

# Topology adaptive network-on-chip design and implementation

T.A. Bartic, J.-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde and R. Lauwereins

**Abstract:** Network-on-chip designs promise to offer considerable advantages over the traditional bus-based designs in solving the numerous technological, economic and productivity problems associated with billion-transistor system-on-chip development. The authors believe that different types of networks will be required, depending on the application domain. Therefore, a very flexible network design is proposed that is highly scalable, and can be easily changed to accommodate various needs. A network-on-chip design, realised as part of the platform that the authors are developing for reconfigurable systems, is presented. This design is suitable for building networks with irregular topologies, and with low latency and high throughput.

## 1 Introduction

The design of larger more complex systems becomes an increasingly difficult task because of the many different issues related to the productivity, design reuse, technology and cost that have to be tackled simultaneously [1, 2]. In this context IP reuse becomes more and more important. However, building a system reusing existing IP blocks requires first of all standardised interfaces [3] and secondly finding a way of exchanging information between them. There are different solutions to this communication problem but a disciplined and scalable one is offered by the network-on-chip strategy [4].

A standard way of communication becomes even more important for dynamically reconfigurable systems. The advantage of such a system is that it can efficiently execute a large variety of tasks with distinct requirements using a limited but reconfigurable amount of resources. Our reconfigurable system [5] consists of an instruction set processor (ISP) combined with reconfigurable hardware. The reconfigurable hardware is divided into blocks, which will be referred to as tiles, that can be reconfigured with hardware implemented tasks, which will be referred as IPs. Tasks can be run either in their software version on the ISP or in their hardware version using one of the available hardware reconfigurable tiles, according to the availability and the performance required from the system. As tasks are swapped in and out the communication between tiles changes accordingly. Implementing *ad-hoc*, point-to-point connections between tasks would imply solving a very complex and time-consuming run-time routing problem. Our approach consists in implementing a network that will provide the necessary flexibility in the communication between tasks.

There have been many network-on-chip designs proposed to date, however, most of them provide only limited flexibility in the number of parameters that can be changed. Network topology is one of the key constraints in the previous solutions. Our present design removes many of these limitations and allows for easy exploration and comparison of the different design choices.

However, the advantage of a standard communication infrastructure is always coupled with concern about performance loss. Applications and systems have various requirements and hence networks with different capacities and services are needed. Applications with similar needs can be grouped into application domains such as: databases, multimedia, Internet, general-purpose parallel computing, etc. Considering the application domain and the system that will run the applications (supercomputer, personal computer, mobile devices, ...), the designer will have to choose a network offering the best compromise between power, size and capacity.

To address the issue of variable communication needs we have designed a highly scalable network. Together with the IP interfaces, the ISP and the operating system (OS) that manages all these resources, it forms a powerful and very versatile platform that is described in detail in [6]. This is our second network design and it differs from the first one [7] in its increased scalability, and its flexibility to choose the topology and the routing algorithm. Our design is written in synthesisable VHDL and it has been implemented on a Xilinx Virtex-II Pro FPGA, but equivalent gate numbers are also provided to enable comparisons with other technologies.

## 2 System overview

Communication between the hardware resources will occupy a very important place in future System-on-Chip designs. From the communication point of view it is convenient to use the OSI model to describe our platform.

As Fig. 1 shows, at the highest-level the application designer uses the services offered by the layer beneath, the OS. For our platform the OS that manages the underlying communication infrastructure is called the operating system for reconfigurable systems (OS4RS) [8]. It is based on real-time Linux on top of which specific capabilities have been

