# Simulating Next-Generation Cyber-Physical Computing Platforms

***Paolo Burgio***
*University of Modena and Reggio Emilia, Italy; email : paolo.burgio@unimore.it*
***Carlos Alvarez, Eduard Ayguadé, Antonio Filgueras, Daniel Jimenez-Gonzalez, Xavier Martorell and Nacho Navarro***
*Barcelona Supercomputing Center, Spain; email : {name.surname}@bsc.es*
***Roberto Giorgi***
*University of Siena, Italy; email : giorgi@dii.unisi.it*

## Abstract

*In specific domains, such as cyber-physical systems, platforms are quickly evolving to include multiple (many-) cores and programmable logic in a single system-on-chip, while including interfaces to commodity sensors/actuators. Programmable Logic (e.g., FPGA) allows for greater flexibility and dependability. However, the task of extracting the performance/watt potential of heterogeneous many-cores is often demanded at the application level, and this has strong implication on the HW/SW co-design process. Enabling fast prototyping of a board being designed is paramount to enable low time-to-market for applications running on it, and ultimately, for the whole platform: programmers must be provided with accurate hardware models, to support the software development cycle at the very early stages of the design process. Virtual platforms fulfill this need, providing that they can be in turn efficiently developed and tested in a few months timespan. In this position paper we will share our experience in the sphere of the AXIOM project, identifying key properties that virtual platforms modeling next-generation cyber-physical systems should have to quickly enable simulation-based software development for a these platforms.*

## 1 Introduction

As the technological scaling for semiconductors predicted by Moore's law hit the so-called *power wall*, and energy consumption became a primary concern for the market of electronic devices, computing platforms shifted to many-core heterogeneous designs [1, 2, 3, 4]. These platforms are perfectly suited to meet the requirements especially of next-generation cyber-physical systems (CPS), where a huge number of peripherals interacting with the surrounding environment are coupled to a computing board delivering high performance/watt through many-core SMPs and hardware accelerators. Sensors and actuators will be integrated in the design through *ad-hoc* bridges/circuits, or more flexible reprogrammable logic (e.g., FPGAs), composing a system made of several communicating nodes with one or more centralized controllers running on general purpose SMP cores. Hardware accelerators are application-specific circuits which increase the power efficiency of portions (*kernels*) of applications by orders of magnitude. The consequence is that, today, software developers must write code that runs on multiple cores and uses the hardware resources available in the platform, in a productive and effective manner: extracting the tremendous performance/watt potential of such a complex platform essentially becomes also a software development problem. Dependability is also improved when adopting programmable logic: for example, systems based on programmable logic can execute a function in a deterministic way, without the need of a continuous push-pull to/from caches. Most systems based on caches tend to offer a good average performance but may fail to respect a hard deadline in the worst case. Moreover, if the specific architecture fails, reconfiguration of the FPGA can help. Concepts like Data-Flow Threads (DF-Threads) [5, 6] can enable the repetition of the execution of a failed thread.

Virtual platforms are the key to fight this problem, as they enable fast software prototyping at the very early stages of the design cycle of a board, where hardware is not yet 100% available. Computer architects are well aware of this, and in recent years a number of simulator infrastructures have been developed [7, 8, 9], and eventually commercialized, that model a generic or specialized computing fabric with also high accuracy (e.g., cycle-accurate [9, 10]). Unfortunately, correctly modeling the behavior of an hardware platform is time-costly: fully cycle-accurate simulators[1] can be orders of magnitude slower than the corresponding hardware counterparts [11]. For this reason, recently, some virtual platforms were proposed (such as Qemu [11]), for pure functional simulation. They can be successfully adopted in an initial phase to enable functional testing/debugging of the alpha versions of applications, and to quickly exploring the hardware/software partitioning of applications into kernels. Then, software developers might resort to slower and fully cycle-accurate simulators in advanced stages of debugging, until the first prototypes of the board are available.

