HW-SW framework for distributed parallel computing on programmable chips

Jaume Joven Murillo, David Castells-Rufas, Jordi Carrabina Bordoll

CEPHIS-UAB (Universitat Autònoma de Barcelona)
QC2034. Edifici ETSE. Campus UAB 08193 Bellaterra, Spain
email: jaume.joven@uab.es, david.castells@uab.es, jordi.carrabina@uab.es

***Abstract*— This paper addresses a new hardware/software Multi Processor System-on-a-Chip (MPSoC) co-design methodology to map PVM/MPI parallel software framework to a developed multiprocessor architecture for distributed parallel computing on a chip. This methodology is composed of two concurrent phases. The first one is the software development of the embedded parallel framework for on-chip platforms. Parallel Virtual Machine (PVM) and Message Passing Interface (MPI) are two traditional software frameworks for distributed parallel computing, so we need to translate one of them towards the on-chip environment. The goal of the second phase is to develop distributed parallel on-chip hardware architecture, based on Multiprocessor System-On-a-Chip (MPSoC) that includes a Network On-a-Chip (NoC) strategy, together with the corresponding distributed memory subsystem. Performance measurements for this hardware architecture have been obtained by synthesizing it for reconfigurable platforms (FPGA).**

Once the parallel software framework is loaded into each processor of our hardware architecture, we can develop typical parallel distributed applications to run over this SoC/NoC platform. The result of this work will be a complete and flexible parallel HW/SW on-chip platform for an embedded distributed computing.

***Index Terms*— Embedded computing, Distributed Parallel Computing, Multi-processor System-on-chip (MPSoC), Networks-on-chip (NoC), Parallel processing, Parallel Virtual Machine (PVM), Message Passing Interface (MPI).**

I. Introduction

This section discusses fundamental concepts together with an extended introduction to understand the parallel processing design methodology.

A. *Evolution of SoC hardware architectures*

Nowadays, the Moore's law gives the opportunity to the designer to quickly develop complex SoC platforms that can use efficiently all available technologic resources. Traditional SoC [1] architectures are composed by three generic elements:

1) Standard system bus, also called on-chip bus (OCB): For example: AMBA, Avalon, CoreConnect or Wishbone.
2) Set of virtual components (IP cores): processors, peripherals, application specific, etc. For our framework, the main IP core will be a soft- or hard-IP processor.
3) Memory subsystem: on-chip RAM, off-chip SRAM (SDRAM, SSRAM, etc.) and Flash or ROM memories.

Nowadays we have a great variety of high capacity programmable chips, also called reconfigurable devices (FPGAs) where we can easily integrate complete SoCs architectures for many different applications. Due to the inherent flexibility of these devices (Field-Programmable), designers are able to quickly develop and test several hardware/software architectures. In this paper, we used Altera reconfigurable devices to implement the hardware-software architecture and get experimental results. Moreover, the proposed methodology is based on the use and development of hardware and software components and architectures, either virtual or physical, so that they can be easily mapped into different complex physical implementations, i.e. FPGAs, ASICs, PCBs, etc.

At present, Altera and Xilinx are main vendors of complex reconfigurable devices and they (either directly or through third parties) provide a set of tools in order to develop the application specific SoCs architectures. In our case of study that SoCs architectures will be more complex in order to reach the implementation of complex MPSoC [2][3][4] architectural implementations. The singularity of our approach is that the SoC will be composed by multiple independent tiles (isochronous regions) composed by mono-processor architectures into the same silicon, with small on-chip distributed memory memories and with large external off-chip distributed memories, thanks to the large I/O pin-count of FPGAs. Therefore, the number of processors we can integrate in the MPSoC architecture will be limited mainly by FPGA resources.

In addition, as we are going to explain later, the proposed MPSoC architecture must include a feasible on-chip interconnection strategy (segmented bus approach is not possible), so that it can be applicable to NoC architectures [5].

B. Fundamentals on parallel computing

There exist several traditional parallel architectural frameworks for conventional parallel computation systems, according to the memory models that these architectures use [6], as shown in table 1.