---

© IEE, 2005

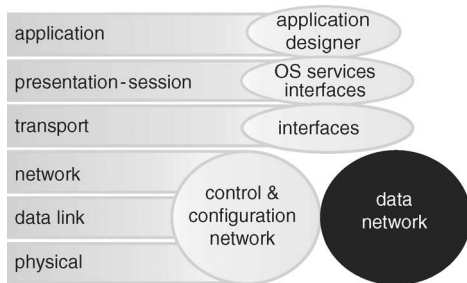
IEE Proceedings online no. 20045016

doi: 10.1049/ip-cdt:20045016

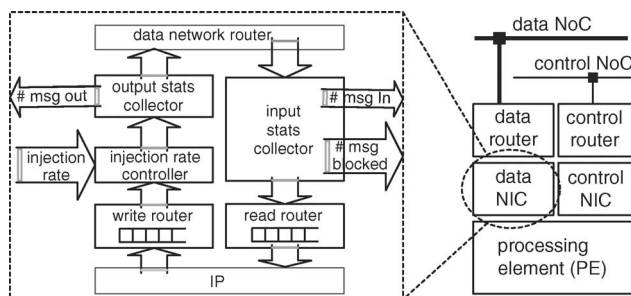
Paper first received 29th April and in revised form 13th July 2004

The authors are with IMEC, Kapeldreef 75, 3001 Leuven, Belgium. D. Verkest is also with Vrije Universiteit, Brussel and Katholieke Universiteit, Leuven. R. Lauwereins is also with Katholieke Universiteit Leuven

E-mail: bartic@imec.be



**Fig. 1** The structure of our platform from the communication point of view



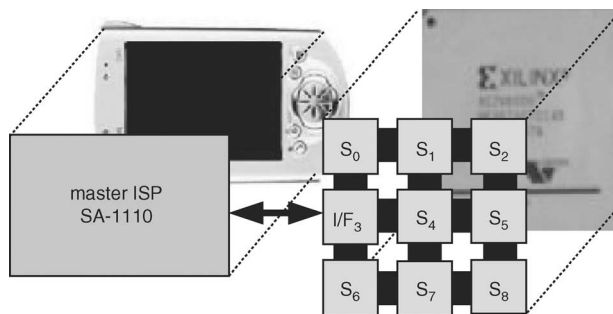
**Fig. 2** The structure of the IP interfaces

The data NIC offers a high-level interface to the data router. The control NIC allows the OS to control the communication resources and the IPs

added. Part of the OS is implemented in hardware [9]. The hardware part provides a standard interface to the IPs and the means to manage the data network. The transport level is also implemented by the interfaces. The structure of the interfaces is presented in more detail in Fig. 2. It can be seen that the interface is made of a data network interface component (data NIC) and a control network interface component (control NIC).

From the IP viewpoint the main role of the data NIC is to buffer input and output messages and to provide a high-level interface to the data router. The data NIC is also responsible for collecting communication statistics. This involves keeping track of the number of messages sent, received and blocked. The blocked message count denotes the number of received messages that were blocked in the data router buffer while waiting for the IP input buffer to be released. Moreover, the data NIC implements an injection rate control mechanism, that allows it to control the amount of messages the attached IP injects in the data network per unit of time [9, 10]. The desired injection rate is supplied by the OS using the control network. In this way the OS can provide a ‘coarse’ quality of service, by regulating the amount of data injected by the IPs in the network. The data NIC also handles possible clock differences between the IP and the data network. To this end it uses double-port buffers, where each port has its own clock.

Each node in our system is also connected to a control network interface component (control NIC), which in turn is connected to the control network. The main role of the control NIC is to provide OS4RS with a unified view of the communication resources. For instance, the message statistics collected in the data NIC are processed and communicated to the core OS by the control NIC. The control NIC also allows the core OS to dynamically set the routing table in the data router or to manage the injection rate control mechanism of the data NIC. Another role of the control NIC is to provide OS4RS with an abstract view of the distributed IPs. Hence, it is considered to be a distributed



**Fig. 3** The platform consists of one main processor and a network-on-chip instance

This Figure shows the case where a StrongARM was used as the main processor together with a  $3 \times 3$  mesh network

part of the OS4RS. This role of the control NIC is discussed in-depth in [9].

The data network provides the communication between the IPs and the rest of this paper focuses on its design.

