Co-Synthesis of a Configurable SoC Platform based on a Network on Chip Architecture

Mário P. Véstias
mpv@fidelio.inesc-id.pt
INESC-ID, Lisboa
Portugal

Horácio C. Neto
hcn@inesc-id.pt
INESC-ID, Lisboa
Portugal

Abstract - The constant increase of gate capacity and performance of configurable hardware chips made it possible to implement systems-on-chip (SoC) able to tackle the demanding requirements of many embedded systems. In this paper, we propose an approach to the design space exploration of a configurable SoC (CSoC) platform based on a network on chip (NoC) architecture for the execution of dataflow dominated embedded systems. The approach has been validated with the design of a color image compression algorithm in an FPGA.

I Introduction

Configurable hardware is constantly being upgraded with higher working frequencies and gate capacity that allow the implementation of faster complex systems in a single chip, making it a competitive solution for embedded systems. This gate capacity leads to a complexity challenge needing new architectures and design methodologies to increase design productivity. An approach to the design of such complex systems is to reuse hardware and software blocks resulting in a number of interconnected IP cores. Gajski et.al. [1] have proposed an IP-centric embedded system design methodology and Vahid et al. [2] have proposed a platform based methodology which not only allows reuse of components but also of system architectures and topologies.

Many architectural templates have been proposed for hardware platforms for future SoCs with a general emphasis on providing efficient and standardized communication infrastructures for connecting multiple resources on the chip, like the Network-on-Chip (NoC) [3], [4]. The NoC has been introduced as a new interconnection paradigm able to integrate a many cores while keeping a high communication bandwidth. The increased computational power and internal communication bandwidth of NoC can provide better timing performances to the embedded applications than the shared medium in current SoC architectures.

Some works have contributed with concepts in the area of networks on chip, like [5]. Among the few implementations of NoC, we are mainly interested on the configurable hardware implementations of [6] and [7]. Marescaux [6] have implemented a bidirectional torus in a Virtex/VirtexII FPGA. The network uses 16 bits data packets, the XY routing algorithm, virtual output buffers and supports up to 320Mbits/s at 40MHz with two virtual channels time multiplexed for QoS support. HERMES [7] is a NoC mesh topology implemented in a VirtexII FPGA. The network supports up to 500 Mbits/s at 25MHz without QoS.

Many co-synthesis tools will be required to develop NoC based architectures. Tools to choose the best platform configuration and to map applications to the target NoC architecture will be essential. There has been a lot of research on co-synthesis for bus-based architectures [8], [9], [10]. NoC researchers can adapt many of the techniques and ideas from these approaches for NoC tool development.

Since NoC is a novel research area only a few mapping and scheduling approaches have been developed. Lei et al. [11] use a genetic algorithm (GA) for task mapping and list-scheduling (LS) for task scheduling. The communication is neither mapped nor scheduled and delay is estimated as the average distance between processors. Shin et al. [12] proposed a methodology with network assignment and link speed allocation for reducing communication energy. They use GA for mapping and network assignment and LS for task scheduling and link assignment. To our knowledge, there is not a methodology for the development of NoC based SoC considering all aspects of the co-synthesis process.

In this paper, we propose a co-synthesis methodology with the integration of allocation, mapping and scheduling steps for the development of SoC based on a parameterizable NoC for the execution of dataflow-dominated applications, like multimedia. The platform supports hardware/software multitasking and includes hardware support for the operating system. Increased productivity is achieved through orthogonalization of communication and computation and design reuse. A real multimedia example has been simulated and implemented on a Virtex II XC2V6000 FPGA.

II. CSoC Architecture

Our CSoC platform consists of an array of tiles interconnected with a NoC. The NoC consists of an array of routers (R), where a router is connected to at most four neighbor routers and to a local IP core. Among the many interconnection topologies, we use a 2D mesh topology because it fits naturally in a 2-dimensional chip (see example in figure 1).

