A Framework for MPSoC Generation and Distributed Applications Evaluation

Guilherme Castilhos[†], Eduardo Wachter[*], Guilherme Madalozzo[*], Augusto Erichsen[*], Thiago Monteiro[*], Fernando Moraes[*]

[†]Santa Cruz do Sul University (UNISC) – Faculty of Informatics – Brazil
guilhermemcastilhos@unisc.br
[*]FACIN – PUCRS – Av. Ipiranga 6681– Porto Alegre – RS – Brazil
{guilherme.madalozzo, eduardo.wachter, thiago.monteiro, augusto.erichsen }@acad.pucrs.br, fernando.moraes@pucrs.br

***Abstract* – The design of MPSoCs is a complex task. From the designer side point of view, a new feature inserted into the system (e.g. a mapping heuristic or a new function in the operating system) must be validated with a large set of the MPSoC configurations. From the application developer side point of view, the performance of a set of applications running simultaneously in the MPSoC platform must be also evaluated for different MPSoC configurations. Therefore, for both designers and application developers a framework enabling the automatic MPSoCs generation and simulation is mandatory for design space exploration. This is the goal of the present work, present a parameterizable MPSoC, including distributed management, and a framework to generate and simulate several MPSoCs configurations automatically. Results show that it is feasible to simulate large platforms, up to 400 processing elements, using a cycle accurate SystemC description.**

Keywords – MPSoC; NoC; framework; distributed management; reclustering

I. INTRODUCTION

The huge design space to develop the hardware/software infrastructure of MPSoCs (Multiprocessor System-on-Chip), or to evaluate the performance of applications requires frameworks able to generate and to simulate the MPSoC. Examples of such frameworks are scarce in the literature. Most frameworks are limited to few PEs (processing elements), use abstract models that does not enable accurate performance evaluation or evaluate only one metric (as power estimation [1] or application parallelization [2]).

To fill this gap, the *goal* of this work is to present a framework able to automatically generate and simulate a NoC-based MPSoC. From a set of *scenarios*, a RTL SystemC description of the MPSoC is generated, along with the software, operating systems and applications. The adoption of RTL SystemC description is due to the simulation time, two orders of magnitude faster than pure VHDL, with the same clock-cycle accuracy [3]. The generated MPSoC may have a centralized or distributed management of resources. Distributed management is suited for large MPSoCs, and centralized management for small/medium MPSoCs (up to 64 PEs) [4].

The *contributions* of the present work include: (*i*) the framework, with results for MPSoCs containing up to 400 PEs; (*ii*) the NoC-MPSoC architecture with distributed management; (*iii*) *regression test*, to uncover system bugs after system changes, or to evaluate the performance of applications for several MPSoC configurations.

II. FRAMEWORKS FOR MPSOC GENERATION

Angiolini et al. [5] propose a methodology to integrate existing standalone CAD tools into a virtual platform. They explore state-of-the-art CAD tools, such as the commercial LISATek suite and the academic MPARM environment. These tools respectively focus on the seamless development of ASIPs, and on the analysis of system-level issues such as multiprocessor performance and communication support facilities (shared-memory infrastructure). Then, they integrate LISA custom designed IPs to the interconnection provided by MPARM. The integration was made taking into account that both tools adopt a SystemC simulation backbone. Cycle-accurate simulations of heterogeneous platforms, where cores interact with the interconnect, compete for shared resources.