The entire platform is shown in Fig. 3. It consists of one central processor (the StrongARM inside the IPAQ handheld device) running OS4RS and a network instance. The figure shows the case of a  $3 \times 3$  mesh network. One of the network tiles is used for the communication with the main processor. The network itself resides on a Virtex II FPGA.

### 3 Router design

At the heart of the data network are the routers. Depending on topology and on the number of IPs attached to them, the routers have a variable number of input and output ports. Separating the input ports from the output ports allows a flexible network design. For example a router that is part of a network with a mesh topology, with bi-directional connections to four neighbouring routers and one IP has five input and five output ports. Our router is independently parameterisable in the number of input and output ports. It therefore allows routers with an unequal number of inputs and outputs. All inputs and outputs are identical no matter if they connect to another router or to an IP. All the extra functionality required by the communication with the IPs is implemented in the interfaces.

#### 3.1 Switching technique

The current implementation uses virtual cut-through switching in order to keep the latency low: packets are forwarded as soon as they arrive if the output channel is free. The base latency (computed for the case when the packets are never blocked), in cycles, is given by the formula [11]:

$$t_{\{vct\}} = t_{\{first\_flit\}} \times NoNodes + NoFlits \quad (1)$$

where  $t_{\{first\_flit\}}$  is the number of cycles required to route and switch the first flit (in our design phits are equal to flits),  $NoNodes$  is the distance in nodes from origin to destination and  $NoFlits$  is the total number of flits in the packet. A flit is smallest data unit on which data flow control is performed. A phit is a data unit on which data flow control is not performed, and is smaller or equal to a flit.

This switching technique requires that the packets have a finite size, such that they can be fully buffered in the router if the next router is not available. In our design the maximum packet size is given as a parameter and it can have a large impact on the network performance. If the packets are very small, the overhead due to the packet header can be

considerable. On the other hand, if the packets are too big the size of the buffers will be very big and the network will cover a lot of silicon area.

Virtual cut-through achieves, at low traffic, the same low latency as wormhole switching, whereas at high loads, it reaches the same high throughput as store and forward [12]. Because in our system the size of the payload is relatively large (544 bytes, for OS efficiency reasons [13]) a wormhole switching network would saturate much faster than the virtual cut-through one. The drawback is the large buffer size required to store the packets. For this reason wormhole switching with virtual channels will be considered for the next version of the network.

### 3.2 Routing algorithm

The routing algorithm is another important design choice. The taxonomy of the routing algorithms is very complex [11] but from the adaptivity point of view they can be divided into adaptive and deterministic algorithms. Our router uses a deterministic algorithm with a limited amount of adaptivity.

The routing algorithm can be implemented as a finite state machine or as a look-up table. We have chosen a look-up table approach in the current implementation. The table has an entry for each IP in the network, and its content can be dynamically changed by the OS, at run time. Having one entry for each IP in every router gives us the possibility to customise the routing for each IP at the link level. The drawback is of course that the required memory space grows proportionally with the number of IPs. This is to be compared with a finite state machine implementation that has a fixed size independent of the number of IPs but has no flexibility.

This higher-level form of adaptivity, could be useful in networks with a known traffic pattern. The IPs can be characterised in terms of the required data rate, and knowing the senders and the receivers for each IP, the routing tables can be modified to balance the network traffic. Each time a task that produces a change in the communication between the IPs is started, the OS can compute a new optimum distribution of the communication channels and the tables changed accordingly.

This feature can be exploited in reconfigurable systems as well. When two IPs exchange many packets it is preferable to place them next to each other, to exploit traffic locality. In reconfigurable systems adjacent positions can be freed by swapping IPs. However, IP swapping can take a significant amount of time, not to mention the temporary loss in performance: the tasks will have to be interrupted, their state saved, then they are swapped and their state restored. The flexibility in reprogramming the routing tables can be used to avoid frequent IP relocation. Providing that there are enough free communication resources, packets can be made to travel over low traffic connections incurring the minimum extra delay but taking full advantage of the maximum available bandwidth and saving the time required by IP swapping and maintaining undisturbed the currently executing tasks.