0-7803-9451-8/06/$20.00

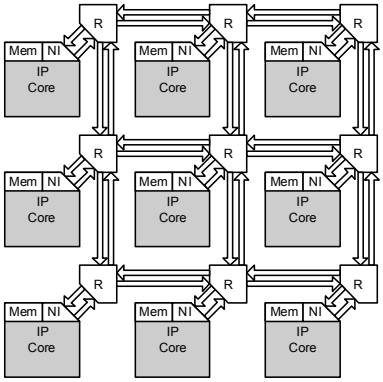

Fig. 1. SoC architecture

A tile consists of an IP core, local memory and a network interface (NI). An IP core is a piece of configurable hardware or a processor. Each core has direct access to local memory and uses the NoC to exchange data with other cores. The link between a router and a core is established with a NI. The platform connects to the environment using IP cores that implement a particular type of interface.

A NoC can be described by its topology and by the strategies used for routing, flow control, switching, arbitration and buffering. Routing determines how a message chooses a path in this graph, while flow control deals with the allocation of channels and buffers to a message as it traverses this path. Switching is the mechanism that removes data from an input channel of a router and places it on an output channel, while arbitration is responsible for scheduling the access to channels and buffers. Buffering defines the approach used to temporarily store messages.

The communication behavior follows a layered approach similar to the OSI communication architecture (see figure 2).

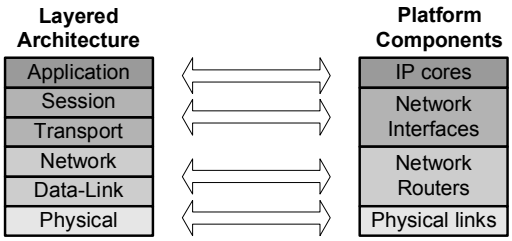

Fig. 2. Communication architecture of the SoC platform

The application layer includes all tasks implemented by a core that consume/produce data. The session layer includes OS services, namely, memory management and task scheduling. The transport layer manages the identification of task ports for the correct end-to-end delivery of data between tasks on different IP cores and the segmentation of data output into packets and reassembly of packets into input data. The network layer includes services for packet routing. The data-link layer includes protocols for reliable data communication between two routers and between a router and an IP core. Finally, the physical layer models the physical links for transmission of bits.

A. Parameterization

The NoC infrastructure has a set of configurable parameters, including:

1) The size of the 2D mesh topology.
2) The type of IP core of each tile.
3) The data width of point-to-point channels between routers. Supported values are 8, 16 and 32 bits.

III. Router Design

A router forwards packets between IP cores. For each packet received, the router reads the destination address and forwards it to the correct output port. Our router consists of a set of input and output ports with buffering and a set of control blocks for routing, flow control, switching and arbitration (see figure 3).

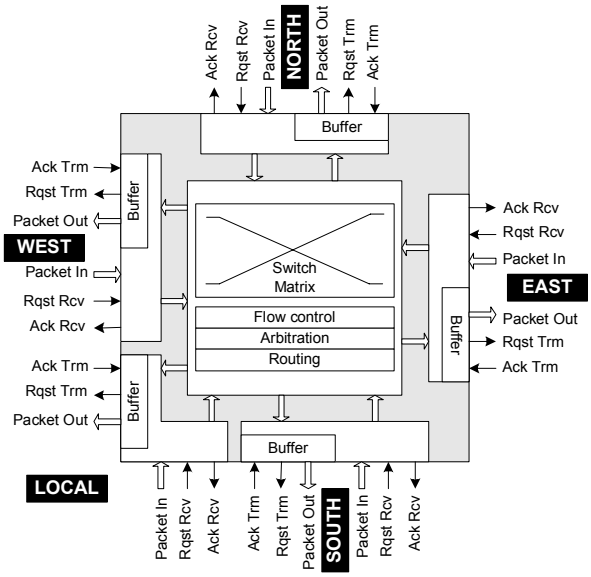

Fig. 3. Architecture of a NoC router

Router ports are used to exchange packets with neighbor routers and with the local IP. A port guarantees the communication reliability through a two-way handshake point-to-point flow control.

