An FPGA-Based Framework for Technology-Aware Prototyping of Multicore Embedded Architectures

Paolo Meloni, Simone Secchi, *Student Member, IEEE*, and Luigi Raffo, *Member, IEEE*

***Abstract*—The use of cycle-accurate software simulators as a foundation for the exploration of all the possible full-system hardware–software (hw–sw) configurations does not appear to be anymore a feasible way to handle modern embedded multicore systems complexity. In this letter, an field programmable gate array (FPGA)-based cycle-accurate hardware emulation framework is presented and proposed as a research accelerator for the exploration of complete multicore systems. The framework provides the possibility to extract from the automatically instantiated hardware-emulated system a set of metrics for the assessment of the performance and the evaluation of the architectural tradeoffs, as well as the estimation of figures of power and area consumption of a prospective application-specified integrated circuit (ASIC) implementation of the considered architecture.**

***Index Terms*—Design exploration, field programmable gate array (FPGA), MPSoC emulation.**

I. Introduction

THE prediction of the performances of modern multicore architectures requires an effective solution of the speed-accuracy tradeoff. The interest has recently shifted from well-established cycle-accurate full-system simulators to the adoption of field programmable gate array (FPGA)-based hardware emulation platforms, whose trends in integration capability, speed, and price propose them as a candidate to speed-up the exploration of large multicore architectures [3]. Moreover, to consider already at system/architectural level the variables related to the low level implementation, the concept of “system-level design with technology-awareness” must be introduced. Detailed area, frequency, and power models can be used to back-annotate the architectural assumptions and the experimental results obtained by means of the prototyping. This letter presents an FPGA-based framework for the emulation of complex and large multicore architectures that allows the easy instantiation of the desired system configuration and automatically generates the hardware description files for the FPGA synthesis and implementation. The prototyping results can be duely back-annotated using analytic models included in the framework, to evaluate a prospective application-specified integrated circuit (ASIC) implementation of the system.

Manuscript received September 25, 2009; revised December 26, 2009: accepted January 31, 2010. Date of publication March 01, 2010. Date of current version April 26, 2010. This work was supported by the EU FP7 project MADNESS (FP7/248424).

The authors are with the Department of Electrical and Electronic Engineering, University of Cagliari, Cagliari, Italy (e-mail: paolo.meloni@diee.unica.it; simone.secchi@diee.unica.it; luigi@diee.unica.it).

Color versions of one or more of the figures in this letter are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LES.2010.2044365

II. Related Work

To date, software cycle-accurate simulation has been the primary tool to allow collaborative hardware and software research [5].

However, for parallel software development, such approaches to simulation do not provide a practical speed-accuracy tradeoff.

A first approach aims at achieving the maximum speed of the simulation by raising the abstraction-level of the described architecture. Simics [14] is one of the best known full-system functional simulators. It offers the level of accuracy necessary to execute fairly complex binaries on the simulated machine, including operating systems. Cycle-accurate timing simulations can be performed including custom modules that extend Simics through its set of application programming interface (APIs). A timing multiprocessor simulator built on top of the Simics library is GEMS [15], a SPARC-based multiprocessor and its memory hierarchy simulation are targeted. Extensions of Simics targeting the simulation of reconfigurable hardware processor extensions have been developed, as reported in [12]. MC-Sim [9] is a multiaccuracy software-based simulator in which the processing cores are simulated with functional accuracy, preserving the highest modularity (through definition of specific APIs) to enable the possible addition of custom processor or cache models. The on-chip interconnection model included in MC-Sim, instead, supports timing simulation. The letter presents also a methodology for automatic generation of fast (claimed 45x over RTL) C-based simulators for coprocessors from a high-level description. ReSP [4] presents a TLM SystemC-based simulation platform that introduces automatically generated Python wrappers that provide increased flexibility, in terms of integration of new components and advanced simulation control capabilities. The Liberty [20] modeling framework emphasizes the reusability of components and the minimization of the specification overhead. The user specifies a structural system description that is automatically translated into a simulator executable.

