable for complex environments. On the contrary, the NoC approach guarantees scalability and flexibility due to the use of a packet–switched paradigm instead of a static circuit one. Also, thanks to dynamic reconfiguration capabilities, CoNoChi ensures an higher level of flexibility because of the possibility to change the topology at run–time, as application requires. Last, reliability depends deeply on the design paradigm; a bus is a single point–of–failure, and this means that such a communication infrastructure is not reliable. On the contrary, a network ensures high reliabilty due to the presence of different paths from the sender to the destination end–point of the communication: the data forwarding is distributed, and not centralized.

IV. A Light–Weight Network–on–Chip for Reconfigurable Architectures

Aim of the work presented in this paper is the definition of a suitable communication infrastructure for dynamically

TABLE I
QUALITATIVE COMPARISON AMONG THE DIFFERENT COMMUNICATION INFRASTRUCTURE

	RMBoC	**BUS–COM**	**XPipes**	**CoNoChi**
Type	Bus–based		Network–on–Chip	
Shared Medium Access	Access is granted by an arbiter		Access is concurrent	
Parallelism Achieved	Low, due to the bus	Low, but higher than RMBoC due to multiple buses	High	High
Connectivity	High, ensured by a shared channel		Depends on the topology of the network, at synthesis–time	Depends on topology, at run–time
Resource Requirements	Low/Medium	Low/Medium	Depends on switch implementation	
Flexibility	Low, a bus is a bottleneck	Low	High	Very high, due to run–time reconfiguration
Scalability	Low	Low	High	High
Reusability	Medium	Medium	Depends on switch design	Depends on switch design
Reliability	Low, bus is a single point–of–failure	Low	High, packet–switched network	High

reconfigurable architectures. The rationale of defining such a novel communication infrastructure is the need to be able to cope with the dynamic changes of application requirements, in order to achieve flexibility and adaptability of the entire system, to provide the end–user with a valid application framework. This section introduces the novel features that the proposed approach exploits, as long as an example application of the solution.

The communication infrastructure will be deployed on an ad–hoc reconfigurable architecture. The architectural model we use is composed of two parts: a static side and a reconfigurable one. The static part contains those components that will be always required by the application during its life–cycle; for example we may have a processor and basic I/O interfaces to the external environment. The reconfigurable part, on the other hand, will be composed of those regions (portions of the device) that will contain the modules that could be loaded or removed after the system has been deployed, at execution–time. Furthermore, in order to maintain a persistent communication among the static part and the reconfigurable modules, a special hardware macro is employed: bus macros will be used to route signals along the boundaries of the reconfigurable modules. An example of such architecture is reported in Figure 4.

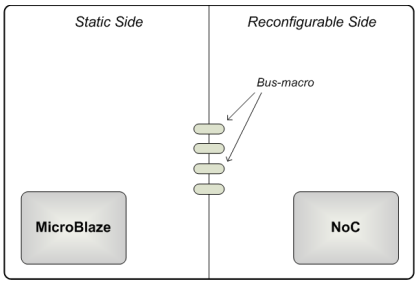

Fig. 4. Reconfigurable architecture model

A. A Network–based Approach

The proposed approach is based on a networked view of the system: the communication infrastructure will then be based on the use of basic network elements and components. The network, also, will be packet–switched. The use of packet–switching instead of circuit, hardwired connections present several advantages, as expressed next [12]:

- the interface design complexity of the cores can be reduced, resulting in *light–weight* interfaces;
- the possibility to tailor the communication protocol gives the flexibilty to extend the design and to define a suitable application–driven communication scheme;
- networks enable the definition of fault–tolerant interconnection schemes, both at wiring level and at logical (protocol) level;
- the wires used to establish the connections among the modules can be easily reused to route different signals, in parallel, so that parallelism and efficiency are increased.

As we can see, a packet–switched network overcomes the flexibility and scalability limitations imposed by classical point–to–point or bus–based systems.