The arbitration mechanism uses a round-robin scheme to arbitrate requests from different input ports and grants the output buffer to an input request port. Among the deadlock free routing algorithms for mesh topologies, we implemented the XY algorithm. The XY algorithm routes packets first along the X direction, then along the Y direction until reaching the target. The switching mechanism is based on the store and forward process.

IV. Network Interface Design

The NI consists of an input and an output controller, shared memory ports to connect to the core, a port to connect to the router and an OS memory (see figure 4).

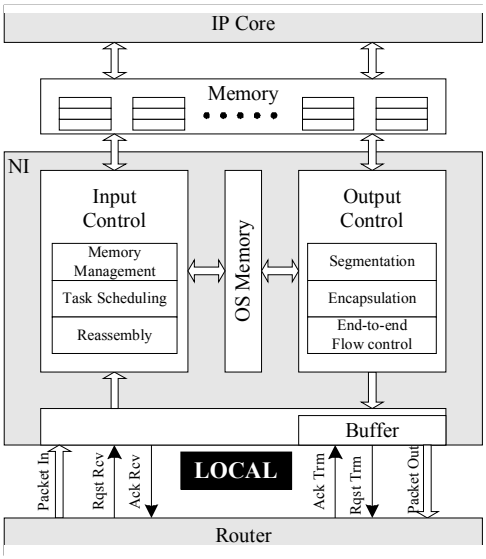

Fig. 4. – Architecture of the network interface

The interface between the NI and the router is identical to the interface between two routers. The interface between the IP core and the NI is implemented with shared memory. The memory may be dual-port RAM or FIFO, depending on the type of the core. For a processor core, the shared memory is always implemented with dual-port RAM. For a hardware core, it may be any of the two kinds of memory. The NI also performs data split and data merge to transmit packets with a data size different than the link size.

The input controller receives data from the router and sends it to the shared memory to an address defined by the port number of the packet. For data transmitted in multiple packets, the controller stores each packet in sequential memory positions until the complete data port is available. When all data of the input ports of a task are available and the corresponding output port is free, the input controller sends a token to the core indicating the task can be executed.

The output controller reads data from the shared memory and sends it in one or more packets. It also implements an end-to-end dataflow control, that is, when the input buffers of a task are available, the output controller sends a token to the producers indicating that they can send more data.

V. Packet Structure

Data to be transmitted is encapsulated in packets at the transmitter and deencapsulated at the receiver. Besides data packets, our NoC implementation uses configuration and token packets. Configuration packets are used to configure the NI and token packets are used for end-to-end flow control.

The highest level of the packet structure is at the application layer where data is produced. If data is too large to fit into one packet it will be divided into several packets. Each packet is encapsulated with its type at the session layer. Next, the transport layer adds the destination port and the network layer adds the destination address.

VI. CSoC Performance Evaluation

We have conducted a simulation of a prototype of the NoC to characterize the following set of parameters (see table 1):

Link latency (LL) - the delay for a packet to move from the output of a router to the output of a neighbor router;

Resource generation latency (RGL) – the delay for a NI to generate a packet;

Resource reception latency (RCL) – the delay for a NI to consume a packet;

Resource to resource latency (R2RL) – the delay for moving a packet from one IP core to a neighbor IP core;

Resource to resource bandwidth (R2RB) - the transmission throughput between IP cores.

