Verification of Embedded System Designs through Hardware-Software Co-Simulation

José Cláudio V. S. Júnior, Alisson V. Brito, and Tiago P. Nascimento

***Abstract*—This work proposes an environment for real- time testing of heterogeneous embedded systems through co-simulation. The verification occurs on real-time between the system software and hardware platform using the High Level Architecture (HLA) as a middleware between the hardware device and the simulated model. The novelty of this approach is not only providing support for simulations, but also allowing the synchronous integration with any physical hardware devices. In this paper we use the Ptolemy framework as a simulation platform. The integration of HLA with Ptolemy and the hardware models open a vast set of applications, like the test of many devices at the same time, running the same, or different applications or modules, the usage of Ptolemy for real-time control of embedded systems and the distributed execution of different embedded devices for performance improvement or collaborative execution. A case study is presented to prove the concept, showing the successful integration between the Ptolemy framework with an implementation using Atmel and ARM microcontrollers.**

***Index Terms*—Functional hardware verification, co-simulation, ptolemy, high level architecture, embedded systems.**

I. Introduction

The need for high performance, available and reliable embedded systems has made computing systems computing systems increasingly complex. Faced with such complexity, to design a project with only one language or even with a single level of abstraction has proved insufficient [1], since currently almost always computational systems are composed by modules developed in software and hardware, which may contain analog modules and mechanical parts [2].

These systems that combine components from different domains and need to cooperate are called heterogeneous systems. To verify this type of system is common to validate each developed module separately. Given this heterogeneity in embedded system design, verify its functionality becomes an increasingly difficult task. According to [3] and [4] the functional verification phase is one of the major problems within the hardware design. It is estimated that around 70% of the resources employed in a hardware design are used in the functional verification step [3], [5].

Co-simulating embedded hardware systems and software models can enable the verification of such hardware systems, when there is available a reference model in software, or validate a simulation model when there is available a reference model in hardware. Such situations can accelerate the design of embedded systems and motivated the development of this work. But some points need to be overcome, as the great difficulty is to integrate software simulators with also heterogeneous hardware implementations.

In this context, this work contributes by presenting a solution for synchronous co-simulation of hardware models with other models, aiming at functional verification of embedded systems. Furthermore this work allows simulation of heterogeneous models in a distributed manner.

For this, the High Level Architecture [6] was used as integration and synchronization platform, Ptolemy [7] to model and simulate the design in software and two hardware plat- forms, Arduino Mini [8] and Freescale Freedom KL25Z [9] for model execution in hardware. Furthermore, the Universal Verification Methodology (UVM) [10] has been adapted to the functional verification.

The results show that the hardware devices and the simulated models were successfully integrated synchronously within the proposed environment, enabling the co-simulation in a single environment for verification, opening opportunities for other applications, such as distributed execution of embedded systems to improve performance, the synchronous control of distributed hardware and the verification of various hardware devices at the same time.

This paper is organized as follows: in Section II theoretical concepts required for the development of this work are presented. In Section III some related works that perform co-simulation are presented. In Section IV the proposed architecture is presented. In Section V experimental results are exposed using the proposed architecture and, finally, Section VI is addressing the main discussions and concluding remarks obtained in this work.

II. Theoretical Foundation

A. Embedded Systems Verification

Verification is a process used to demonstrate whether the functionality of the project under development is in accordance with their specification or not [5]. The most popular types of verification are the formal verification, functional verification and hybrid verification.

According to [11], Formal Verification, also called Static Verification, makes comparisons by formal or mathematical methods of an implementation against a specification in order to detect if the implementation violates the specification. The Functional Verification, also known as Dynamic Verification, sets some steps in order to compare the model to be de- signed with your respective specification, being responsible for detecting logical errors in the system. This type of check is performed through simulation. The

Manuscript received April 15, 2014; revised June 20, 2014.

José Cláudio V. S. Júnior, Alisson V. Brito, and Tiago P. Nascimento are with Federal University of Paraiba – Center of Informatics, João Pessoa–PB–Brasil, Brazil (e-mail: cl4udio@gmail.com, alisson@ci.ufpb.br, tiagopn@ci.ufpb.br).

Hybrid Verification, also called Semi-formal, combines techniques of formal and functional verification.

