CODIS+: Co-simulation environment for heterogeneous systems

Mossaad Ben Ayed*, Faouzi Bouchhima, Mohamed Abid*****

**Computer Science Department, College of science and humanities at AlGhat, Majmaah University, KSA*
Computer and Embedded System laboratory, University of Sfax, Tunisia
(e-mail:mm.ayed@mu.edu.sa , mossaad_benayed@yahoo.fr)
*** ENET'Com college, University of Sfax, Tunisia*
(e-mail: f_bouchhima@yahoo.fr)
**** National School of Engineers of Sfax, University of Sfax, Tunisia*
(e-mail: mohamed.abid@enis.rnu.tn)

Abstract: Heterogeneous systems ensure the integration of diverse specific components to different applications in various domains such as the electrical, mechanical and optical fields. They can be defined as the combination of discrete and continuous models at the same application. This kind of system presents an important challenge for designers. The main problem is the difference between algorithms of continuous simulators and algorithms of discrete ones. One of the key challenges is to overcome problems of the existing environments supporting heterogeneous models. So far, research in Computer-Aided Design (CAD) tools has attempted to offer a global view of the designed systems and to enable their overall verification. Indeed, the Continuous DIscrete Simulation (CODIS) tool based on co-simulation environment proved to be a powerful tool for global verification in heterogeneous systems (Bouchhima et al., (2005,2007)). This paper will attempt to contribute an extension of the CODIS tool in two ways. Firstly, it will attempt to integrate SystemC, Simulink simulators and a hardware accelerator automatically in order to generate global simulation model instances. Secondly, the evaluation of the simulation model will be performed by using an illustrative application.

Keywords: Co-simulation, Synchronization model, CODIS, Heterogeneous system, Continuous/Discrete model.

1. INTRODUCTION

Modern systems are increasingly complex and heterogeneous like mixed-signal systems and real-time controllers. Nowadays, systems on chip are used for convergence of multiple technologies. In 2001, Ferroelectric Random Access Memory (FRAM) and Field Programmable Gate Array (FPGA) were integrated on chip followed by Micro Electro Mechanical Systems (MEMS) and chemical sensors in 2003; electro-optical in 2005; and electro-biological in 2006 (ITRS, 2006).

These systems will be ubiquitous in communications, automotive, medical and other domains. The global verification of these systems requires new environments supporting new techniques enabling reusability, high abstraction levels and simulation accuracy. Currently, heterogeneous systems are designed by reusing pre-designed components to respect the tight constraints of time-to-market. This type of design represents an important challenge; one of the key issues being the integration of the pre-built components specific to different application domains such as the electrical, the mechanical, and the optical domains (Balarin et al., 2003; Nicolescu et al., 2002; Urrea et al., 2015; Minh et al., 2012; Li et al., 2015). Nowadays, designers build different components to be integrated in one chip by using the existing powerful tools specifically designed for a particular application domain such as SystemC for the electronic digital part and Matlab/Simulink for the mechanical parts. Nevertheless, they often prefer to keep using their current tools. Consequently, the best solution of new CAD tools is to be based on global simulation models defined independently from specific languages or simulators. This solution would permit the integration of the best and most adequate existing tools to benefit from their full capabilities. For this reason, the purpose of this study was to use the co-simulation technique. This technique is expected to allow the description of both the continuous and the discrete models in a specific and appropriate language despite its decrease of the simulation speed.

Nevertheless, the co-simulation model must be based on generic models (independent from languages or simulators). These models present essentially two difficulties in the definition of a continuous/discrete co-simulation model:

(1) The heterogeneity of the definitions of the concepts manipulated by the discrete and the continuous components.

(2) The need for continuous/discrete communication and synchronization.

Therefore, co-simulation interfaces are used for overcoming these difficulties. These interfaces have a great influence on the accuracy and the performance of the global simulation. Their automatic generation is very important, since their design is time consuming and can be an important source of errors.

As a consequence, this paper, proposes a generic architecture of a discrete/continuous simulation model for an accurate global verification in heterogeneous system design. Solutions are provided to implement this model in the case of SystemC, Simulink simulators and hardware accelerator. In addition, the continuous/discrete simulation (CODIS+) is presented as an extension of the CODIS tool, for the automatic generation of the global simulation model (co-simulation model) instances. Finally, the proposed model is evaluated using an illustrative application.

The rest of this paper will be organized as follows:

Section 2 describes the related work. Section 3 presents the different basic modelling concepts for discrete and continuous models. Section 4 introduces the CODIS+ tool by defining the synchronization models and their interfaces. Section 5 exhibits the experimental results and accuracy analysis. Finally, section 6 concludes the paper.

2. RELATED WORK

In the past years, although research in the discrete/continuous simulation area proliferated, it remained behind its digital counterpart. An overview of related work and existing tools is given in this section. There are mainly two approaches to overcome the difficulties of discrete/continuous systems in modelling and verification; i.e. common approach and co-simulation approach.

2.1. Common approach

The common approach used a unique language for the specification of the overall system. Some of these languages could be obtained by extension of well-established languages. Illustrative examples were VHDL-AMS (IEEE Std, 2007), Verilog-AMS (Pecheux et al., 2005) and recently, SystemC-AMS (Vachoux et al., 2003) extending, respectively VHDL, Verilog and SystemC for mixed-signal systems design. The first trials, contributing to the deeper understanding of the problems in using SystemC and VHDL for mixed-signal systems design were presented by (Bonnerud et al., 2001; Pichon et al., 1995). In (Enwich et al., 2001), a SystemC based framework supporting signal processing-dominated applications was proposed. The synchronization between the synchronous dataflow and linear continuous time is using fixed time step. These solutions allowed the continuous design and an accelerated time simulation. These methods suffered from the use of one solver to resolve EDOs and limitation of the design for different abstraction layers. Another framework was proposed in (Bonnerud et al., 2001) where the authors presented a mixed-signal simulation to simulate an analog to digital data converter. The framework included C++mixed-signal modules. They implemented a virtual clock for the scheduling of the analog blocks to avoid multiple executions of them due to the SystemC scheduler.

Other research based on common approach proposed a novel language that supports discrete/continuous description as Ptolemy (Ptolemy, 2015), MLdesigner (Schorcht et al., 2003), and Modelica (Modelica, 2012).

Ptolemy (Ptolemy, 2015), is an open source java-based environment that considers heterogeneous systems represented as a set of components whose interaction styles are governed by computation models implemented as ‘‘domains’’. It provides a unified infrastructure to assure hierarchical composition of these computation models. The necessity to learn a novel language presents its major drawback.

MLdesigner (Schorcht et al., 2003) has a similar interface as Simulink but its environment complexity presents a challenge.