TABLE I
NoC Characterization

LL	RGL	RCL	R2RB
1 cycle	4 cycles	5 cycles	$\frac{f(frequency)}{5}$ Packets/s

From these parameters, we calculate the communication delay of a packet between two tiles. In a NoC, the communication delay depends on the distance between tiles, the size of transmission data and the network traffic. In the execution of an application, it is possible to have many concurrent transmissions, which conflicts in the use of the communication resources. If tasks have variable execution times, the network traffic is non-deterministic, which makes the analysis more difficult. To simplify, we assume that at any time a link is dedicated to a single data transfer. Hence, the transmission delay of a packet between two tiles separated by NR routers, *EdgeDelay*, is given by:

$$EdgeDelay = \frac{RGL + RCL + LL \times NR}{f(\text{working frequency})} \qquad (1)$$

Since the packets are buffered at the routers, the transmission of data requiring more than one packet can be pipelined. In this case, the transmission delay, $EdgeDelay_{pipe}$, of NP packets is given by:

$$EdgeDelay_{pipe} = \frac{1}{R2RB} \times (NR + NP) \qquad (2)$$

Besides performance, we have also determined the maximum area occupied by a router and a NI after the synthesis and placement of the components on the target FPGA with the Xilinx ISE 6.2i software (see table 2).

TABLE II
Slice Areas of the Interconnection Components

Block	Size (slices)	BRAM	% XC2V6000
Router (8bits)	189	0	0,56
NI	121	1	0,36

A router can forward five packets per clock cycle at 150MHz. Therefore, a router can forward at most 6Gbps.

VII. CSoC Co-Synthesis

To configure our CSoC architecture for a specific application, we have developed a platform-based co-synthesis methodology. It finds a hardware/software architecture that runs the application with optimized performance and meets the design constraints (see figure 5).

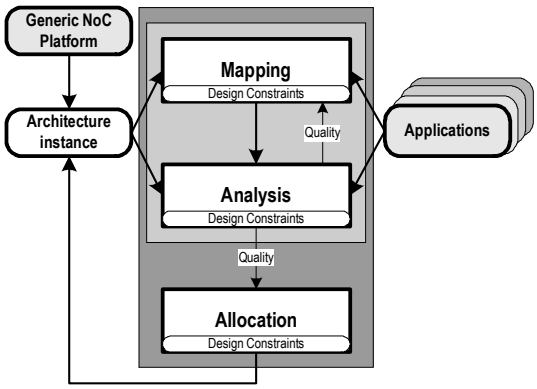

Fig. 5. Co-synthesis flow

It starts with an architecture instance, maps the application onto the architecture and uses the analysis step to determine the quality of the architecture based on cost and performance metrics. The analysis yields quality values that together with design constraints over cost and performance guide the allocation to improve the architecture.

A. Application Model

The applications are modeled with an iterative dataflow graph (IDFG) that can represent iterative behaviors. This model is a directed cyclic graph G = (V, E), where each vertex $v \in V$ represents a task (atomic computation) and each edge $e \in E$ represents intra or inter data dependencies between tasks.

A task vertex has associated its worst and/or average case execution time, data, and program memory size or hardware area in each of the available cores. An edge has an associated vector (v,d), where v is the data size to be transferred and d is the inter delay between two connected tasks.

B. Allocation

The allocation step determines the most appropriate SoC architecture from the generic CSoC platform for the execution of the application. It determines the size of the NoC topology and the type of core associated with each tile. This is a hard problem. Therefore, we have used an heuristic (see figure 6).

From the generic CSoC platform, the algorithm instantiates an initial set of IP cores. From this set, it generates an initial architecture. To generate an architecture from a set of cores, the algorithm reads sequentially the set of tiles with its IP core association (including the NI) and fills the FPGA (including the routers) until it is full or all tiles are associated.

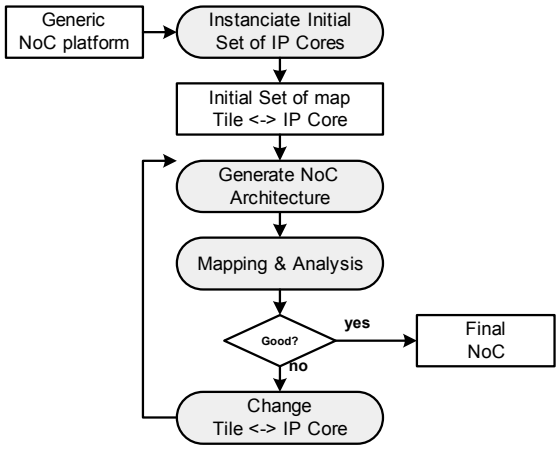

Fig. 6. Allocation algorithm

Next, it maps the application and finds the quality of the architecture with the analysis tool. If the quality is considered acceptable, the algorithm stops. Otherwise, it changes the IP core of a tile and restarts the evaluation flow. The iterative process of the algorithm is controlled with a simulated annealing (SA) algorithm [13] as follows:

Move generation function: generates a new architecture instance from the previous by changing the IP core of a tile.