In this position paper we describe our preliminary analysis on building a virtual platform for simulating cyber-physical systems, in the context of the AXIOM project [12]. AXIOM explores energy-efficient, many-core platforms for next-generation cyber-physical systems. We will briefly describe the guidelines that drive the development of the AXIOM board, in section 2. In section 3 we decompose a simulator for many-core heterogeneous platforms in its basic building blocks, and for each of them we discuss in detail the main issues in simulating it, and how it can (should) be accurately modeled in the quickest way possible. We will do this bringing our expertise on already existing simulation platforms,

[1] cycle accurate virtual platforms mimic the behavior of each component of a system at every clock cycle

both industry [7, 8] and academical solutions [10]. Finally, section 4 draws some conclusions.

## 2 Requirements for a cyber-physical system: The Axiom project

We are entering the *cyber-physical age*, where both objects and people will become nodes of the same digital network for exchanging information. This vision is also referred to as *"Internet of Things"* (IoT) becauses the general expectation is that "things" or systems will become somewhat smart as people, allowing a tight system-to-human and device-to-environment interaction. As a consequence, we expect that such cyber-physical systems (CPS) will at least react in real-times, consume the least possible energy for a given task, scale up through modularity, and allow for an easy programmability across performance scaling. The whole set of these expectations impose scientific and technological challenges that AXIOM project (Agile, eXtensible, fast I/O Module[12, 13]) tries to address, exploring new hardware/-software architectures for CPSs.

Communities [14, 15, 16] that are using CPSs are devising more and more the need for more powerful embedded platforms that could be: i) easy programmable through an almost standard software toolchain; ii) be customizable with programmable logic (i.e., FPGAs), iii) be extensible to one or more boards (e.g., two robotic arms that need to be closely synchronized toward a single real-time task); iv) provide an easy way to integrate sensors (e.g., through widely available Arduino [15] shields). Current solutions providing enough energy-efficient computational power for fulfilling this needs are starting to rely more and more on multi- and many-core architectures (e.g., UDOO [14] and RaspberryPi2 [16] rely on a quad-core and GPUs) . For example, some current research projects (such as ADEPT [17] or FP7 P-SOCRATES [18]) are already investigating how to join efforts from the high-performance computing (HPC) and the embedded computing domains, which are both focused on high power efficiency, while GPUs and new dataflow platforms such as Maxeler's [19], or in general FPGAs, are claimed as the most energy efficient.

AXIOM research mainly targets designs coupling power-efficient multiple cores, such as ARM ones, and FPGA accelerators on the same die as in the Xilinx Zynq [1], and produce prototypes of single-board computers, similar to UDOO [14], Arduino [15] and RaspberryPi [16]. This architecture includes capability to high-speed board-to-board interconnects and controllers for commodity CPS peripherals such as *Arduino Shields*. AXIOM partners will start the development using a virtual platform: this paper reports the preliminary results of such investigations. At the same time, the tested parts, when ready, are progressively migrated on the FPGA prototype (a Xilinx ZC-706). As a consequence, the AXIOM project requires a virtual platform which simulates general-purpose cores, programmable logic (for accelerators), and peripherals ASIC circuits that integrate sensors and actuators. Figure 1 shows the scheme of a computing platform including two general-purpose cores, FPGA logics and a few peripherals/sensors connected to it.

From the software viewpoint, the AXIOM system will interact with and react to the surrounding environment by properly managing actions in real-time through an operating system (such as Linux), a well-known parallel programming model:

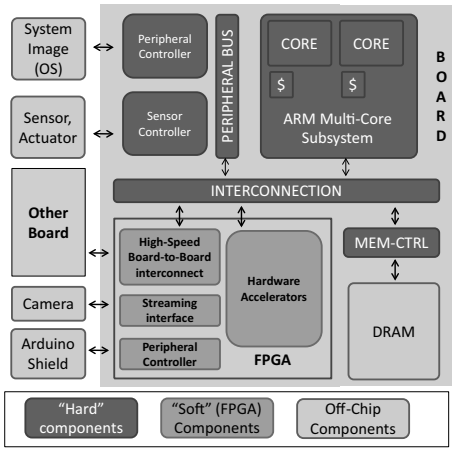


**Figure 1: Heterogeneous computing platform with sensors**

OmpSs [20]. By using OmpSs, applications will be hardware/software partitioned, i.e., decomposed in parallel tasks that can be mapped on multiple software execution units (OS threads) and/or hardware execution units, e.g. the accelerators in the FPGA. This provides a huge number of options for mapping tasks to resources, considering the device on which a task is mapped, the size of the input data, the data transfer time, or the different speed of the devices in executing the task. Tools and techniques for quickly finding the optimal HW/SW partitioning of applications according to performance and power metrics, and to validate them against real-time constraints, are therefore crucial for the project. The issue is that, when all tasks are mapped either on hardware or software, a complete FPGA synthesis flow for hardware accelerators can spend from hours to days, depending on the size of the computational kernels to process. With virtual platforms, on the contrary, new accelerator elements can be quickly added in the simulation environment, and we can run a timing accurate full system simulation of the applications partitioned on the SMP cores and FPGA accelerators in a matter of minutes to few hours.

## 3 Virtual Platform requirements

This section describes how to build a virtual platform for a computing system such as the one targeted by AXIOM. Starting from AXIOM specifications, we will first describe its basic building blocks, and discuss how a proper design for each of them will enable fast prototyping of the target board. We will bring our expertise, previously gained using/developing two simulator for heterogeneous systems, namely COTSon [7] in the TERAFLUX project [21, 22], and a prototype built after the open-source academical VirtualSoC [9] by University of Bologna: HC-VSoC [10, 23]. We will also refer to other existing simulator infrastructure of potential interest.

From AXIOM specifications, the simulator must enable quick software prototyping of a system *whose hardware architecture is not 100% defined at early stages of the project*. We identify these four key requirements:

1)immediate availability of at least a first functional version of the simulator, to let the software development cycle start;
2) possibility of defining architectures and their timing model for cycle accurate evaluations, to be selectively used in combination with functional models;

3) the virtual platform must be capable of integrating multiple modules (such as proximity sensors), that *generate/simulate events coming from the surrounding environment*, hence whose behavior must be random, or driven by user/parametrizable;
4) we must be capable of easily putting new hardware models in the design, and to replace (fast and inaccurate) behavioral models of its components, e.g., sensors and actuators, with more timing-accurate (and slower) versions.

Requirements 1) and 2) match the experience of the COTSon [7] simulator, while requirements 3) and 4) match the experience of the HC-VSoC [10, 23] simulator, which models platforms with user-defined hardware accelerators called HWPUs, i.e., whose functionality is defined by the end-user. HC-VSoC architects reduce the problem complexity by specifying a clearly-defined common communication contract and infrastructure for all the blocks modeled in the systems. SMP cores and HWPUs are equipped with a memory shell that supports that communication protocol. Also the COTSon [7] virtual platform, whose primary design requirement was to build a highly scalable architecture, employs a similar solution, providing a well-defined communication API.

This brings us to the first component of a virtual platform, the **simulation engine**, which supports/simulates the interaction between modeled blocks. A number of tools for this exist, both coming from industry and academia, and the most known is probably SystemC [24] by Accelera. The SystemC package is a very flexible macro library (C++ language) and a simulation engine, to simulate the behavior of hardware blocks with different levels of abstraction and accuracy, from RTL- to cycle-accurate, to transactional-level modeling. Higly-scalable infrastructures (such as – as explained before – COTSon [7], or OVP [8]) expose a very simple API to integrate hardware blocks in their engines, and come with a few prebuilt architecture models. These are the best solution if the architecture models included in these packages partially or totally match the one being developed.