The update of the routing tables would be performed by the OS, and would therefore require considerably more time than just a few clock cycles. Therefore, this type of adaptivity does not offer all the power of a hardware implemented adaptive algorithm. However, for systems with well known communication pattern, this type of flexibility will be adequate for many situations, presenting the advantage of much lower hardware requirements than would be needed for true adaptive routing algorithms.

### 3.3 Router structure and implementation

The router is built from input and output blocks connected through a shared crossbar switch (see Fig. 4). Any input can be connected to any output, and there is an arbiter in every output to resolve conflicts between the different inputs. Different arbitration schemes could be used, the one that is presently implemented is an acceptance-dependent round robin.

The crossbar switch can have different designs, the current one is the most straightforward one, using large multiplexers in a single stage. Other designs using smaller multiplexers in multiple stages would result in a smaller area. The only requirement is that the crossbar switch should remain non-blocking, connections should not wait on each other, to make full use of the available resources and to maintain latency as low as possible.

The data link width is parameterisable, such that the network can be easily adapted to accommodate different capacity requirements. There is, however, a minimum width equal to the size of an IP address, because the address decoding is done in one cycle. For example a 16-node network, needs four address bits, hence the smallest link width is four.

The handshaking between routers is realised through request/acknowledge lines. When a request arrives at the input block there is one clock cycle needed for the routing and then one (or more) cycle to get the arbiter acknowledgment, depending on the status of the output buffer. Once the request is acknowledged one flit is transmitted every clock cycle. The flow control is made at the packet-level, meaning that an entire packet is sent before another one can begin.

As previously discussed, for virtual cut-through switching, packets have to be stored if blocked. There are several different buffering strategies. The buffers can be placed before the crossbar switch i.e. input queuing, or after i.e. output queuing. Our implementation uses output queuing and there is one buffer per output block. The advantage of this solution is that it avoids head-of-line blocking which occurs in the case of input queuing. In this case, if a packet is blocked, once it is buffered the input becomes free again to receive and forward a new packet. If the following packet requires the same output then depending on the size of the output buffer the input may or may not be blocked. If the output buffer can hold more than one packet then subsequent packets can be buffered until the maximum

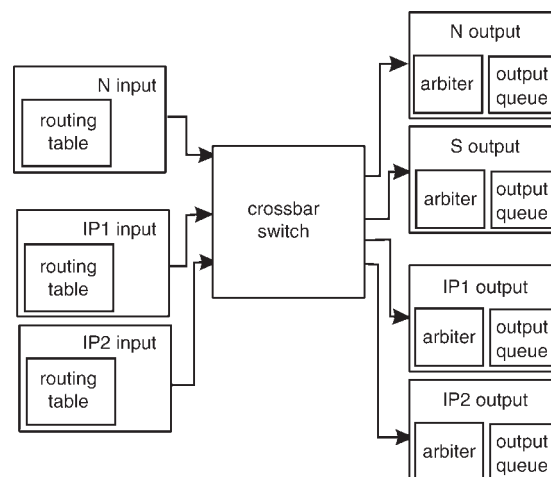


Fig. 4 Structure of a one-input, two-output router with the possibility of connecting two IPs

number is reached. In the case of heavy network traffic the throughput increases if buffers are capable of storing several packets, but so does the network area. For our implementation on the Virtex-II Pro architecture we used the available 2kbyte block RAMs (BRAMs). For packets of 544 bytes we can buffer up to three packets in one BRAM.

#### 4 Network design

As discussed in the Introduction, different networks will be required for different application domains. A flexible, easy to customise network design allows many different solutions to be simulated and synthesised in a short time, making the network optimisation easier and shortening the development time.

