A Dynamic FPGA-based Hardware-in-the-Loop Co-simulation and Prototype Testing Platform

Clément Foucher, Alexandre Nketsa

A Dynamic FPGA-based Hardware-in-the-Loop Co-simulation and Prototype Testing Platform

Clément Foucher, Alexandre Nketsa

CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France
E-mail: {Clement.Foucher, Alexandre.Nketsa}@laas.fr

***Abstract*—The base idea of co-simulation is to couple heterogeneous simulators in a single environment. This allows for choosing the best-suited simulator to represent each part of a complex system. Hardware-in-the-loop co-simulation introduces physical components in a software co-simulation. In this paper, we propose a hardware-in-the-loop co-simulation platform using a dynamically reconfigurable architecture on FPGAs. Main uses for this technology include introducing prototypes of digital systems directly in the simulation environment, and accelerating simulation by using FPGAs to implement models that can take advantage of parallelism. The platform was validated by coupling a C++ application with hardware modules loaded on-demand on a FPGA.**

Keywords–Co-simulation; Prototyping; Hardware-in-the-Loop; Reconfigurable Architectures.

I. Introduction

Co-simulation is the process of making multiple simulators collaborate in a larger simulation. In a co-simulation, each simulator is in charge of simulating a part of the system, and has an interface to exchange data and synchronize with its environment, which is constituted of other simulators. Using a co-simulation approach rather that simulating the whole system within a single simulator is an answer to growing simulation needs as systems get increasingly complex.

Indeed, in systems composed of many heterogeneous sub-systems, very different simulation needs can emerge. There are integrated tools, supporting most of these needs, such as COMSOL Multiphysics [1], which integrates a large number of modules to simulate a heterogeneous system. But as it is difficult to design software that answers each and every simulation need, the co-simulation paradigm seems a promising alternative approach. Profession-centric tools are also developed in that way, such as CNES' BASILES [2] co-simulation framework, which integrates different tools to provide a methodology targeting satellite simulation.

The separation of models in a co-simulation allows for easy replacement of a model by another, as long as the interfaces are preserved. This capability is well-suited for model-based approaches [3], in which models are progressively refined from a very high level description to a precise description of the target system. Extending this mechanism to system's prototype by replacing a model by its physical implementation introduces the Hardware-in-the-Loop (HIL) notion. HIL allows for test and validation in a fully managed environment. This notably makes possible replaying simulation scenarios to ensure the implementation behaves as expected and to compare prototype outputs with the one obtained from the higher level model. This is done by wrapping the prototype in an interface that allows simulators to send commands to the physical system through actuators, as well as sensors to provide data the other way.

Hardware implementations can be of many natures, but we focus here on digital hardware systems, e.g., systems on chips, or control-command parts of a system. Thus in the following, the term *hardware* will refer to logic circuits, by opposition to *software*. We use dynamically programmable hardware, taking advantage of System on Programmable Chip (SoPC) architecture introduced by Field-Programmable Gate Array (FPGA) devices.

In this kind of HIL co-simulation, the programmable hardware can be of two uses:

- Implementing a prototype of a digital system to join the simulation, replacing its higher-level software model,
- Implementing a model which can take advantage of parallelism to see its performances improved being executed on hardware rather than software.

These two different applications of SoPC HIL share a large part of the deployment approach.

In the former, we use a SoPC to implement a logic circuit prototype. FPGAs deliver various advantages for this consideration, the first being the reusability of the hardware. Indeed, nothing but development time is lost in case the circuit is faulty, as opposite as an error in a specifically made circuit. Moreover, programmable logic enhances high debug capabilities. It is easy to add observer logic that will not interfere with the actual purpose of the circuit, thanks to the high parallelism offered by the programmable resources.

In the latter, FPGA is just a computing resource, as those that can be found in High Performance Reconfigurable Computers (HPRCs) [4]. Indeed, hardware implementation enhances concurrent execution and parallelism. This generally improves the speed of algorithms, and may even achieve impressive speedups from single-core software execution for some algorithms. Generally, the more parallelizable is the algorithm, the more substantial is the speed gain.