An second component that must be carefully designed at early stages is the **interconnection**, which emulates on- or off-chip connectivity. Designing an interconnection infrastructure with acceptable good tradeoff of simulation speed, scalability, and timing accuracy is not trivial and it is probably the most time-consuming part in developing virtual platforms. In addition, in AXIOM, the interconnection must enable fast integration of the future versions of the hardware models, to meet requirement 4). An example of scalable communication *medium* is the one connecting multiple COTSon [7] nodes, or the one of HC-VSoC [23], whose protocol is called PINOUT. The difference between them is in the way they are implemented: in the former it is exposed as a pluggable model rigorously decoupled as a functional model and a timing model. Hence the simulated hardware blocks access to the interconnection by directly invoking a simple given API. The latter is itself an instantiated as a SystemC modules with its own model of hardware ports, and a timing accuracy given by design. An amenable property of a simulated interconnection (although not a critical requirement for AXIOM), is that it should be possible to customize its internals and the modeled communication delay should be driven, e.g., *via* simulator parameters or configuration files. An example of configuration files for a simulator is shown in Figure 2. It was developed in the PREDATOR FP7 project [25].

Rows and columns in the figure simulate a hierarchical crossbar by specifying the communication delay among each master port (e.g., SMP cores) and each slave ports (memories and peripherals), respectively. In this example, we are modeling an N-to-M crossbar with user-defined delay for each master-to-slave (core-to-mem bank) path: for instance, we note that the M-th slave implements the controller for an off-chip DDR, because the delay that each master "sees" to get to it (specified in the last column) are significantly higher than for other memories.

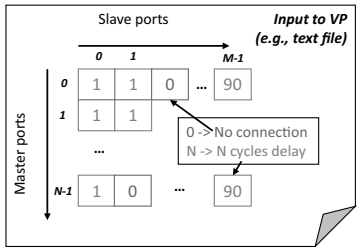


**Figure 2: Parametrizable interconnection model**

**General Purpose cores** are the most complex component of a simulator, as they must correctly mimic the functional behavior, e.g., of modern superscalar cores, with branch-prediction units and multiple deep pipelines, or the complex hierarchical shared cache systems and prefetch buffers of next-generation many-core architectures. Luckily, the choice of the instruction-set architecture (ISA) and core model to adopt is usually made at the very beginning of the project, and it does not change in the following. Moreover, most of the simulation infrastructures provide a portfolio of processor models, which is often freely available as a library (see for instance OVP [8]).

The key point in integrating core models in a bigger design is that, in order to support the development of software, *each core model must come with the required toolchain for compiling the code of applications, deploying them on the simulator and – possibly – to support debugging* to do what ultimately is the main purpose of a virtual platform: support software development. In AXIOM, this is reflected in requirement 1).

A few examples can be:

- COTSon [7], which includes x86_64 processor models together with the associated toolchain;
- The HC-VSoC package [10, 23] targets for ARM-based embedded systems, and it comes with a "standard" GNU Compiler Collection (GCC [26]) cross-compiled for it;
- Open Virtual Platforms by Imperas [8] provides a wide portfolio of core models, including ARM (32 and 64 bit), Imagination MIPS (32 and 64 bit), PowerPC, Xilinx Microblaze, and many more.

A project can also adopt a proprietary ISA from a specific provider: they also usually come with a library/software package that simulates a single processing core, using "open" simulation engines (e.g., SystemC), or again with in-house engines or define ISEs (ISA Extensions).

Due to its complexity, the processor model is usually the component from which the development of a virtual platform starts, together with the simulating engine. For this reason, the preliminary version of the platform provided to programmers typically embeds only one or multiple SMP cores, the interconnection model, and a few memories, with limited set

of peripherals. Using this, software developers for an heterogeneous platform (such as AXIOM's) can immediately compile, deploy and test the "host/SMP part" of their code.

