

Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs

T. Marescaux, A. Bartic, D. Verkest*, S. Vernalde, and R. Lauwereins**

IMEC vzw.,
Kapeldreef 75, 3001 Leuven, Belgium
marescau@imec.be

Abstract. Multimedia support appears on embedded platforms, such as WAP for mobile phones. However, true multimedia applications require both the computation power that only dedicated hardware can provide and the flexibility of software implementations. To this end, we are investigating reconfigurable architectures, composed of an instruction-set processor running software processes and coupled to an FPGA on which hardware tasks are spawned by dynamic partial reconfiguration. This paper focuses on two main aspects. It explains how separating communication from computation enables hardware multi-tasking and it describes our implementation of a fixed communication-layer that decouples the computation elements, allowing them to be dynamically reconfigured. This communication layer is an interconnection network, implemented on a Virtex FPGA, allowing fast synchronous communication between hardware tasks implemented on the same matrix. The network is a 2D torus and uses wormhole routing. It achieves transfer rates up to 77.6 MB/s between two adjacent routers, when clocked at 40 MHz. Interconnection networks on FPGAs allow fine-grain dynamic partial reconfiguration and make hardware multi-tasking a reality.

1 Introduction

Nowadays, numerous multimedia applications are emerging on portable devices such as personal digital assistants (PDA) or mobile phones. Typical applications such as MPEG players or 3D games are usually computationally intensive, preventing them from being implemented on general-purpose embedded processors. To achieve the minimal Quality of Service (QoS) required for these applications, traditional designs of multimedia platforms contain dedicated hardware accelerators, which lack flexibility, or application specific instruction-set processors (ASIP), which are limited to their specific application domains.

Based on our experience in reconfigurable systems [1], we believe that the combination of instruction-set processors (ISP) with reconfigurable hardware is the best trade-off for such a platform. The platform has to support true hardware/software (HW/SW) multitasking, i.e. tasks are executed either on the ISP

* also professor at Vrije Universiteit Brussel

** also professor at Katholieke Universiteit Leuven

or on the reconfigurable hardware, a Field Programmable Gate Array (FPGA) in our case. A hard real-time operating system (RT-OS) manages the applications by distributing the different tasks on the available resources. Our platform is composed of a Compaq iPaqTM PDA, running RT-Linux [3] on its Strong-Arm processor SA-1110 (206MHz) and controlling a Xilinx VirtexTM XCV800 running hardware tasks.

The applications running on our platform [2] are composed of several software threads and of several hardware tasks. These HW and SW components must therefore be able to inter-communicate, i.e. a specific HW communication layer, compatible with SW communication, has to be designed.

Hardware resources are shared by dividing the FPGA into logical tiles of coarse granularity, such as a JPEG decoder. However, the reconfiguration grain is fine, i.e. an AES encryption module could replace the JPEG decoder. Tasks can be dynamically instantiated in the tile matrix by partial reconfiguration. Our communication-layer is a packet-switched Inter-Connection Network (ICN) and is fixed in place to allow Dynamic Partial Reconfiguration (DPR).

This paper presents the use of on-FPGA interconnection networks to enable fine-grain dynamic partial-reconfiguration. To this end, Sect. 2 develops how separating communication from computation allows hardware multitasking on FPGAs. Sect. 3 presents a simple packet-switched network that is used to create a fixed communication layer. Sect. 4 details the implementation of our interconnection network on the Virtex family. Specific issues about Virtex column-based DPR are treated in Sect. 5. Sect. 6 discusses the performances of our reconfigurable System on Chip (SoC) platform. Finally, Sect. 7 concludes.

2 Separating Communication from Computation Enables Fine-Grain Dynamic Partial-Reconfiguration

In order to do multitasking, the FPGA must be partitioned into an array of identical tiles, each tile running a hardware task, equivalent to a software thread. Whereas the granularity of the tiles is coarse, a JPEG decoder for example, the reconfiguration grain is fine: the same tile can be configured to run an image filter as well as a data encryption module. This approach differs from previous works [4] where the FPGA is divided into simple computation elements, difficult to manage by an OS and requiring specific component libraries.

The ability to dynamically reconfigure tasks depends on the control over the boundaries between them. Indeed, with the traditional design-flow, if we reconfigure an AES encryption module in place of a Laplace edge detector, their interfaces do not match and we have to perform a Place and Route (P&R) on the whole FPGA. However, by adding constraints to the positioning of the interface, the P&R phase is only required at design-time because it yields hardware components with an identical input/output topology (Sect. 5).

