2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

Towards Architectural Design Space Exploration for Heterogeneous Manycores

Benard Xypolitidis, Rudin Shabani, Satej V. Khandeparkar, Zain Ul-Abdin, Süleyman Savas, Tomas Nordström
Centre for Research on Embedded Systems (CERES),
Halmstad University, Sweden
benard.xypolitidis@gmail.com, rudin_shabani@msn.com, satejk5@gmail.com,
{zain-ul-abdin, suleyman.savas, tomas.nordstrom}@hh.se

***Abstract*—Today many of the high performance embedded processors already contain multiple processor cores and we see heterogeneous manycore architectures being proposed. Therefore it is very desirable to have a fast way to explore various heterogeneous architectures through the use of an architectural design space exploration tool, giving the designer the option to explore design alternatives before the physical implementation.**

In this paper, we have extended Heracles, a design space exploration tool for (homogeneous) manycore architectures, to incorporate different types of processing cores, and thus allow us to model heterogeneity. Our tool, called the Heterogeneous Heracles System (HHS), can besides the already supported MIPS core also include OpenRISC cores. The new tool retains the possibility available in Heracles to perform register transfer level (RTL) simulations of each explored architecture in Verilog as well as synthesizing it to field-programmable gate arrays (FPGAs). To facilitate the exploration of heterogeneous architectures, we have also extended the graphical user interface (GUI) to support heterogeneity. This GUI provides options to configure the types of core, core settings, memory system and network topology.

Some initial results on FPGA utilization are presented from synthesizing both homogeneous and heterogeneous manycore architectures, as well as some benchmark results from both simulated and synthesized architectures.

I. Introduction

The latest step in processor architecture is the introduction of heterogeneity in manycore System on Chip (SoC). An asymmetric (heterogeneous) SoC consists of large cores that handle complex and heavy computations in combination with a large number of small cores that execute less complex computations and use less energy. Heterogeneity facilitates an energy efficient architecture, used in portable devices, or a high performance architecture which handles complex and heavy calculations.

Development of a System on Chip with multiple and heterogeneous cores need a huge design space to be explored, verified, and tested. With a design space exploration tool, the architecture can be modified according to the requirements of specific application without going through the whole design cycle, which reduces the development cost drastically.

Now the design space exploration can be performed at different levels. The two most common approaches are based on either performing software simulation using machine learning techniques to determine the design space or to carry out register-transfer level (RTL) simulation to achieve cycle-accurate execution results. The software simulation suffers from the lack of accuracy, whereas the RTL simulation approach has the disadvantage of increased simulation time. However, the increased simulation time could be mitigated by performing hardware emulation on FPGAs from the RTL design description.

In this paper, we introduce our Heterogeneous Heracles System (HHS) as a tool for Architectural Design Space Exploration (ADSE) for heterogeneous manycore systems that generates synthesized hardware configurations for FPGA emulation. It is an extension to the open source Heracles Designer [1], which is a (homogeneous) manycore design space exploration tool. The proposed system extends Heracles to incorporate a new type of processing core, the OpenRISC, whereas it originally only have a MIPS core available. The newly integrated OpenRISC core supports multiplication operation as well as a floating-point unit, which were not suppoerted by the previously present MIPS core. A new graphical user interface (GUI) has also been added to support parameterization of heterogeneous systems design.

The outline of the paper is as follows. Section II provides background on the system architectural overview for the Heracles System and the OpenRISC. A brief literature review of the related work is presented in Section III. Section IV elaborates on architectural system overview of the HHS. Section V presents the evaluation tests that are perfomed and the corresponding obtained results. Section VI provides concluding remarks and suggestions for future work.

II. Background

A. Heracles System

The Heracles System [2] is an open-source architectural design space exploration (ADSE) tool, which can configure system architectures into different topologies, routing schemes, processing elements or cores and memory system organizations. In order to support fast exploration of future manycore architectures, the Heracles System is constructed with a high degree of modularity. The main purpose of this platform is architectural exploration in research and teaching environments [1]. Based on a set of global parameters the Heracles System generates a number of Verilog files which implements the manycore architecture. These Verilog files can then be simulated on ModelSim [3] or synthesized to FPGAs. The Heracles System is provided with a GUI called Heracles