**Programmable logic and peripherals (and sensors/actuators)**. The platform template targeted by AXIOM embeds on-chip programmable logic, as well as a number of peripherals controllers to interact, e.g., with sensors or Arduino shields. Once the communication infrastructure has been set, and a scalable model of the on-chip interconnection implemented as explained before, it is extremely easy to include in the simulator in-house customized models for peripherals and hardware accelerators. For instance, the PINOUT interface in [23] is implemented in the so-called COMU of HC-VSoC HWPUs. Internally, each of HWPU model can be implemented with a different simulation speed/timing accuracy tradeoff, as required by project specification.

**Integration with external components**. In the AXIOM project, peripheral components will either interact with the surrounding environment, or connect the board to COTS components or 3rd party subsystems such as the Arduino Shields, and the virtual platform will simulate these behaviors. In the first case, we can employ parameters or proper input files for the simulator that mimic the surrounding environment. For instance, the behavior of temperature sensors can be easily defined *via* simple input text files describing the variation of the temperature in time. The second scenario, in turn, has a great impact on the simulation infrastructure, and raises a potential problem. Simulator developers might need to integrate pre-existing models of the two platforms (e.g., the core model running on SystemC, and the model of an Arduino running on a proprietary simulation engine), which are potentially not designed to communicate each other, or can even be written in different programming language. This possible incompatibility in the communication between simulator models, may require to implement stub functions to transform the information between formats understood by the two components.

**Memories**. In current virtual platforms, typically memories are implemented as "wrappers" that simply add a delay for accessing big arrays of data modeling the memory banks. For this reason, it is not uncommon that virtual platform developers create their in-house simulation models of memories, when possible. More complex or "fancy" memory models, such as smart memories, can be easily implemented starting from these components.

**Support software libraries** Applications running on the simulator might employ specific standard libraries, such as `libc` and `libsdtc++`, or custom runtimes, such as `nanos++` [27], or `libhwpu` (in HC-VSoC) to do their work. This is also the case of AXIOM. In this case, the simulator infrastructure must support the same set of required APIs as the "real" board, to ensure code portability.

# 4 Conclusions

We presented in this paper the approach used by the AXIOM project for flexibly simulating a realistic Cyber-Physical System, soon to be implemented as single board computer. Mainly, besides a FPGA prototype, we developed the preliminary steps through virtual platforms. In particular two platforms had been selected: COTSon and HC-VSoC as they can provide the best support for our design needs. In particular, the inclusion of FPGA in the simulation toolchain provides support for exploring dependability options.

# 5 Acknowledgment

This work is supported by the AXIOM project, funded by EU H2020 program (grant ICT-01-2014 GA 645496), the Spanish Government, through the Severo Ochoa program (grant SEV-2011-00067) the Spanish Ministry of Science and Technology (TIN2012-34557) and the Generalitat de Catalunya (MPEXPAR, 2014-SGR-1051). Authors would also like to thank the anonymous reviewers for their precious feedback.

# References

[1] Xilinx Inc., *Zynq Series*.

[2] G. Kyriazis (2012), *Heterogeneous System Architecture: A Technical Review*.

[3] AMD, *The AMD Fusion Family of APUs*.

[4] R. Giorgi (2015), *Scalable embedded systems: Towards the convergence of high-performance and embedded computing*, in Proceedings of the 13th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing.

[5] R. Giorgi and P. Faraboschi (2014), *An Introduction to DF-Threads and their Execution Model*, in IEEE Proceedings of MPP-2014, (Paris, France), pp. 60–65.

[6] S. Weis, A. Garbade, B. Fechner, A. Mendelson, R. Giorgi, and T. Ungerer (2014), *Architectural support for fault tolerance in a teradevice dataflow system*, Springer International Journal of Parallel Programming, pp. 1–25.

[7] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega (2009), *COTSon: Infrastructure for Full System Simulation*, SIGOPS Oper. Syst. Rev., vol. 43, pp. 52–61.

[8] Imperas Software, *OVP – Open Virtual Platforms*.