Simulation is the most widely used means to perform the verification, since the formal verification techniques are still immature and complex [12]. There are two types of simulation, Functional Simulation, which is used to verify if the project performs the desired function without including timing and Temporal Simulation models, where delays are added, resulting more accuracy than the Functional Simulation.

B. Heterogeneous and Distributed Co-Simulation

The fundamental principle of co-simulation is the support for cooperative implementation of different simulators [13]. The co-simulation process requires simulators capable of simulating software and hardware parties jointly, guaranteeing consistency in the interaction between them. Each co-simulation component is responsible to execute any individual part of the system, which is described in some language and Model of Computation (MoC) at some specific level of abstraction [14].

A heterogeneous distributed co-simulation enables simulators from different domains to cooperate in a decentralized manner. The co-simulation environment must have appropriate structures that provide communication support, synchronization and data handling between the simulators [15]. Co-simulation uses some of these structures also applied in distributed simulation, mainly related to synchronization features.

In the area of distributed simulation, there are two possibilities: the Parallel and Distributed Simulation (PADS) and Distributed Interactive Simulation (DIS) [16]. A PADS is focused on the implementation of distributed discrete simulation based on events [17]. DIS aims to meet the real-time simulations.

III. Related Works

The co-simulation enables parallel execution of simulators to perform the validation of a system. Such simulators can be characterized by a computer system, hardware resources or a real part of a system. These simulated environments can be homogeneous or heterogeneous. Given these two possibilities in which the simulator is characterized, there are some works in the literature that use all of these characteristics in order to perform functional verification of embedded systems.

The authors in [18] presented a methodology for functional verification of hardware through parallel co-simulation. The work uses "thread" and "handshaking"' provided by the Verilog PLI (Programming Language Interface) functions to ensure that the co-simulation will be correctly made. Parallel threads are inserted, which are autonomous in relation to the execution flow of the simulator of Hardware Description Language (HDL) threads, enabling the data-exchanging amount the simulations simultaneously. The synchronization among hardware and software is made using handshakes through the Application Program Interface (API) provided by the Programming Language Interface (PLI) to Verilog, combined with specific synchronizations functions.

The authors in [19] presented an environment for verification and validation of hardware using the technique of Hardware-in-the-loop (HIL). It focuses in an integration of Matlab Simulink with HIL. A proposal for time synchronization between the simulation models is presented, from that it is possible to establish a real-time communication among the involved models. For this, the compute running Simulink performs the control logic to synchronize the time. In this paper it is used a PIC microcontroller as physical device. A Programmable Interval Timer (PIT) controls the time.

The authors in [13] presented an environment that aims at integrating virtual components on models of distributed co-simulation. The co-simulation is based on a modified version of HLA, named Distributed Cos-simulation Backbone (DCB). It imposes proprietary standards for data exchanging, and requires explicit calls to functions of Run-time infrastructure (RTI). In general the operation is the same as HLA, which is a Federated co-simulation comprising autonomous and distributed federations, where Federations may be written in different languages.

There are also works that use the technique of Hardware-in-the-loop with the purpose of verifying embedded systems, but they are restricted to a limited set of models and devices. It is not possible to simulate and verify any device or application being restricted to homogeneous simulation models. Works that perform only co-simulations of homogeneous models of- ten are unable to implement the heterogeneity due to the great effort that is employed for adaptation to different hardware platforms and communication and synchronization protocols. Such characteristics like heterogeneity and distribution of a verification and simulation platform are provided in this work through the integration of Ptolemy and HLA environments.

IV. Proposed Architecture

The step of functional verification is considered the most important phase in the development of embedded system design. At this stage it is possible to detect logical errors in the system, it must guaranty that errors are detected in the early stages of the design, not passing on to future steps. Traditional methodologies for functional verification basically consist of injecting a set of input stimuli on hardware, so that the expected results are known in advance, to be compared to the result of the reference [20].

However when it comes to heterogeneous systems containing both software as hardware models, this process of obtaining the inputs and outputs is not straightforward, due to its dependence on technology. Therefore we use concepts of Universal Verification Methodology (UVM) [10], particularly the testbench model, which was modified and implemented in Ptolemy and integrated with HLA. The Ptolemy enables modeling and simulating embedded systems using different Models of Computation, while the HLA allows the integration of heterogeneous technologies in real-time, synchronously.