Hardware tasks are thus encapsulated into a fixed layer providing them with a unified way of communicating. This communication layer raises the abstraction level of the hardware blocks, allowing easy Intellectual Property (IP) block integration.

Various fixed communication layers such as buses and on-chip interconnection networks can be used to this end. However, we prefer a packet-switched network to a bus for three reasons. A simple bus is a major bottleneck because its routing resources are shared by all connected blocks, whereas in a network routing resources are distributed. A network is therefore easily scalable, whereas the complexity of a bus arbiter increases with the number of blocks controlled. Finally, networks are more power efficient than buses, because idle parts can be powered off, whereas buses must always drive long lines.

3 Choosing a Packet-Switched Network for SoC Reconfigurable Platforms

3.1 SoC Reconfigurable Platforms Require Specific Interconnection Networks

One can think of an interconnection network as being an array of routers interconnecting an array of processors (Fig. 1(a)). In general each processor has direct access to a local memory, without using the ICN. Interconnection net-

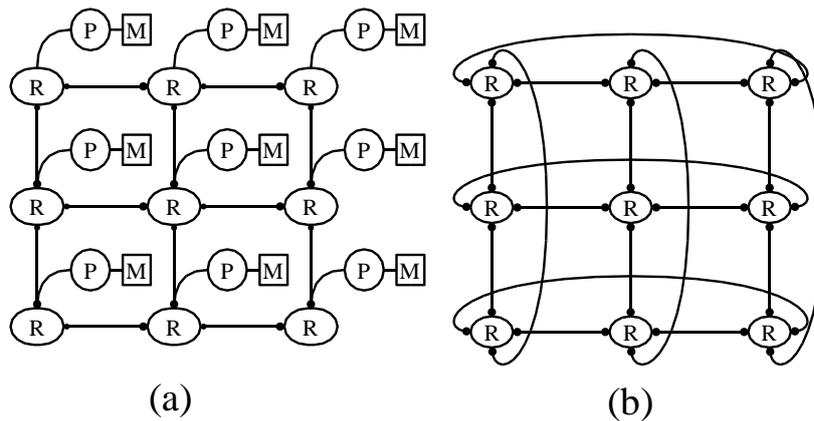


Fig. 1. In an ICN (a), each processor (P) is connected to a router (R). Each processor has access to local memory (M). In a 2D torus (b), each row and column of routers is connected in a ring, reducing router complexity with respect to a 2D mesh.

works have been successfully used in the world of multi-processor computing, such as the J-Machine [6], with various forms of architectures (k-ary n-cubes, hyper-cubes, butterflies, crossbars) and routing policies (virtual cut-through, wormhole, mad-postman switching) [9]. However, they are multi-chip circuits, whereas for reconfigurable SoC we have to implement the whole network and all the processors on the same FPGA ! The choice for our ICN is therefore dictated by the need for low hardware overhead. A network, and therefore its complexity, are described by two parameters: topology and routing algorithm [9].

3.2 The Interconnection Network has a 2D Torus Topology to Limit Hardware Overhead

An FPGA is a 2-dimensional chip, so we naturally considered 2D network topologies. In a mesh-topology such as the one in Fig. 1(a) a router has to be able to route in all directions: North, South, East and West. It is possible to reduce the router complexity by using a similar topology, called a torus network (Fig. 1(b)). Such a network folds the mesh along the horizontal and vertical directions and therefore only requires routing along two directions, i.e. East and South. However, this complexity reduction comes at the expense of a 15% increase (for a $4 * 4$ folded torus) [7] in power consumption with respect to a mesh network.

3.3 The Network uses Wormhole Packet-Switching

The routing algorithm we use on our 2D torus is called Wormhole Routing [8]. It is a blocking, hop-based, deterministic routing algorithm. It uses relative addresses and thus does not require a global knowledge of the network. In wormhole switching, message packets are pipelined through the network. This technique relieves routers from buffering complete messages, thus making them small and fast [9]. A message is broken into flits (flow control units). Flits come in two types: header flits, containing routing information and data flits containing the message itself.

The two header flits give the number of channels that must be traversed respectively in the X and Y directions. Packets are routed first along the X direction, then along the Y direction before reaching their target. The value in the X header flit is decremented each time a router is traversed. When it reaches zero, the X heading flit is discarded and routing starts on the Y dimension. When Y equals zero, the message enters the destination hardware block.