Modelica (Modelica, 2012) can also be considered as one of these languages. Several commercial simulation tools such as Dymola and Math Modelica are based on it. This language provides a set of libraries for several application domains (electrical, thermal, etc.). However, the concept of discrete events is difficult to manipulate in this language.

In (Pinki et al., 2003; Riihimaki et al., 2005; Kajtazovic et al., 2005; Azam, 2005), the authors presented a tool based on UML approach. This method allows the creation of a model from a GUI. The use of UML language makes the design description difficult due to the requirement of a clock mechanism and events scheduler.

Other work based on approximate metrics (Girard et al., 2007) accelerates simulation and resolves the problem of Ordinary Differential Equations (ODEs) complexity, although the accurate simulation is decreased.

In (Prabhakar et al., 2015; Ghomri et al., 2013), the authors used Hybrid Automata to explore the real-time property of heterogeneous systems. But this method is more suitable for continuous models than for discrete models.

In summary, several attempts were proposed for mixed-signal simulation on common approach. They required abandoning well established efficient tools for specific domains (IEEE Std, 2007; Vachoux et al., 2003; Ptolemy, 2015; Oltean et al., 2010). Common approach allowed results in shortest simulation time but suffered mainly from three disadvantages:

- No use of libraries and dedicated IPs developed by each domain (discrete/continuous).
- The need to learn a new language.
- The restriction of the design use to few abstraction layers.

To overcome these shortcomings other researchers, proposed the use of co-simulation approach.

2.2. Co-simulation approach

The most important benefit of the co-simulation approach was the synchronization between different simulators. In (ElTahaway et al., 1993), the authors presented a simulation model based on VHDL and ELDO. Based on the lock-step approach with a fixed time step, their model was at the physical level. Other works used the notion of co-simulation backplane functions to integrate mixed-signal simulators. In (Zwolinski et al., 1995), the authors developed a collection of backplane that allowed simulators to share data during simulation. They integrated an open source SPICE simulator

and a logic simulator. In a similar way, the authors in (Hickey et al., 2006; Martin et al., 2002) proposed aco-simulation environment based on Xyce (a SPICE parallel simulator) and SAVANT (a parallel VHDL simulator). The interfacing is enabled by C++ classes containing methods for signal conversion and data exchange between simulators.

The Nexus-PDK environment proposed by Celoxica (Celoxica) supported co-simulation of cycle accurate C, C++ and Handel-C models with SystemC, MATLAB/Simulink, VHDL and Verilog simulators. All the models integrated in this environment were discrete. A similar approach was adopted by Active-HDL (Active-HDL, 2015).

These different attempts to create a co-simulation environment suffered from:

- The increase of simulation time.
- The limitation on the number of abstraction layers.

In order to overcome the problems met in co-simulation approach, Bouchhima et al., (Bouchhima et. al., 2005, 2007) proposed the CODIS tool. The work of these scholars presented the following advantages:

- They proposed a generic model for an accurate continuous/discrete simulation that was independent from languages and simulator;
- They provided implementation solutions in the case of SystemC and Simulink simulators;
- They introduced the CODIS tool which can automatically produce instances of the global simulation model. Hence, CODIS environment presented the most powerful tool in Co-simulation approach due to the generic model proposed. Unfortunately, CODIS still suffers from the increased simulation time.

Moreover, as was rightly argued by (ElTahaway et al., 1993; Hickey et al., 2006), Celoxica, (Enwich, et al., 2001; Dehghanimmohammadabadi et al., 2017), most of the proposed heterogeneous approaches were application-specific extensions or specific for a pair of simulators.

In light of this brief review, this paper purports to contribute the following tentative proposals:

(1) a synchronization model between a discrete simulator and an emulator for HW/SW design;

(2) a synchronization model between a continuous simulator and an emulator using Hardware Software In the Loop (HSIL) techniques;

(3) a synchronization model between a discrete simulator/continuous simulator and an emulator for continuous/discrete systems;

(4) the implementation of the proposed solutions for SystemC (Al-Junaid et al., 2004) as discrete simulator, Simulink (Matlab/Simulnik) as continuous simulator and emulator based on target architecture;

(5) the introduction of the CODIS+ tool which can verify heterogeneous systems in the first step of design; and finally,

(6) the presentation of an accuracy and performance analysis using an illustrative application.

3. BASICS MODELING CONCEPT

System description is based on the definition of the required model. The understanding of different simulation models is the aim of the definition. As will be mentioned below, the simulation model can be assigned as a discrete model, as a continuous model or as a heterogeneous model. Each model has a different definition that will be presented in the next Sub-sections.

3.1. Discrete event simulation model

The simulation of discrete models is based on events. Each executed event can generate new events with the smallest time stamp. An event is associated with a process.

The simulation depends on description and supported layers. Many simulators are proposed to increase performance. The simulation of synchronous systems is based on signals that can have events only at clock ticks (Chang et al., 1997). The simulation for data flow models depends on the ordering of events. A scheduler of discrete events is used for increasing performance (Patel et al., 2004).

3.2. Continuous simulation model

Systems described by differential and algebraic equations are considered as continuous models. The simulation step demands the numerical solution of these equations even some ODEs are complexes. As a solution, many algorithms, (Gupta et al., 1985), are proposed to discretize the continuous time into different discrete time instants. Then the numerical value of state is computed at these ordered time instants. The integration step presents the interval between two consecutive time instants. This integration step can be fixed or variable according to the used algorithm in order to ensure the stability, the accuracy and the continuity of the signals.

3.3. Continuous/discrete simulation: Discussion

Most dynamic systems are modelled using continuous and discrete models. The difference between the two models is presented essentially in the notion of time, the means of communication and the process activation rules.

To overcome the heterogeneity of modelling, two models were adopted in literature: common and co-simulation approaches, which are presented in related work section.

As will be shown below, the co-simulation approach has many advantages and is more suitable for research because it builds the results of past researchers.

Bouchhima (Bouchhima et al., 2007) findings on CODIS were the result of cooperative work between GRM and CES laboratories. CODIS tool was based on co-simulation approach. It did not only ensure synchronization between continuous and discrete simulators but also allowed the creation of generic interfaces. CODIS model contained essentially two layers: synchronization layer and communication layer. The proposed synchronization scheme for the CODIS tool was applied in both cases of

Matlab/Simulink and SystemC/Modelsim.

The next section describes the CODIS concept for heterogeneous simulation.

4. ACCELERATED CONTINUOUS DISCRETE SIMULATION (CODIS+) MODEL

The CODIS+ proposes an accelerated co-simulation method using hardware acceleration. This extension is not only to accelerate the simulation but also allows hardware description using via on board emulation. This idea will be described in the next sections.