Memory Models	Parallel architectures	
	UMA	NUMA
Shared Memory	SMP (Symmetric Multiprocessors)	ccNUMA (NUMA with cache coherency)
Distributed Memory	Not used	MPP (Massive Parallel Processing)

Table I. Parallel architectures according to memory models

There are two main parallel architecture paradigms:

1) UMA (Uniform Memory Access): in this architecture memory access time is independent from which processor makes the request or which memory contains requested data. Usually, we use this architecture on SMP systems where normally use a global shared memory for all processors.

2) NUMA (Non-Uniform Memory Access): this architecture is used in multiprocessors where the memory access time depends on relative memory location to specific processor. An example of this architecture is ccNUMA. Under this architecture, any processor can access its own local memory faster than non-local memory, usually a shared by all processors. Opposite to NUMA, with shared memory, are the distributed memory systems. These systems are also called massive parallel processing (MPP) systems because many processors are used to work together in a distributed way.

The main disadvantage of UMA architectures against NUMA is scalability. So, under UMA architectures when number of processors increases over a threshold the system does not scale well.

For this reason, and due to the fact that the ccNUMA parallel architecture will be already integrated on a chip with soft core processors with shared memory [7], in this paper we focus on NUMA architectures with distributed memory model. In addition, as previously commented, we face a complete MPSoC/NoC hardware design, but also a software design. The result is a complete platform-based design methodology [8][9] to build distributed parallel computing in the on-chip environment.

II. MPI/PVM SOFTWARE FRAMEWORK

Nowadays, all multiprocessor systems for parallel computing are clusters of processors that are interconnected through a network infrastructure, usually any type of Ethernet, and they use message passing protocol to communicate.

The Parallel Virtual Machine (PVM) [10] and Message Passing Interface (MPI) [11][12] are frameworks defined in the 90s as standard software libraries to describe parallelism for heterogeneous concurrent computing in networked environments. In other words, these software frameworks were designed to allow a network of heterogeneous processors to be able to collaborate together to solve huge computation problems.

These software frameworks for parallel distributed computing also define other relevant concepts: hardware architecture (NUMA with distributed memory), the communication model (message passing) and also the computing taxonomies (Multiple Instruction Multiple Data or MIMD) [6]. The software framework is based on message passing, so that communication protocol is based on transmitting messages over an underlying network using the fastest available mechanism, for example, on Internet environments, either UDP or TCP can be used.

That point is really important for the on-chip mapping, because when the communication across the underlaying network runs on an on-chip interconnection mechanism, we can reach a higher bandwidth (around Gbps [13]) between every on-chip processor.

These two parallel software frameworks, PVM and MPI, are quite similar but are not exactly the same. Both define a communication protocol among the processors running a parallel program on distributed memory systems, being PVM the original one and MPI its evolution that nowadays is still being actively developed. Therefore, the MPI framework should be considered a priori as a better choice for its better porting facilities to other environments (in our case on chip environment).

III. COMPONENTS OF A GENERIC ON-CHIP ARCHITECTURE FOR DISTRIBUTED PARALLEL COMPUTING

Once presented both the software framework (section II) and a quick view of hardware architectures (section I), it is necessary to detail the whole set of components needed to generate the on-chip distributed parallel computing architecture. All required components are:

1) *Soft/Hard IP core processors (Pi)*
2) *Distributed memory subsystem (Mi)*
3) *Network Interface Controller (NICi)*
4) *Network interconnection (routers, switch, crossbars...)*
5) *Porting of parallel software framework*

Items 1 to 4 are hardware components whereas the last one is a software component. Figure 1 shows a block diagram of the structure and connectivity of components in our distributed parallel processing architecture.

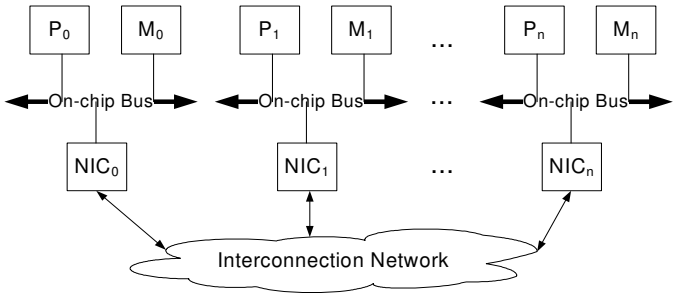

Fig 1. MPSoC/NoC architecture for distributed parallel processing

As commented, in order to map this architecture into experimental platforms, we used Altera FPGAs and, consequently, we selected its NIOSII as soft core processor. NIOSII is a flexible processor (in terms of FU, pipe-line, peripherals, memory, etc.) optimized for these reconfigurable devices. The most important configurable properties considered in our work are cache size and processor width architecture (16 or 32 bits). User can also add custom instructions to be implemented in optimized functional units into the processor data-path.