Roth et al. [6] presented a framework for multi-resolution and multi-device (MultiX-Simulation) of MPSoCs. The framework is based in the SystemC federate library simulation, which is based on a generic simulation backbone called High Level Architecture (HLA). This library implements an event based simulator where the logical representation of an interconnection of different simulators is called a Federation and includes multiple modules (Federates) that communicate via a Runtime Infrastructure (RTI). The RTI provides different management services, which are relevant for simulation control, synchronization and data exchange. To interconnect PEs, a NoC (Network-on-Chip) executes an XY routing-scheme with dedicated FIFOs, implemented in SystemC. The MPSoC is partitioned by assigning one PE to each federate, and splitting the FIFO channels which results in similar workload for each federate. They shows results comparing four applications executing in MPSoC sizes of up to 6x6 comparing the speed-up from the RTL-model to the proposed Mixed-model. Results show a simulation speedup up to 229 times in a 6x6 MPSoC.

xENoC [7] is an environment for hardware/software automated design of NoC-based MPSoC architectures. The core of this environment is an EDA tool, called NoCWizard, which can generate RTL Verilog NoCs. The whole system is described in an XML file (NoC features, IPs and mapping), which is used as input for the automatic generation tools. In addition to the hardware infrastructure, xNoC also includes an Embedded Message Passing Interface (eMPI) supporting parallel task communication.

Lemaire et al. [8] propose a framework for simulating a MPSoC platform. This framework enables the simulation of each module (NoC or PE) in different levels, e.g. RTL and TLM. The NoC is modeled in three different levels: (*i*) an untimed packet-level without NoC contention; (*ii*) an approximately-timed packet-level mode with contention; (*iii*) an accurate flit-level mode, very close to the hardware behavior. For a mixed simulation, they used a wrapper for co-simulation purposes. The PE is composed by (*i*) two VLIW processors; (*ii*) network interface (NI), (*iii*) RAM; (*iv*) one MIPS, responsible for the PE management. The MIPS is the only module that is modeled at high-level in an approximate-timed SystemC TLM and ISS (Instruction Set Simulator) mode. A RTL description of the other modules is available, interconnected by co-simulation wrappers. The simulation time results show that the high-level SystemC TLM models are 40 times faster than the RTL description. The presented scenario consists in computing a 1K-FFT on the MIPS core in one PE, taking around 0.35s to execute.

Table I compares relevant features of the reviewed works. As most proposals, the present work adopts a SystemC modeling. The differentiation of our proposal includes: (*i*) accuracy, since the SystemC model was derived from the synthesizable VHDL model; (*ii*) simulation speed, enabling to evaluate platforms with 400 PEs in few hours; (*iii*) design space exploration, including system size, tasks per PE, management techniques, mapping heuristics, among other parameters.

Table I - State-of-the-art in MPSoC Frameworks Generation.

Proposal	Description Language	Interconnection	Processor Type	Debugging	Accuracy	Model Engine	Max network size	Simulation time
Angiolini et al. [5]	SystemC	Shared memory	LISA 2.0 processor models	Graphical interface	Cycle Accurate	Event-based	Depends on the shared memory bottleneck	N/A
MultiX-Simulation [6]	SystemC	NoC	MIPS	N/A	Quasi Cycle Accurate Level (CAL)	Event-based	N/A	1797s for a 6x6 MPSoC with four tasks application
xENoC [7]	VHDL/XML	NoC	NIOSII soft-core	N/A	N/A	Cycle-based	Size parameterized according to NoC dimensions	N/A
SMEP [8]	SystemC TLM	NoC	MIPS and 2 VLIW	N/A	7% compared to RTL	TLM	N/A	0.35s for 1 PE executing a FFT
Proposed Work	SystemC RTL	NoC	MIPS / Microblaze	Performance reports	Cycle Accurate	Event-Based	256x256 (theoretical maximum size)	286s for a 8x8 MPSoC.

III. MPSoC Architecture

The present work adopts a NoC-based MPSoC architecture, interconnecting PEs through a 2D-mesh topology, using a 32-bit flit width. Each PE contains a RISC processor, a network interface (NI), a DMA module, and a private memory (RAM) for code and data. The PE may support Von Neumann and Harvard memory organizations. The PE private memory is a true dual port memory. In processors with a Von Neumann memory organization, as the Plasma processor [9], the memory can be shared between the processor and the NoC (through the DMA module). In a Harvard organization, as the Microblaze processor, one of the memory ports is shared with the DMA module, resulting in a smaller communication performance compared to the Von Neumann memory organization. It is important to remember the advantages of the Harvard organization for computation, since instructions and data are accessed in parallel, reducing stalls in the processor pipeline.