CODIS+ is based on co-simulation/emulation bus that comprising two layers: (1) Communication layer and (2) Synchronization layer. Figure 1 describes the general architecture of CODIS+. The communication layer handles: (1) data exchange and (2) signals conversion. The first function allows writing or reading signal values that connect the continuous model and the discrete model. Data exchange is possible in both directions and depends on the type of the context switch. When the switch is between discrete simulator and continuous simulators, the data exchange is ensured by shared memory. When the switch is to hardware accelerator, the data exchange is ensured by interrupt method (Ben Ayed et al., 2013; Ben Ayed et al., 2012). Signals conversion takes care of the conversion between analogue signals and discrete signals.

CODIS+ can be presented as three kinds of simulation as shown in figure 2: (1) Hardware Software Co-Simulation model (2) Hardware Software in The Loop (3) Continuous/Hardware/Software Co-Simulation/Emulation.

The next sub-sections present the synchronization schemes for different kinds of CODIS+ simulation.

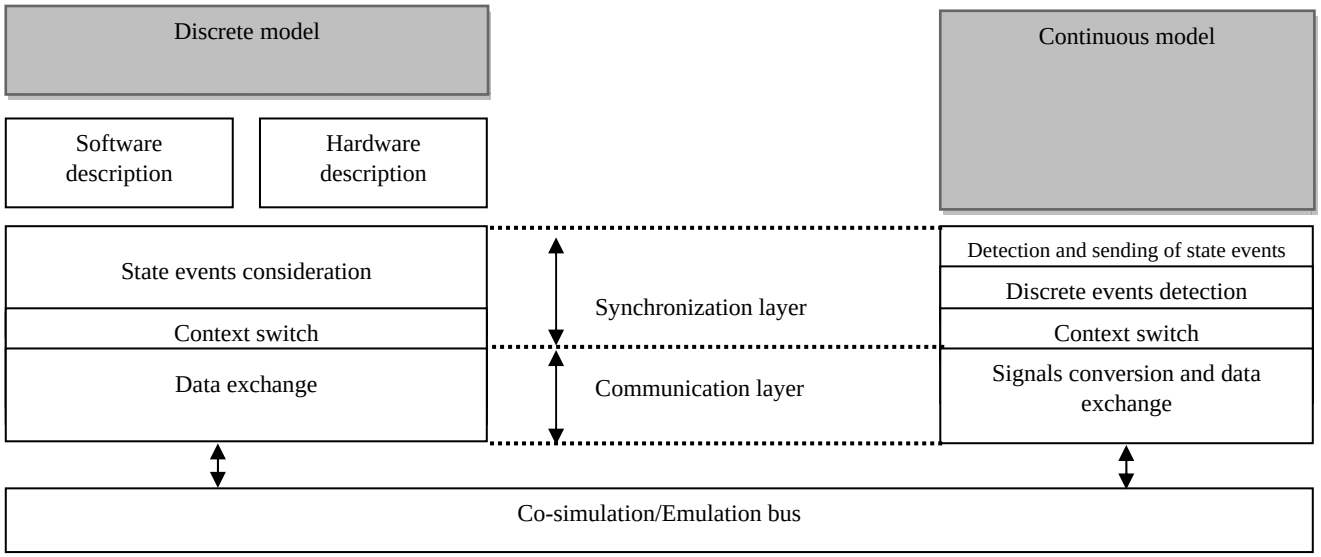


Fig. 1. Generic architecture for CODIS+ simulation.

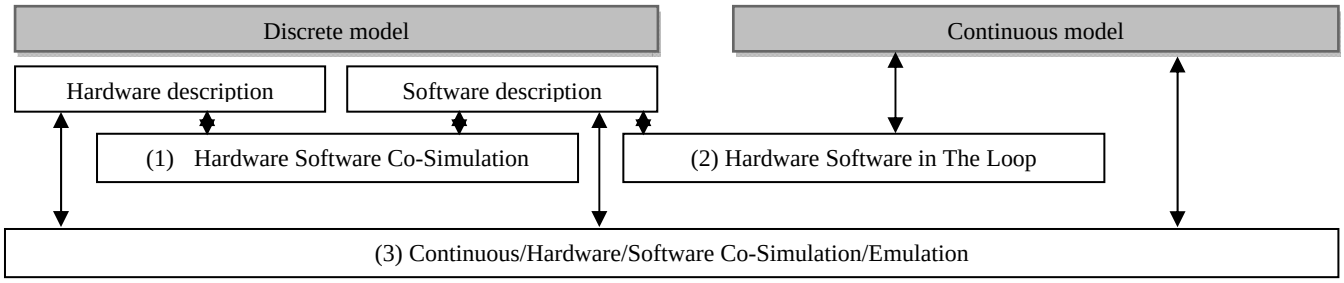


Fig. 2. Supported simulations of CODIS+.

4.1. Hardware Software Co-Simulation model

The hardware software co-simulation model is based on synchronization schemes which respect not only the continuous and discrete model proposed in the CODIS tool, but also the interaction style that can be involved between HW and SW components. In the same design, different synchronization schemes can be used for HW and SW components. The simulation time of SW applications represents the execution time in the target base architecture. Nevertheless, it should be stated that the task scheduling policy is not the aim of this paper. The goal is to define the relationship between simulators based on synchronization schemes. SC time in the following figure represents the advance time of discrete simulator.

- **Scheme 1: The SW Task receives data periodically from the hardware Task.**

As is shown in figure 3, the scheme is based on FIFO memory between SW Task and HW Task. The main idea is based on fixed synchronization time between simulator and emulator. Because of the difference in speed, the HW imposes a synchronization Time (Tsync). This Tsync must be greater than Task time of HW or SW.

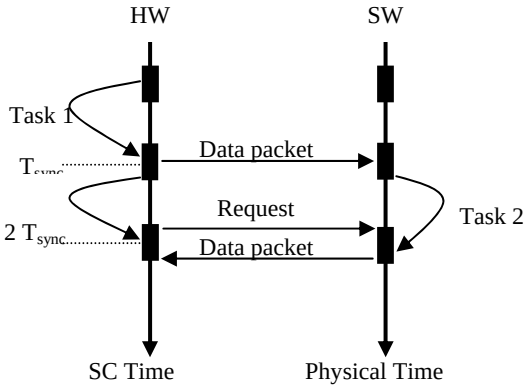


Fig. 3. Synchronization model: scheme 1.