3.4 Two Virtual Channels are Used to Avoid Deadlocking

Our network uses two time-multiplexed Virtual Channels (VC) to avoid deadlocks. Deadlocks in an interconnection network occur when no message can advance toward its destination because all queues are full with messages not destined to neighboring routers (Fig. 2(a)) [8]. Fig. 2(b) shows the virtual channel usage on a 1D torus. Router 0 only sends messages on VC_0 , whereas the other routers may initiate messages only on VC_1 . This virtual channel management policy avoids deadlocks by breaking the torus into a spiral [9].

4 Architecture of our Interconnection Network

This section details our implementation of a packet-switched ICN on a Virtex XCV800. However, the figures can be easily extrapolated to the Virtex II XC2V6000, which is the target of our final demonstrator. Reconfigurable designs should target the Virtex II, because it features 6 columns of block-RAMs (BRAMs), allowing implementation of up to 6 routers, whereas the Virtex I has only 2 (Sect. 5).

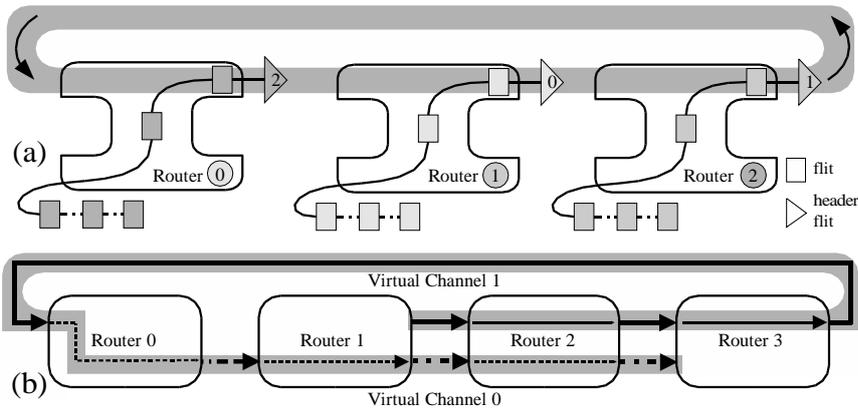


Fig. 2. When message queues are full with messages destined to non-neighboring routers (a) no message can be sent, the network is deadlocked. Using two virtual channels avoids dead-locking by breaking the 1D torus into a spiral(b).

4.1 Implementation Overview

The data-path width is chosen to maximize the network's throughput. However, it can not be too wide, because on the Virtex I family the BRAM element is 16-bit wide. Moreover, reconfigurable modules have to use the scarce number of long-lines connecting tri-state buffers (Sect. 5) to connect routers together and to get access to resources such as BRAMs or multipliers. We have chosen to prioritize the availability to these resources. Therefore, on our network, messages are segmented into 16-bit flits. The Maximum Transfer Unit (MTU) is fixed to 128 data flits per message, enabling a Virtex I to buffer two messages in a BRAM. The ICN is fully pipelined and achieves, between two routers, a peak rate of 38.8 MBytes/s per virtual channel (Sect. 6), when clocked at 40 MHz.

Hardware tasks can be slow compared to the network's bitrates. Therefore to avoid blocking, the hardware tasks are decoupled from the network using interfaces. These interfaces use dual-port BRAMs to buffer messages and work as a network abstraction layer for the hardware task.

In our demonstrator, the ICN is connected to the memory bus of the Strong-Arm SA1110 on a Compaq iPaq 3760. A specialized interface (IO interface) resides at the border of the FPGA to enable communication with the CPU. The IO interface uses control registers and interrupts to communicate with the CPU and its message buffers are memory mapped to the SA1110. The scheduling of the hardware tasks is done on the CPU.

Moreover, the reconfiguration of the Virtex is also done by the CPU, which accesses the Virtex select-map port through some glue-logic. At 50MHz a Virtex XCV800 can be reconfigured in less than 11ms (39.5ms on a XC2V6000). For partial reconfiguration we can therefore go well under 5ms, which allows hardware tasks to start in a time a user cannot perceive.

4.2 Router Architecture

On a 2D torus, rows are equivalent to columns, therefore the torus can be decomposed into rows of 1D toruses connected to columns of 1D toruses. We can chain two simple 1D routers to obtain a 2D router and produce a more modular network.