Applications are modeled as task graphs $A=<T,C>$ (example in Figure 1), where $T = \{t_1, t_2, ..., t_m\}$ is the set of application tasks corresponding to the graph vertices, and $C= \{(t_i, t_j, w_{ij}) \mid (t_i, t_j) \in T$ and $w_{ij} \in \mathbb{N}^*\}$ denotes the communications between tasks, corresponding to the graph edges. An external MPSoC memory, named *task repository*, contains all applications tasks (set T), which are loaded into the system at runtime.

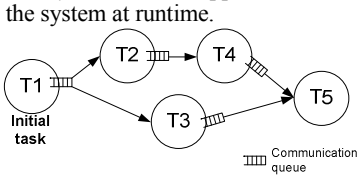

Figure 1 - Application modeled as a task graph.

Each PE runs a small operating system, *microkernel*, responsible for task scheduling and communication between tasks. Tasks communicate through message passing, using MPI-like send/receive primitives, and *microkernel* services, as "Task Allocated", "Location Request", "Message Request", "Message Delivery". Non-blocking *sends* insert messages in the communication queues (Figure 1), while blocking *receives* read from the communication queues. Such communication scheme reduces the overall NoC traffic, since a message is only injected into the network when it is required. If the communication queues are placed at the receiver side, messages could block the NoC when the queues become completed filled.

The memory is organized in equally sized pages, favoring task mapping and task migration. When a given task is required to be mapped, it may use any available page in the PE set. In the same way, if a task is required to be migrated from one PE to another, the transmission of the page contents and the task context is sufficient to ensure the correct task migration. All PEs have a parameterized memory size, enabling to increase the number of tasks running simultaneously at each processor, as well as the execution of large applications.

IV. Resource Management

One relevant features of the MPSoC design is how resources are managed. NoC-based MPSoCs offer scalability at the hardware level. However, the management of the MPSoC resources requires scalable methods, to effectively extract the computational power offered by dozens of processors. State-of-the-art proposals adopt different approaches to tackle such problem, using the MPSoC clustering as the most common approach [10][11].

The management of the MPSoC may vary from a centralized approach (one PE responsible to manage all MPSoC resources) to a distributed approach. The distributed management architecture divides the MPSoC in *n* equally sized regions, named *clusters*, defined at design time. At execution time, if a given application does not fit in a cluster, the cluster may ask resources to adjacent clusters. Therefore, the cluster size may increase at execution time, according to the resources required by applications. When resources became available in a given cluster, tasks belonging to applications mapped in other clusters may be migrated to the cluster. Therefore, PEs of the MPSoC may act as:

- *LMPs* (Local Master PEs): control the cluster, executing functions such as task mapping, task migration, monitoring, deadlines verification, and communication with other LMPs and GMP;
- *GMP* (Global Master PE): contains all functions of the LMP, and functions for the overall system management, such as choose in which cluster a given application will be mapped, control the available resources in each cluster, receive debugging and control messages from LMPs, access the application repository, and receive new applications requests from an external interface;
- *SP* (Slave PEs): responsible for task execution. SPs may execute *k* simultaneous tasks, being *k* the number of pages for tasks.

When the MPSoC is specified with one cluster, the management is centralized, without LMPs.

LMPs and the GMP do not execute applications tasks, only system management. Therefore, they represent an overhead, and the numbers of clusters must be carefully chosen. For example, in a 12x12 MPSoC (144 PEs), with 9 4x4 clusters, 9 PEs execute only management. This represents a cost of 6.25% of PEs dedicated to management. Smaller clusters induce a large overhead. Therefore, distributed management is suited for large MPSoCs, and centralized management for small/medium MPSoCs (up to 64 PEs) [4].