The entire interconnection scheme is based on a single element: the *switch*. A switch can be seen, at high–level, as a black–box forwarding the incoming packets toward the desired destination point, by enabling the appropriate output–port. The sequence of switches that a data packet has to pass through before getting to the desired destination end–point is the routing path. Each switch is composed of a set of basic I/O interfaces and computational elements, used to buffer the incoming requests and to forward the data packets. The main features of the switch will be reported in the next sections.

The systematic interconnection of several switches defines the connectivity of the network, and the structure of this interconnection is the topology. The network has to guarantee a predefined level of connectivity among each sender and each receiver, and this can be done statically or dynamically by appropriately choosing the best fitting topology for the application needs.

B. The Need of a Light–Weight Infrastructure

The infrastructure we are hereby proposing is based on a *light–weight* Network–on–Chip architecture. Being light–weight for an architecture is a real requirement, since the

implementation has to take into account several constraints: device resource utilization, performance, power consumption, reliability and efficiency. A light–weight approach is necessary to cope with strict physical constraints and application requirements. As a matter of fact, the communication infrastructure has to guarantee a predefined level of QoS, and this has to be done in a cost–driven environment. A light–weight architecture, as defined in the next sections, is able to guarantee a valid yet efficient communication backbone for the (current) application.

Several are the aspects that define an infrastructure as being *light–weight*. The most meaningful aspects follow:

- interface design;
- communication among switches and cores (protocol);
- switching and routing component;
- inter–switch connection;
- reconfiguration task.

The interface design relies directly on the protocol definition. The protocol plays an important role in the entire communnication infrastructure, since it defines the way messages are exchanged, and it defines the way messages are acknowledged. Furthermore, it directly impacts on the performance of the system, for example setting up the power consumption and wire length requirements. A light–weight protocol should guarantee complete information, but in an efficient way. The interface design is directly affected by the protocol scheme in that the encoding and decoding logic might be more or less complex. Complexity in the interface means an higher time–consuming task by the cores, and this is reflected in the user perspective of the system performance.

A light–weight switching (or routing) component, ineed, is necessary to guarantee the *minimal* required interconnection logic to forward the packets from the sender to the destination end–point. This avoids high computational overhead and also avoids resource usage overhead. Also, the connections among the switches[2] are relevant. In this case the logical links have to guarantee reliable connections and, at the same time, low wiring.

The aforementioned aspects are application–dependent, because of the high heterogeneity of modern on–chip applications, and for this reason a dynamically reconfigurable communication infrastructure is useful; as a matter of fact, the possibility to change the properties of the basic components of the network allows us to tailor the communication scheme as required by the *current* application at execution–time, i.e. without any need neither of restarting the system nor of using additional physical devices: the same area (the same chip) can now be reused to host different applications at run–time, lowering the costs of the design and improving the area usage efficiency.

The use of dynamic reconfiguration adds flexibility to the system, in that the network parameters can be changed at run–time depending on the application work–load and requirements. For example, a new switching or a new routing element could be loaded at run–time, provided that the communication among the cores is such that a new component is required, with different parameters; further, the topology can be changed by adding (or removing, if required) a new (an existing) switch. In both cases the reconfiguration process must not prevent the remainder of the communication to continue, and also the reconfiguration process should not require a large time overhead. For these purposes and in this context, a light–weight approach (as previously defined) is necessary; since the reconfiguration time is linear with the bitstream size and the bitstream size is tied to the dimension (in terms of physical resources in the FPGA) of the component to be reconfigured, a light–weight network is a suitable approach for dynamically reconfigurable systems and architectures.

The approach we propose is intended to be both light–weight, in the sense previously described, and dynamically reconfigurable.

C. A Layered Approach

The proposed solution is based on a layered approach to the network design: on one side the computational elements, while on the other side the actual communication. In this way, we are able to decouple the logic used to assemble and disassemble the user–space level data into (from) packets, from the actual data forwarding. While the former task is performed by the network interfaces connecting the module end–point to the switch, the latter is inherently ensured by the switches interconnections. The independent design of communication and computation of the same system allows for independent optimization of the two aspects, thus resulting in a better performance featured architecture.