A 1D-router has two input/output channels. A message entering a router can either be forwarded along the current direction or sent orthogonally, either on a column if the Y header-flit is non-zero, or into a hardware task. Each channel is composed of a 16-bit data path and of 3-bit control signals. Two control-bits are used to signal the presence of a message and its destination and the third is a back-pressure signal (nack) used to block a message entering a busy router or interface.

The routers handle two time-multiplexed VCs to avoid deadlocking (Sect. 3.4) [9]. These VCs are interleaved, with one clock cycle each.

For efficiency, the router is fully pipelined. Because the data channels are interleaved the control signals are also interleaved and precede data by one clock cycle. The nack signal, used for blocking, is back-propagated along the message path. It takes two ticks for a flit to get out of a router, therefore at each clock cycle data is transmitted on, alternating VCs.

A 1D-router (Fig. 3), is composed of one input controller per channel, one arbiter and an output-controller. The input controllers issue output requests to a round-robin arbiter and decrement header-flits when routing. Each output channel has a 2-flit deep buffer to be able to resume blocked messages. The output controller is composed of a 2-input crossbar-switch and of the nack logic.

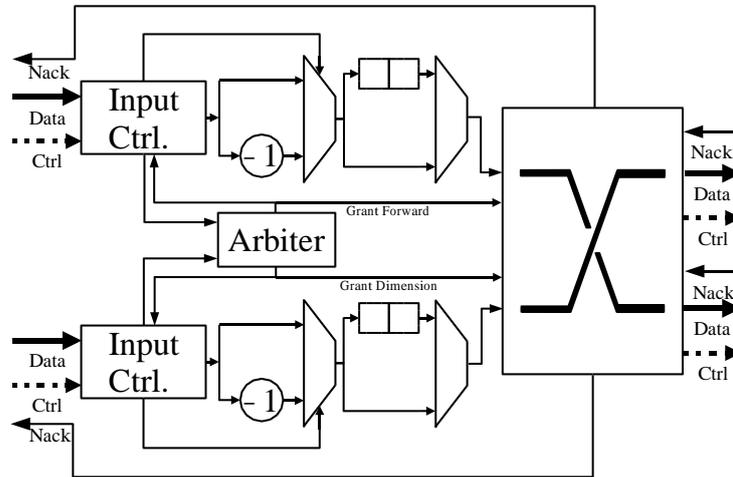


Fig. 3. 1D-Router has 1 input controller per channel, 2-flit deep buffers are included in the output controller/switch.

4.3 Architecture of the Interface between Task and Router

An interface decouples the hardware task from the network using DP-RAMs as message buffers. The hardware task can then use independent data-width and clock rates, allowing easy IP integration. The interfaces are designed to cope with the sustained bit-rates required by the ICN and perform multiple-message buffering to reduce network congestion.

Moreover the interface, called a "net-cell", provides the hardware task (or IP-block) with high-level communication by means of routing tables. Hardware tasks within an application communicate through a tuple composed of a logical address and a port number, similar to the IP address and UDP port number for the UDP protocol. Routing tables transform a destination logical address into the number of X and Y hops on the network. The routing tables are updated by the RT-OS to match the position of the IP-block in the network. Therefore, a task does not need to know where it is situated in the network and whether the tasks it communicates with are running in hardware or software. The RT-OS adapts the routing tables after an IP-block reconfiguration. Therefore, there is no need for complex run-time circuit re-routing as required in previous works [5] and circuit integrity is guaranteed. Routing tables can also be modified at any time without having to reset or stop the hardware task. This is very useful if some other task from the same application is switched in or out of the network as a function of the available resources and the QoS policy.

Our implementation of a net-cell buffers two input and two output messages on a Virtex I and eight on the Virtex II. Each message buffer is complemented by a control register bank giving the length and origin/destination of the message. This high-level protocol information such as port number or net-cell origin is piggy-backed in the Y-header flit and does not require extra bandwidth.

As Fig. 4 shows, our net-cell is composed of a process reading the local router and steering the storage of messages in a circular linked-list of message buffers and control registers. The hardware task is encapsulated in a block that always presents the same fixed interface giving access to a message-in presentation layer and a message-out presentation layer. Moreover this fixed block also gives ports to extra local resources such as BRAMS or even multipliers on a Virtex II. The presentation layers abstract the internals of the net-cell from the IP-block.