- **Scheme 2: The SW Task waits the end of the hardware Task.**

Figure 4 presents the synchronization model: scheme 2. It is clear that when a hardware component is simulated by SystemC, the SW Task uses a waiting loop for data. Once the hardware Task (Task1) is finished, the simulator sends data to the SW Task and a context switch from SystemC to board takes place. Then, the SW Task receives data and resumes the execution. After that, the execution time of Task1 is modeled by the SystemC wait () function. The amount of time used by the wait function is sent to the SW part to inform it about the duration of the waiting loop as can be seen in figure 4. The SystemC and the emulator need to exchange the time stamp at every context switch.

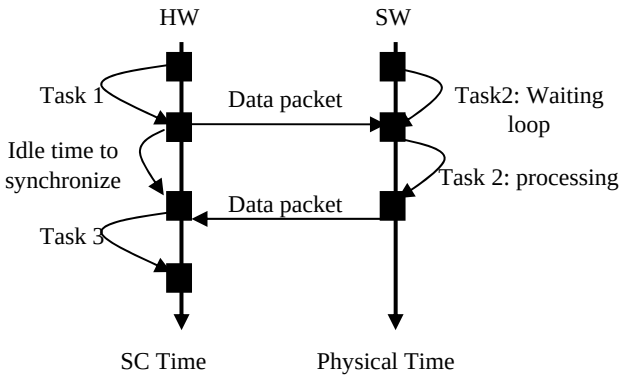


Fig. 4. Synchronization model: scheme 2.

- **Scheme 3: The SW Task receives an interrupt to indicate the end of the hardware Task**

Figure 5 illustrates the synchronization model: scheme 3where the software does not use a waiting loop. However, the end of the Task is indicated by interrupt. Therefore, the software can execute the Task instead of waiting. The Simulation scheduler, running on the target processor, sends data to the simulation interfaces as shown by arrow 0. This activates the hardware Task1. At the end of Task 1 process, and before sending data to SW Task, the wait_for_interrupt(sc_time) function is called as can be seen in Figure 6. This makes the simulator advance its time as shown by arrow 1. Furthermore, it sends an interrupt packet to inform the emulator of the next time stamp as shown by arrow 2. Simultaneously, the simulation scheduler activates a timer with a period that coincides with the received interrupt time stamp and begins the execution of an intermediate Task; i.e. an eventual user background Task. When the timer is reached, it interrupts the background Task. Thus the simulation scheduler activates Task 2 implying that the number of the interrupt is received with the interrupt packet. The last one may request data, thus Task 1 resumes execution and sends data packet As shown by arrow 4; which activates Task 2. Figure 7 shows the template of the code.

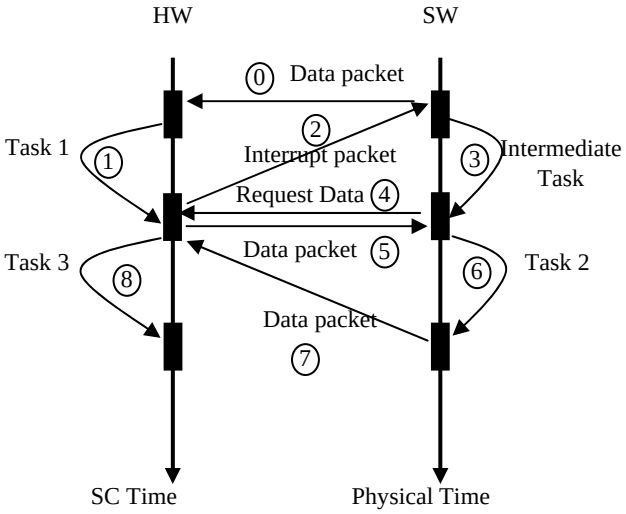


Fig. 5. Synchronization model: scheme 3.

```
Void wait_for_interrupt (sc_time t)
{wait(t);
 send_interrupt_packet(…..) ;}
```

Fig. 6. Wait_for_interrupt code.

Where t is an estimation of the Task 1 duration

```
                /* Task1 code */
 Instructions
 Wait_for_interrupt (t);
 Switch_context();/* context switch to SystemC */
```

Fig. 7. Template of synchronization code.

- **Scheme 4: The SW Task may receive a random interrupt resulting from externally received data**

This scheme is illustrated by Figure 8. The SystemC begins the execution of Task 1 and, when finished, sends a data packet to the SW Task. Task 2 starts and the SystemC executes the Hardware_Input_Interface. This is a process that models the input interfaces of the hardware subsystem. However, its execution does not advance the SystemC local time. The process may generate a random interrupt packet which informs the SW task of the reception of new data. The sent packet via USB generates a USB interrupt which interrupts Task 2. Thus, the USB interrupt plays the same role as the hardware interrupt. Once the interrupt takes place. Once the interrupt takes place, Task 3 begins. Therefore, the information needed to be executed by the interrupt routine

can be executed. This information can be found in the received interrupt packet.

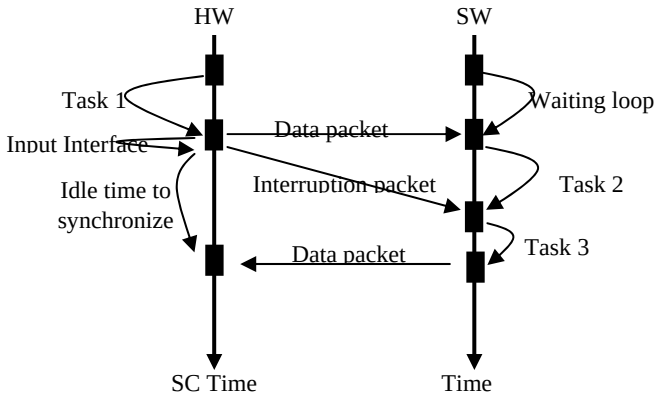


Fig. 8. Synchronization model: scheme 4.

To ensure communication and to save the synchronization context, an array of shared registers was used. We assumed that the HW / SW partitioning was static. Also, the scheduler was static and based on data dependence. The shared registers were used as a shared connection bus. Although this register-based bus modelling was not the same as the actual chip bus modelling, it was easy to set up. The register array was implemented on the FPGA board and could be accessed from the simulator using interrupt services routines.

The Simulator Engine ensures the verification of sequential and parallel applications. Two examples are presented. The first one is based on sequential Tasks and the second one is based on parallel and sequential Tasks.

4.2. Hardware Software in The Loop (HSIL) model

HSIL model is a new co-simulation technique for different layers. It synchronizes the continuous simulator and the HW accelerator.In line with (Ben Ayed et al., 2013), we consider that the time synchronization between the Simulink simulator and the processor emulated on the FPGA board a key issue of the proposed approach. The verification method is based on the following synchronization scheme which respects the interaction style between the continuous and the discrete model.