Also, decoupling two aspects allows us to define a simple, light–weight interface toward the network and, furthermore, to clearly define the boundaries of the reconfigurable and static sides of the architecture. As a matter of fact, in the architectural model previously defined, the communication is entirely given in the reconfigurable side of the system. This feature will be used to demonstrate – through the experiment reported in Section V – the dynamic adaptability of the network.

D. A Reliable Communication

In Section III we have seen how the design of the communication infrastructure is relevant to the reliability of the target system. The proposed approach tries to ensure a reliable communication among the cores, at any time instant of its life–cycle. The reliabilty in the proposed approach is achieved thanks to two aspects:

- a persistent communication among the switches of the network;
- a redundant communication path among each communicating core.

The first aspect addresses a lower–level communication fabrics that interconnects each switch within the system. The idea

[2]Notice that in this case we are not considering the structure of the interconnections, defining the topology, but the real physical (and logical) properties of these links.

is to ensure a persistent communication among these network elements in order to gain a certain level of control of the communication. This persistent interconnection is minimalist with respect to resource requirements, since only low–width lines are used to exchange control information. For example, a single–bit line among each switch could be used to keep the system aware of the local conjestions of the switches. Suppose for example that a switch is subject to an high work–load; also, imagine that the network topology is able to ensure load balancing. Such a control line is doubly useful, since *i)* it avoids the use of a (time and power consuming) centralized system control and *ii)* it can be used to inform the other switches (and, possibly, the central processor) of the conjestion situation. In this way the topology or some of the interconnections could be changed at run–time in order to find a new configuration for the network.

A redundant communication path, indeed, is inherently ensured by the network approach between pairs of cores. Since we use a packet–switched approach and the interconnection of the switches can be tailored, both at synthesis and run–time, the path from a given sender to a given destination is not unique. In this way, if a link of the path is broken, another path is available to us, and the connection is held. This is a redundant path, but it does not impact on the resource requirements, since different path segments can be reused to interconnect different cores. The routing path can be chosen at run–time, either by dynamically reconfiguring routing tables or by providing network nodes with appropriate additional features. The first approach – as explained in Section IV-E – is simpler from the switch design point–of–view, but it requires dynamic reconfiguration decreasing timing performance. On the other hand, the latter approach requires additional features to the switch, increasing resource requirements.

E. Dynamic Features

The routing scheme is simple, based on a sequence of ports to be followed to reach the destination point. The issue is in the choice of the location where to store the routing path information. Two are the possible approaches: a LUT–based storage and a memory–based storage. The choice has to deal with the design requirements of reliable communication and dynamic topology.

With a LUT–based approach, the routing path information is hard–coded in the component, directly in the HDL code; in the second case, on the other hand, the information is kept in the BRAM blocks of the FPGA. The former approach lacks of flexibility and efficiency, since once the component is synthesized the routing path is *embedded* in the LUT equations, and a change of this information would mean to reconfigure the entire component; a more flexible approach is given in the latter case, in which the information contained in the BRAMs can be easily reconfigured without interfering with the logic of the component. In this way, the routing information is completely decoupled from the actual routing logic. This can be easily done in two ways: either using the microprocessor in the architecture to directly change the content of the memory blocks, or by reconfiguring the BRAMs with a new configuration bitfile. At this point, the two approaches should be explored with respect to efficiency, performance, and design convenience.

Reconfiguring at execution–time the content of the memory enables the network to define the routing path, from a given sender to a given destination, at run–time: dynamic routing can be performed. In this way, the network ensures flexibility of the communication scheme, and the requirements of reliable communication are ensured; furthermore, fault–tolerant communication can be realized, by means of controlling the traffic of the packets and changing the routing path when needed by the application.

