Embedded Systems Requirements Verification Using HiLeS Designer

C-E. Gómez[1,3], J-C. Pascal[1,2], P. Esteban[1,2], Y. Déléris[4], J-R. Devatine[5]
1: CNRS; LAAS; 7 avenue du colonel Roche, F-31077 Toulouse, France
2: Université de Toulouse; UPS, INSA, INP, ISAE; LAAS; F-31077 Toulouse, France
3: Universidad de Los Andes; Carrera 1 N° 18A 10; Bogotá, Colombia
4: Airbus; 316 route Bayonne, F-31300 Toulouse, France
5: Aéreoconseil; 3 rue Dieudonné Costes; F-31700 Blagnac, France

Abstract: One of the issues related to systems design is the early verification in first design steps: system specifications verification. Nowadays, it is common to use text-based specifications to begin a system design. However, these specifications cannot be verified until a software model is made. In this work, we show how can we use HiLeS Designer to model and verify, formally and by simulation an embedded system specification. This tool makes easier to build the model, using graphical concepts which are familiar to designers. It also helps to verify formally the structure and some logical behavior, and by simulation, it is possible to verify the consistence of the embedded system specification. We model and verify System Display Selector Requirements applying HiLeS Designer.

Keywords: Verification, Validation, Virtual Prototype, VHDL-AMS, Petri-nets.

1. Introduction

In order to manage the complexity of a system, it is necessary to increase the abstraction level to describe in highlights how the system will be conceived before to take into account details such as specific time, protocols, and the partition hardware software. Electronic System Level (ESL) [Martin07] tries to introduce a new abstraction level called "System level" in the design flow of a system. This abstraction level is used to model the general functionality and architecture of the system without seeing detailed information and especially which system part will be developed in software and which part in hardware.

The tendency in ESL is to create models based in traditional model language such as Java, C, C++ and more recently SystemC [IEEE05] and SpecC [Fujita01]. These models are verified by simulation using an specific test scenario. However this test does not guarantee the model can have a non-wished behavior, behavior that was not evaluated in the different test that the model is evaluated.

There is another tendency to use model language such as UML [Vanderperren08] and SysML [Friedenthal08]. These languages are tied to the MDA [Boulet03] in order to verify and to generate a virtual prototype base on the specification model created in these languages.

We propose to use HiLeS Designer to model and verify embedded systems design representing textual requirements in block concepts where the functionality and the constraints are encapsulated into the block. HiLeS Designer has the feature to use Petri net as part of the model language. This feature allows to verify formally the structure and logical behavior described in this formalism without using a test scenario. HiLeS Designer also allows to generate an executable virtual prototype in VHDL-AMS where the analogical and logical behavior can be evaluated at the same time.

This paper is organized as follows: Section 2 shows the related work with HiLeS Designer, Section 3 gives a HiLeS Designer overview. Section 4 describes how HiLeS Designer is used to model and to verify a System Display Selector. Section 5 provides the conclusions and directions for future works.

2. Related Work

In the market, there are different tools which use models of computation (MoC) concept to model systems in different domains. These MoC is the way to represent physical behavior of specific domains. For instance, image processing is modeled in a MoC called Kahn's Processes Networks, these are processes which interact by channels that can buffer messages (e.g. FIFO), or electrical circuits use Continuous-Time MoC which are functional and storage components communicating with continuous signals [Liu01]. Tools such as MLDesigner [Schorcht03], Ptolemy II [Ptolemy09] and Visual Sim [MD03] are examples that use MoC.

There are other tools which use an specific language to describe the functional behavior of the systems. Matlab [Mathworks09], Modelica [Fritzson98] and Cofluent Studio [Calvez93] are examples of this tendency. Matlab uses a C-based language to describe a system, Modelica use its own oriented-object language to model and Cofluent Studio is based on SystemC.

Some of this tools allow the mix of MoC. In MLDesigner, it is possible to model in the same

workspace continuous-time and Discrete events MoC; it also is possible to build in Matlab/simulink/Statechart mix but its execution is done in its own execution engine, e.g., there is communication transformation between the two MoC in order to execute in its own domain.