The second state-of-the-art direction aims at preserving the maximum accuracy of the simulation (cycle-accuracy), by exploiting FPGAs to build hardware-based emulators. Several approaches use FPGAs as a means for accelerating simulation: FAST [7] and A-Ports [19], for instance, map timing and functionally accurate performance models of the circuit under emulation onto reconfigurable logic. Protoflex [8], on the other hand, defines a methodology for implementing a hybrid hardware–software (hw–sw) approach to full-system multiprocessor emulation. Part of the target architecture is prototyped on an FPGA device and part is simulated by a host software-based functional environment. For every simulator the specific hw–sw partitioning depends on the activation rate of the elementary subsystems. Moreover, Protoflex virtualizes the execution of different logical processors on the same hardware resources, to minimize device utilization. The authors claim an average speed-up of 38x over Simics.

Other approaches build the prototyping platform implementing the full-system under test on FPGA. The most important contribution to this field is brought by the RAMP project [2]. Atlas [21] is the first operational design within the RAMP project. It provides an FPGA design composed by processing and transactional memory cores featuring rich software support. In this work the speed-up achievable with on-hardware emulation is clearly assessed. Another example of full-system FPGA-based emulator is [10], where application specific architectures are considered. Moreover, various approaches to FPGA-based emulation have been developed also on the industrial side, such as [11].

Our work is intended to enhance usability and flexibility of FPGA full-system prototyping, by provision of a system-level composition framework with novel features not elsewhere available in literature, such as:

- a "topology builder" that is able to translate high-level directives input by the user into a prototyping platform;
- support for performance extraction during the prototyping, with modalities and granularity specified already at system level;
- support for technology awareness, introduced by means of back-annotation of the prototyping data with power/frequency models, in order to obtain detailed activity-based power figures;
- capabilities of investigating new generation multicore architectures (e.g., including NoC interconnects programmed according to message passing, shared memory, or hybrid models of computation).

Finally, we provide other fundamental information derived by our experience in building the proposed tool.

III. Framework Description

Fig. 1 gives a schematic view of the proposed framework, employed in a possible design space exploration loop.

Input: The designer, starting the exploration, inputs the parallel application to be executed. The application can be parallelized according to shared-memory, message-passing,or hybrid models of computation, employing libraries provided by the framework. Moreover, a high-level structural description of the candidate system, whose composition is described in detail in Section III-A, is taken as input.

Step 1: The system-level description file is passed to the SHMPI (see Section III-A) builder which, building upon a repository of soft-cores, generates the files that are needed for the actual FPGA implementation. The reference repository of HDL IPs contains different Xilinx proprietary processing, memorization, and interconnection cores, along with custom modules for hardware synchronization. Moreover, the Xpipes architecture [6], [1] has been adopted as the template for interconnection network infrastructures. In order to preserve the

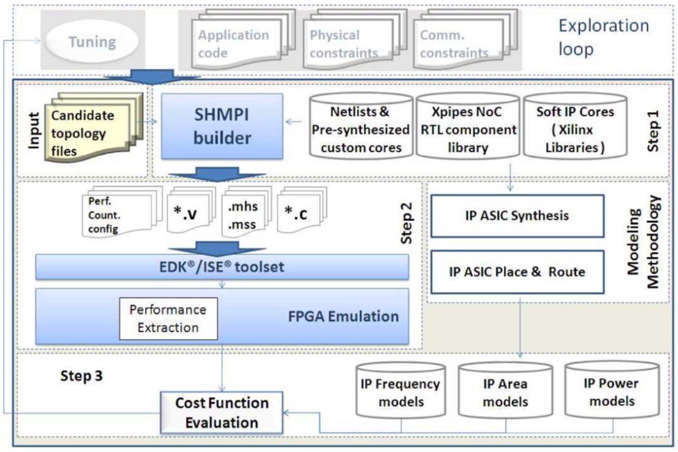

Fig. 1. View of the proposed framework employed in a possible design space exploration loop.

possibility of extending the library with little effort, dedicated wrappers have been developed to adapt the cores to the OCP communication protocol [17].

Step 2: The architectural hardware description and the software libraries generated by the SHMPI builder are then passed to the Xilinx proprietary tools for the FPGA implementation flow; the toolchain handles the configuration and the synthesis of the hardware modules, as well as the compilation of the software code (application kernel, libraries, and drivers). The FPGA emulation, by means of adequate support for performance extraction (described in detail in Section III-B), allows an accurate profiling of the target application on the configured architecture.

Step 3: The cycle-accurate information on the switching activity collected during the emulation, can be passed as input to analytic power and area models, in order to investigate on a prospective ASIC implementation of the system. A deeper insight of the models is provided in Section III-C.