What we propose here is a solution that achieves the following objectives:

- Easy integration and automated deployment of hardware modules in a distributed heterogeneous co-simulation,

- Dynamic loading/unloading of hardware modules thanks to partial reconfiguration,
- Intuitive interface definition for communication between modules,
- Automatic handling of FPGAs partial reconfiguration, turning a single chip in multiple independent programmable resources.

We developed a platform by combining an existing co-simulation solution, CosiMate [5], with a previous work on managing partially reconfigurable resources, the Simple Parallel platform for Reconfigurable Environment (SPoRE) [6].

SPoRE is a tool for handling FPGA-based computing resources. It allows executing computation kernels on distributed reconfigurable resources from a remote workstation. CosiMate is a co-simulation bus supporting various software simulation tools, and providing a communication and synchronization interface between them. By combining the SPoRE platform and the CosiMate environment, we set up a heterogeneous co-simulation environment using both software and dynamically reconfigurable hardware.

In the following, we begin by taking a look at existing solutions in Section II. Then we present our platform building blocks in Section III, the platform itself in Section IV, and how we build a co-simulation for this platform in Section V. Finally, we analyze the platform usages in Section VI and present the perspectives in Section VII.

II. Related Work

Connecting reconfigurable hardware logic to software in order to take advantage of both kinds of computations allows for powerful applications. This process is notably used in the rising generation of HPRCs [4], which are massive computing farms containing FPGAs tightly coupled to processors, the latter delegating intensive computation kernels to the former, acting as application-specific co-processors.

In [7], Liang *et al.* use a combination of MATLAB/Simulink and Xilinx System Generator (XSG) to communicate between software and hardware. Their co-simulation process follow the MDE guidelines [3], beginning by simulating a module, then coding it in HDL and finally executing it on a FPGA as a HIL process. Nevertheless, the communication between software and hardware is relying on proprietary protocols inherent to the tools. Using this solution, there is no control on how the FPGA is handled by XSG, which is done in a static way. This is the major difference with our solution, which notably allows partial reconfiguration of a FPGA, thus allowing for multi-IP design on a single FPGA chip. Moreover, our tool is able to handle multiple FPGA running in parallel on a network, thus multiplying the available resources.

Liao *et al.* present a coupling technique between a HDL simulator and a hardware module running on FPGA [8]. Their solution implements an efficient synchronization technique as hardware clock signal is generated from software, allowing for synchronous operations. Nevertheless, this solution prevents from taking full advantage of the speedup allowed by hardware implementation. Moreover, this is not clear how the communication ports, on both hardware and software, are generated: is this an automatic process or does ports have to be manually tailored. In our solution, hardware ports rely on bus-based registers, while software ports are based on CosiMate formalism.

An important part of introducing a prototype in a HIL co-simulation is to be able to reproduce on the hardware the exact stimuli applied to the corresponding software model. In [9], authors instantiate a hardware module on a FPGA, and link it to a testbench in a simulator. This technique allows for simulating a HDL design in a simulator, and then deport the design itself on a reconfigurable device while preserving the test scenario. They use the SCE-MI API [10] to communicate between a HDL simulator and hardware implemented on FPGA. While SCE-MI is a very interesting approach for heterogeneous communication, it does not handle the hardware deployment, where our solution allows for automatically handling FPGA configuration using partial reconfiguration.

III. HIL Co-Simulation Platform Base Blocks

To build the co-simulation platform, we combined two existing platform. On one hand, we used the CosiMate software [5] from ChiasTek, a co-simulation bus allowing putting together various simulators. On the other hand, we extended the SPoRE platform [6], previously developed by our means, allowing remote control and managed reconfiguration of FPGA-based nodes through a network.

CosiMate, as shown on Figure 1, is a bus on which standard simulators are plugged. CosiMate offers a standard interface through ports and synchronization mechanisms for simulators.