4.4 Architecture of the Interface between CPU and FPGA

Our reconfigurable platform is composed of an ISP (SA-1110) coupled to an FPGA through its memory bus. A special interface, similar to a net-cell (c.f. Sect. 4.3), has been designed to allow fast and efficient communication between them by means of memory-mapped registers and interrupts.

The Virtex I implementation can buffer 8 input and 8 output messages in its BRAMs. These are mapped in the memory space of the SA-1110 to allow fast access. Indeed, on an iPaq 3760 the SA-1110 has a maximum access speed of up to 103 MHz on its memory bus.

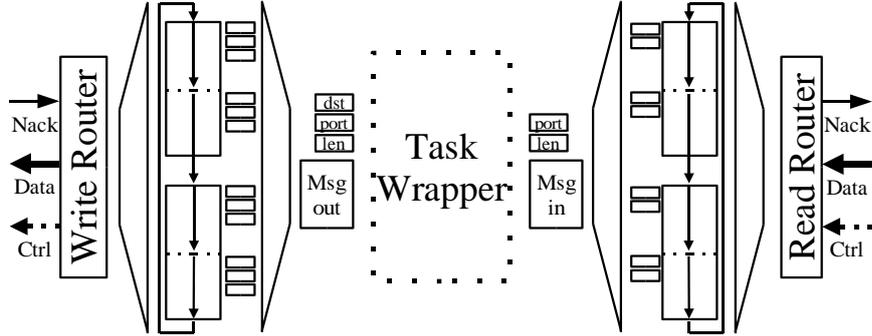


Fig. 4. Net-cell encapsulates the hardware task and provides it with high-level communication primitives (destination, port, message length).

5 Dynamic Partial Reconfiguration on Xilinx Virtex FPGAs

The full design is composed of fixed router modules and replaceable IP modules. The IP modules can be loaded dynamically, according to the user actions. When a new IP module is to be loaded into the FPGA it will be placed into a free predefined area, or will replace a module that is no longer needed. In order to be able to dynamically reconfigure the design, partial bitstreams of the IP modules must be available.

We have generated the partial bitstreams for the IP modules following the Partial Reconfiguration (PR) methodology developed by Xilinx [10] (ISE tool suite v4.2 and higher). According to the Xilinx methodology, the design has to be partitioned in fixed and reconfigurable modules. Naturally, routers are fixed and hardware tasks are the reconfigurable modules.

In principle, during the partial reconfiguration process all the modules can continue working, except for the reconfigured module. However this is not possible in our design, because the network has to maintain communication between modules. According to the PR methodology, the reconfigurable modules must span the whole chip from top to bottom [10]. Therefore the 2D torus must be folded into a 1D structure and consequently the communication between routers has to traverse the reconfigurable modules (Fig. 5). The communication with the reconfigurable modules can only take place through a "Bus Macro" (BM). The BM ensures the reproducibility of the design routing and is implemented using tri-state buffers. The tri-state buffers force the routing to always pass through the same places. At the same time they decouple the modules from each other during reconfiguration, avoiding possible harmful transitory situations. In this way, a 4 bit per row communication channel is possible between adjacent modules. This limitation comes from the current Virtex architecture and its limited routing resources.

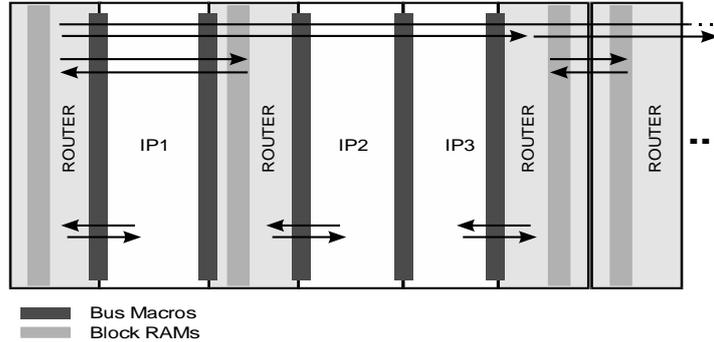


Fig. 5. Placement of routers and IP modules a Virtex II 6000.

The Virtex II 6000 has 96 rows, limiting the total number of bits passing through the interface to 384. Moreover, the BMs only support unidirectional signals, strongly limiting the minimum size of an FPGA that can be used to implement our network.