Since the network is defined as an interconnection of several switches, the dynamic capabilities exploited by the architecture defined in the previous section allows to dynamically change the topology of the entire network. The example reported in the next section reports how the topology can be changed by simply loading at run–time the new configuration, i.e. the new network (seen as a black–box).

A communication infrastructure for reconfigurable architectures should support addressing of the cores, independentely on their physical location. In architectures supporting dynamic core relocation, this is an hard requirement since at any instant the module could be placed into a (new) different position. One possible approach is to define how a core could be accessed by means of its *logical address*. Logical address masks the physical effective placement of the module by using a mapping mechanism, each entry corresponding to a specific physical placement. When the core is relocated, the only information to be changed is the physical location. Using a centralized map, relocation is performed in a transparent way with respect to the end–user, i.e. the sender of the message.

F. A Light–Weight Communication Protocol

In a packet–switched network, a protocol is required to manage the information flow among the cores. The protocol definition is a relevant task to the final network performance, since the protocol has impact on different aspects of the system:

- the packet structure;
- the communication performance;
- the sytem performance, e.g. the effective throughput of the system;
- the design of the network interfaces.

A simple protocol should be realized, in order to achieve the desired performances. This will then be used to define the required packet structure to carry the information along the network.

V. Preliminary Results

This section reports some results obtained while exploring the network capabilities of the proposed architecture on a Xilinx Virtex–II Pro FPGA device. An example of topology reconfiguration will be explained, and the resource requirements

of the two networks used in the experiment are reported with respect to several different Xilinx FPGA devices.

A. Experiment Description

The purpose of the experiment is to demonstrate how to use a Network–on–Chip approach to design a reconfigurable communication infrastructure for dynamically reconfigurable architectures. The network that will be used is dynamically reconfigurable, in that its topology can be changed during execution time without losing the connectivity of the system modules.

The experiment setup follows. A reconfigurable architecture has been realized in order to support dynamic reconfiguration of the NoC, as described in Section IV; the target device is a Virtex–II Pro FPGA from Xilinx, hosted on a Virtex–II Pro Development Board (VP20) from Avnet. The chosen FPGA has enough resources for our entire demonstrating application, and it supports dynamic reconfiguration. The static side contains those components that are never reconfigured during the system execution; a MicroBlaze soft–core microprocessor is used to execute the software. The processor is connected through the On–Chip Peripheral Bus (OPB) to other peripherals. The entire application requires that an *initiator* sends (receives) some data to (from) the destination component, known as the *target*. The interfaces of both the initiator and the target have been used to connect to the NoC; for this reason, they are part of the boundaries among the static and the reconfigurable side of the architecture, and their signals will be routed through the use of bus–macros [2].

The two different topologies that are loaded (in two different time instants) onto the device are reported in Figure 5. Figure 5 (a) relates to the starting situation, while in Figure 5 (b) the network that will be loaded at run–time is reported. The

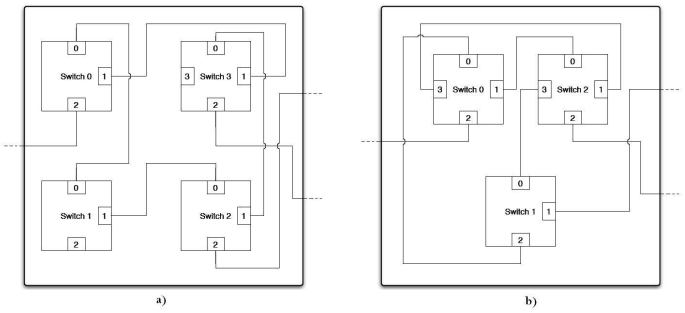

Fig. 5. (a) initial topology and (b) dynamically loaded new topology

dotted lines crossing the boundaries of the network represent the signal connections with the sender (left–hand side of the box) and the destination points (right–hand side). The simple software application executed on the MicroBlaze processor will read and write integer data from and to specific addresses, corresponding to a predefined peripheral. The output of the computation is read from the serial port that the development board provides. The given output is the proof that the data is sent throughout the network and the data is correctly written and read.