Output: The user can compare the evaluated metrics/cost functions with the constraints that the final system is desired to satisfy and tune the design accordingly.

A. The SHMPI Topology Builder

The SHMPI topology builder generates the actual RTL cores (processing, interconnection, memories) of the platform in a library-based approach (HDL instantiation), based on the system-level specification file input by the designer. The SHMPI topology builder includes the parsing engine of Xpipes compiler, a tool developed for the automatic instantiation of application-specific interconnection networks [13]. New functions have been developed, enabling composition of the entire multicore hardware platform (including processing elements and memory hierarchy definition), automatic configuration of the software libraries, and integration with the Xilinx development tools.

The topology is described, inside the input file, in terms of the following.

- A high-level component-wise topology description of processing cores, memories, and interconnection architecture (switches and links).

- The routing tables to be used by the network interfaces in case a source-routing interconnection network is instantiated.
- All the necessary parameters for the configuration of the processing and interconnection modules.
- The address map of the memory cores and the different memory-mapped peripherals to be included in the platform.
- Specific directives for the inclusion and the connection of the performance/event counters needed for metrics extraction.

The functions developed for the SHMPI builder, on the basis of this system-level description, perform the following operations.

- Instantiation and sizing of the HDL code of processors, peripherals, semaphores, and memory blocks.
- Generation of the input platform hardware and software description files (*.mhs* and *.mss*) for the Xilinx toolchain; the hardware is tailored according to the specified architectural parameters. The compilers are configured according to the chosen kind of processing elements. The linker script and dedicated header files are modified to be compliant with the defined memory map.
- Instantiation of the performance counters and configuration of their access mode. When instantiating a processing or interconnection core, if requested by the system-level description, dedicated performance extraction modules are connected to the module pins.
- Generation of software libraries for performance counter accessing. According to the memory-mapping defined for the performance counters, adequate C functions are created and included for compilation.
- Configuration of the system to support shared memory, message passing, or hybrid models of computation, and relative customization of the synchronization/communication software libraries and processor-to-memory interfaces. In particular, when message-passing support is enabled for a processing core, the related private memory is implemented using two dual-port BRAMs, respectively, for data and instructions. In this case, a memory-mapped DMA module is instantiated and programmed via software to execute send/receive operations.

B. Performance Extraction

The framework provides the possibility to connect dedicated low-overhead memory-mapped event-counters to specific logic chunks. The designer is allowed to monitor the processing core interface, the switch output channel interface (in case a NoC structure is instantiated) and the memory port. The declaration of these event-counters can be included in the topology file that is input to the whole framework; the SHMPI topology builder then handles the insertion and the connection of the necessary hardware modules and wires.

The overhead introduced by the insertion of the performance counters depends on three factors, under full control of the user. First, *which* core is intended to access the performance extraction subsystem. A dedicated processing core can be added and placed on the FPGA, to access all the event-counters without affecting the application execution. Conversely, in order to save hardware resources, the performance counters can be read by the same system processors under emulation. The second factor is *when* the performance counters are accessed; they can be read offline, at the end of the execution, adding dedicated BRAM buffers to store the event traces, or they can be read at runtime, enabling dynamic resources management mechanisms. Finally, *how* they are accessed; the designer can tradeoff additional hardware resources to reduce traffic overhead, instantiating dedicated point-to-point connections instead of using the interconnection layer already used in the system.

C. Models for Prospective ASIC Implementation

In classic hw–sw embedded system design flows, several assumptions made at the system-level design phase are often verified only after the actual back-end implementation. The effort required by this kind of process is increasing with the technology scaling. Thus, introducing “technology awareness” already at system-level would be crucial to increase the productivity and reduce the iterations needed to achieve a physically feasible and efficient system configuration. To this aim, the adoption of some kind of modeling is unavoidable to obtain an early estimation of technology-related features and to take effective technology-aware system-level decisions.

Within the proposed framework, the use of analytic models, is coupled to FPGA fast emulation, to obtain early power and execution time figures related to a prospective ASIC implementation, without the need to perform long postsynthesis software simulations. The FPGA emulation provides event/cycle-based metrics that can be back-annotated using the analytic models for the estimation of the physical figures of interest. This allows to evaluate the timing results according to the modeled target ASIC operating frequencies and to translate the evaluated switching activity in detailed power numbers.