In the proposed environment, the DUT (Design under Test) is an actor who performs direct communication with the hardware. It receives stimuli and generates outputs that are compared with the expected results received from the reference model.

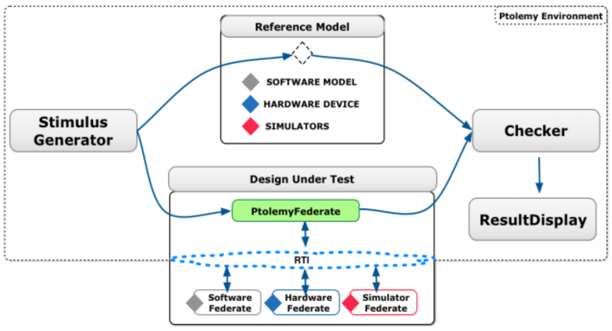


Fig. 1. Architecture of proposed co-simulation environment.

As can be seen in Fig. 1, some elements of the UVM test bench were implemented in Ptolemy, where each actor represents a step of the flow of the proposed test bench, they are:

- **Stimulus Generator:** It is the actor responsible for sending predefined data stored in a file that will serve as input to the system models;
- **Reference Model:** Application model that can be software, written graphically in Ptolemy model or physical hardware, and that complies with the specification;
- **Checker:** Responsible for collecting and analyzing data sent by Reference Model and the DUT. Upon receipt of the data, they are compared, taking into account the time and the value;
- **Result Display:** Responsible for displaying the results generated by the checker. It can be observed whether the model was validated or not, showing the instants when the results differ;
- **Design under Test:** The DUT is usually a description of the device being developed, coded in a hardware description language HDL (Hardware Description Language) or simulated in software. However, what we propose is that the DUT is the prototyped hardware de- vice. Here is where Ptolemy simulator communicates with other simulators or hardware devices through the HLA. This integration is detailed in IV-B section.

A. Heterogeneous Simulation Environment

The Ptolemy project, created and maintained by the University of Berkley, presents a support for modeling, design and simulation of any systems based on actors, especially embedded systems that mix technologies such as analog and digital electronics, and electromechanical hardware devices plus complexity in systems with multiple levels of abstraction [21]. It is a software framework written in Java with a graphical interface called Vergil. Its main difference from the other tools is the support to heterogeneous models interacting in the same environment [7].

As can be seen in Figure 1, the entire verification environment is inserted into the Ptolemy environment, thereby enabling those different systems can be checked in a single co-simulation environment. The Ptolemy offers a heterogeneous simulation environment where multiple systems can be modeled at different levels of abstraction, which can be used as reference models. Heterogeneity among simulators (external to Ptolemy) is guaranteed through integration between Ptolemy and HLA, which provides mechanisms for communication and synchronization.

In the Ptolemy environment (see Figure 4) there is SlaveHLADirector, which was developed in this work to manage the simulation of Ptolemy respecting policies based on discrete events, coordinating the execution order of the actors in the model, verifying the data when sending from the actors to RTI and also from RTI to the actors.

The PtolemyFederateActor is the actor that performs the communication with RTI from Ptolemy model. All data sent to it will be passed through the RTI and send to the appropriate Federate.

B. Communication Mechanism

For the effective communication between the simulators in the environment, it is necessary that the co-simulation environment knows the semantics of all the involved models and how actions in one can affect the state of the others [22]. To enable this communication, it is necessary to use suitable interfaces for all devices (Federates) involved.

There are some technologies that provide the necessary resources to enable communication between components in a distributed simulation. Among the most used are Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA) and High Level Architecture (HLA). From these three architectures HLA is the only one specific for distributed simulation. CORBA and RMI are generic architectures for distributed systems.

The HLA architecture [6] was defined under the leadership of Defence Modeling Simulation Office (DMSO) to support reuse and interoperability through a vast number of different types of simulators maintained by the U.S. Department of Defense. This architecture is defined by three IEEE standards: the first deals with the framework in general and its main rules, the second contain the specification of the interface between the simulator, and the third is the specification of the data model exchanged between simulators.