Our network is implemented as a VHDL component, parametrised using generics. The number of network nodes can be easily changed during the design, allowing networks of different sizes. The topology is passed as a bi-dimensional array, mentioning all the connections between the different routers, and between the routers and the IPs.

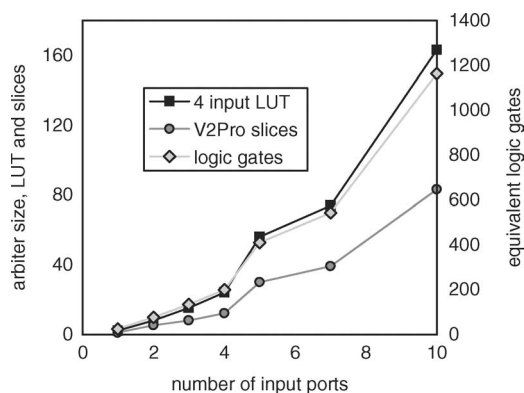
As previously mentioned, the router is independently parameterizable in the number of input and output ports, which allows us to build irregular topologies. Moreover, several IPs can be connected to one router.

##### 4.1 Area and timing performance

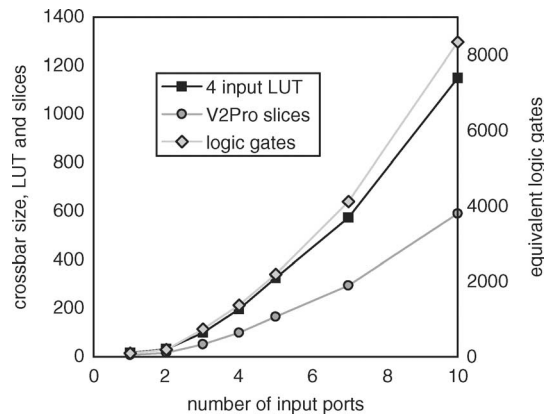
The design is optimised for performance: every input block has a routing table, and every output block an arbiter and a buffer (see Fig. 4). This solution provides the shortest possible latency at the cost of silicon area. The  $t_{\text{first\_flit}}$  from (1) is three cycles. Therefore, a packet travelling five nodes has a base latency of 15 cycles plus a number of cycles equal to the number of packet flits. For a large packet the delay to receive the first flit is only a small fraction of the time needed to receive the entire packet. For a streaming application once the first flit arrives the IP can start processing the data right away, the next flits will follow every clock cycle. The overhead delay incurred by the routing through the network is kept to a minimum.

To save area the routing table and the arbiter could be shared by introducing a mechanism where inputs and outputs can access them only in a certain time slot. However, for routers with a small number of ports multiplying the routing table might be a better option. For Virtex-II Pro the tables are very efficiently implemented using distributed dual-port select RAMs.

The blocks that have the worst scaling properties are the crossbar and the arbiter. Both blocks (see Figs. 5 and 6)



**Fig. 5** Arbiter size scaling with the number of input ports (Virtex-II Pro look-up tables (LUTs), slices and equivalent logic gates as reported by Xilinx mapping tool)



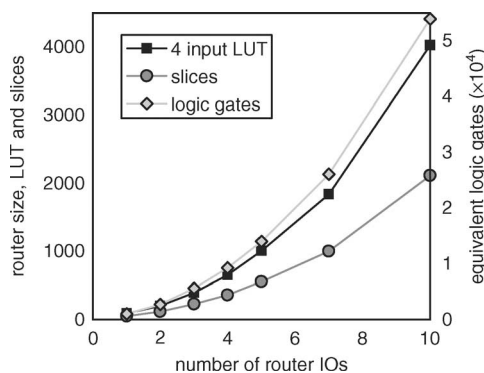
**Fig. 6** Crossbar size scaling for an equal number of input and output ports

(Virtex-II Pro LUTs, slices and equivalent logic gates as reported by Xilinx mapping tool)

grow quadratically in size with the number of ports. For a five-inputs, five-output router the arbiter takes 30 Virtex-II Pro slices (equivalent to 410 logic gates), representing 50% of the output block size and about 25% of the router size. For routers with more ports, as would be needed for high-dimensional network topologies, its size would increase considerably. For these topologies a centralised arbiter would result in considerable area reduction. The crossbar has almost the same scaling behaviour. For the above-mentioned router size the crossbar takes 165 slices (2300 equivalent gates), but because there is only one crossbar per router its proportion in the total router size is almost the same as of all the arbiters.