Furthermore, NIOSII contains a 5-stage pipeline, and uses Harvard memory architecture with separated bus masters for data and instruction memories. This processor offers a throughput around 60-90DMIPS@80MHz and needs about 1200-1800 LEs of FPGA (aprox. 25kgates) resources depending on the specific architecture selected (economic, standard or full feature core).

Usually, when selecting an embedded processor there is an attached on-chip bus to build the SoC architecture. In our case, the system bus of all processing elements (PE) is the Altera's Avalon bus. It is a multiplexed multi-master on-chip bus, so we can address multiple masters through an arbiter. Altera is also providing a tool (SOPC builder) to easily build a complete NIOSII tile architecture with its Avalon bus.

In addition, this tool allows changing the whole set of Avalon bus properties (boot device and its memory address, memory mapping of all IP cores, assignments and priority levels of interruptions, …).

The unique software component of our methodology is the parallel software framework that implements the message passing protocol over the hardware architecture. This software will be executed inside the RAM memory available in every processor and interacts with the network interface controller through the Avalon bus in order to communicate (send, receive) all software messages across the interconnection network.

The next component in our architecture is the memory subsystem. The RAM memory could be either on-chip RAM available as structural FPGA resources or external RAM (as the SSRAM available on the prototyping board used for our experiment). Depending on the memory architecture, users will get different system performance. When using on-chip RAM better performance is achieved, but this is a strongly limited resource. On the other hand, if we use external SRAM, there are more resources available but we have worse efficiency on each memory access.

The network interface controller (NIC) will be designed at hardware level, i.e. using existing hardware description languages (HDL), like Verilog or VHDL, and must be connected to the Avalon bus (detailed in section IV). NIC are sometimes referred as resource network interface or RNI in NoC literature.

For our experiments, any PE will also have other components on the Avalon bus, like JTAGs and UARTs for software debugging and message printing through the console; and timers or performance counters to acquire performance information about the system (software time execution, number of clocks spent by a piece of software, etc).

Finally, the interconnection network is the structure that joins all pieces of this special puzzle, MPSoC/NoC architecture. There are many ways to develop it, but its implementation depends on the system requirements and the number of processors that we integrate in its MPSoC/NoC solution. Typical network interconnection can be a channel with handshake protocol between two sender-receiver processors, or a crossbar interconnection network with dedicate switch or router.

We must remark that the architecture shown in figure 1 is valid for any type of embedded processors (ARM, MIPS, MicroBlaze, PowerPC, Leon, …), on-chip buses following VSIA specifications (AMBA, CoreConnect…) and architectural development tools (UltraWizard [14] to develop ARM-AMBA based MPSoC designs, Platform Express, Xilinx ISE,…).

For any case, hardware components for the parallel distributed architecture are almost identical, and the proposed methodology is still valid.

IV. HARDWARE DESIGN: MULTIPROCESSOR SYSTEM ON A CHIP IMPLEMENTATION BASED ON NETWORK ON A CHIP INTERCONNECTION

This section presents the hardware architecture. Hardware design is essentially based on building the MPSoC based on NoC interconnection. As commented, it is easy and quick to build a SoC based on NIOSII with Avalon bus. Therefore, we only need to develop the NIC IP core for our Avalon bus and all the NoC communication resources (switch/router).

A. NIC implementation

The NIC implements from the hardware point of view, the interface between the mono-processor system bus (tile) and the interconnection network. From the SW point of view, it is the manager of messages between the software framework and the interconnection network.

Therefore, the NIC implementation will be designed at hardware level (Verilog HDL code) and it has a strong relationship with the porting of the parallel software framework over an on-chip environment.