The models included in the framework are built by interpolation of layout-level experimental results obtained after the ASIC implementation of the reference library IPs. More detailed information can be found in [16], referring to the Xpipes NoC building blocks. The accuracy of the models described in [16] is assessed in the letter to be lower then 10% when complete topologies are considered, with respect to post layout analysis of real ASIC implementations.

IV. Use Cases

This Use Case compares three system configurations featuring three alternative NoC topologies as interconnection infrastructure.

The used hardware FPGA-based platform includes a Xilinx Virtex5 XC5VLX330 device. The target application is a shared-memory implementation of the *RadixSort* algorithm, included in the SPLASH2 benchmark suite [18]. The considered topologies are shown in Fig. 2. Each one includes eight processors, eight private memories, and three shared modules (a shared memory *shm*, a bank of hardware semaphores for synchronization purposes *t&s*, and an I/O controller *uart*).

The first explored topology (hereafter called “star”) features a 11×11 central switch, limiting the number of hops between sources and destinations of the communication scheme imposed by the target application. In the second topology (hereafter

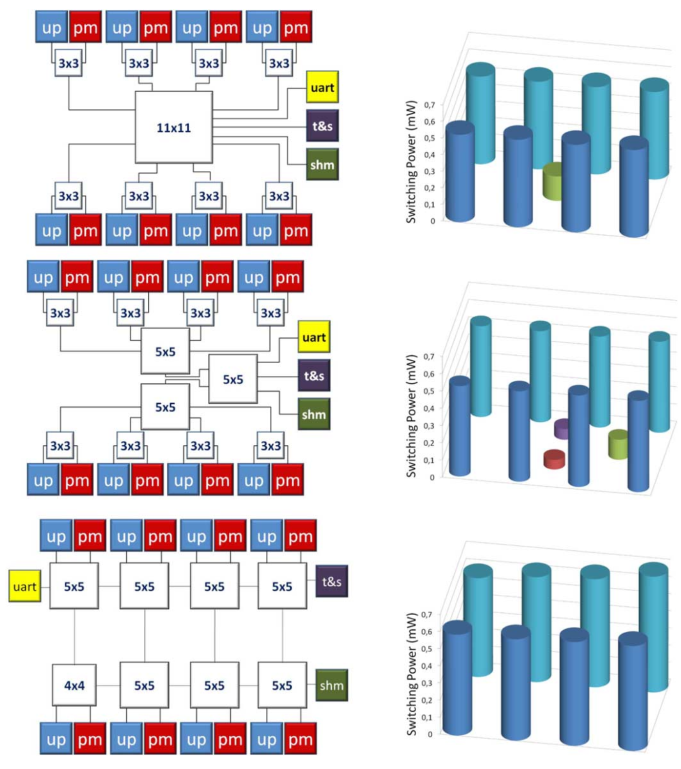

Fig. 2. Explored topologies (left), compared with the per-switch dynamic power consumption (right).

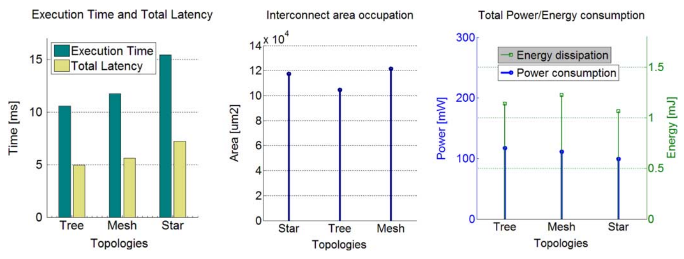

Fig. 3. Execution times and total latencies, area obstruction, energy/power consumption of the compared topologies.

called "tree") the 11×11 switch is replaced with three 5×5 switches, in order to increase the operative frequency at the cost of an increased latency to the shared devices. The last topology under comparison is a quasi-mesh 4×2 topology. The largest switch is still 5×5. Fig. 3 plots various functional and physical metrics evaluated on the examined topologies. The execution times and the total latencies are plotted, both in terms of seconds, thus accounting for the different maximum operating frequencies of the three architectures, estimated with the analytic models. Moreover, the modeled occupied area and the total power/energy consumption of the NoC modules are indicated. The right part of Fig. 2 shows the contribution of each single switch (a single bar) of the topologies to the dynamic power consumed in the different system configurations, estimated observing the flit congestion at every single output channel.