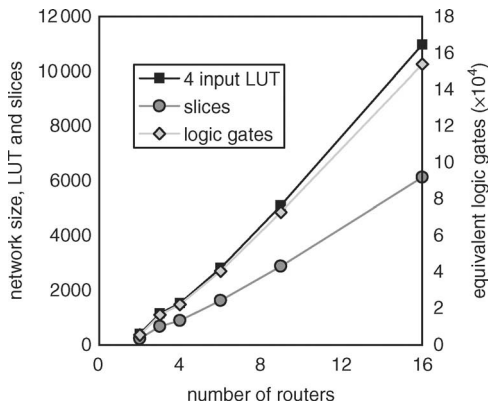
The router size, as a function of the number of ports, for an equal number of inputs and outputs is shown in Fig. 7. Besides the router size expressed in LUTs and slices, the Figure shows the equivalent number of logic gates. The buffers are not included in the total number of gates. The number of slices shows a quadratic increase with the number of ports due to the non-linear scaling of the arbiter and of the crossbar switch. A better implementation of these two blocks would result in an improved scalability. Even so, the size of a router required for low-dimensional topologies, excluding the buffers is very small.

Figure 8 shows the network size as a function of the number of nodes for a mesh topology. Because of the fixed dimensionality of the topology (and therefore a fixed number of router ports) the size increases linearly with the number of nodes. The size of the buffers is not included in



**Fig. 7** Router scaling, for an equal number of input and output ports

(Virtex-II Pro LUTs, slices and equivalent logic gates as reported by the Xilinx mapping tool)



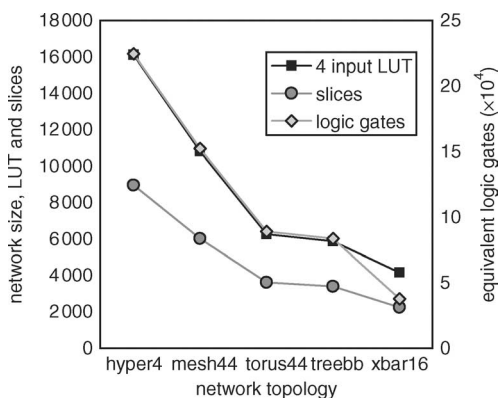
**Fig. 8** Network scaling as a function of number of nodes for mesh topology

(Virtex-II Pro LUTs, slices and equivalent logic gates as reported by the Xilinx mapping tool)

the total number of equivalent gates. If the network uses virtual cut-through switching, with large packets of 544 flits, for an ASIC implementation the buffers would represent a considerable part of the total network area, around 90% by our estimations, for all network sizes. However, in an FPGA, BRAMs are very efficiently implemented and therefore relatively, the cost of the buffers compared to the cost of logic is strongly decreased. For a nine-node mesh network, implemented in Virtex-II Pro 40 the logic takes 14% of the logic resources and the buffers 17% of the available memory.