There are two different ways to develop a NIC inside the on-chip bus:

1) NIC Slave: when NIC receives data from processor, relays it to interconnection network resource. When NIC receives data from interconnection network resources it must notify that this data is available and wait for the processor to acquire it. In this implementation there are two mechanisms to notify the processor that a packet has been received. One based on processor polling and the other in interrupts (up to 64 levels of ISR are available in Avalon).

2) NIC Master-Slave: in this implementation when the NIC receives data from the processor, the process is the same as in previous NIC slave implementations. The difference appears when it receives data from interconnection network. Then the NIC becomes a bus

master, does a bus request, takes the bus, writes received packet to the memory location that the processor assigns, and notifies that its data is available.

Figure 2 shows the actual NIC composed of two registers to transmit and receive data respectively, and also the status and control registers to manage and get the NIC status at any moment. The width of transmit and receive registers depend on the on-chip bus width (in our case fixed to 32 bits of the Avalon bus) and XY address location width of our NxN Mesh Interconnection Network.

In addition, the NIC also contains a FSM to control all internal register signals and also the two interfaces: the Avalon bus interface and the router/switch interface. The router/switch interface has a 4-phase handshake protocol with request/acknowledge signalling and txData/rxData buses. The bus interface is a conventional OCB master or slave interface, depending on which NIC's implementation has been used.

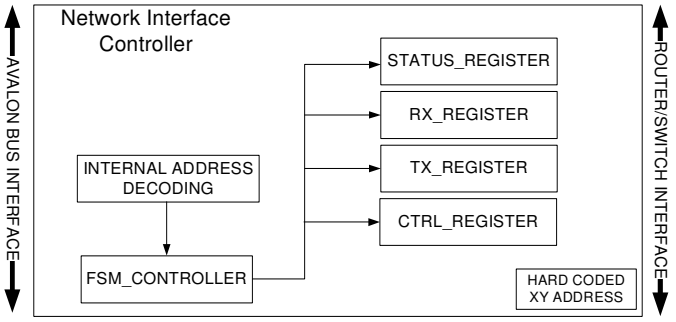

Fig 2. Generic bloc diagram of network interface controller

Therefore, NICs must be partially specifically designed for every specific on-chip bus. Our NIC core has a master or slave Avalon interface and it has been designed taking into account its portability to other OCBs, according to the VSIA recommendations.

B. MPSoC/NoC design

The MPSoC/NoC [5][13][15] hardware design is more complex and it is based on the idea to map a network infrastructure on the on-chip environment in order to create a complete distributed parallel computing architecture. Mapping components to the distributed parallel architecture is straight forward. Every tile of the MPSoC/NoC will have its own NIOSII embedded soft core processor with its NIC, and within the same tile or inside another tile we mapped the RAM memory. On the other hand, interconnection network resources must include a switch or router to communicate the NoC tiles. On that kind of architectures a large number of processors can be interconnected through a complex communication protocol.

Main packet-switching protocols [16] used in MPSoC/NoC literature are:

1) Store & Forward Packet Switching: each switch/router must receive the whole packet before relaying that packet to the next switch/router.

2) Virtual Cut Through Packet Switching: each switch sends pieces (flits) of the whole packet, every time a packet arrives to a switch/router.

3) Wormhole switching: this protocol reserves a virtual channel (wormhole) during a limited space of time in order to transmit packets.

To generate an efficient architecture, it also is important to select the optimal topology (Star, Mesh, Torus 2D, Hypercube, Butterfly…) according to the specific parallel application we want to run.

For each topology, we must dispose of a specific switch/router interface [17][18][19] in order to communicate all tiles composing our MPSoC/NoC architecture. So, the routing resource implementation depends on the selected topology. In this work, we initially developed a 2x2 Mesh NoC architecture (based on tiles composed of NIOSII and Avalon bus) using Virtual cut Through Packet Switching, as shown in figures 3 and 4.