The main idea of HLA is to separate the specific features of each simulator through a general purpose infrastructure. Each simulator must use a Runtime Infrastructure (RTI) to communicate the other simulators. RTI is an infrastructure that provides the interface between the simulators and the overall structure of HLA. It aims at the control of distributed transactions (exchange of messages) and is responsible for managing the total simulation time and provides a synchronous communication protocol. Each simulator connected to a RTI is represented by a Federate. The Federates can be seen as an entity or component model and could be represented by a simulation model on physical hardware or a specific purpose simulator.

For a better understanding, Figure 2 is presented, which details the entire design flow. Each co-simulation project uses a RTI since it is responsible for providing a single communication protocol. The RTIAmbassador is responsible for managing all entry and exit of the federation, data exchange and synchronization. In other words, it is responsible for the specific structure of each simulation interface structure of HLA. The FederateAmbassador enables communication between Federates, through the exchange of messages with the RTI.

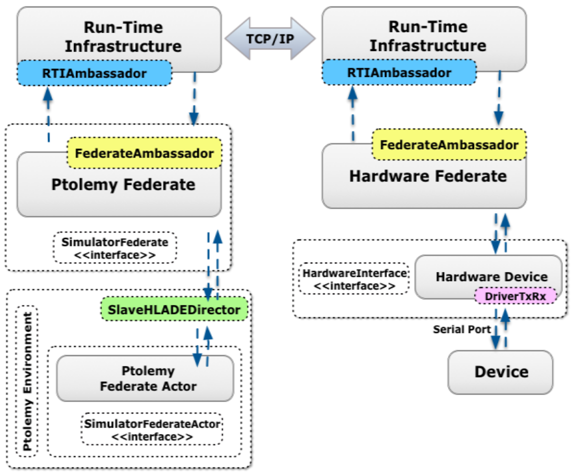


Fig. 2. Architecture of ptolemy and hardware federations.

The integration between the software model (Ptolemy) with the HLA can be seen in detail in [23]. As shown in Fig. 2, the federates are connected via a TCP/IP connection. The PtolemyFederate is responsible for simulation and verification of heterogeneous models. It contains all the logic for the verification. Theoretically it can be considered a master project, because it is responsible for starting the simulation and is waiting for the hardware modules (slaves) to start the verification. The HardwareFederate is responsible for the integration between the physical hardware with the software model.

C. Synchronization Model

In the co-simulation environment, the various simulators that make up the system can have different execution speed and time model. Thus, there is the need to define a synchronization model. With synchronization, there will be a coordination of events sent and received between the simulators, so there is greater assurance that the triggered events may occur in the correct order during the simulation.

The HLA provides information necessary to use PADS and DIS simulation models. This work makes use of the concepts of PADS. The DIS simulation concerns real-time simulation that is not our focus at this time, and it would turn implementation more complex. In PADS two approaches are used in relation to synchronization. In the synchronous approach, also called conservative, the protocol must ensure the correct order of occurrence of events during a simulation. In the optimistic approach, unlike the conservative, it allows events to be run out of a temporal order.

As the verification environment was initially developed to perform a functional simulation, the approach used was the conservative one. In HLA, through Time Management that is managed by RTI, an approach called Stepped Time Simulation is used for time forwarding, which ensure that time does not proceed to the next step until all activities at that current global time are completed.

Fig. 3 shows PtolemyFederate initially sending a given data (event) for the HardwareFederate, and keeps waiting for a response, and just then it can process information. The HardwareFederate initially expects to receive a data to process and send a response to PtolemyFederate, then it gets waiting for a new event.

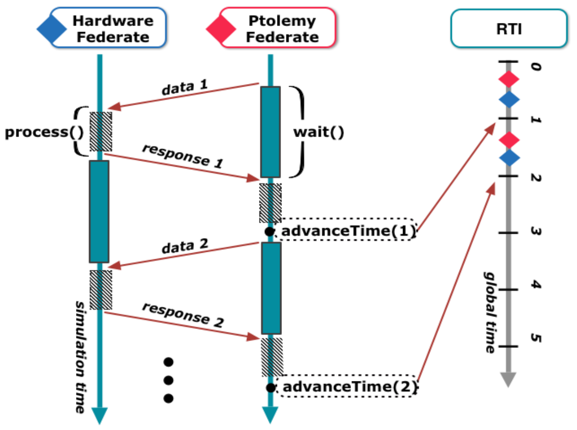


Fig. 3. Implemented synchronization model.