It is supposed that at least one task does not have dependences on other tasks, being the *initial(s) task(s)* (Figure 1). According to user requests, a new application may be requested to execute in the system. The GMP executes a "*cluster selection*" heuristic to choose the cluster that will receive the new application. The heuristic choose the cluster that best fit the application in terms of available resources. Once a given cluster is selected, the GMP sends the application description to the LMP of the selected cluster. Two mapping situations may arise: mapping of initial tasks and mapping of remaining tasks. The initial task mapping searches for the SP with the highest number of available resources around it. This increases the probability of the remaining tasks of the application to be mapped close to each other, reducing the communicating distance between tasks, and therefore the communication energy. The mapping of the remaining tasks adopts the PREMAP-DN multi-task mapping heuristic [12] to select the SP to receive a new task. This mapping heuristic minimizes the energy consumed in the NoC, by approximating the tasks with higher communication volume.

When the mapping heuristic cannot map tasks in the cluster due to lack of available resources, the LMP of the cluster try to borrow resources from neighbor clusters. This is a 5-step process:

1) The LMP of a cluster (LMP_{CL}) sends a "*loan request*" message, requesting resources to all LMPs in neighbor clusters (LMP_{NBO}).

2) Each LMP_{NBO} search for available resources in theirs clusters. If there is only one available resource, this resource is *reserved* to be borrowed; otherwise, if there is more than one available resource, the LMP_{NBO} reserve the one as close as possible, in number of hops, between the task to be mapped and the source task in the cluster managed by LMP_{CL}.
3) After the reservation, all LMP_{NBO} send a "*loan delivery*" message to the LMP_{CL}, notifying the resource position, if it exists.
4) The LMP_{CL} chooses the closest resource from the one that requested the task, sending a "*loan release*" message to all LMP_{NBO} that were not selected. If there are no available resources in neighbor clusters, the search space increases, extending it to the neighbors of the neighbor clusters, returning to step 1.
5) Finally, the LMP_{CL} send a "*task allocation request*" message to the GMP requesting the task mapping on the borrowed resource.

Therefore, the cluster size increases at runtime, because the borrowed resource is now part of this cluster. This process optimizes the system management, since applications can be mapped in clusters, even if the cluster has no sufficient resources. This process is named *reclustering*, which may expand the cluster size with the above protocol, and restore the cluster size using task migration.

V. Scenario-based Verification

This Section describes the process to automatically generate the MPSoC, and to obtain the performance reports. From the designer side point of view, a new feature inserted into the system (e.g. a mapping heuristic) must be validated with a large set of MPSoC configurations. From the application developer side point of view, the performance of a set of applications running simultaneously in the platform must be also evaluated for different MPSoC configurations. The flow presented in Figure 2 enables the automation of the MPSoC generation, simulation and analysis. The flow is divided in 5 phases.

The first phase generates the *scenarios* with different applications and different configurations of the MPSoC. The in-house *Scenario Generator* tool has as inputs: (1) the project name; (2) MPSoC size; (3) cluster size; (4) memory size; (5) page size; (6) GMP Address; (7) configuration file containing the application set that will be inserted at runtime in the MPSoC.

In the second phase, a second in-house tool *MPSoC Generation*, is responsible for generating and compiling the hardware and software of the MPSoC. As input, this phase receives the MPSoC hardware description, the *microkernel* files, the applications C codes, and the MPSoC configuration file for each scenario. The configuration files contain the parameters defined at the first phase. The MIPS processor is described with a SystemC instruction set simulator, while all other modules are modeled in a cycle accurate SystemC.

The result of the second phase is an executable file for each scenario. The numbers in the Figure describing the result of the second phase corresponds to the parameters of the first phase. For example: (2) is the MPSoC size; (3) cluster size; (4) memory size; etc.