TABLE I
FPGA HARDWARE RESOURCES UTILIZATION AND EMULATION ACTUAL TIME FOR ONE OF THE THREE TOPOLOGIES IN USE CASE

	Tree topology
BRAM blocks	80(27%)
Slice Registers	28618(13%)
Slice LUTs	42529(20%)
Critical Path	12.43 ns
Emulation Time	< 2 sec

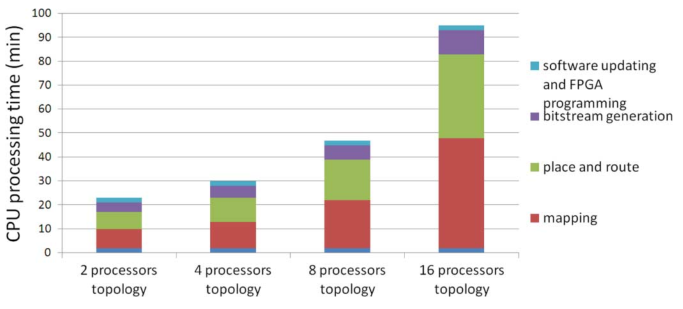

Fig. 4. Computational effort needed to complete the implementation flow for different architectural configurations.

In Fig. 3, "star" and "tree" topologies show contrasting advantages with respect to execution time and energy consumption. In fact, the "star" topology, with its limited operating frequency, shows a higher execution time, that is however, mitigated by the lower power consumption, resulting in a lower energy dissipation.

In Table I, we show the hardware resources required to implement one of the topologies under test. The use case can fit easily on commercial devices. Moreover, we report in the table the critical path inside the system and the obtained emulation time.

V. Simulation Speed and Accuracy Assessment

A point of interest in using FPGA-based emulators is certainly the speedup achievable over software-based cycle-accurate simulators. All the results related to functional and physical metrics showed in Fig. 3 and Fig. 2 are obtained, for each topology, with a single FPGA emulation. The time needed for application execution and performance data outputting is 0.8 sec. This result is coherent with [10] and [21], where multicore FPGA-based emulators are assessed to be three orders of magnitude faster than software-based simulators, when not accounting for the time spent on HW implementation flow. When the emulation platform is instead used inside a design space exploration cycle, a factor limiting the mentioned speedup is the time needed to traverse the whole FPGA implementation flow.

In Fig. 4, we provide an overview of how the FPGA implementation effort scales for regular quasi-mesh topologies with increasing number of processors. The implementation flows have been performed by Xilinx ISE 10.1.3 on a DualCore AMD Opteron (@2.2 GHz) with 6 GB RAM. To shrink the synthesis time, we managed to build a library of reusable presynthesized components with different parameter configurations. For topologies not larger than eight processors, the total FPGA implementation time does not exceed one hour. Thus, we can consider iterative optimization and exploration loops as a potential field of use for the proposed framework, especially when the target application is complex, while the number of integrated cores does not increase very much, like in the embedded systems domain.

To compare with software-based cycle-accurate simulation, we simulated the SystemC RTL models of the NoC modules for a 16-processor mesh topology. The simulation has been performed using *RadixSort* memory-access traces as stimulus and simulation time resulted in more than 12 h. The FPGA-based prototyping of the system, accounting for the implementation effort, takes, according to Fig. 4, about 1 h and 40 min (8x). In those cases in which the exploration extends only to software configuration (mapping, scheduling, TLP granularity), only one synthesis is needed, resulting in a much higher speedup as reported in literature.

Regarding simulation accuracy assessment, it is worth noting that in the proposed use-case the same RTL code could be synthesized for FPGA (for evaluation) and for ASIC (for real production). The prototyping does not insert any error in the estimation of "functional related" (execution time, latency, congestion) performances. Cycle/signal-level accuracy is thus guaranteed without the need of a test comparison (emulated versus prospective implementation). The inaccuracy that might be inserted by the technology-aware models, as mentioned in Section III-C, is affordable in most design cases.

VI. Conclusion

In this letter, an FPGA-based framework for the exploration and characterization of MPSoC architectures is presented, with particular emphasis on NoC-based systems. The two main points of strength of the proposed framework are high-level automatic hw-sw platform instantiation, integrated with Xilinx proprietary tools for FPGA implementation, and the use of analytic models that, exploiting the functional information provided by the FPGA emulation, are able to estimate different technology-related parameters of a prospective ASIC implementation. The presented use case validates the usefulness of the framework in all the contexts where rapid simulation methodologies are required, as an effective support to quantitative design space exploration or simply as an environment for rapid prototyping of complex multicore platforms.