978-1-4673-8776-7/16 $31.00 © 2016 IEEE
DOI 10.1109/PDP.2016.79

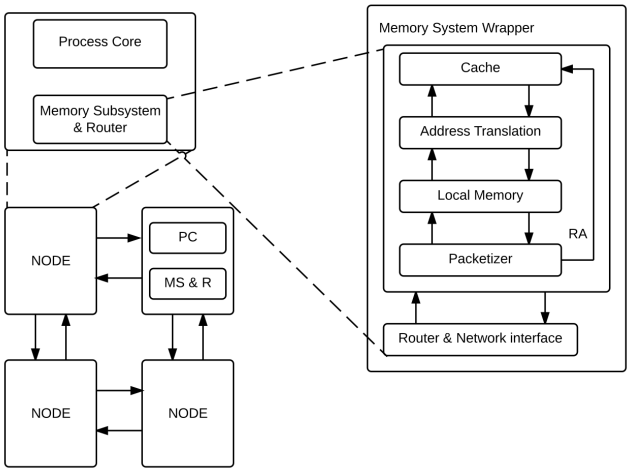

Fig. 1. Generic Heracles System Overview.

Designer. Heracles Designer allows the user to modify the global variables to quickly generate a new architectural design. Heracles Designer also provides the option to download the synthesized architecture to a Field Programmable Gate Array (FPGA) [4]. Heracles Designer generates a file which contains a set of global Verilog variables representing the explored manycore architecture.

A generic overview of the overall structure of the Heracles System is depicted in Figure 1. Each node consists of three parts: the processing core, the memory subsystem and the network interface. The memory subsystem consists of the cache and the packetizer intended for communication. The network interface is responsible for communicating between the NoC and the CPU through the router. The Heracles System is based on Verilog, thus can be synthesized for FPGA.

The available memory is distributed among the cores as local memory under the control of the memory subsystem. Every core's local memory is mapped into its own section of the global address space by the Heracles Designer.

B. MIPS

The processing core that is available in the Heracles System is based on a Microprocessor without Interlocked Pipeline Stages (MIPS) [5] implemented using a 7-stage 32-bit pipeline. The choice of a 7-stage pipeline is due to block RAM access time on the FPGA. The 32-bit registers are mapped to the block RAM, which frees up some LUTs and saves the resources of the FPGA. The MIPS CPU is fully bypassed, meaning no branch delay slot or branch prediction table are present. The MIPS-III instruction set architecture (ISA), used in the Heracles System, does not include floating point [6] operations. By using block RAMs, the instruction and data caches are implemented. Instruction fetch and data memory access take two cycles. To support the extended pipeline, stall and bypass signals are modified. Instructions and data memory accesses are issued and executed in-order. The MIPS architecture used is a single threaded design which includes one program counter and a set of 32 data registers. The ALU of MIPS supports a set of operations limited to integer addition and subtraction, logical operations, shift operations, and comparison operations. With such a limited set of operations this processor is not able to efficiently run applications which involve multiplication and division operations, or signal processing application relying on floating point operations.

C. OpenRISC

The OpenRISC processor, which this paper studies, consists of a power management unit, debug unit, tick timer, programmable interrupt controller (PIC), central processing unit (CPU), and memory management hardware [7]. The OpenRISC CPU is based on 5-stage pipelined *RISC* architecture. By using the standardized 32-bit Wishbone bus interface, peripheral system and a memory subsystem may be added. The OpenRISC is intended to have a performance comparable to an ARM10 processor. Furthermore, OpenRISC is a 32 and 64-bit processor which supports floating-point operations [8]. OpenRISC is inspired by DLX [9], which in turn is a cleaned up (and modernized) MIPS CPU. The OpenRISC closely resembles the MIPS CPU available in the Heracles System. This facilitated the integration of the OpenRISC into the Heracles System, which is the main reason for it's selection. The ALU of OpenRISC can perform the multiplication and division operations besides the addition, subtraction, logical and comparison operations. The FPU of OpenRisc is capable of performing floating point additions and subtractions using a or1200_fpu_addsub Verilog module. It also instantiates a separate module, or1200_fpu_mul, or1200_fpu_div and or1200_fpu_cmp for floating point multiplication, division and comparison operations. Other modules are used for integer to float conversion and pre and post normalization required during floating point arithmetic. The OpenRISC CPU also has a exception unit, which raises a exception such as; *Bus Error*, if there is any non-existing physical location, *Alignment* exception is raised if the address is unaligned with the data memory and *Illegal instruction* this exception is raised if the CPU encounters a instruction which it cannot decode. The exception unit is of utmost importance when integrating a CPU with a new system, as it gives us deeper insight into the workings of the new system and accelerates the process of integration. The OpenRISC CPU has a optional implementation of a debug unit which assists software developers in debugging their systems. It provides support for watch-points, breakpoints and program-flow control registers.