[9] D. Bortolotti, C. Pinto, A. Marongiu, M. Ruggiero, and L. Benini (2013), *VirtualSoC: A Full-System Simulation Environment for Massively Parallel Heterogeneous System-on-Chip*, in Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, pp. 2182–2187.

[10] P. Burgio, A. Marongiu, D. Heller, C. Chavet, P. Coussy, and L. Benini (2012), *OpenMP-based Synergistic Parallelization and HW Acceleration for On-Chip Shared-Memory Clusters*, in 15th Euromicro Conference on Digital System Design, DSD 2012, Cesme, Izmir, Turkey, pp. 751–758.

[11] F. Bellard (2005), *QEMU, a Fast and Portable Dynamic Translator*, in Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC '05, (Berkeley, CA, USA), pp. 41–41, USENIX Association.

[12] D. Theodoropoulos *et al.* (2015), *The AXIOM project (agile, extensible, fast i/o module)*, in IEEE Proceedings of the 15th International Conference on Embedded Computer Systems: Architecture, MOdeling and Simulation.

[13] C. Alvarez, E. Ayguade, J. Bueno, A. Filgueras, D. Jimenez-Gonzalez, X. Martorell, N. Navarro, D. Theodoropoulos, D. N. Pnevmatikatos, C. Scordino, P. Gai, C. Segura, C. Fernandez, D. Oro, J. R. Saeta, P. Passera, A. Pomella, A. Rizzo, and R. Giorgi (2015), *The AXIOM software layers*, IEEE Proceedings of the 18th EUROMICRO-DSD, pp. 117–124.

[14] UDOO, *Android Linux Arduino in a tiny single-board computer*.

[15] M. Banzi (2008), *Getting Started with Arduino*. Sebastopol, CA: Make Books - Imprint of: O'Reilly Media.

[16] The Raspberry Pi Foundation., *The Raspberry Pi Board*.

[17] The ADEPT Consortium, *ADEPT – Addressing Energy in Parallel Tehcnologies*. [Online]. Available: http://www.adept-project.eu/.

[18] L. M. Pinho, V. Nélis, P. M. Yomsi, E. Quiñones, M. Bertogna, P. Burgio, A. Marongiu, C. Scordino, P. Gai, M. Ramponi, and M. Mardiak (2015), *P-SOCRATES: a Parallel Software Framework for Time-Critical Many-Core Systems*, Microprocess. Microsyst., vol. 39, no. 8, pp. 1190–1203. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2015.06.004.

[19] Maxeler Technologies, *MPT Hardware*.

[20] J. Bueno, L. Martinell, A. Duran, M. Farreras, X. Martorell, R. M. Badia, E. Ayguade, and J. Labarta (2011), *Productive Cluster Programming with OmpSs*, in Proceedings of the 17th International Conference on Parallel Processing - Volume Part I, Euro-Par'11, (Berlin, Heidelberg), pp. 555–566, Springer-Verlag.

[21] R. Giorgi *et al.* (2014), *TERAFLUX: Harnessing dataflow in next generation teradevices*, Microprocessors and Microsystems, vol. 38, no. 8, Part B, pp. 976 – 990.

[22] R. Giorgi and A. Scionti (2015), *A scalable thread scheduling co-processor based on data-flow principles*, ELSEVIER Future Generation Computer Systems, pp. 1–10.

[23] P. Burgio, A. Marongiu, P. Coussy, and L. Benini (2014), *A HLS-Based Toolflow to Design Next-Generation Heterogeneous Many-Core Platforms with Shared Memory*, in 12th IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2014, Milano, Italy, pp. 130–137.

[24] Accelerat Systems Initatives, *SystemC*.

[25] P. Burgio, M. Ruggiero, and L. Benini (2010), *Simulating Future Automotive Systems*, tech. rep., DEIS - University of Bologna.

[26] The Free Software Foundation, *GCC – The Gnu Compiler Collection*.

[27] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas (2011), *OmpSs: A proposal for programming heterogeneous multi-core architectures*, Parallel Processing Letters, vol. 21, pp. 173–193.