The third phase simulates each scenario using its respective executable file. Each simulation can be executed sequentially in the same workstation, or distributed in a grid. The results of this phase are performance reports for each SP, identifying the initial and final time for each task/application. The report of the GMP contains also the total execution time for the simulated scenario.

In the fourth phase, a script reads the performance reports for each scenario, illustrating in a graphical view where each task was mapped, together with its initial and final execution time. This phase provides to the user an overview of how tasks were mapped in the MPSoC, and the performance of each application. It is possible to indicate in the script the simulation time, to obtain a "picture" of the system in a given moment of the simulation.

In the last phase, another script extracts the performance information from the reports and verifies if all applications have finished. This phase also generates reports, charts, and tables used to compare the simulated scenarios.

The script used in the last phase is able to automatically execute phases 2, 3 e 4, using as input the configuration files generated during phase 1. Therefore, this flow enables *regression tests*, since it is possible to generate and simulate several MPSoCs configurations without requiring any user interference. At the end, analyzing the reports it is possible to verify if all scenarios passed (enabling to debug the platform, evaluating the test-case with non finished applications), or to evaluate the performance of a set of applications using different configurations

VI. Results

The employed benchmark is a MJPEG encoder, with five tasks. Each SP is able to execute 3 simultaneous tasks. Table II details the MPSoC configurations, varying the MPSoC size, management strategy, and 30/60% load (it corresponds to the number of tasks the MPSoC may execute simultaneously: |SPs|*|pages|). Therefore, 28 test cases were evaluated. The number of SPs per MPSoC size is obtained subtracting the number of manager PEs from the total number of PEs. For example, in a 20x20 MPSoC, with 25 4x4 clusters, there are 25 managers PEs (1 GMP and 24 LMPs). Therefore, the number of SPs is 375. Considering this configuration, the MPSoC is able to execute up to 1,125 simultaneous tasks (375 SPs * 3 pages per SP). For a load of 60%, the simulation runs 675 tasks, which is equivalent of 135 instances of the MJPEG running simultaneously.

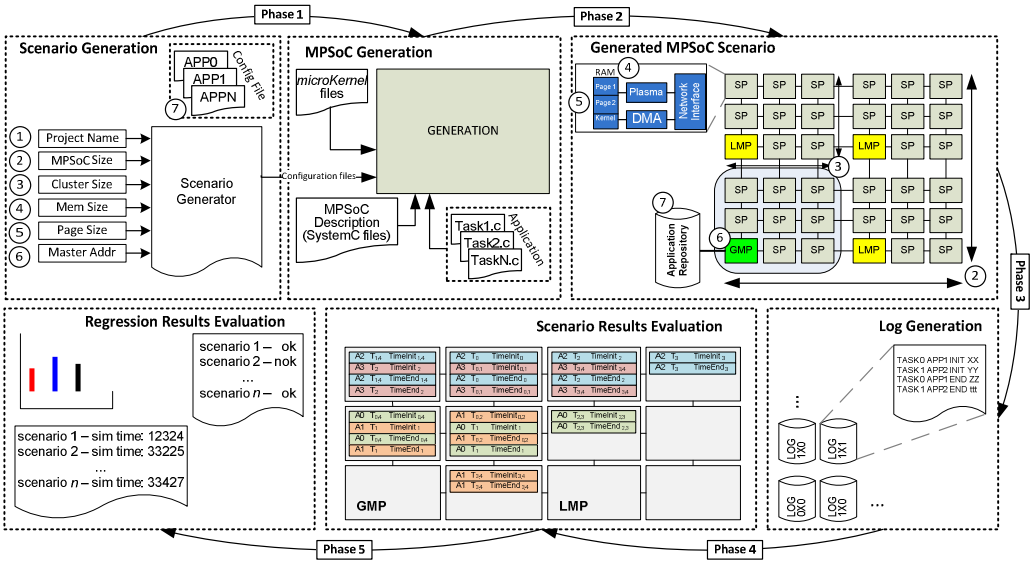

Figure 2 – Framework for MPSoC automatic simulation and analysis flow, divided into five main phases.