The OpenRISC system architecture is still an ongoing project.

III. Related Works

A fair amount of work has been done on architectural description, automatic code generation, design space exploration and simulation of processors. This section provides a brief literature review of some of the significant work performed in the context of architectural design space exploration. For a

detailed study of the various software based design mapping techniques and tools, see [10].

At the micro-architectural level, Dubach et al. [11] have proposed a machine learning based simulation approach to train the architecture-centric model. The data gathered by off-line learning experiments can then be used to predict the performance and power consumption of benchmark applications with reasonable accuracy. Eyerman et al. [12], on the other hand, used a two-phase genetic local search algorithm to simulate out-of-order processors. The first stage applies statistical simulation to prune the design space, which is followed by the detailed simulation of a specific region of interest to reduce the simulation time.

A scalable manycore processor architecture with OpenRISC as a processing element is proposed by Chien et al. [13]. The processing cores are connected via a mesh-based Network on Chip (NoC) and has access to an external memory. They propose XY routing to avoid any deadlock on routing paths. The developed framework is intended for analysis, verification and validation of manycore processor architecture for embedded parallel applications. While their framework is similar to the system presented in this paper and uses the same processor, OpenRISC, to design manycore architectures, it currently do not include the support for exploring heterogeneity.

A design space exploration tool is presented by Lahiri et al. [14]. This design space exploration tool is used for optimization of system-level on-chip communication architectures. The tool consists of two algorithms. The first algorithm is a clustering algorithm for mapping the SoC communications to network and topology. The second algorithm is an iterative algorithm that dynamically improves the previous algorithm. Lahiri et al. [14] contribute with an architectural view of how communications can be mapped on SoCs, which gave a basis towards the understanding of the communication between the cores in the Heracles System.

For the NoC topologies design exploration, Genko et al. [15] have created an emulation framework, which is implemented on an FPGA. The emulation framework has been designed as a modular NoC programmable platform. It consists of Traffic Generators, Traffic Receptors and user defined interconnections between the switches of the network. Although the framework enables a vast and extensive exploration of NoC topologies, it only uses one hard coded processing core, thus limiting the possibility to explore the heterogeneity at the core level. Similarly, Öberg et al. [16] have proposed a NoC exploration platform, which can generate 1D, 2D or 3D Mesh and Torus topologies for multicore platforms. Their NoC generator generates an arbitrarily large multicore platform through an XML configuration file. Their NoC design solves the scalability issue by complimenting the topology design, with an on-chip interconnect system in the form of packet-based communication. At the moment their framework do not readily support heterogeneity among the cores. While the Heracles System can generate 2D and 3D Mesh it currently does not support Torus topologies.

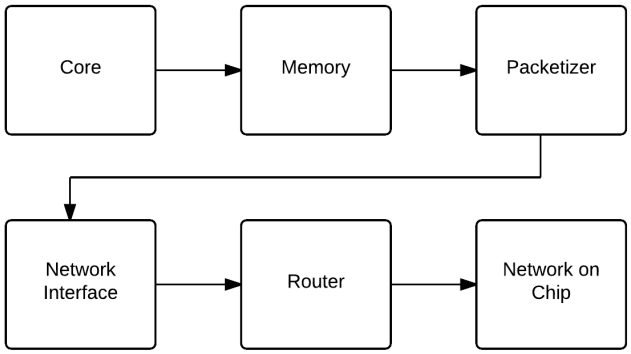

Fig. 2. Dataflow diagram of the HHS.

IV. Heterogeneous Heracles System

Our Heterogeneous Heracles System (HHS) integrates an additional processor core type, OpenRISC, to the Heracles System. The OpenRISC processor is integrated into the Heracles System via an interfacer module. The HHS fully maintains the original properties of the Heracles System, such as varying the number of processing cores and reconfiguring both the memory and network topologies. The data flow of the HHS can be observed in Figure 2.

A. ADSET

Throughout the Verilog files that build the HHS there are a set of global variables that are used in order to have a modular and reconfigurable design. Architectural Design Space Exploration Tool (ADSET) is a graphical user interface (GUI) created to manipulate these global variables that configure the system generation of the HHS. Through ADSET, these global variables are changed based on the system architecture that is to be explored. We have created our own GUI in order to support OpenRISC and heterogeneous architectures, while the source code for Heracles native GUI, Heracles Designer, is not available.