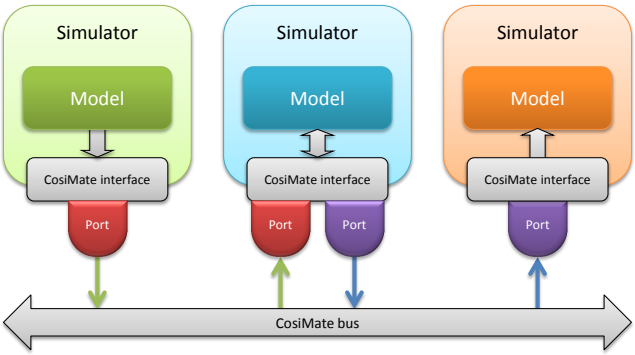

Figure 1. CosiMate bus architecture.

A port is defined with a direction and a data type, e.g., an output integer port on an IP will provide 32-bit data to the bus. Ports can also be defined as arrays to allow transferring blocks of data with larger size than basic types.

A simulator needs an extension library to be compatible with CosiMate. In such a simulator, user can define ports using a specific syntax, which will allow for generation of a XML-based description. A CosiMate project gathers these description files and allows links to be made between ports.

An output port from a simulator can be linked to one or various input ports of the same type and size of another simulator. Then, launching the co-simulation requires all the simulators to be running, and CosiMate automatically handles communication between ports.

Two modes are supported by CosiMate: synchronized and event-driven. In synchronized mode, all simulators wait on a barrier at each simulation step, the barrier being released by CosiMate environment when all simulators have reached it. Communication synchronization is done at each simulation step. This mode enables simulation time to be managed.

In event-driven mode, data flow though ports without any barrier, thus the synchronization process is up to the user. Moreover, event-driven mode also requires user to define a protocol for communication, as there is no time step to indicate when data is available.

The SPoRE platform is a distributed-node platform, which nodes contain FPGAs and are linked by a network, as displayed on Figure 2. The arrows indicate which node initiates the communication.

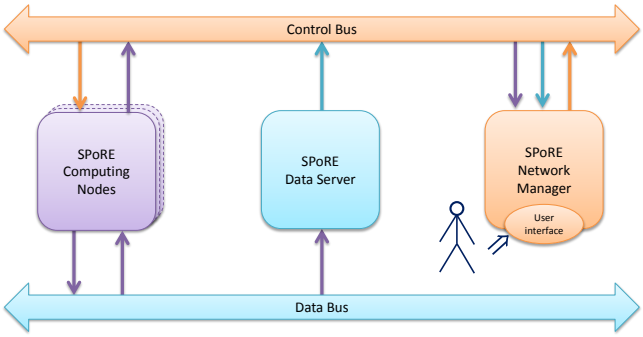

Figure 2. SPoRE buses representation.

As communication between nodes relies on Sockets, the network itself is abstracted, and could take different shapes. In our case, we use an Ethernet-based network. The SPoRE platform contains one or more computing nodes, at least one data server, and exactly one network manager node, from which the user commands the platform.

SPoRE uses a XML description to build an application. SPoRE applications are wrappers that indicate how to use FPGA partial configuration files implementing IPs. SPoRE partial reconfiguration mechanism and node management are discussed in [6] and [11].

When user already has a HDL description of needed computing kernels, a simple vision of SPoRE by user is as follows:

- The user describes computation kernels as black boxes containing in/out ports,
- The user writes an application by describing which kernels are to be used, and for each one, associate ports to application's *message paths*,
- The application description (XML), the kernels descriptions (XML) and the FPGA partial configuration files implementing kernels (binary) are stored on a data server,
- The user launches the application from the network manager,
- The user retrieves the results from the data server.

SPoRE automatically handles application description and bitstream download on computing nodes, does the reconfiguration process, download and compute data, and upload results to the data server.