TABLE II
BITSTREAM SIZE. (*) THE BITSTREAM INCLUDES BOTH HARDWARE AND SOFTWARE.

Bitstream	Type	Size
Static side and `noc_0` (*)	Complete	1.003 KB
Static side and `noc_1` (*)	Complete	1.003 KB
`noc_0`	Partial	227 KB
`noc_1`	Partial	227 KB

TABLE III
RECONFIGURATION TIME OF THE NETWORK.

Starting configuration	Final configuration	Reconfiguration Time
`noc_0`	`noc_1`	$222ms$
`noc_1`	`noc_0`	$222ms$

During the normal execution, the NoC has been reconfigured with the topology shown in Figure 5 (b). Notice that during the reconfiguration process the connections crossing the boundaries of the reconfigurable modules are persistent. After the NoC has been reconfigured, the software run by the microprocessor starts again, and the computation gives us the expected results.

Table II reports the size of the bitstreams (both total and partial), and Table III shows the reconfiguration time required to reconfigure the predefined portion of the device.

B. Area Requirements

Table IV reports the area requirements of the two network topologies for different Xilinx FPGA devices. The network refered to as `noc_0` is made up by 4 switches, while the `noc_1` is composed of 3 switches, as depicted in Figure 5. Different Xilinx FPGA devices have been chosen, in order to give a quantitative comparison. Aside from the least recent Spartan–3 FPGA, the area occupation of the network is acceptable, if we think that the switches realize the entire communication scheme.

Figure 6 represents the implementation of the entire reconfigurable architecture with the topology reported in Figure 5 (a). On the left–hand side we can see the static part of the system, comprising the MicroBlaze microprocessor; the reconfigurable part is on the right–hand side. After the reconfiguration process, the new topology is loaded onto the device, and the situation is as in Figure 7.

VI. CONCLUSIONS AND FUTURE WORKS

The use of the Network–on–Chip paradigm in the design of communication infrastructure for dynamically reconfigurable architectures has been shown as a suitable approach to guarantee flexibility and adaptability to the run–time application changes. The benefits of partial reconfiguration [4] show the validity of such an approach. This support to the communication layer allows the topology of the network to be adapted, according for example to the network traffic status and (local) conjestion. The routing mechanism has been done dynamic by directly reconfiguring the content of the BRAM blocks, storing the information about the routing path from the sender to the receiver end–points. In this way, a dynamic routing mechanism

TABLE IV
AREA REQUIREMENTS. (*) VIRTEX–5 DEVICES USE 6–INPUT LUTS.

Device		noc_0		noc_1	
Family	*Code*	*Available Slices*	*Used resources*	*Available Slices*	*Used resources*
Spartan–3	XC3S200	1920	970 (50%)	1920	863 (44%)
Spartan–3	XC3S400	3584	970 (27%)	3584	863 (24%)
Virtex–II Pro	XC2VP7	4928	962 (19%)	4928	854 (17%)
Virtex–II Pro	XC2VP20	9280	962 (10%)	9280	854 (9%)
Virtex–II Pro	XC2VP30	13696	962 (7%)	13696	854 (6%)
Virtex–4	XC4VFX12	5472	1152 (21%)	5472	1035 (18%)
Virtex–4	XC4VSX25	10240	1152 (11%)	10240	1035 (10%)
Virtex–4	XC4VLX15	6144	1152 (18%)	6144	1035 (17%)
Virtex–5 (*)	XC5VLX85	51840	628 (1%)	51840	553 (1%)