After each event cycle (send, wait and process) the PtolemyFederate execute the advanceTime() operation, which notifies the RTI that the global simulation time must be advanced, thereby ensuring that all events are ordered according to the timestamp of the transmitted data in relation to the global time.

D. Communication Hardware Interface

In order to allow the communication of any hardware device with software model, a communication interface and a data protocol were developed. The Hardware Device is a federate that should be implemented according to each device. Its implementation should contain an instance of the class DriverTx/Rx which is responsible for acquiring data from the hardware device interface. It automatically detects the port and the ID of the hardware that is connected to the USB port. However, there is the need to set the information for each device in advance. Since most devices use serial communication via USB, this interface will be fully compatible with any device, needing only the protocol implementation.

The main idea of the protocol is that the message is among two bytes previously defined as starting and ending of message. Thus, it enables any message size. The protocol was developed in a simplified manner to facilitate the functioning of the experiments of the case study presented here, but the interface allows more complex protocols to be developed in the future. The tested devices, Arduino Mini and Freescale Freedom KL25Z worked perfectly, and were not necessary any change in the interface, required only that the protocol was running in both architectures.

V. Case Study

Initially as a case study an implementation in software was used for temperature conversion. The application received a value related to a temperature on the Fahrenheit scale and returning the converted value on the Celsius scale. This application was chosen in order to validate the developed environment. With advances in the development of the environment, further tests were conducted using AES128 encryption algorithm. It was defined a 16 byte standard key and messages were pre-defined and randomly sent to be encrypted.

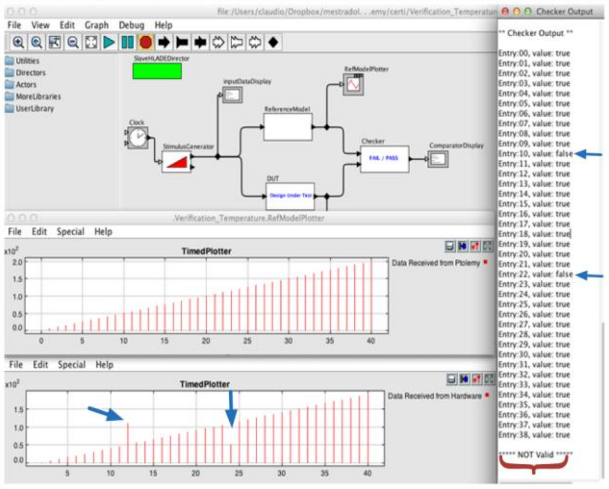
Arduino Mini and the Freedom KL25Z platforms were used as hardware model and software applications were written in Java language for conducting both experiments. According to Table I, the models of hardware and software were reference model and to be the Design under Verification (DUV). All implementations were developed by different programmers, using different languages. To realize the implementation of applications on the Arduino, Wiring language was used, and for Freedom the C language was used.

TABLE II: HARDWARE AND SOFTWARE VERIFICATION EXPERIMENTS

Reference Model	**Design Under Test**		
	Arduino	Freedom	Java
Arduino	-	X	X
Freedom	X	-	X
Java	X	X	X

Is shown in Fig. 4 the results obtained from developed co-simulation. It is shown the result of the functional verification from a co-simulation written in the Arduino hardware platform, which is expected to conform to the specification from a software model, which complies with the specification model. So in this case, it is expected that the environment returns a valid output as a result if the equivalence between software written in (reference model) model and hardware model is true.