HiLeS designer contains Continuous-Time, Discrete-Time and Discrete event MoC, thanks to the use of VHDL-AMS as base of the HiLeS formalism. VHDL-AMS allows modeling Continuous and Discrete time systems, but it is not appropriated to model Discrete events systems. For that reason, we transform the discrete-event model to VHDL-AMS in order to execute in a same environment the three MoC. This transformation gives HiLeS Designer the advantage of simulating faster and creating, in one standard language, Intellectual Properties (IPs).

3. HiLeS Designer

3.1. HiLeS

High Level Specification or HiLeS is a graphic language developed by Jimenez [Jimenez00]. Its objective is to help systems designers to model the architecture and behavior of a system into system level, e.g., without specifying software and hardware features.

HiLeS has four main concepts:

- Structural Block
- Functional Block
- Control Net
- Channels
- Ports

The structural block represents the system architecture. Each structural block can be composed by other Structural blocks, functional blocks and a control net. These elements represent a specific group of behaviors encapsulated in a block. The system hierarchy is decomposed in structural blocks and, at the same time, the detail level.

The functional block is the lowest level in a system functional description. It defines a function represented by an equation.

The Control Net is an ordinary Petri net which controls the execution sequence of each block defined in the model. Each block is represented by a place in the control net. By arcs, drawn with dashed arrows, the process defined in the related block is executed. Once the process finishes, the block fires the transition after the place which represents the executed block and the control net sequence continues.

Channels are routes which transmit the information from the environment to the system, from one block to another and from the control net to blocks. There are defined two channel types: Control and Continuous. The Control channels manage the execution of the blocks, and the Continuous Channels route the signals from blocks to blocks or environment to system, according to their nature.

Finally, ports are interfaces between the blocks and its environment. They put the blocks or the environment information within channels in order to transmit or receive information (out-port and in-port respectively) to other blocks or environment. They have nature and type such as they are defined in VHDL-AMS.

In Figure 1, we show an example of the HiLeS graphic representation. The figure illustrates two flows: signal flow and sequence flow and also two hierarchical levels.

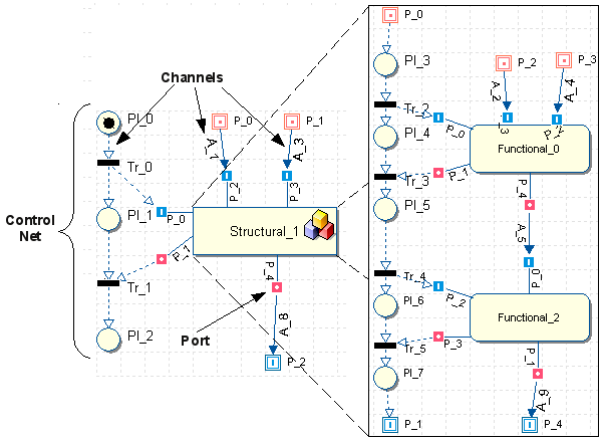

Figure 1: HiLeS Concepts example

In this example, the first hierarchical level is represented by the “Structural_1” block. This block is internally described by two functional blocks (Functional_0 and Functional_1) and these blocks define the “Structural_1” block functionality. The “Structural_1” intern definition is considered as the second hierarchical level an it is the last level because there are only functional blocks.

The signal flow is represented by the signal transformation made in the structural and functional blocks flow. In the example, there are two signals continuous that go from the environment to the system through A_1 and A_3 channels. These signals are transformed inside “Structural_1” structural block. The signals go in though P_2 and P_3 in-ports. Inside the structural block, the signals are conduced by A_2 and A_4 channels to the “Functional_0” block where they are processed. The result is only one signal which goes out though P_4 out-port. The new signal is conduced from the “Functional_0” block to “Functional_2” block through A_5 channel. This signal is processed for “Functional_2” functional block through P_0 in-port. The final result goes out through the P_1 out-port and it is transported to the P_4 out-port through A_8 channel. The resulting signal is sent from “Structural_1” through P_4 out-port to the environment using P_2 out-port conduced by A_8 channel.

The sequence flow is represented by the control net. A token in a place indicates which block is in execution. In the example, the execution begins in Pl_0 place. When Tr_0 transition is fired, a token is sent to Pl_1 and another to execute “Structural_1”. Then, inside “Structural_1”, Pl_3 is marked. The sequence continues, executing “Functional_0”. Once “Functional_0” finishes its process, Tr_3 transition is fired by a signal from the functional block. The sequence continues in the same way finishing in Pl_2.