The current Xilinx tool suite requires the modules to have widths in multiples of 4 columns. A Virtex II 6000 has 88 columns which sets the upper limit of the possible modules to 22. However, the main limitation in the number of modules comes from the fact that all resources, such as BRAMs, present inside the area reserved for a module can only be used by that module. Because the routers use BRAMs, the maximum number of routers is given by the number of BRAM columns available on the FPGA: 2 for a Virtex I and 6 for a Virtex II 6000.

6 Results

The use of an ICN brings in some hardware overhead (Tab. 1). The synthesis has been performed with Synopsys Design Compiler on a Virtex *XCV800*. For the 2*2 torus network we implemented, the hardware overhead on a Virtex *XCV800* amounts to 35%, but drops to 9.5% on a bigger Virtex *XC2V6000*. The ICN is

Table 1. Hardware overhead induced by an ICN.

Element	<i>XCV800</i> (slices)	<i>XCV800</i> (%)	<i>XC2V6000</i> (%)
1D-Router	223	2.4	0.7
Net-Cell	259	2.8	0.8
FPGA-CPU Interface	716	7.6	2.1
1 * 4 1D-Torus (estimated)	2385	25.4	7.1
2 * 2 2D-Torus (estimated)	3227	34.8	9.5

fully pipelined and it takes 2 clock cycles to transmit one 16-bit flit on a given VC. There are 128 data flits in a message and two extra header flits plus two

more for the message tail. The bandwidth between two adjacent routers, per VC is therefore: $(16 - bit * 20MHz * 128/132)/8 = 38.8MBytes/s$. On a $2 * 2$ torus network, the total bandwidth is therefore $310.4MBytes/s$. The throughput of the network is however limited by the deterministic routing scheme. If each task is simultaneously sending a message, the throughput drops to 20%. A technique to overcome this problem is to clock the network faster than the hardware tasks.

7 Conclusions

This paper presents the three steps that enable us to use interconnection networks to perform fine-grain dynamic multi-tasking on FPGAs. In the first place, one must separate communication from computation by using a fixed communication layer. To this end, interconnection networks are to be preferred over bus architectures because they are more scalable and consume less power. Our interconnection network is a fully-pipelined 2D-torus that uses wormhole routing to minimize hardware overhead and achieves $77.6MBytes/s$ at $40MHz$. Finally, dynamic partial reconfiguration is possible on Virtex FPGAs by folding the 2D-network into a 1D-structure fitting the Virtex column-based architecture.

Interconnection networks enable fine-grain dynamic multi-tasking on FPGAs for a low hardware overhead. Hardware tasks can be dynamically instantiated in the network by partial reconfiguration, opening the way to a new class of hybrid applications dynamically mixing hardware and software components.

References

1. D. Desmet, P. Avasare, P. Coene, S. Decneut, F. Hendrickx, T. Marescaux, J.-Y. Mignolet, R. Pasko, P. Schaumont, D. Verkest: Design of Cam-E-Leon: A Run-time Reconfigurable Web Camera, in Embedded Processor Design Challenges, LNCS Springer 2002.
2. J.-Y. Mignolet, S. Vernalde, D. Verkest and R. Lauwereins: Enabling hardware-software multitasking on a reconfigurable computing platform for networked portable multimedia appliances, ERSA 2002.
3. P. Coene, V. Nollet: RT-Linux Strong-Arm port (<http://www.imec.be/rtlinux>).
4. G. Brebner. A virtual hardware operating system for the Xilinx XC6200, Proc. 6th International Workshop on Field Programmable Logic and Applications, Springer LNCS 1142, 1996, pp 327-336.
5. J. Burns et al.: A dynamic reconfiguration run-time system, Proc. 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines, IEEE, 1997, pp 66-75.
6. W.J. Dally et al.: The J-Machine: A Retrospective, Retrospective in 25 Years of the International Symposia on Computer Architecture - Selected Papers. pp 54-58.
7. W.J. Dally and B. Towles: Route Packets, Not Wires: On-Chip Interconnection Networks, Proceedings of the 38th Design Automation Conference, June 2001.
8. W.J. Dally and C.L. Seitz: The torus routing chip, Distributed Computing, 1:187-196, 1986.
9. J. Duato, S. Yalamanchili, L. Ni: Interconnection networks, an engineering approach, September 1997. ISBN 0-8186-7800-3.
10. Xilinx Inc., An Implementation Flow for Active Partial Reconfiguration Using 4.2i, XAPP290 (v0.5) February 2002.