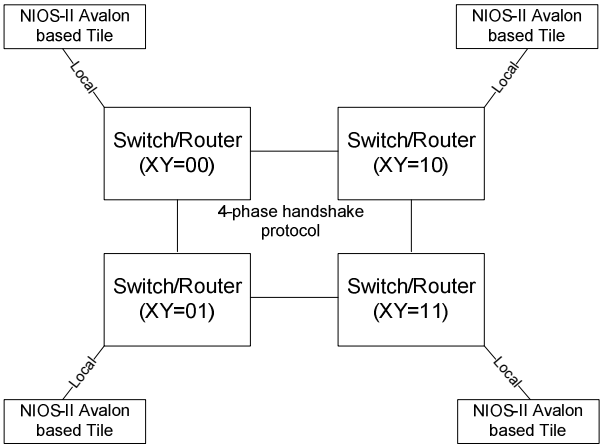

Fig 3. 2x2 Mesh NoC based on NIOS-II MPSoC

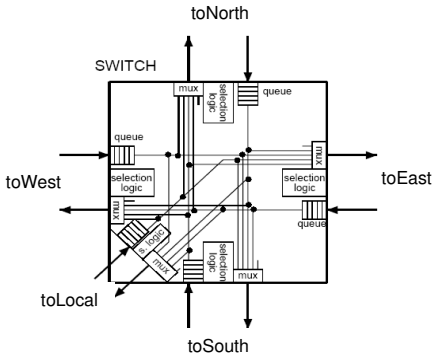

Fig 4. Switch/Router for Mesh NoC architecture

In general, the Mesh topology is based on the fact that each tile is able to communicate with their four neighbours. To do that, the switch interface (that uses also 4-phase protocol) must be defined with five ports or interfaces: north, south, east, west and local. The local one is the port that communicates to its attached tile.

In addition, our Mesh interconnection network is processor independent, and it can be also parameterized according to different criteria (type of connection, network topology, packet size, buffering capabilities, etc.).

This type of MPSoC/NoC is practically a conventional network, so the idea to develop a methodology to implement distributed parallel computing has the same importance than in the classical processor platform domain.

V. SW DESIGN: PORTING OF PARALLEL SOFTWARE FRAMEWORK FOR EMBEDDED ON-CHIP ENVIRONMENTS

Both PVM and MPI parallel software frameworks described in section II are based on TCP/IP Protocol because the conventional distributed parallel systems usually transmit messages over Local Area Network (LAN) environments. In the MPSoC/NoC case, the interconnection between software and NIC goes through the Avalon bus and a more or less complex NoC structure.

Therefore, we have done the porting of the lower layers of the selected software framework, to manage differently the message passing mechanisms to the interconnection network through the NIC. Finally, by using this porting, from the application perspective, the underlying network implementation will be transparent. In figure 5, we show the porting process from MPI to an embeddedMPI software framework on top of the OSI model [20].

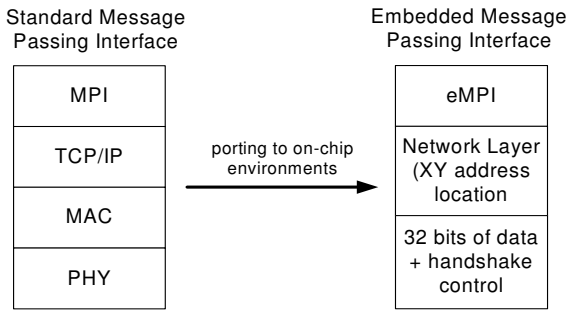

Fig 5. On-chip embedded MPI OSI protocol vs Standard MPI comparison

In MPSoC/NoC architectures, the transport layer is technology independent and the packet size has usually variable size. This layer is in charge of managing the packet building at NICs for data coming from eMPI layer through the network layer. The network layer defines the way the packet is transmitted through the interconnection network from any producer to a given receiver, by obtaining the corresponding information from the network assigned address (XY address location inside MPSoC/NoC). Data-link and physical layers define the signals for the physical interfaces and the packet formats at bit level.

Detailed examinations of MPI code [11][12] let us to classify the different routines offered for message passing: point to point communication (blocking and non-blocking), collective communication (broadcast, gather, scatter,…), etc. Those functions have a standardized definition and prototype (MPI_XXX) that should not be modified in order to keep portability. In our work, we selected a basic subset to be used on most of the parallel programs.