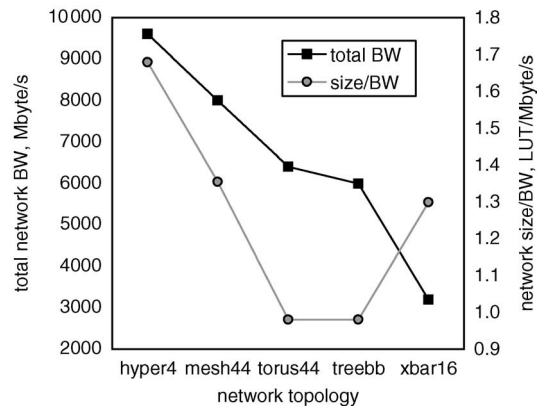
The network size for different network topologies is shown in Fig. 9. All the networks shown in the Figure interconnect 16 IPs, with one IP per router. The different topologies are: four-dimensional hypercube, a  $4 \times 4$  mesh, a  $4 \times 4$  torus, a balanced binary tree and a crossbar switch. As expected the networks with a high interconnectivity have a higher size. The total network bandwidth for the different topologies is shown in Fig. 10. It was assumed that the network can run at 50 MHz and that the links are 16 bits wide, which results in a data rate of 100Mbyte/s per link. The crossbar switch has the smallest area of all but also the smallest total bandwidth. In reality the total bandwidth should be corrected with the saturation coefficient of the network. In real life the maximum network throughput will be much lower and it will depend on a variety of factors, including the routing algorithm, buffer sizes, switching technique and so on.

Figure 10 also shows the ratio between the network area and the total network bandwidth, which could be considered



**Fig. 9** Network size as a function of topology

All the networks interconnect 16 IPs. (Virtex-II Pro LUTs, slices and equivalent logic gates as reported by the Xilinx mapping tool)



**Fig. 10** Cumulative network bandwidth as a function of topology and their respective area-bandwidth efficiency

All the networks interconnect 16 IPs

as a measure of the area efficiency. It can be seen that for a given bandwidth the most area-efficient topologies would be the tree and torus topologies, followed by the crossbar switch and at a larger distance by meshes and hypercubes. However, as mentioned before, these numbers will have to be corrected in order to obtain the real efficiency, with a coefficient that is dependent on the particular choices of that design and on the applications.

The choice of the topology cannot be based solely on bandwidth considerations. Physical considerations also play a role. If the network routers have to be distributed at a certain distance from one another, to help diminish deep sub-micron issues related to long lines for instance, then the topology cannot be a crossbar switch. Another important aspect is latency. From all the presented topologies the hypercube has the highest degree of interconnectivity and will provide the smallest average latency. Therefore, although hypercubes are not very efficient in terms of area waste for a given bandwidth this topology could be chosen to reach low latencies.

## 5 Related work

Network-on-chip designs have been proposed by Guerrier and Greiner [14] with the SPIN network, Rijkema *et al.* with Aethereal [15] and Kumar *et al.* [16]. Although all these designs are scalable, the proposed networks can be extended only by increasing the number of nodes, while keeping a fixed, regular topology. We believe, however, that a single topology cannot suit all the applications. Moreover, support for irregular topologies is important as they allow us to make better trade-offs and permit a smooth transition from the present, mainly bus-based designs.

Recently, Saastamoinen *et al.* [17] have proposed a network design that supports several parametrisation options including topology. Their approach is to build a library of components that can be combined in order to realise different networks. However, for the time being their design does not support irregular topologies.

## 6 Conclusions

Network-on-chip systems can have different requirements in terms of latency, throughput, area and power depending on the application domain at which they are targeted. A flexible network design gives the opportunity to easily search the design space for an optimum solution as well as to quickly develop the communication infrastructure of a

new system. The possibility of dynamically changing the packet routing for each IP, in every router allows us to better balance the network traffic, assuming that the IPs have known traffic pattern and requirements. A network using virtual cut-through switching has a low latency while maintaining a high throughput. The price to pay is the large amount of silicon area required by the buffers. However, for a reconfigurable platform, implemented on Virtex FPGA, the available resources are used almost in the same proportion for logic and for memory. The network area can be rapidly assessed for different topologies. However, the topology choice will have to take into account in addition to the area and total bandwidth other parameters such as that latency and the network saturation coefficient, and physical constraints.