Table II – MPSoC configurations used to evaluate the execution and simulation times. In the distributed management the cluster size is 4x4.

MPSoC Size	Centralized Management	Distributed Management
	N# of SPs	N# of SPs
8x8	**63**	**60**
12x8	**95**	**90**
12x12	**143**	**135**
16x12	**191**	**180**
16x16	**255**	**240**
20x16	**319**	**300**
20x20	**399**	**375**

Figure 3 presents the execution time for all 28 test cases. The execution time using the distributed management grows linearly with the number of SPs (R^2=0.988 for loads 30% and 60%). On the other hand, the centralized management has a quadratic increase (R^2=0.999 for loads 30% and 60%). Such difference is due to the load in the managers. Each LMP in the distributed version is responsible for mapping, monitoring, and control messages. The GMP in the centralized approach is overloaded due to execution of a large number of task mappings (more than 1,000 tasks to be mapped in the larger scenarios), and to control of the availability of resources. In addition, in the centralized management scenarios, the traffic around the GMP generates hot-spot regions, compromising the communication performance, and in long-term the reliability of the system. *This result is a clear demonstration that distributed management contributes to the scalability at the application level.*

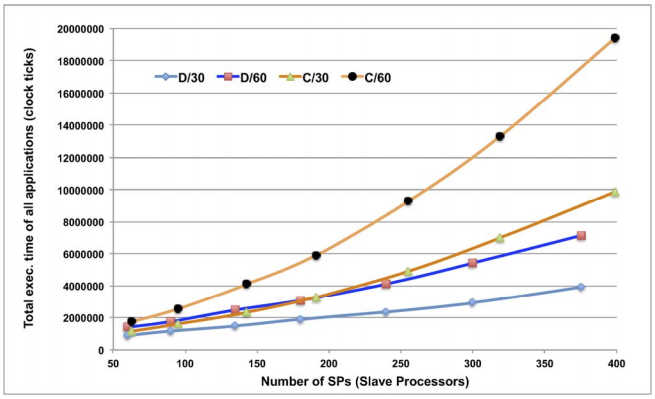

Figure 3 – Total execution time for all applications (in clock cycles), for different MPSoC sizes, load and management techniques.

Figure 4 presents the simulation time for the 28 test cases. The simulation time is presented in the y-axis, in seconds. The simulation time grows exponentially with the number of PEs. Even if this result shows the limitation to evaluate large MPSoCs with clock cycle accuracy, it is important to note that the simulation of 375 SPs with 675 simultaneous tasks (60% load) took 4.7 hours (17,164.28 seconds). Therefore, it is still reasonable to simulate an MPSoC with a RTL level model, since it enables to evaluate throughput, latency, jitter, power, and energy. The simulation time of the centralized management is higher because the execution time of these scenarios took longer (the simulation time is a function of the number of events created during simulation).

To conclude this section, it is important to present the cost of the *reclustering* process, specifically the migration process. The total migration time is around 11,000 clock cycles, for a page size equal to 32 KB. The performance during task migration is reduced, since the task has to be stopped. After tasks migration, results shown an average improvement in the applications' throughput in order of 18%.

Such results demonstrates the effectiveness of the distributed management adopted in the MPSoC platform, enabling to have clusters with variable size at run-time, and when possible restore the initial cluster shape by migrating tasks. Even if the task migration cost is some thousands of clock cycles, for the executing software this amount of time is very small, and the penalty induced during migration is quickly translated into performance gains.