For each structural block is possible to create different architectural solutions as it is depicted in Figure 2. Figure 2a shows the structural block with its in-ports which process the signals from the environment and out-ports that are the processing result. This structural block has a specified functionality. However, to solve this functionality it is possible to propose different solutions. Figure 2b and 2c shows two different architecture examples in order to compose “Structural_2” block.

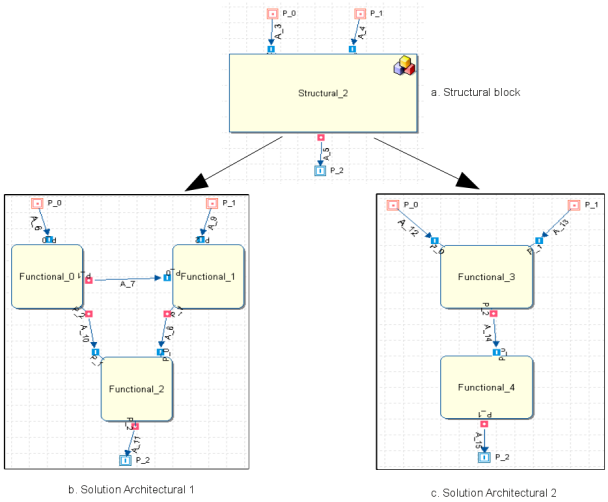

Figure 2: Different architectural solutions.

3.2. HiLeS Designer

HiLeS Designer is the tool that supports the HiLeS language and it was conceived by Hamon [Hamon05]. In Figure 3, there is a schema of how HiLeS Designer works. The designer interprets and represents in HiLeS concepts the specifications described in natural language. The model is built not only with the HiLeS concepts, but also with VHDL-AMS description in the functional blocks.

Once the model of the first executable level is created, e.g., the first level abstraction, it is possible to verify before continuing with levels more detailed. HiLeS Designer has two types of verification, formal logical verification and verification by simulation.

Formal logical verification is a process related to the control net described in the model. Since the control net is a Petri net, it is possible to verify mathematically the structure and the behavior of the execution sequence design. Features such as deadlock (when a marking is not enabled to activate a transition), liveness (when the Petri net is not blocking), boundedness (when the tokens in each Petri net place is limited in every marking) and reachability (when every place is reachable) are analyzed. This process is presented in HiLeS Designer due to the integration with a Petri net analysis tool call TINA [Berthomieu06]. HiLeS Designer extracts the control net from the model which wants to be analyzed, it translates the control net to the TINA language and create a flat text file. HiLeS Designer executes TINA with this file and TINA generates a flat text file with the analysis results. HiLeS Designer reads the file and prints in a output text field. With these results, the designer identifies the logic problems in its design without need a test bank to find them out.

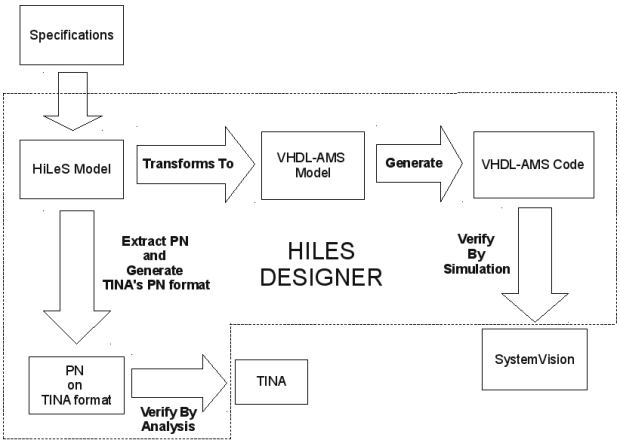

Figure 3: HiLeS Designer functionalities

When the logic verification is made, the verification by simulation is executed. HiLeS Designer generate a virtual prototype from the model in VHDL-AMS. Each concept from the HiLeS language is connected to a concept in VHDL-AMS. At the same time, the test scenario is created according to the designers' configuration. The VHDL-AMS code is compiled and executed in VHDL-AMS tools such as SystemVision [Mentor09]. The designer analyzes the simulation results and compares if these results are according to the specification.

Once the first level is verified, the designer can descend the abstraction level to model in more detail the system.