IV. Building the Platform

The way we chose for linking CosiMate and SPoRE was to use the base representations of each platform to build a bridge. The bridge should then be viewed as a simulator by the CosiMate environment and as a computing node by the SPoRE platform, as presented on Figure 3. CosiMate supports C/C++ written simulators, and SPoRE only needs Socket support to declare a node. We then choose to build the bridge using C++ and Qt, to enhance portability.

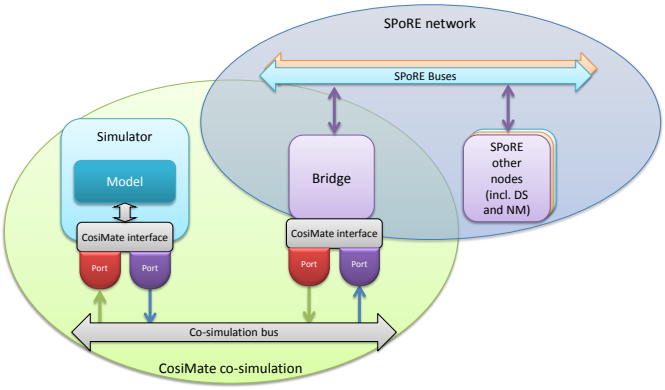

Figure 3. Bridge between CosiMate and SPoRE.

SPoRE platform has been extended to support the bridge. First, we replaced the scheduler to allow IP reconfiguration being done depending on the co-simulation needs. Due to SPoRE modular nature, this replacement does not override the previous scheduler, the scheduler choice is proposed to the user. The scheduler dynamically instantiates the modules when needed in co-simulation, and can erase them when not needed any more. The reconfiguration process itself relies on SPoRE capabilities and is transparent to the user.

In SPoRE application descriptor, we added a reference to a co-simulation descriptor. This reference is ignored by common SPoRE nodes, but can be interpreted by the bridge. The co-simulation descriptor indicates the relation between SPoRE message paths and CosiMate ports. Using this file, the bridge dynamically creates CosiMate ports, allowing for the generation of the CosiMate configuration file. The CosiMate ports are thus dynamic, and depend on the application. There is no need to write a specific XML configuration file for CosiMate, this is done automatically to match SPoRE IP ports.

We use the co-simulation environment in event-driven mode, using a simple request/acknowledge protocol for data

handling, as shown on Figure 4. Indeed, when declaring an

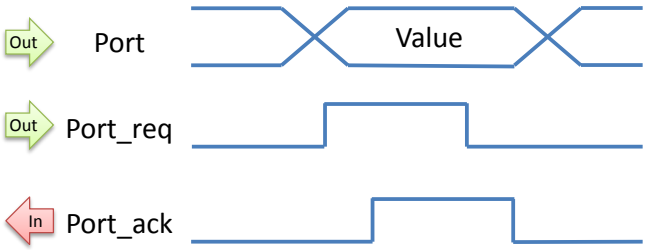

Figure 4. Bridge protocol for an output port.

output (resp. input) CosiMate port, the bridge actually declares two additional single-bit ports, one output (resp. input) for request, and one input (resp. output) for acknowledge.

In SPoRE environment, messages are referenced by a data server. SPoRE uses a *message path* mechanism to link messages to kernels. Message paths are FIFOs containing messages identifiers (IDs) and owners.

In an application description, each kernel port actually used in the application is linked to a message path. All messages produced by an output port will be declared by node to data server in the corresponding message path using a unique message ID. Conversely, when a message is available in a message path linked to an input port, the node will be advised by data server, that will indicate which node hosts the message. Messages are then downloaded directly between nodes. Note that if a node hosts both ends of a message path, the data server will still manage the message path to ensure data coherency, but no download will be necessary. This message path mechanism was initially added to SPoRE while building this bridge, but is now the default behavior for port management.

When an event comes to the bridge from CosiMate, a SPoRE message is dynamically generated, and SPoRE data server is advised of that creation. If a SPoRE node is listening on the according message path, it will then be able to download the message from the bridge. Conversely, the bridge listens on output messages path linked to CosiMate ports. When a message is produced in such a message path, the bridge is advised by SPoRE data server, and will download the message from the producer node, eventually initiating an event on corresponding CosiMate port to transmit the message content.