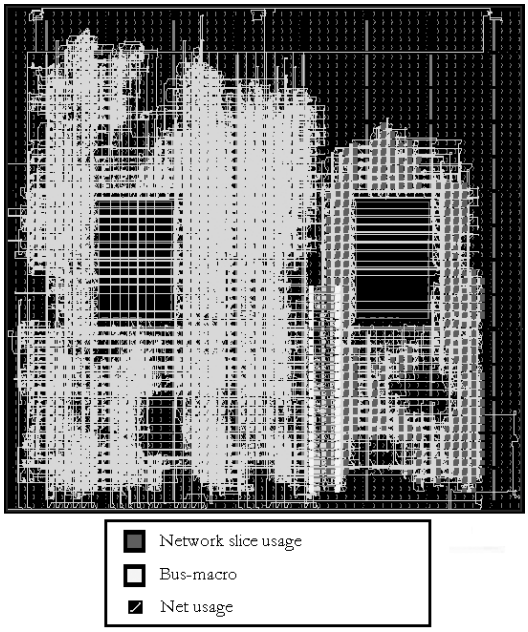

Fig. 6. Reconfigurable architecture; static side and reconfigurable side with first topology, before the reconfiguration process.

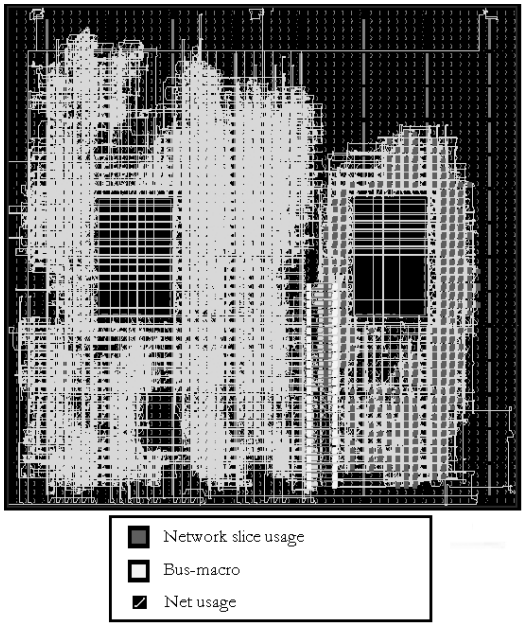

Fig. 7. Reconfigurable architecture; static side and reconfigurable side with second topology, after the reconfiguration process.

has been realized, in which the routing information relies on the current network status.

Future works have to be done to improve the capabilities of the network; we will define a way to make the communication fault–tolerant with respect to the application execution. For this reason, a persistent communication channel among the switches of the network is defined. Further, to keep the network consistent with previous approaches, a bridge should be realized in order to be able to use the proposed approach in architectures with (necessary) bus–based connections. As explained in Section IV, the direction that we are following is to keep the requirements low, as the light–weight approach we defined clearly states.

REFERENCES

[1] Xilinx. Xapp290 - two flows for partial reconfiguration: Module based or small-bit manipulation. 2002.

[2] Xilinx. Early access partial reconfiguration user guide. March 2006.

[3] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38:1–51, 2006.

[4] Cindy Kao. Benefits of partial reconfiguration. *XCell*, pages 65–67, 2005.

[5] Xilinx. Ug070 - virtex-4 user guide, April 2007.

[6] A. Ahmadinia, C. Bobda, J. Ding, M. Majer, J. Teich, S.P. Fekete, and J.C. van der Veen. A practical approach for circuit routing on dynamic reconfigurable devices. *Rapid System Prototyping, 2005. (RSP 2005). The 16th IEEE International Workshop on*, pages 84–90, 8-10 June 2005.

[7] C. Bobda and A. Ahmadinia. Dynamic interconnection of reconfigurable modules on reconfigurable devices. *Design & Test of Computers, IEEE*, 22(5):443–451, Sept.-Oct. 2005.

[8] T. Pionteck, C. Albrecht, R. Koch, E. Maehle, M. Hubner, and J. Becker. Communication architectures for dynamically reconfigurable fpga designs. pages 1–8, 26-30 March 2007.

[9] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002.

[10] D. Bertozzi and L. Benini. Xpipes: a network-on-chip architecture for gigascale systems-on-chip. *Circuits and Systems Magazine, IEEE*, 4(2):18–31, 2004.

[11] 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.

[12] W.J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. pages 684–689, 2001.