Cooling schedule: the cooling schedule includes the initial temperature, t0, the decrement rule for the temperature, the stop criterion and the length of the Markov chain. t0 is obtained by incrementing the temperature until the percentage of accepted transitions is higher then 70%. The decrement rule is given by $tk = t0 \times 0.95k$. The algorithm stops when three consecutive Markov chains end with the same value. The length of the Markov chain is equal to the size of the neighborhood.

Cost function: obtained with the mapping and analysis steps.

C. Mapping

Mapping consists on assigning each application object (task, data transfer and variable) onto an architectural element (IP core, link and memory, respectively) in order to maximize the quality of the architecture while satisfying design constraints. Even for small instances, the mapping problem has exponential complexity so that heuristics must be used.

Our mapping approach uses SA to improve an initial solution found with LS while exploring the advantages of both pipelining and unrolling to increase the throughput. Pipelining allows tasks belonging to different iterations to be executed at the same time and unrolling increases the number of tasks within an iteration to explore more parallelism. The NoC structure can easily implement pipelining since the output of a task is buffered and the flow of data is easily controlled by the end-to-end flow control of the NI.

The algorithm starts with the IDFG of the application and a NoC architecture and iteratively runs a mapping design space exploration step with different unrolling values (U) (see figure 7).

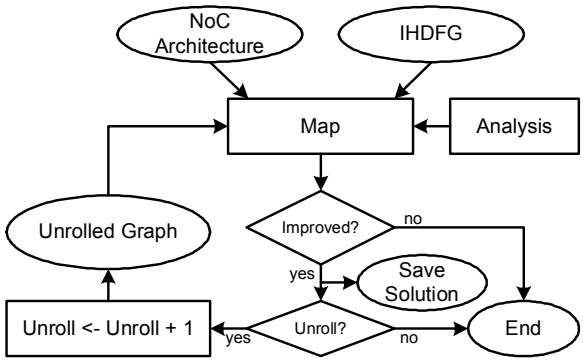

Fig. 7. Mapping algorithm

The iterative process is controlled with SA using the same cooling schedule of the allocation process. Each mapping solution is evaluated with the analysis tool.

D. Analysis

The analysis finds the quality of an architecture based on its performance, cost and memory requirements.

For performance evaluation, it uses LS to order the execution of tasks assigned to a single core to optimize the throughput of the graph, $C_{throughput}$. To schedule tasks on software processors, the algorithm assumes the processor executes one task at a time and the program code of the task is in local memory before starting its execution.

The memory requirements of a CSoC architecture depends on the local memory requirements of each IP core. The local memory of a core is used to store the program instructions of tasks (for a software core), the OS data of the NI and the tasks data. We assume the local memory is implemented with BRAM and the instruction, the OS and data memories use independent BRAM.

To calculate the memory utilization of a core, the analysis uses a table with the instruction memory size of each task on each software core and the size of input and output ports of each task. The analysis process determines the number of local BRAMs, BR, necessary to implement core *k* as follows:

$$BR(k) = \frac{\sum_{i \in \{coreTasks\}} instrMem(i)}{BRAMSize} + \frac{2 \times \sum_{i \in \{coreTasks\}} \sum_{j \in \{taskPorts\}} portSize(i,j)}{BRAMSize} + 1 \quad (3)$$

where *instrMem(i)* is the code memory size of task *i*, portSize(i, j) is the data size of port *j* of task *i*. The total number of BRAM used, C_{memory}, is given by:

$$C_{memory} = \sum_{i \in \{setofcores\}} BR(i) \quad (4)$$

The final quality of the architecture is given by $C_t + C_m$ where

$$Cx = \begin{cases} K \times \dfrac{Cmetric(P)}{\overline{Cmetric}}, & \text{w/o constraint} \\ Ka \times \left| \dfrac{(Cmetric(P) - Cmetric\ Constra\ int)}{CmetricCon\ stra\ int} \right|, & \text{w/ constraint} \end{cases} \quad (5)$$

where K and Ka are weighting factors specified by the user with typical values of 0.5 and 100, respectively. *Cmetric*(P) is the metric value (throughput or memory) of a solution P and $C_{metricConstraint}$ is a metric (throughput or memory) constraint. For a non-constrained metric, we use the average value ($\overline{Cmetric}$) calculated from the values of the metric in some (<20) previous solutions.

VIII. Design Evaluation

This section describes the design of a JPEG encoder with the proposed CSoC environment. A simulation has been executed and a prototype is under development based on the Xilinx VirtexII XC2V6000.

A. JPEG Encoder

The standard JPEG compression with a block size of 8×8 pixels for color images was implemented. The compression method used is based on the DCT (see IDFG of the JPEG encoder in figure 8).

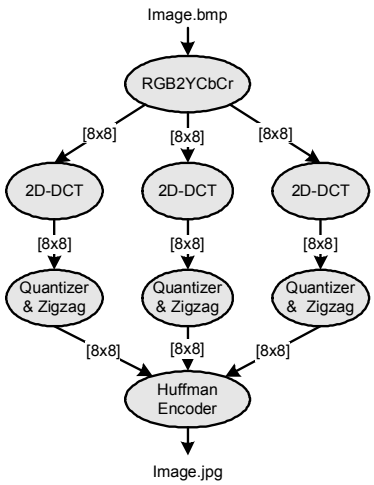

Fig. 8. IDFG of the JPEG encoder

The tasks of the JPEG where characterized considering a hardware implementation (see table 3).

TABLE III
JPEG Hardware Task Characterization

Task	Size (slices)	BRAM	Latency
RGB2YCbCr	204	0	64
2D-DCT	1612	1	168
Quantizer	312	1	64
Huffman	176	1	192

For this application, the co-synthesis tool took less than 5 minutes to found a hardware only solution that can process two blocks of [8×8]×24 bits in 3.8 μs (cores frequency = 100MHz, NoC frequency = 150MHz with 16 bit packets), which is equivalent to a processing capacity of 800 Mbps. With this throughput, we can process color images at the processing times of table 4.

TABLE IV
Execution Time of Hardware Solutions

Image size	HW solution (NoC) seconds (fps)	Pentium 4 at 1.7 GHz	HW solution (bus) seconds
640×480	0.009 (108)	0.046	0,055
800×600	0.015 (67)	0.071	0,086
1024×768	0.024 (42)	0.110	0,14

The JPEG solution was placed & routed and simulated on the FPGA with the Xilinx ISE 6.2i software (see figure 9).

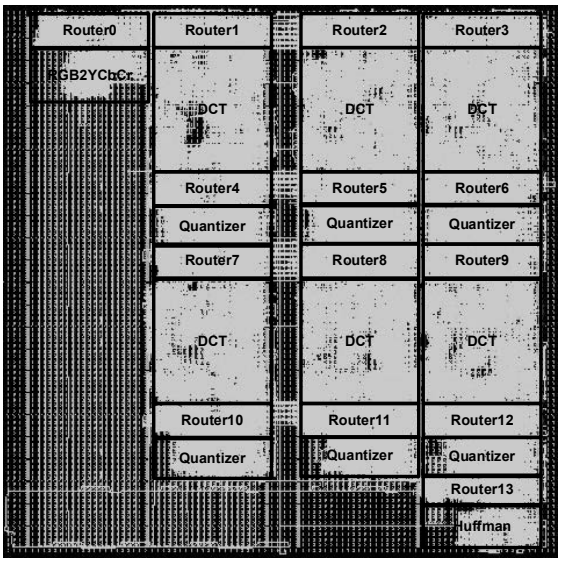

Fig. 9. FPGA implementation of the JPEG case study

In the figure, we have outlined all resources used in the implementation (13877 out of 33792 slices, 45%). We can see the co-synthesis tool have unrolled the graph one time, almost doubling the processing rate. The design was easily routed because of its regularity and the mapping constraints over the IP cores, the routers and block RAM.