V. Implementing Models as Hardware

Following the model-based prototyping method [3], the co-simulation process begins by simulating high-level models of the system. The model is then progressively refined, until a HDL implementation is done. This implementation can be done manually by describing the IP core in HDL language, or make use of model-to-text [12] or high-level synthesis tools [13] to generate the code.

Second phase would be to simulate this code using a HDL simulation tool plugged to the co-simulation bus. ModelSim [14] is an example of tool supported by CosiMate. Some minor signal adaptation may have to be done at this phase, as data type representation can differ between software and hardware, notably number of bits used to represent a signal. Afterward the results of this simulation should not differ from the simulation using high-level models, or the error between simulations should be in an acceptable range to validate the implementation [15].

Finally, the synthesis of the hardware module has to be done following SPoRE bus-based architecture. SPoRE implements computing kernels as *cells* plugged to a bus, which notably allows IPs for direct access to data in RAM. This bus-based interface will generally be easy to implement for most IPs. In certain cases however, this mandatory interface will cause some architectural restriction. Typically, this can be the case of some IPs requiring to be fed two or more messages at the same clock cycle. Nevertheless, this can be easily worked around by adding small controllers that will store messages and write them to the IP when all are available.

When correctly wrapped in a SPoRE cell interface, the user describes the IP ports and their protocol using SPoRE descriptor syntax. This allows for use in any SPoRE application, including co-simulation applications by instantiating the bridge.

This integration can be seen on multiple levels. As the bridge concentrates all data exchanges between the two platforms, each platform can be seen by the other as *being* the bridge. This means that we can see the complete platform as a CosiMate co-simulation integrating a FPGA-based simulator, or as a SPoRE platform integrating software simulators. Moreover, due to SPoRE distributed nature and CosiMate allowing multi-host co-simulation, we can easily integrate two or more SPoRE platforms by using as many bridges as necessary, in order to overcome a bottleneck if needed.

For now, the bridge only supports integer transmission between the platforms. If data size between a SPoRE message path and a CosiMate port does not match, a stack is automatically defined. As an example, we tested a FPGA-based AES encryption that requires 128-bit word length, and coupled it with 32-bit integer ports in CosiMate. To do that, the bridge waits for 4 messages from the co-simulation environment before it generate a SPoRE message. Conversely, a SPoRE message will generate 4 successive CosiMate messages when received. Another way to handle this difference is to force data size matching by using CosiMate array mechanism, and treat an array of 4 integers as a single 128-bit message.

The test scenario we built is an AES encoder/decoder prototype testing. The hardware part contains two modules: an encoder and a decoder. The software part consists in a small C++ software that allows for selecting a local file and choosing whether to encrypt or decrypt it. The software module then emits the data words composing the file one after the other as events on the CosiMate bus. The hardware part of the platform then automatically instantiates the required module, reads the inputs, and sends the outputs back to the bridge. The outputs are used by the software application to build a new file. This test application allowed

us to validate the hardware/software communication part as well as the dynamic behavior.

VI. Platform Uses

This platform can be used for different purpose. As explained in a previous section, it allows accelerating a simulation by implementing highly parallelizable models in hardware, as well as testing a logic prototype in the same environment its higher-level models were tested.

In both cases, debug features are of matter, as it is important to obtain information from inside a model. This can be easily done by adding Embedded Logic Analyzers (ELA) in the FPGA. But using SPoRE also enables one to add its specific observers inside the IP. Data will then be retrieved from additional ports and uploaded to the data server. Doing so will use Ethernet bandwidth, which may interfere with IP if communication timing is of matter. But this allows observability of the model without needing a specific debug connection such as JTAG. This is especially useful when using multiple FPGAs to deploy models, in which case it is difficult to have specific debug links to all devices.