- **The Continuous model waits the end of the hardware/software task.**

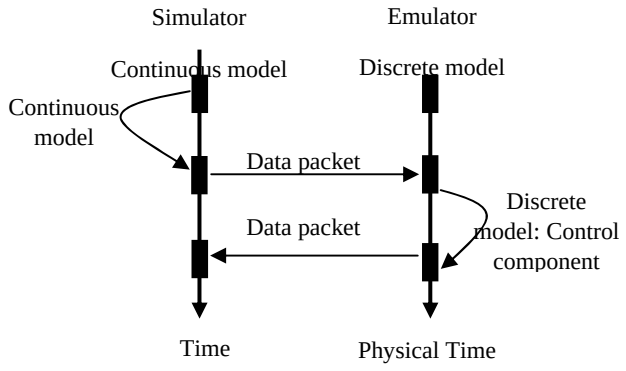


Fig. 9. Synchronization scheme for HSIL.

As can be seen in Figure 9, when a hardware/software component is emulated by the board, the continuous model uses a waiting loop for data.

Once the controller task is finished, the emulator sends the data to the simulator and a context switch from the board to Simulink simulator is taken. Simultaneously, the continuous model receives the data and resumes execution. Note that the Simulink and the emulator need to exchange information about the time.

4.3. Continuous/ Hardware/Software Co-Simulation/Emulation model

Figure 10 summarizes the presented synchronization scheme. The key part of the CODIS+ synchronization is localized in the hardware description that represents the simulation master because the time advancing is tied to time stamp obtained from the discrete event.

CODIS+ proposes a co-simulation tool for heterogeneous systems i.e. Continuous/HW/SW. It is based on the co-simulation/emulation model between continuous simulator, discrete simulator and HW accelerator.

We assume our system contains a continuous model and a discrete model comprising four events (A,C,D: HW event, B: SW event).

First, the HW accelerator executes the wait() function that makes the processor wait for HW interrupt and switches context to the discrete simulator. Second, the Discrete Simulator (DS) executes event A without advancing the time and without changing the output variables. Third, a switch context is made to Continuous Simulator (CS). The continuous model begins its execution until the time stamp is reached and returns to DS. Fourth, the update step is made i.e. update outputs and time. Fifth, the DS synchronises to the HW accelerator to execute the SW event B, and executes the Idle() function. Sixth, after the execution of event B, a context switch to CS is invoked to align the time and return to DS. Seventh, event C is executed. But in this case, we assume that the CS generates a state event before the time stamp is attained. In this case, a context switch to DS is performed and the discrete time is advanced to the time of the state event followed by a context switch back to CS.

5. EXPERIMENTAL RESULTS

The presented synchronization schemes are implemented for Matlab/Simulink for continuous model and SystemC/Hardware accelerator for discrete model. It should be clearly underscored that the defined synchronization scheme presented in section 4 is independent of language or tool and tied only to the continuous model and the discrete model.

CODIS+ provides a generic interface for different models as shown in Figure 11.

To analyze the capabilities of the simulation tool, an illustrative example was used. This example was a speed limiter system used to limit the speed of the car according to the driver. The figure 12 exhibits the proposed system functionalities.

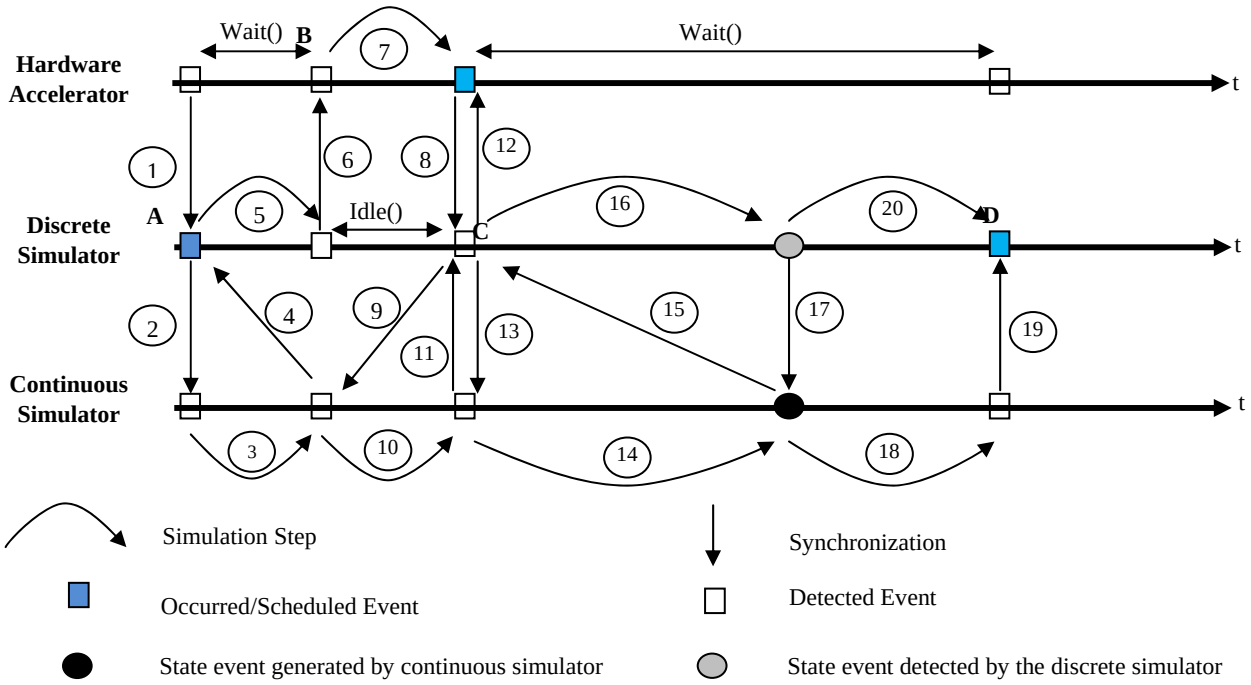


Fig. 10. Synchronization scheme for CODIS+ simulation.

The discrete part of the system was modelled using SystemC/Hardware accelerator and the continuous part was modelled using Simulink.

Speed limiter system was the combination of two studied systems in (Ben Ayed et al., 2013; Ben Ayed et al., 2010): Fingerprint recognition and Closed-Loop Engine Speed Control. The fingerprint system was described by event discrete model. It was based essentially on five steps: (1) Filtering (2) Binarization (3) Skeletonization (4) Minutia extraction (5) Matching. In (Ben Ayed et al., 2016) the different parts of fingerprint recognition were portioned using the time consumption as follows:

- Hardware components: Binarization, Filter and Matching.
- Software applications: Minutia extraction and Skeletonization.