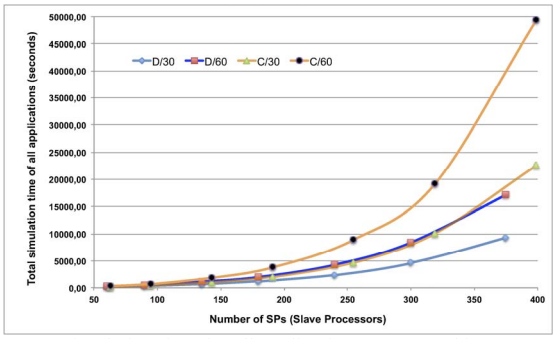

Figure 4 – Simulation time for all applications (Xeon 64 bits, 4 cores, 32 GB, 3 GHz), for different MPSoC sizes, load and management techniques.

VII. Conclusions and Future Works

The presented framework for NoC-based MPSoC generation and simulation enables to evaluate large platforms with an acceptable simulation time. The MPSoC contains adaptive techniques for management, enabling to parallelize tasks as mapping, migration, and monitoring. The distributed management enabled the addition of a large set of applications in the MPSoC, with a linear increase in the total execution time. The total execution time should theoretically be the same, regardless the number of instances of the applications being executed. However, the management plays an important role in the total execution time, since the application must be read from the memory, mapped and monitored. Therefore, in such large MPSoCs (e.g. 20x20), a distributed management approach is mandatory.

Future works includes the integration of all tools of the framework in a unified environment; enrich the performance reports with communication performance parameters (as latency and throughput); add QoS techniques in the MPSoC to control applications deadlines. The use of more abstract models, employing for example OVP, is research direction to reduce the simulation time.

Acknowledgements

The author Fernando Moraes acknowledge the support granted by CNPQ, processes 472126/2013-0 and 302625/2012-7; and CAPES process 708/11.

References

[1] Ben Atitallah, R.; et al. *An Efficient Framework for Power-Aware Design of Heterogeneous MPSoC*. IEEE Transactions on Industrial Informatics, v.9(1), pp.487-501, 2013.

[2] Ceng, J.; et al. *MAPS: An integrated framework for MPSoC application parallelization*. In: DAC, 2008, pp. 754-759.

[3] Petry, C.; Wachter, E.; Castilhos, G.; Moraes, F.; Calazans, N. *A Spectrum of MPSoC Models for Design and Verification Spaces Exploration*. In: RSP, 2012, pp. 30-35.

[4] Castilhos, G.; Mandelli, M.; Madalozzo, G.; Moraes, F. *Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes*. In: ISVLSI, 2013.

[5] Angiolini, F.; et al. *An Integrated Open Framework for Heterogeneous MPSoC Design Space Exploration*. In: DATE, 2006. 6p.

[6] Roth, C.; et al. *Modular Framework for Multi-level Multi-device MPSoC Simulation*. In: IPDPSW, 2011, pp.136-142.

[7] Joven, J.; et al. xENOC – *An eXperimental Network-on-Chip Environment for Parallel Distributed Computing on NoC-based MPSoC Architectures*. In: Euromicro, 2008, pp. 141-148.

[8] Lemaire, R.; Thuries, S.; Heiztmann, F. *A flexible modeling environment for a NoC-based multicore architecture*. In: High Level Design Validation and Test Workshop, 2012, pp. 140-147.

[9] Plasma CPI. Available at http://opencores.org/project,plasma

[10] Shabbir, A.; Kumar, A.; Mesman, B.; Corporaal, H. *Distributed Resource Management for Concurrent Execution of Multimedia Applications on MPSoC Platforms*. In: SAMOS, 2011, pp. 132-139.

[11] Fattah, M.; Daneshtalab, M.; Liljeberg, P.; Plosila J. *Exploration of MPSoC Monitoring and Management Systems*. In: ReCoSoC, 2011, 3p.

[12] Mandelli, M.; Amory, A.; Ost, L.; Moraes, F. *Multi-task dynamic mapping onto NoC-based MPSoCs*. In: SBCCI, 2011, pp. 191-196.