This basic subset porting functions are composed by these six functions:

1) *MPI_Init():* initializes MPI software framework.
2) *MPI_Finalize():* ends MPI software framework.
3) *MPI_Comm_size()*: initializes the number of concurrent processes to run over the network.
4) *MPI_Comm_rank():* initializes each nodes with our rank inside MPI software framework.
5) *MPI_Send():* blocking send that sends messages over the underlaying network.
6) *MPI_Recv():* blocking receive that receives messages from underlaying network.

VI. EXPERIMENTAL RESULTS

This section presents the results obtained using several test traffic generators over the experimental architectural platform composed of NIOSII processor, Avalon bus and the presented MPSoC/NoC infrastructure. The physical platform used is an Altera development kit, based on an EP1S25 Stratix FPGA device, containing 25.660 LEs and 1.944.576 bits of on-chip RAM.

The development board also includes two independent banks of 1MB of external SRAM, 32Mbits of flash memory and other resources.

Our 2x2 Mesh MPSoC architecture that includes a NoC strategy to communicate all the tiles requires 15.120 LEs (58% of FPGA resources). On this architecture, we achieved a message passing of 10Mbps running our NIOSII at only 20MHz. With this devices and parameters, Altera tools estimate a power consumption of 1092mW that lets to a power efficient implementation compared with standard parallel applications. In addition, our embeddedMPI parallel software framework contains around 1500 C/C++ source code lines and takes small overhead of memory footprint usage. For each NIOSII processor, we need a RAM memory of at least 64KB per processor to download the parallel software framework.

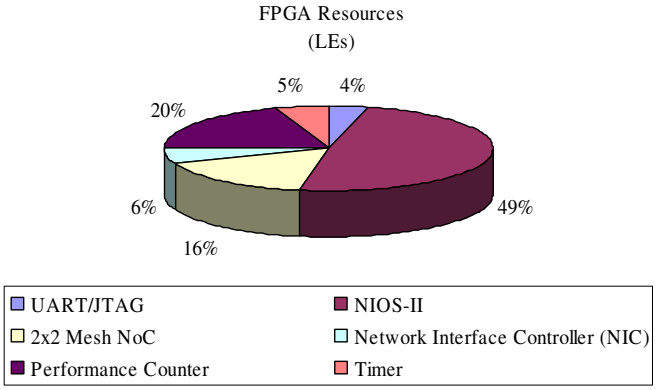

Fig 6. Hardware area distribution of our 2x2 Mesh MPSoC/NoC

VII. Conclusion and future work

The presented hardware/software co-design methodology to develop distributed parallel computing architecture on-a-chip is a feasible alternative to increase computational capabilities. Furthermore, it can be easily mapped on FPGA devices.

Nowadays, a lot of huge computational problems exist and must be parallelized in a distributed way in order to quickly get the results, even in the embedded applications domain. Some of those problems, like distributed fractal generation, molecular dynamics simulations, superconductivity studies, or matrix & sorting algorithms could be efficiently computed in embedded on-chip environments. For this reason, the methodology presented and developed in this paper is shown as a good alternative approach to solve these special problems on embedded environments.

With this methodology, we improve embedded on-chip system performance because processor tiles can be added when HW (FPGA) resources are available. This is possible due to the fact that NUMA distributed memory systems scale properly. A further optimization work will be devoted to the memory access bottleneck. Nowadays, in our experiments from memory to NIC we spend 24 clock cycles, whereas in pipelined burst accesses we should greatly improve this number that directly affects performance and message passing bandwidth.

Mixing well-known modern hardware on-chip architectures, MPSoC/NoC, together with standard parallel software frameworks, PVM/MPI, in a application-specific design methodology produces robust and capable platform with high reuse capabilities for the research community.

HW and SW overheard will mainly depend on the NIC infrastructure and topology, but our experimental results are showing that these are small enough compared to the computational structures (22% versus 53% area occupation).

In addition, the use of FPGA devices and associated tools to design and validate architectures eases the hardware design process, so we don't need to spend that much time to develop both general purpose and application specific (compared to ASIC developments) platforms.