The Closed-Loop Engine Speed Control is described by discrete and continuous model. This system is composed of the following components:

- Event-discrete parts: Controller and Compression.
- Continuous parts: Throttle & Manifold, Combustion and Vehicle Dynamics.

Based on CODIS+ model the speed limiter components are simulated using:

- Event-discrete simulator (SystemC): Binarization, Filter and Matching.
- Hardware Accelerator (FPGA board): Minutia extraction and Skeletonization.
- Continuous simulator (Matlab): Throttle & Manifold, Combustion and Vehicle Dynamics.

Figure 13 shows the added interfaces for the Closed-Loop Engine Speed Control to support CODIS+ simulation. Table 1 demonstrates that CODIS+ exhibits the shortest simulation time.

CODIS+ is approximately three times faster than CODIS and approximately 8 times faster than Matlab. CODIS+ presents an excellent tool based on co-simulation for heterogeneous systems.

The experimental results support the initial hypothesis. CODIS+ ensures a minimum simulation time because the SW part is simulated by a hardware accelerator instead of an ISS (Instruction Set Simulator).

Table 1. A comparison of the simulation time between different simulation tools.

	Matlab Simulation	CODIS	CODIS+
Simulation time (s)	19.2	8.4	**2.55**

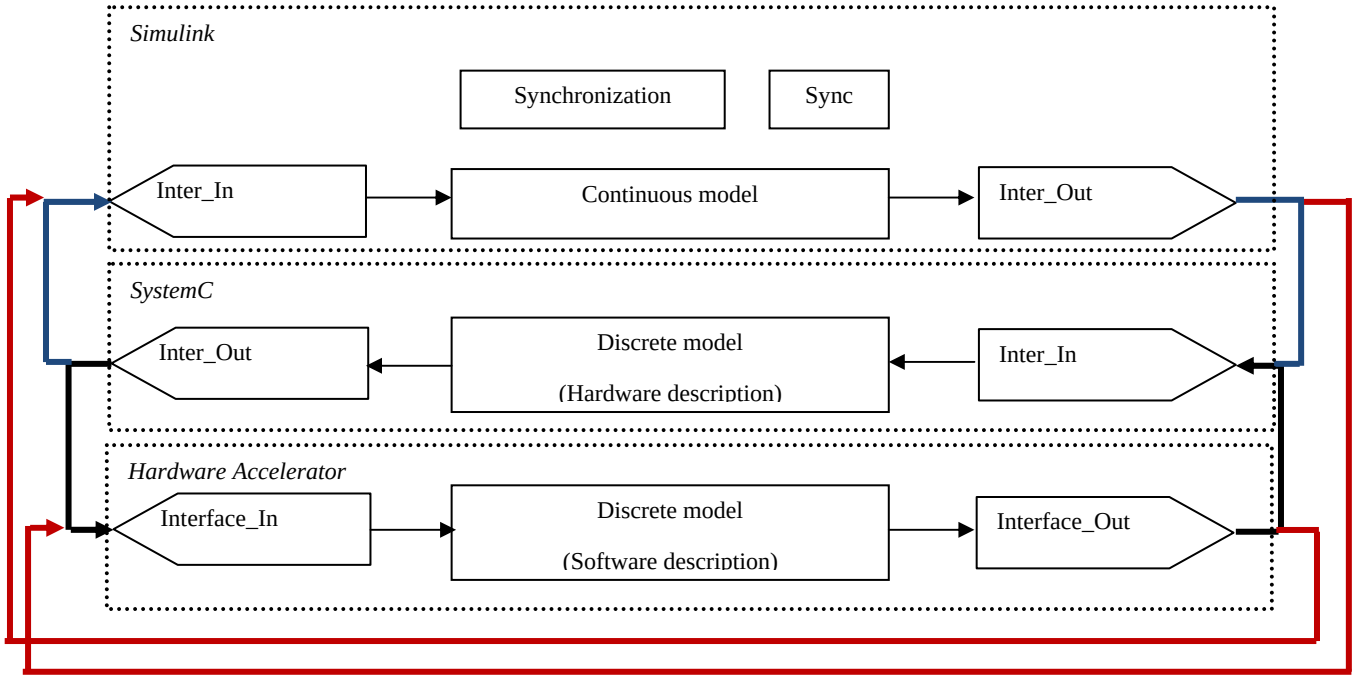


Fig. 11. CODIS+ interfaces.