4. System Display Selector

In order to show how HiLeS designer works and it is applied in a real system, we present this example. The example system is modeled using HiLeS Formalism starting from the specification document. The formal verification and the verification by simulations are applied.

4.1 SD Selector Modeling on HiLeS Designer

The System Display Selector (SD Selector) is a device which is used in Airbus aircraft. This system selects the correct page to be displayed in a screen

according to their input signals. The system requirements are described in natural language in a document. The work is to interpret the system requirements, to build a model and to verify the model coherence and completeness satisfying the requirements. This verification is done formally and by simulation using HiLeS Designer.

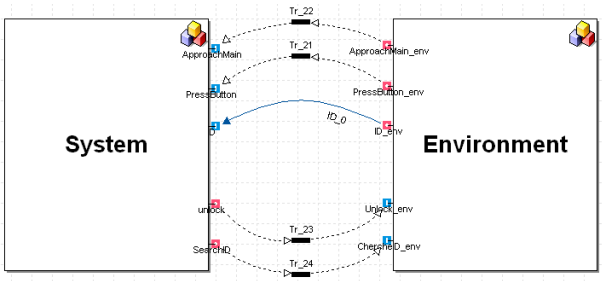

Figure 4: Level 0 of the SD Selector model

To model the system in HiLeS Designer, we have to use different abstraction levels in order to make step by step the model based on the textual requirements. The first level, called “Level 0”, serves to model the system and its environment as a black box (Figure 4). The environment is used to model different scenarios that the system can have following the textual requirements. This scenario model helps to generate the test scenarios for the virtual prototype in VHDL-AMS.

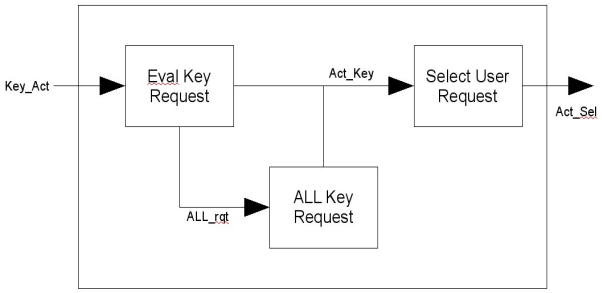

Figure 5: System Architecture requirement.

The following levels are made based on the architecture proposed by the textual requirements. In Figure 5, there is an architecture example described in the system requirements, and in Figure 6, the same architecture expressed on HiLeS.

We also maintain the hierarchy expressed in the requirements document, using the structural block concept in HiLeS. Additionally, to make easier the functional description with more detailed requirements, we build new hierarchies to specify each functionality, until we reach to the lower level of description using functional blocks.

In order to show in more detail how a model can be built using the HiLeS formalism, we choose the “Valid key Action” subsystem. The subsystem's functionality evaluates the user’s action on the keyboard, when the user selects the information they want to show on the screen. The “Valid key Action” subsystem requirements are described in a requirement document subsection. The textual requirements are interpreted and the subsystem model is deduced. In Table 1, there are some requirement examples extracted from the document.

In Figure 7, the HiLeS model of this subsystem is depicted. This model follows every requirement described for the subsystem in the requirements document. To illustrate how the textual requirements are interpreted using HiLeS concepts, we will use the requirements described before. The state position presented in the Requirement 1 is represented by the places Pl_170 (Pressed) and Pl_66 (Released). The state Status is represented by the places Pl_179 (Valid) and Pl_178 (Invalid). According to the user's action, the states change. Following the Requirement 3, when a key is pressed in a keyboard, the place Pl_64 is filled by a token. Since the Pl_66 (Unpressed) and Pl_64 (PressKey) have token, the Tr_61 is fired, then the Pl_68 (Timer) and Pl_170 (Pressed) is filled with a token. That means the key status changes from “released” to “pressed” achieving the textual requirement. The requirement 2 is identified in the figure by the token presented in the places Pl_66 and Pl_179. The timer, which is a functional block, is described in VHDL-AMS (Figure 8) and its behavior is to wait a defined time period. In our example, the time is the period where the pressed key is valid without releasing it.