However, it is possible to see that both are not identical. The "Checker Output" displays the entire result of the simulation, taking into account the content and delay. Both models need to be synchronized; otherwise the data diverge, generating a not validated message.



```
** Checker Output **

Entry:00, value: true
Entry:01, value: true
Entry:02, value: true
Entry:03, value: true
Entry:04, value: true
Entry:05, value: true
Entry:06, value: true
Entry:07, value: true
Entry:08, value: true
Entry:09, value: true
Entry:10, value: false
Entry:11, value: true
Entry:12, value: true
Entry:13, value: true
Entry:14, value: true
Entry:15, value: true
Entry:16, value: true
Entry:17, value: true
Entry:18, value: true
Entry:19, value: true
Entry:20, value: true
Entry:21, value: true
Entry:22, value: false
Entry:23, value: true
Entry:24, value: true
Entry:25, value: true
Entry:26, value: true
Entry:27, value: true
Entry:28, value: true
Entry:29, value: true
Entry:30, value: true
Entry:31, value: true
Entry:32, value: true
Entry:33, value: true
Entry:34, value: true
Entry:35, value: true
Entry:36, value: true
Entry:37, value: true
Entry:38, value: true

***** NOT Valid *****
```

Fig. 4. Co-simulation output with an invalid model.

The scenario in Fig. 4 was created just to test if the tool could detect when the models are not equivalent. In the window "Data received from hardware " representing the data received by the Arduino, the two arrows indicate in the chart the results that were intentionally sent in the wrong way, unlike the window "Data received from software" showing the data received correctly.

The environment proved quite effective in all tests that were performed, either with a simple model of application such as temperature, either with a more complex model such as encryption. Even when a large number of data have been entered, the environment behaved synchronously and verified correctly.

VI. FINAL CONSIDERATIONS

This paper presents an environment for verification and test of embedded systems through co-simulation using physical hardware. This approach allowed a physical hardware test, which unlike simulated, is being tested under external factors that directly influence the hardware behavior.

As the communication of most hardware devices is performed via serial port, in practice, our proposal will allow the integration of any hardware device using the same method. In addition, HLA provides consistent and synchronous communication of any simulator via a TCP/IP connection. It allowed initially the integration of Ptolemy, but can also allow in the future the integration with other simulations tools such as Matlab, VHDL, SystemC, etc.

To the best of our knowledge, this is the first work that uses the integration between HLA and Ptolemy for hardware co-simulation. This integration was first presented in [23] and a performance evaluation was presented in [24]. Now, for the first time the integration with hardware devices is presented for testing embedded systems using this innovative approach.

Integrating HLA with Ptolemy and hardware models opens a wide range of applications, such as verify multiple devices at the same time, running the same or different applications or modules, distributed execution of various embedded devices for improving performance or collaborative execution.

As future work, improvements need to be made in the communication interface between the physical hardware and the software model. Such as an automated way to map the ports of the hardware device taking into consideration the amount and type of data to be exchanged, and automatically create the data model to be used by HLA.

REFERENCES

[1] B. A. De Mello and F. R. Wagner, "A standardized co-simulation backbone," *SoC Design Methodologies*, Springer, 2002, pp. 181–192.

[2] P. Le Marrec, C. Valderrama, F. Hessel, A. Jerraya, M. Attia, and O. Cayrol, "Hardware, software and mechanical cosimulation for automotive applications," *Rapid System Prototyping*, 1998, pp. 202–206.

[3] J. Bergeron, *Writing Testbenches: Functional Verification of HDL Models*, Second Edition, Norwell, MA, USA: Kluwer Academic Publishers, 2003, vol. 2.

[4] A. Piziali, *Functional Verification Coverage Measurement and Analysis*, First Edition, Massachusetts, USA: Kluwer Academic Publishers, 2004.

[5] J. Bergeron, *Writing testbenches using system Verilog*, Springer Heidelberg, 2006.

[6] IEEE, "Ieee standard for modeling and simulation (m&s) high level architecture (hla) – framework and rules," IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000), 2000.

[7] E. A. Lee and I. John, "Overview of the ptolemy project: Electronics research laboratory," College of Engineering, University of California, 1999.

[8] M. Banzi, *Getting Started with Arduino*, O'Reilly Media, Inc., 2009.

[9] Freescale. (2013). Frdm-kl25z: Freescale freedom development platform. [Online]. Available:

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=FRDM- KL25Z

[10] U. V. M. Accellera, *Universal verification methodology i. 0 user's guide*, 2011.

[11] J. Bergeron, E. Cerny, A. Hunter, and A. Nightingale, *Verification Methodology Manual for System Verilog*, Secaucus, NJ, USA: Springer - Verlag New York, Inc., 2005.