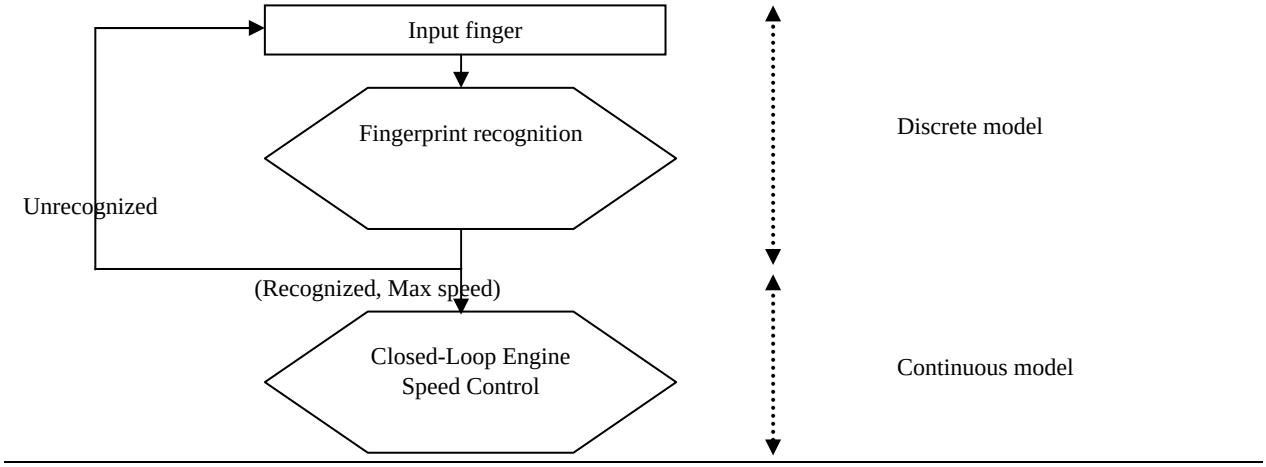
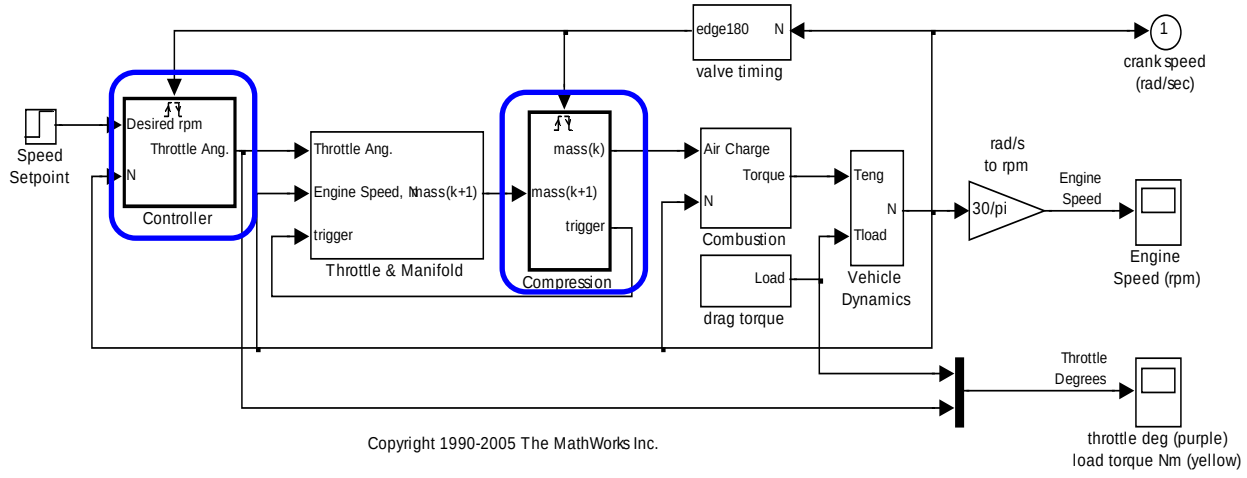
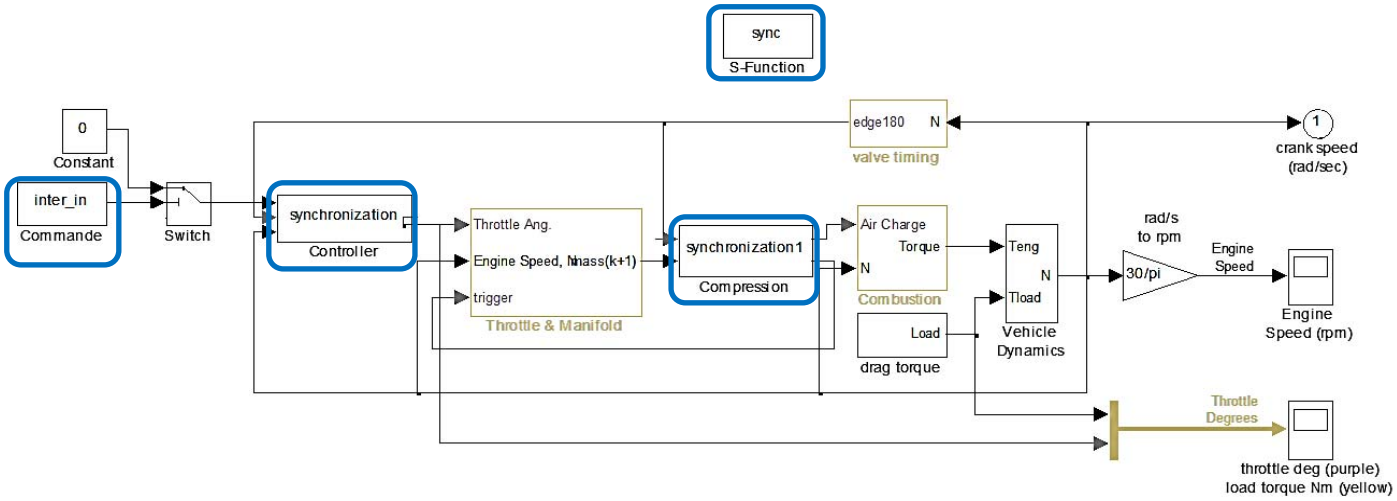


Fig. 12. Speed limiter system.



(a)Closed-Loop Engine Speed Control model



(b)Closed-Loop Engine Speed Control with CODIS+ interfaces

Fig. 13. Generic interface for CODIS+.

6. CONCLUSION

System complexity, heterogeneity of models in the same system and time-to-market constraints, present a challenge for designers. Dynamic systems consist of pre-designed components with different models i.e. continuous model and discrete model. That is why, new CAD tools are required for modelling and simulation.

CODIS+ environment is proposed for the simulation of heterogeneous systems. This tool provides a generic simulation model that generates interfaces for continuous models (implemented in Matlab/Simulink) and discrete event models (implemented in SystemC and HW accelerators).

It was shown on a test example that the simulation time was reduced by 86% compared to Matlab and 69% compared to CODIS environment.

7. ACKNOWLEDGEMENT

The authors would like to express their gratitude to Dr. Ayadi Hajji for his help with proofreading, correcting and improving the English of the manuscript.

The author would like to thank Deanship of Scientific Research at Majmaah University for funding this project under the number 37/112.

REFERENCES

Active-HDL (2015), available on line at */http://www.aldec.com/ActiveHDLS*, Established in 1984, final revision in 2015.

Al-Junaid, H., and Kazmierski, T.J. (2004), SEAMS - a SystemC Environment with Analog and Mixed- Signal Extensions, *IEEE International Symposium on Circuits and Systems*, IEEE.

Azam, F., Zhang, L., and Ahmad, R. (2005), Using UML profile for connecting information architecture and detailed information design, *Proceedings of the IEEE Symposium on Emerging Technologies*, 423 – 428, IEEE.

Ben Ayed, M., Bouchhima, F., and Abid, M. (2013), A Fast Simulation Emulation Engine, *Informacije MIDEM - Journal of Microelectronics, ElectronicComponents and Materials*, Volume (43), 162 – 172.

Ben Ayed, M., Bouchhima, F., and Abid, M. (2013), A novel verification technique for control units, *The International Journal of Engineering and Technology (IJET)*, Volume (5) 1990-1999.

Ben Ayed, M., Bouchhima, F., and Abid, M. (2010), A Novel Application of the Classifier DECOC Based on Fingerprint Identification, *Interactive Multimodal Pattern Recognition in Embedded Systems IMPRESS Workshop on Database and Expert Systems Applications DEXA*, IEEE.

Ben Ayed, M., Bouchhima, F., and Abid, M. (2012), A Fast Hardware/Software Co-Verification Method using a real hardware acceleration, *the 24th International Conference of Microelectronics ICM*, IEEE.

Ben Ayed, M., and Elkosantini, S. (2016), An Accelerated Architecture Based on GPU and Multi-Processor Design for Fingerprint Recognition, *International Journal of Advanced Computer Science and Applications*, Volume (7), 337-348.