From this and other results, we conclude the following:

- Designs can be quickly and easily designed without bus design complications. The same design with a single bus could not achieve the same throughput.
- For certain cores (Huffman), a router uses more slices. This area overhead may become a serious bottleneck for NoC architectures. Other NoC topologies with shared routers and local memory are being considered.
- Many NoC parameters can and should be explored with the cosynthesis tool, including buffer size, switching capacity, routing algorithm and arbitration policy. Adjusting these parameters to specific applications means smaller size routers and consequently less overhead.

IX. Summary and Conclusions

The well-structured design of the CSoC platform and the acceptable computation times of the co-synthesis tool allow the rapid development of SoC architectures.

Our approach has been used to design a JPEG application with a throughput of 800 Mbps. The results are very promising since we where capable of easily integrate several IP cores in a single chip and obtain high quality solutions.

Future research includes developing a more flexible CSoC with different network topologies and including a generic parameterizable router as part of the design space exploration in order to improve the area, performance and energy dissipation of the final SoC architecture.

Acknowledgments

The authors thank the support granted by INESC-ID.

References

[1] D. Gajski, R. Dömer and J. Zhu, “IP-Centric Methodology and Design with the SpecC Language”, in *System Level Design*, Nato Science Series 357, 1999.

[2] F. Vahid and T. Givargis, “Platform Tuning for Embedded Systems Design”, in *IEEE Computer,* 34, 3.

[3] W. Dally and B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks”, in *Proc. of DAC*, 2001.

[4] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg and D. Lindqvist, “Network on Chip: An Architecture for Billion Transistor Era”, in *Proceedings of the IEEE NorChip Conference*, Nov. 2000.

[5] L. Benini and G. de Micheli, “Networks on Chips: a New SoC Paradigm”, in *Computer*, v.35(1), Jan. 2002, pp.70-78.

[6] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, R. Lauwereins, “Interconnection Networks Enable Fine-Grain Dyanmic Multi-Tasking on FPGAs”, in *Field-Programmable Logic and Applications*, 2002, pp. 795-805.

[7] F. Moraes, N. Calazans, A. Mello, L. Möller, L. Ost “HERMES: an Infrastructure for Low Area Overhead Packet-Switching Networks on Chip”, in *Integration*, the VLSI Journal 38, 2004, pp. 69-93.

[8] R. Dick and N. Jha, “CORDS: Hardware-Software Co-Synthesis of Reconfigurable Real-Time Distributed Embedded Systems”, in *Proc. of ICCAD*, pp. 62-68, 1998.

[9] R. Szymanek and K. Kuchcinski, “Design Space Exploration in System Level Synthesis under Memory Constraints”, in *Proceedings EuroMicro*, pp. 8-10, 1999.

[10] U. Shenoy, et al., “A System-Level Algorithm with Guaranteed Solution Quality”, in *Proceedings of DATE*, pp. 417-422, 2000.

[11] T. Lei and S. Kumar, “Algorithms and Tools for NoC Based System Design”, in *Proceedings of SBCCI*, 2003.

[12] D. Shin and J. Kim, “Power-Aware Communication Optimization for Networks-on-Chips with a Voltage Scalable links”, in *Proceedings of CODES+ISSS*, pp. 170-175, 2004.

[13] S. Kirkpatrick, C. Gelatt, and M. Vecchi, “Optimization by Simulated Annealing”, in Science, 220(4598): pp. 671-680, May 1983.