Next step and future work in this methodology is to map architectures in virtual platforms using SystemC modelling languages [21]. Simulation and modelling elements available in SystemC [22] (sc_fifo, sc_thread, wait()…) should offer higher levels of abstract that let to realize architectural exploration and profiling even faster for early stages of platform design. By using SystemC to model parallel HW/SW process execution, including all models of hardware components of the distributed parallel architecture, we can perform more complex MPSoC/NoC topology exploration, bandwidth estimation of the interconnection network, select optimal packet size and communication protocol, or monitor the network traffic to avoid possible congestions without using any physical platform.

References

[1] H. Chang et al, "Surviving the SOC Revolution: A Guide to Platform Based Design", Kluwer Academic Publishers. 1999

[2] Ahmed Amine Jerraya and Wayne Wolf . The Morgan Kaufmann Series in Systems on Silicon. "Multiprocessor Systems-on-Chips". 2005 ISBN: 0-12385-251-X

[3] A. Baghdadi, N-E. Zergainoh, D. Lyonnard, A.A. Jerraya. "Generic Architecture platform for multiprocessor System-on-Chip design". 2000

[4] Ahmed Amine Jerraya, Amer Baghdadi, Wander Cesário, Lovic Gauthier, Damien Lyonnard, Gabriela Nicolescu, Yanick Paviot, Sungjoo Yoo. "Application-Specific Multiprocessor Systems-on-Chip". 2001

[5] Benini, L.; De Micheli, G. Networks on chips: a new SoC paradigm. IEEE Computer, v. 35(1), Jan. 2002, pp. 70-78.

[6] D. E. Culler and J. P. Singh, Parallel Computer Architecture - A Hardware Software Approach, Morgan Kaufmann Publishers, Inc., ISBN 1-55860-343-3, 1999.

[7] A Austin Hung, William Bishop, Andrew Kennings. "MPSOC: Symmetric Multiprocessing on Programmable Chips Made Easy". DATE'2005 (1530-1591/05)

[8] A. L. Sangiovanni-Vincentelli et al "Platform-Based Design and Software Design Methodology for Embedded Systems" IEEE Design and test of computers

[9] Juha-Pekka Soininen et al "Extending Platform-based design to network on Chip Systems" IEEE Proceedings of International conference of VLSI Design

[10] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek and Vaidyalingam S. Sunderam. "PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Network Parallel Computing". November 1994. ISBN 0-262-57108-0299

[11] Message Passing Interface Forum. "MPI: A message-passing interface standard". Technical Report UT-CS-94-230, University of Tennessee, 1994.

[12] W. Gropp, E. Lusk, and A. Skjellum, "Using MPI: Portable Parallel Programming with the Message-Passing Interface". MIT Press, 1994.

[13] David Bertozzi, Luca Benini "A Network-on-chip Architecture for gigascale Systems-on-Chip" IEEE Circuits and systems magazine

[14] Carrabia, J.; Montón, M.; Martínez, R.; Joven, J.; Font, O.; Ruíz, R.; García, P. & Terés, L. Bus-centric SoC Architecture Generation Tools. GSPx, The International Embedded Solutions Event, 2004.

[15] Shashi Kumar et al "A Network on Chip Architecture and Design Methodology" IEEE Computer Society Annual Symposium on VLSI 2002

[16] P. Guerrier and A. Greiner, A Generic Architecture for On-Chip Packet-Switched Interconnections, In Proc. of DATE 2000, March 2000.

[17] Rijpkema, E.; Goossens, K.; Rădulescu, A. Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip. Design, Automation and Test in Europe (DATE'03), Mar. 2003, pp. 350-355.

[18] Rijpkema, E.; Goossens, K; Wielage, P. A Router Architecture for Networks on Silicon. 2nd Workshop on Embedded Systems (PROGRESS´2001), Nov. 2001

[19] K. Goossens et al., Networks on Silicon: Combining Best-Effort and Guaranteed Services, In Proc. of DATE 2002, March 2002

[20] M. Millberg, "The Nostrum Protocol Stack and Suggested Services Provided by the Nostrum Backbone". Technical Report TRITA-IMITLECS R 02:01, LECS, Dept. of IMIT, KTH, Stockholm, Sweden, 2003.

[21] Martin, G. "SystemC and the future of design languages: opportunities for users and research", Proc. ISCAS, 2003.

[22] David C. Black, Jack Donovan. "SystemC from the ground up", 2004