Bonnerud, T.E., Hernes, B., and Ytterdal, T. (2001), A mixed-signal, functional level simulation framework based on SystemC for system-on-a-chip applications", *IEEE Custom Integrated Circuits proceedings*, IEEE.

Bouchhima, F., Nicolescu, G., Aboulhamid, M., and Abid, M. (2007), Generic discrete-continuous simulation model for accurate validation in heterogeneous systems design, *Microelectronics Journal*, Volume (38), 805-815.

Bouchhima, F., Nicolescu, G., Aboulhamid, M., and Abid, M. (2005), Discrete-continuous simulation model for accurate validation in component-based heterogeneous SoC design, *The 16th IEEE International Workshop on Rapid System Prototyping*, IEEE, 181 – 187.

Balarin, F., Watanabe, Y., and Hsieh, H. (2003), Metropolis: an integrated electronic system design environment, *Computer*, Volume (36), 45–52.

Celoxica, available on line at */http://www.celoxica.com/methodologyS*.

Chang, W.T., Ha, S., and Lee, E.A. (1997), Heterogeneous simulation—mixing discrete-event models with dataflow, *Journal of VLSI Signal Processing Systems - Special issue on the rapid prototyping of application specific signal processors (RASSP) program*, Volume (15), 127-144.

Dehghanimmohammadabadi, M., and Keyser, T.K. (2017), Intelligent simulation: Integration of SIMIO and MATLAB to deploy decision support systems to simulation environment, *Simulation Modelling Practice and Theory*, Volume (71), 45-60.

ELTahaway, H., Rodriguez, D., Sabiro, S.G., and Mayol, J.J. (1993), VHDeLDO: a new mixed mode simulation, *Proceedings EURODAC'93*.

Enwich, K., Schwarz, P., Grimm, C., and Waldschmidt, K. (2001), SystemC extensions for mixed-signal system design, *System Specification & Design Languages*, 19-28, Chapter 2, Springer US.

Ghomri, L., and Hassane, A. (2013), Continuous flow Systems and Control Methodology Using Hybrid Petri nets, *Journal of control engineering and applied informatics*, Volume (15), 106-116.

Girard, A., and Pappas, G. J. (2007), Approximation Metrics for Discrete and Continuous Systems, *IEEE Transactions on Automatic Control*, Volume (52), No. 5, p.782-798, 2007.

Gupta, K., Davis, R.S., and Tescher, P.E. (1985), A review of recent developments in solving ODES, *ACM Computing Surveys (CSUR)*, Volume (17), 5-47.

Hickey, D.R. , Wilsey, P.A. , and Hoekstra, R.J. (2006), Mixed-signal simulation with the Simbus Backplane, *39th Simulation Symposium*, IEEE.

IEEE Std 1076.1 (2007), IEEE Standard VHDL Analog and Mixed-signal Extensions, *IEEE*.

ITRS (2006), International Technology Roadmap for Semiconductor Design, available on line at */http://public.itrs.net/S*.

Kajtazovic, S., Steger, C., and Pistauer, M. (2005), A HDL-independent modeling methodology for heterogeneous system designs, *IEEE Behavioral Modeling and Simulation Workshop*, p. 88 – 93.

Li, S., and Cao, J. (2015), Distributed adaptive control of pinning synchronization in complex dynamical networks with non-delayed and delayed coupling, *International Journal of Control, Automation, and Systems*, Volume (13), 1076–1085.

Martin, D.E., Wilsey, P.A., and Hoekstra, R.J. (2002), Integrating multiple parallel simulation engines for mixed-technology parallel simulation, *Simulation Symposium*, IEEE.

Matlab/Simulink, available on line at *www.mathworks.com*.

Nicolescu, G., Yoo, S., and Bouchhima, A. (2002), Validation in a component-based design flow for Multicore SoCs, in *Proceedings of ISSS*, IEEE.

Minh, T., and Pumwa, J. (2012), Simulation and Control of Hybrid Electric Vehicles, *International Journal of Control, Automation, and Systems*, Volume (10), 308-316.

Modelica (2012), A unified object-oriented language for physical systems modeling, *specifications report, www.modelica.orgS*.

Oltean, V.E., Dobrescu, R., Popescu, D., and Nicolae, M. (2010), Hybrid modelling and simulation approaches for a class of mechatronic systems, *Journal of control engineering and applied informatics*, Volume (12), 47-54.

Patel, H.D., and Shukla, S.K. (2004), Towards a heterogeneous simulation kernel for system level models: a SystemC kernel for synchronous data flow models, *IEEE Computer society Annual Symposium on VLSI*, IEEE.

Pecheux, F., Lallement, C., and Vachoux, A.(2005), VHDL-AMS and Verilog-AMS as Alternative Hardware Description Languages for Efficient Modeling of Multidiscipline Systems, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 204-225.

Pichon, F., Blanc, S., and Candaele, B. (1995), Mixed-signal modeling in VHDL for system-on-chip applications, *European Design and Test Conference*, IEEE.

Pinki, M., Francis, M., Chandrasekhar, V., Austin, A., and Mantooth, H.A. (2003), Achieving language independence with Paragon, *International Workshop on Behavioral Modeling and Simulation*, p. 149-153.

Prabhakar, P., Duggirala, P.S., Mitra, S., and Viswanathan, M. (2015), Hybrid automata-based cegar for rectangular hybrid systems, *Formal Methods in System Design*, Volume (46), 105-134.

Ptolemy (2015), *University of California, Berkeley, /www.ptolemy.eecs*.

Riihimaki, J., Kukkala, P., and Kangas, T., Hannikainen, M., Hamalainen, T.D. (2005), Interfacing UML 2.0 for Multiprocessor System-on-ChipDesign Flow, *International Symposium on System-on- Chip*, p. 108-111.

Schorcht, G., Troxel, I., Farhangian, K., Unger, P., Zinn, D., Mick, C.K., George, A., and Salzwedel, H. (2003), System-level simulation modeling with MLDesigner, Modeling, Analysis and Simulation of Computer Telecommunications Systems, *11th IEEE/ACM International Symposium*, IEEE, p 207 – 212.

Urrea, C., and Coltters, J. P. (2015), Design and implementation of a graphic 3D simulator for the study of control techniques applied to cooperative robots, *International Journal of Control, Automation and Systems*, Volume (13), 1476–1485.

Vachoux, A., Grimm, C., and Einwich, K. (2003), Analog and mixed signal modeling with SystemC-AMS, *Proceedings of the 2003 International Symposium on circuits and systems*, IEEE.

Zwolinski, M., Garagate, C., and Mrcarica, Z. (1995), Anatomy of a simulation backplane, *IEE Proceedings - Computers and Digital Techniques*, Volume(142), 377-385.