B. System Overview

Through ADSET, a module is generated which holds the starting addresses of all the local memories for each generated CPU. This module also updates the global parameters which configure the system based on the desired architectural buildup. The generated module instantiates a real cores mesh where the architectural variation is set. Every node (see Figure 2) in the HHS consists of either a MIPS or an OpenRISC core which instantiate the memory router system module. The memory router system is responsible for the caches, local memories, and the packetizer modules. It is also responsible for the communication between the modules. The real cores mesh forwards a starting address for each processing core in the system, which are provided by the generated Verilog file from Heracles Designer. It uses this starting address to

communicate with the caches and executes the program that is loaded into the local memory.

When a core wants to write data to another cores memory in the system, the address is sent to where the data is supposed to be written in the memory. Once the memory has received the address it checks whether it belongs to the cores local memory or not. If the address is not inside the address space of the local memory then it forwards the address and the data onto the packetizer. In the packetizer the data is split and put into data packets and then sent to the network interface which in turn sends the data packets over to the router. The router checks the address, identifies to which processing core it belongs and once the address is verified it establishes a point to point connection. The router then sends the data packets over to the NoC. When a core wants to read data from another cores memory, it first sends the address to where it want to access the data along with a read request to its memory. The memory then checks if the address received from the core is inside its own address space. If the address is not inside the memory address space it forwards the address over to the packetizer. The packetizer forwards the address to the network interface which in turn forwards the address to the router and the NoC. The router checks the address, identifies to which processing core it belongs and once the address is verified it establishes a point to point connection. The router then forwards the read request to the core where data is to be accessed. The core which receives the read request, sends the data over to the core that generated the read request.

C. Interface Module

The interface module (see Figure 3) works as a bridge between the Heracles System and the OpenRISC processor. This module was created to keep the original modular design structure of the Heracles System. The OpenRISC CPU communicates with the memory sub system & router (see Figure 1) using the address translation logic for the instruction and data addresses. The address translation in the HHS concatenates the processing core number, generated by the Heracles System, with the address generated by the OpenRISC CPU. The new address is forwarded to the *memory router system* which utilizes the address to find the instruction or data at that particular address.

V. Results

The Heterogeneous Heracles System (HHS) is synthesized and realized on to a Virtex 6 FPGA. Furthermore, a set benchmark applications are tested to obtain the systems performance.

A. Synthesizing to FPGA

The device utilization is presented individually for the MIPS and OpenRISC processing cores in Table I. The synthesis of the processing cores were mapped on the Virtex 6 (xc6vlx75t-1ff484) FPGA. In order to compare the OpenRISC and MIPS processors, detailed knowledge about their inner structure is necessary. One of the most important difference is that the

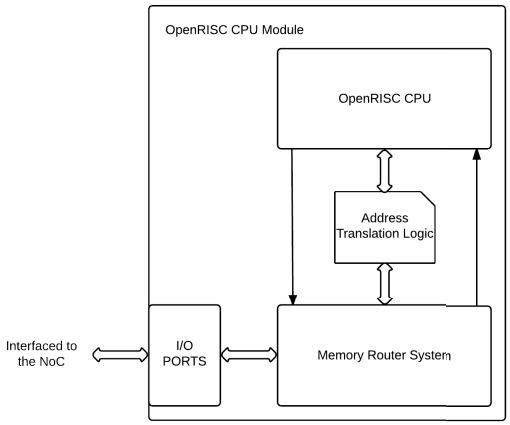

Fig. 3. Interface Module.

TABLE I
Device Utilization of the MIPS and OpenRISC cores.

FPGA Resources	MIPS	OpenRISC
Number of Slice Register	4151(2%)	4555(3%)
Number of Slice LUTs	7553(10%)	8467(12%)
Number of fully used LUT-FF pairs	3249(7%)	3268(8%)
Number of Block RAM/FIFO	8(5%)	12(7%)
Number of BUFG /BUFGCTRLs	5(15%)	5(15%)

MIPS processing core only supports arithmetic and logical operations, excluding multiplication and division. OpenRISC supports arithmetic and logical operations as well as multiplication and division. Furthermore the OpenRISC supports floating-point operations. Taking that into account, having a more powerful processor as OpenRISC allows the Heracles System to run complex algorithms more efficiently. When comparing the Number of Slice Registers, LookUp Tables (LUTs), Global Control Buffers (BUFG/BUFGCTRLs) we can see that the OpenRISC processor utilizes a bit more resources than the MIPS.