## 7 Acknowledgments

Part of this research has been funded by the European Commission through the IST-AMDREL project (IST-2001-34379) and by Xilinx Labs, Xilinx Inc. R&D group.

## 8 References

- 1 Jantsch, A., and Tenhunen, H.: 'Will networks on chip close the productivity gap?', in 'Networks on chip' (Kluwer Academic Publishers, 2003), pp. 3–18
- 2 Benini, L., and De Micheli, G.: 'Networks on chips: A new SoC paradigm', *Computer*, 2002, **35**, (1), pp. 70–78
- 3 Rowson, J.A., and Sangiovanni-Vincentelli, A.: 'Interface-based design'. Proc. 34th Annual Design Automation Conf. (ACM Press, 1997) pp. 178–183
- 4 Dally, W.J., and Towles, B.: 'Route packets, not wires: on-chip interconnection networks'. Proc. 38th Annual Design Automation Conf. (ACM Press, 2001), pp. 684–689
- 5 <http://www.imec.be/reconfigurable>
- 6 Marescaux, T., Nollet, V., Mignolet, J.-Y., Bartic, A., Moffat, W., Avasare, P., Coene, P., Verkest, D., Vernalde, S., and Lauwereins, R.: 'Run-time support for heterogeneous multitasking on reconfigurable SoCs', *Integr. VLSI J.*, 2004, to be published
- 7 Marescaux, T., Bartic, A., Verkest, D., Vernalde, S., and Lauwereins, R.: 'Interconnection networks enable fine-grain dynamic multi-tasking on FPGA's'. Proc. Field-Programmable Logic and Applications Conf. 2002, pp. 795–805
- 8 Nollet, V., Coene, P., Verkest, D., Vernalde, S., and Lauwereins, R.: 'Designing an operating system for a heterogeneous reconfigurable SoC'. Proc. Int. Symp. on Parallel and Distributed Processing, 2003, pp. 174–180
- 9 Marescaux, T., Mignolet, J.-Y., Bartic, A., Moffat, W., Verkest, D., Vernalde, S., and Lauwereins, R.: 'Networks on chip as hardware components of an OS for reconfigurable systems'. Proc. Field-Programmable Logic and Applications Conf. 2003, pp. 595–605
- 10 Nollet, V., Marescaux, T., Verkest, D., Mignolet, J.-Y., and Vernalde, S.: 'Operating system controlled network on chip'. Proc. 41st Design Automation Conf., 2004
- 11 Duato, J., Yalamanchili, S., and Ni, L.: 'Interconnect networks, An engineering approach' (IEEE Computer Society Press, Los Alamitos, CA, USA, 1997)
- 12 Shin, K.G., and Rexford, J.: 'Support for multiple classes of traffic in multicomputer routers'. Proc. Parallel Computer Routing and Communication Workshop, 1994, pp 116–130
- 13 Mignolet, J.-Y., Nollet, V., Coene, P., Verkest, D., Vernalde, S., and Lauwereins, R.: 'Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip'. Proc. DATE 2003, pp. 986–991
- 14 Guerrier, P., and Greiner, A.: A generic architecture for on-chip packet-switched interconnections. Proc. Conf. on Design, Automation and Test in Europe, ACM Press, 2000, pp. 250–256
- 15 Rijpkema, E., Goossens, K.G.W., Radulescu, A., Dielissen, J., van Meerbergen, J., Wielage, P., and Waterlander, E.: 'Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip'. Proc. Conf. on Design Automation and Test in Europe, 2003
- 16 Kumar, Sh., Jantsch, A., Soininen, J.-P., Forsell, M., Millberg, M., Berg, J., Tiensyrj, K., and Hemani, A.: 'A network on chip architecture and design methodology'. Proc. IEEE Computer Society Annual Symp. on VLSI, 2002
- 17 Saastamoinen, I., Siguenza-Tortosa, D., and Nurmi, J.: 'An IP-based on-chip packet-switched network', in 'Network on chip' (Kluwer Academic Publishers, 2003), pp. 193–213