Compared to other solutions depicted in Section II, this platform adds support for partial reconfiguration. This allows seeing an FPGA as a real SoPC, in which IPs are independent from each other. This means if the simulation needs some model at one point of the simulation, and some other model at a different time, we can use both models on the same device even if there is not enough logic to handle both models at the same time. This is done by reconfiguring the FPGA, replacing the unused model.

Moreover, by implementing a timeslicing scheduling approach, we could do so even when both models are needed at the same time. Timeslicing is a technique used in software to simulate application parallelism by attributing one resource (processor core) to different tasks depending on the time. Here, resources are reconfigurable logic, but this can be done the same way. However, this case needs specific cares. This will be discussed in Section VII.

But the most promising use for this platform is for modelling of dynamic systems. Indeed, by adding SPoRE into the co-simulation environment, we provide support for these systems. As for now, Partial Dynamic Reconfiguration (PDR) is only used for resources management. But we can imagine extending the PDR management to the SPoRE application itself. This can be done by adding explicit reconfiguration directives in the SPoRE application. Using this, we will be able to simulate the behavior of dynamic systems using native FPGA technology.

VII. Conclusion and Future Work

In this paper, we presented a co-simulation platform allowing to put together software and hardware models. The main point of this platform is that is handles partial reconfiguration of FPGAs to turn these in dynamic multi-IP designs. One FPGA can then handle various model implementations at the same time, and the models implemented can vary during the co-simulation. Moreover, various FPGAs can be used at the same time if more reconfigurable logic is needed. All reconfigurations are handled automatically without any need for the user to be aware of the partial reconfiguration mechanism.

The HIL co-simulation platform depicted in this article is only a first step in our research projects. One main extension we would like to do is to support synchronized mode up to the FPGA. This will be the object of further work, and we are exploring several leads on the subject. One idea would be using a clocking process on hardware IPs that is independent of the real time hardware clock, rather being provided by the co-simulation environment, as done by Liao *et. al* [8].

Moreover, we would like to automatize the process of designing SPoRE hardware kernels based on HDL IPs. This could be done using a parsing tool that scans the HDL top level entity interface, and generates both a co-simulation interface for software simulation, and a SPoRE wrapper for integration into HIL co-simulation. Some interface issues should also emerge from this, notably type conversion handling and floating/fixed point values representation. Using standard interface definitions, such as IP-XACT [16] can help automating the integration process.

Support of timeslicing simulation will also be investigated. Indeed, if the user is stuck with a number of FPGAs, which does not allow for implementing all models at the same time, timeslicing can solve this issue. It would consist in instantiating one model on the resources, treat the data related to it, then replace it by another model and do the same. In synchronized mode, this would allow to deal with multiple models on the same resources at each simulated time step. The downside of this approach is the reconfiguration time. Indeed, the reconfiguration time is an uncompressible overhead in an IP lifetime. This overhead gets negligible when the IP use time growth, but is of matter if the reconfiguration is frequent.

This overhead then needs to be taken in consideration if hardware is used for better performances. However, if the hardware is used for implementing prototypes, with no considerations of performances, this can be a useful approach.

Finally, this platform has a potential for the simulation of dynamic systems. If PDR is now used in background by SPoRE, we could make it explicit. This approach will concentrate our efforts, as we see here a promising use of the platform. Indeed, if the user is able to indicate how the reconfiguration should be handled, this opens new perspectives for the simulation of dynamic and auto-adaptive systems. To do so, we need to extend the SPoRE scheduler used for co-simulation to allow direct reconfiguration orders from the application itself.

References

[1] COMSOL Inc. (2015). Comsol multiphysics, [Online]. Available: https://www.comsol.com, [retrieved: March, 2015].

[2] F. Quartier and F. Manon, "Simulation for all components, phases and life-cycles of complex space systems.," in CSDM (Posters), 2013, pp. 167–174. [Online]. Available: http://ceur-ws.org/Vol-1085/15-paper.pdf, [retrieved: March, 2015].