It is quite surprising that even though the OpenRISC core is much more powerful compared to the MIPS core it utilizes almost the same amount of FPGA resources. The reason for this needs to be explored further but we suspect it to be the level of optimization that has gone into the OpenRISC implementation compared to the MIPS implementation.

The tests performed on the HHS were simulated to ascertain the system performance with various common benchmarks. Three different architectures were synthesized, the first one containing four MIPS cores, the second one containing four OpenRISC cores and the third one containing two MIPS and two OpenRISC CPUs. Table II shows the device utilization and clock frequencies of the synthesized architectures. The synthesis could not be performed with a larger architectural implementation due to the FPGA size limitations. The Heracles System uses a lot of buffers and flip-flops to set up the network topology and the core-to-core communication in a modular way. This results into a heavy utilization of Slice Reg-

TABLE II
DEVICE UTILIZATION FOR DIFFERENT ARCHITECTURAL DESIGNS.

FPGA Resources	4 MIPS cores	4 OpenRISC cores	2 MIPS + 2 OpenRISC cores
Number of Slice Register	19806(10%)	21509(12%)	20729(11%)
Number of Slice LUTs	38131(41%)	42532(50%)	40003(44%)
Number of fully used LUT-FF pairs	15431(42%)	15269(32%)	15214(37%)
Number of Block RAM /FIFO	29(28%)	45(28%)	45(28%)
Number of BUFG /BUF- GCTRLs	16(50%)	16(50%)	16(50%)
Clock Frequency	103MHz	103MHz	103MHz

isters, LUTs, LUT-FF pairs and BUFG/BUFGCTRLs. Thus, it was not possible to have more than four cores realized on the Virtex 6 FPGA. Based on the clock report obtained during synthesis (cf. Table II) the clock cycle for the synthesized cores is 9.7 ns.

B. Applications Testing

We have used two common benchmarks to simulate the execution on a 64-core architecture generated by the HHS. The system generated for these tests, has a uniform distributed memory of 64KB and 64 OpenRISC cores, where each core has 64 bytes of instruction cache and 64 bytes of data cache. The network topology used is a 2D Mesh with XY-routing. Each core was loaded with the machine instructions of the benchmark programs.

The first test is a *matrix-matrix multiplication* using 8x8 matrices and 16x16 matrices. In both of the cases, integer values were used as matrix elements. In the 8x8 matrix multiplication the input matrix data is divided among 64 processing cores and each core computes one element of the resulting matrix. This was executed in 441 clock cycles and the achieved throughput is 14.9 million samples per second obtained at a clock frequency of 103 MHz. The 16x16 matrix multiplication was done with the same procedure and each processing core computed 4 elements of the resulting matrix. This was executed in 1278 clock cycles (see Table III). Based on the clock frequency at which the architecture is synthesized on a FPGA (see Table II), an estimation is made as to how much actual time the test program requires to execute on a FPGA. Thus the resulting execution time for 16x16 matrix multiplication when realized on the FPGA is 12.39μs.

The second test is a *Fast Fourier Transform (FFT) calculation*. The 16-point FFT is computed in a similar way as the matrix multiplication. The total run-time for the FFT is 693 cycles, resulting in a throughput of 2.4 million samples per second. The total cycles and the total simulation time can be seen in Table III. The estimated execution time on the FPGA for the FFT is 6.72μs. Comparing this to the simulation time, in Modelsim, of 4.43 minutes (266 sec) we see that the execution on the FPGA can be expected to run 40000 times faster.

TABLE III
SYSTEM PERFORMANCE.

	Total Execution Cycles	Throughput (Million Samples/sec)	Total Simulation Time (sec)
8x8 Mat. Mult.	441	14.9	205.2
16x16 Mat. Mult.	1278	20.6	558
16-point FFT	693	2.4	265.8

VI. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we have presented Heterogeneous Heracles System (HHS), a design space exploration tool for heterogeneous manycore architectures. This tool is an extension to the open-source design space exploration tool Heracles, to which we have added the functionality to add new types of processors, and as an example incorporated the OpenRISC processor. We have also added a graphical user interface to determine the parameterization of the architectures. This new tool allows the designer to quickly model and explore heterogeneous manycore architectures based on two different types of processor cores: MIPS and OpenRISC. The output from HHS is a set of Verilog files that can be simulated directly or synthesized in FPGAs.