[12] S. Edwards, L. Lavagno, E. Lee, and A. Sangiovanni-Vincentelli, "De-sign of embedded systems: formal models, validation, and synthesis," in *Proc. the IEEE*, vol. 85, no. 3, pp. 366–390, Mar. 1997.

[13] U. R. F. Souza, J. K. Sperb, B. A. de Mello, and F. R. Wagner, "Tangram-virtual integration of heterogeneous ip components in a dis- tributed co-simulation environment," in *Proc. Integrated Circuits and Systems Design*, 2003, pp. 125–130.

[14] F. Hessel, "Concepção de sistemas hererogêneos multi-linguagens," *Jornada de Atualização em Informática – JAI. XXI Congresso da Sociedade Brasileira de Computação. SBC*, 2001.

[15] W. Bishop and W. Loucks, "A heterogeneous environment for hard-ware/software cosimulation," in *Proc. Simulation Symposium*, Apr. 1997, pp. 14–22.

[16] *R. M. Fujimoto, "Parallel and distributed simulation," in Proc. the 31st conference on Winter simulation: Simulation—a bridge to the future*, vol. 1. ACM, 1999, pp. 122–131.

[17] Parallel simulation: parallel and distributed simulation systems, in *Proc. the 33nd conference on Winter simulation, IEEE Computer Society*, 2001, pp. 147–157.

[18] D. Deprá, B. Zatt, M. Santos, and S. Bampi, "Metodologia para verificação funcional de hardware através de co-simulação paralela dentro de sistemas de software complexos usando pli: Decodificador h.264/avc como estudo de caso," *HÍFEN*, vol. 31, no. 59, 2008.

[19] S. M. Shah and M. Irfan, "Embedded hardware/software verification and validation using hardware-in-the-loop simulation," in *Proc. Emerging Technologies*, 2005, pp. 494–498.

[20] B. Zatt, A. Azevedo, L. Agostini, and S. Bampi, "Validação de uma arquitetura para compensação de movimento segundo o padrão h. 264/avc," presented at XII Iberchip Workshop, Costa Rica, 2006.

[21] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Sachs, Y. Xiong, and S. Neuendorffer, "Taming heterogeneity - the ptolemy approach," in *Proc. the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[22] J. Adams and D. Thomas, "The design of mixed hardware/software systems," in *Proc. Design Automation Conference*, Jun 1996, pp. 515–520.

[23] A. V. Brito, A. L. V. Negreiros, C. Roth, and O. Sander, "Development and evaluation of distributed simulation of embedded systems using ptolemy and hla," presented at 17th IEEE / ACM International Symposium on Distributed Simulation and Real Time Applications, 2013.

[24] A. L. V. d. Negreiros and A. V. Brito, "The development of a method- ology with a tool support to the distributed simulation of heterogeneous and complexes embedded systems," in *Proc. 2012 Brazilian Symposium on Computing System Engineering (SBESC), IEEE*, 2012, pp. 37–42.

José Cláudio V. S. Júnior received the B.S degree in computer science in 2008, currently is a master's student in informatics in the Federal University of Paraiba (UFPB). His research area focuses in the use co-simulation for real-time verification of mobile robots. He is a member of the brazilian computer society (SBC). His main research interests are in embedded systems and robotics.

Alisson V. Brito has Ph.D. in electrical engineering from Federal University of Campina Grande (UFCG) in the area of microelectronics in cooperation with the Karlsruhe Institute of Technology (KIT) (2008), MA (2003) and a BS in Computer Science (2001) also by UFCG. He is a professor at the Center of Informatics and is coordinator of the Graduate Program in Informatics (PPGI) at UFPB.

He has experience in computer science, acting on the following topics: simulation, hardware design, computers in education, and social networking.

Tiago P. Nascimento received the B.S. in mechatronics engineering at the College of Technology and Science - FTC in 2007, the M.S. degrees in electrical engineering from Federal University of Bahia in 2009, and the Ph.D. degree from Oporto University in Electrical and Computer Engineering, in 2012. In 2013, he joined the Department Computer Systems, Federal University of Paraiba, as an assistant professor. His main research interests are in mobile robotics, multi-robots systems,process control, vehicle dynamics and intelligent control.