[3] A. G. Kleppe, J. B. Warmer, and W. Bast, MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional, 2003, ISBN: 978-0321194428.

[4] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The promise of high-performance reconfigurable computing," IEEE Computer, vol. 41, no. 2, pp. 69–76, Feb. 2008, ISSN: 0018-9162. DOI: 10.1109/MC.2008.65.

[5] ChiasTek. (Feb. 2015). Cosimate, [Online]. Available: http://www.cosimate.com, [retrieved: March, 2015].

[6] C. Foucher, F. Muller, and A. Giulieri, "Online codesign on reconfigurable platform for parallel computing," Microprocessors and Microsystems, vol. 37, no. 4–5, pp. 482–493, 2013, ISSN: 0141-9331. DOI: 10.1016/j.micpro.2011.12.007.

[7] L. Guixuan, H. Danping, J. Portilla, and T. Riesgo, "A hardware in the loop design methodology for fpga system and its application to complex functions," in VLSI Design, Automation, and Test (VLSI-DAT), 2012 International Symposium on, Apr. 2012, pp. 1–4. DOI: 10.1109/VLSI-DAT.2012.6212666.

[8] Y. B. Liao, P. Li, A. W. Ruan, Y. W. Wang, and W. C. Li, "A hw/sw co-verification technique for field programmable gate array (fpga) test," in Testing and Diagnosis, 2009. ICTD 2009. IEEE Circuits and Systems International Conference on, Apr. 2009, pp. 1–4. DOI: 10.1109/CAS-ICTD.2009.4960748.

[9] C.-Y. Huang, Y.-F. Yin, C.-J. Hsu, C.-Y. Huang, and T.-M. Chang, "Soc hw/sw verification and validation," in Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific, Jan. 2011, pp. 297–300. DOI: 10.1109/ASPDAC.2011.5722202.

[10] Accellera Systems Initiative, Standard co-emulation modeling interface (sce-mi) reference manual, 2014. [Online]. Available: http://www.accellera.org/downloads/standards/sce-mi, [retrieved: March, 2015].

[11] C. Foucher, "Méthodologie de conception pour la virtualisation et le déploiement d'applications parallèles sur plateforme reconfigurable matériellement," PhD thesis, Sciences et Technologies de l'Information et de la Communication, Oct. 2012. [Online]. Available: https://tel.archives-ouvertes.fr/tel-00777511, [retrieved: March, 2015].

[12] D. Foures, V. Albert, J.-C. Pascal, and A. Nketsa, "Automation of sysml activity diagram simulation with model-driven engineering approach," in Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium, ser. TMS/DEVS '12, Orlando, Florida: Society for Computer Simulation International, 2012, 11:1–11:6, ISBN: 978-1-61839-786-7. [Online]. Available: https://dl.acm.org/citation.cfm?id=2346616.2346627, [retrieved: March, 2015].

[13] P. Coussy and A. Morawiec, High-level synthesis, From Algorithm to Digital Circuit. Springer Netherlands, 2008, ISBN: 978-1-4020-8587-1. DOI: 10.1007/978-1-4020-8588-8.

[14] Mentor Graphics. (Feb. 2015). Modelsim, [Online]. Available: http://www.mentor.com/products/fpga/model/, [retrieved: March, 2015].

[15] U. Fahrenberg and A. Legay, "Generalized quantitative analysis of metric transition systems," in Programming Languages and Systems, ser. Lecture Notes in Computer Science, C.-c. Shan, Ed., vol. 8301, Springer International Publishing, 2013, pp. 192–208, ISBN: 978-3-319-03541-3. DOI: 10.1007/978-3-319-03542-0_14.

[16] IEEE Computer Society, IEEE standard for IP-XACT, standard structure for packaging, integrating, and reusing IP within tool flows, version 1685-2014, Jun. 2014. [Online]. Available: https://standards.ieee.org/getieee/1685/download/1685-2014.pdf, [retrieved: March, 2015].