To test our tool we have generated a first set of architectural designs which we synthesized and mapped on a Virtex 6 FPGA using HHS. These designs have then been tested and evaluated through two common benchmark programs: a matrix-matrix multiplication and a Fast Fourier Transform. From these simulation and synthesis experiments we see a clear benefit in being able to synthesize heterogeneous manycore architectures using parameterization, as the benchmark programs can run about 40000 times faster, when synthesized on FPGAs rather than a Verilog simulation of the architecture.

Even though the OpenRISC core is much more powerful than the MIPS core, and includes support for floating-point operations (which MIPS lacks), our findings show that OpenRISC is utilizing almost the same amount of FPGA resources as the studied MIPS implementation. Future investigations into Heracles and other possible MIPS implementations might shed more light onto the underlying reasons for this. One hypothesis is the different levels of optimization applied to OpenRISC as compared Heracles' MIPS implementation used during our study.

In the future, we plan to perform more complex benchmarking on the generated architectural designs and include additional types of processor cores into HHS in order to expand the design exploration space for heterogeneous manycore architectures.

ACKNOWLEDGEMENT

This research is part of the CERES research program and is conducted in the ESCHER project funded by the Knowledge Foundation, Sweden and HiPEC project funded by Swedish Foundation for Strategic Research (SSF).

REFERENCES

[1] M. A. Kinsy, M. Pellauer, S. Devadas. Heracles: a tool for fast RTL-based design space exploration of multicore processors, *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA'13*, pages 125-134, New York, 2013.

[2] Heracles. http://projects.csail.mit.edu/heracles/, [Accessed online: 2014-12-19]

[3] Mentor Graphics, ModelSim. http://www.mentor.com/products/fv/digital_verification/modelsim_se/index.cfm [Accessed online: 2015-09-27]

[4] S. Kilts. Advanced FPGA Design: Architecture, Implementation, and Optimization, ISBN: 0470054379, 2007.

[5] M. A. Kinsy, M. Pellauer, S. Devadas. Heracles: Fully Synthesizable Parameterized MIPS-based Multicore System, *Proceedings of the International Conference on Field Programmable Logic and Applications, FPL'11*, pages 356-362, China, 2011.

[6] MIPS Instruction Reference, http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html, [Accessed online: 2015-08-26]

[7] Open Cores. http://opencores.org/, [Accessed online: 2015-08-26]

[8] OpenRISC1200 IP Core Specification. http://opencores.org/websvn,filedetails?repname=openrisc&path=/openrisc/trunk/docs/openrisc-arch-1.0-rev0.pdf, [Accessed online: 2015-08-26]

[9] D. Patterson, J. Hennessy. *Computer Architecture: A Quantitative Approach (1st ed.)*, Morgan Kaufmann, ISBN 978-1-55-860329-5. 1996

[10] A.K. Singh, M. Shafique, A. Kumar, J. Henkel. Mapping on multi/many-core systems: survey of current and emerging trends,*Proceedings of the 50th Annual Design Automation Conference, DAC13*, 2013.

[11] C. Dubach, T.M. Jones, M.F.P. O'Boyle. Microarchitectural Design Space Exploration Using an Architecture-Centric Approach, *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*, 2007.

[12] S. Eyerman, L. Eeckhout, K.D. Bosschere. Efficient design space exploration of high performance embedded out-of-order processors, *Proceedings of the Design, Automation and Test in Europe Conference, DATE'06*, 2006.

[13] H. Chien, et al.. Design of A Scalable Many-Core Processor for Embedded Applications, *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, Chiba, 2015.

[14] K. Lahiri, A. Raghunathan, S. Dey. Design Space Exploration for Optimizing On-Chip Communication Architectures, *Computer-Aided Design of Integrated Circuits and Systems*, Vol. 23, No. 6, June 2004.

[15] N. Genko, D. Atienza, G. De Micheli, J.M. Mendias, R. Hermida, F. Catthoor. A complete network-on-chip emulation framework, *Proceedings of the Design, Automation and Test in Europe Conference, DATE'05*, 2005.

[16] J. Öberg, F. Robino. A NoC system generator for the Sea-of-Cores era, *Proceedings of the 8th FPGAWorld Conference, FPGAWorld'11*, 2011.