References

[1] F. Angiolini, P. Meloni, S. Carta, L. Benini, and L. Raffo, "Contrasting a NoC and a traditional interconnect fabric with layout awareness," in *Proc. Design, Autom., Test Eur. Conf.*, Munich, Germany, 2006.

[2] Arvind, K. Asanovic, C. Kozyrakis, S. L. Lu, and M. Oskin, RAMP: Research Accelerator for Multiple Processors—A Community Vision for a Shared Experimental Parallel HW/SW Platform 2005.

[3] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The Landscape of Parallel Computing Research: A View From Berkeley" Univ. of California, Berkeley, CA, Tech. Rep. UCB/EECS-2006-183, Dec. 2006.

[4] G. Beltrame, C. Bolchini, L. Fossati, A. Miele, and D. Sciuto, "ReSP: A non-intrusive transaction-level reflective MPSoC simulation platform for design space exploration," in *Proc. 2008 Asia South Pacific Design Autom. Conf.*, Los Alamitos, CA, 2008, pp. 673–678.

[5] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri, "MPARM: Exploring the multi-processor SoC design space with SystemC," *J. VLSI Signal Process. Syst.*, vol. 41, no. 2, pp. 169–182, 2005.

[6] D. Bertozzi and L. Benini, "X-pipes: A network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits Syst. Mag.*, vol. 4, no. 2, pp. 18–31, Sep. 2004.

[7] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat, "FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle-accurate simulators," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Washington, DC, 2007, pp. 249–261.

[8] E. S. Chung, M. K. Papamichael, E. Nurvitadhi, J. C. Hoe, K. Mai, and B. Falsafi, "ProtoFlex: Towards scalable, full-system multiprocessor simulations using FPGAs," *ACM Trans. Reconfig. Technol. Syst.*, vol. 2, no. 2, pp. 1–32, 2009.

[9] J. Cong, K. Gururaj, G. Han, A. Kaplan, M. Naik, and G. Reinman, "MC-Sim: An efficient simulation tool for MPSoC designs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2008, pp. 364–371.

[10] P. G. Del Valle, D. Atienza, I. Magan, J. G. Flores, E. A. Perez, J. M. Mendias, L. Benini, and G. De Micheli, "Architectural exploration of MPSoC designs based on an FPGA emulation framework," in *Proc. 21st Conf. Design Circuits Integrated Syst.*, 2006, pp. 12–18.

[11] Emulation and Verification Engineering—EVE. Zebu XL and ZV Models Tech. Rep., 2005.

[12] W. Fu and K. Compton, "A simulation platform for reconfigurable computing research," *FPL—Field Programmable Logic Appl.*, pp. 1–7, 2006.

[13] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "XpipesCompiler: A tool for instantiating application specific networks on chip," in *Proc. Conf. Design, Autom. Test Eur.*, Washington, DC, 2004, p. 20884.

[14] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[15] M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[16] P. Meloni, I. Loi, F. Angiolini, S. Carta, M. Barbaro, L. Raffo, and L. Benini, "Area and power modeling for networks-on-chip with layout awareness," *VLSI-Design J.*, 2007.

[17] *OCP International Partnership (OCP-IP)*, , 2003 [Online]. Available: http://www.ocpip.org/home, Open Core Protocol Standard

[18] J. Pal Singh, S. C. Woo, M. Ohara, E. Torrie, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. Int. Symp. Comput. Architecture*, 1995.

[19] M. Pellauer, M. Vijayaraghavan, M. Adler, Arvind, and J. Emer, "A-Ports: An efficient abstraction for cycle-accurate performance models on FPGAs," in *Proc. 16th Int. ACM/SIGDA Symp. Field Programmable Gate Arrays*, New York, 2008, pp. 87–96.

[20] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, S. Malik, and D. I. August, "The liberty simulation environment: A deliberate approach to high-level system modeling," *ACM Trans. Comput. Syst.*, vol. 24, no. 3, pp. 211–249, Aug. 2006.

[21] S. Wee, J. Casper, N. Njoroge, Y. Tesylar, D. Ge, C. Kozyrakis, and K. Olukotun, "A practical FPGA-based framework for novel CMP research," in *Proc. 2007 ACM/SIGDA 15th Int. Symp. Field Programmable Gate Arrays*, New York, 2007, pp. 116–125.