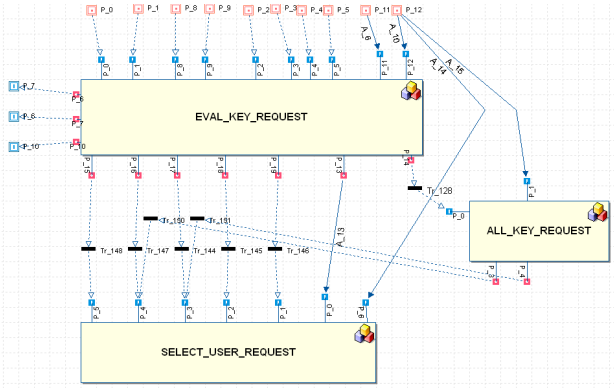

Figure 6: System Architecture requirement in HiLeS.

4.2 Formal Verification Analysis

Once the model is complete, the model verification can be started to ensure the requirements are considered. The first verification step is the formal verification. Its purpose is to find inconsistencies in our design (Blocking, deadlocks, behaviors which are not taking into account in the requirements). In this paper, we choose the requirements 4 and 5 to verify partially the model. These requirements are represented by the actions on the Places Pl_71 (Selected), Pl_170 (Pressed) , Pl_179 (Valid) and Pl_178 (Unselected or None) in Figure 7.

Req. 1	Each key shall be associated to two couples of states: Position: PRESSED-RELEASED and Status: VALID-INVALID
Req. 2	The initial state of the keys shall be status VALID and position RELEASED.
Req. 3	While the system receives an action on the keyboard, the corresponding key position shall be set to PRESSED.
Req. 4	SELECTED KEY shall be the identifier of the first PRESSED AND VALID key.
Req. 5	If there is not a (PRESSED AND VALID) key then SELECTED KEY shall be set to NONE.

Table 1: Part of System Requirements

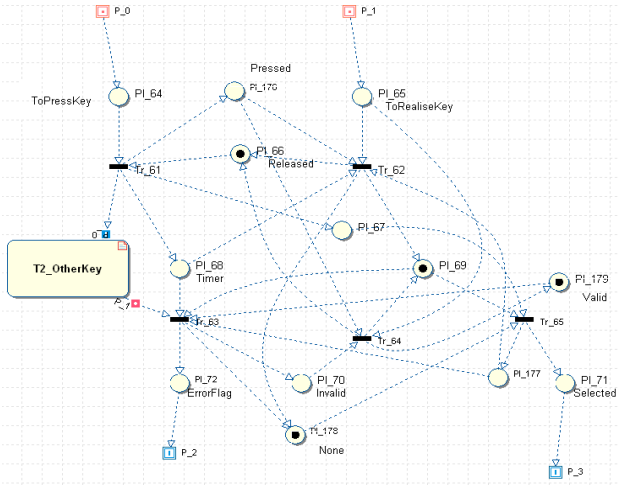

Figure 7: Valid Key Action Model in HiLeS

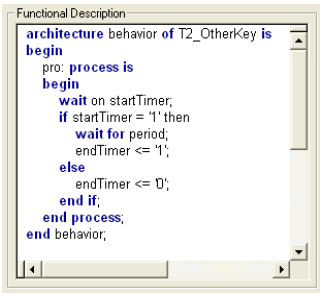

Figure 8: "T2_OtherKey" Functional Block description on VHDL-AMS.

To evaluate the behavior of the model, Table 2 is built with the cases that are tied to the requirements 4 and 5.

As an example, to verify the requirement 5 it is necessary to consider: if "Pressed" or "Valid" has a token, or if neither "Pressed" nor "Valid" has a token, then the place "None" must have a token. The formal verification is based on Petri net accessible markings search. The behavior of this requirement is verified by lines 0, 1, 4, 5 and 6 of Figure 9.

Finally, we can find others cases that are not taken into account in Table 2. For example, the lines 2, 9 and 10 show other cases that are summarized in one: "Pressed", "Valid" and "None" have tokens. This case is identified when the transition that changes the key status from "None" to "Selected" is not fired yet when there is a token in the places "Pressed" and "Valid".

Req	Pressed (P)	Valid (V)	None (N)	Selected (S)
4	1	1	0	1
5	0	0	1	0
	0	1	1	0
	1	0	1	0

Table 2: Cases to evaluate the requirements 4 and 5.

The formal verification can show that some requirements are missing; for instance, if some combinations of states are found but not initially defined. On the other hand, some requirement inconsistencies can also be shown during the modeling process.

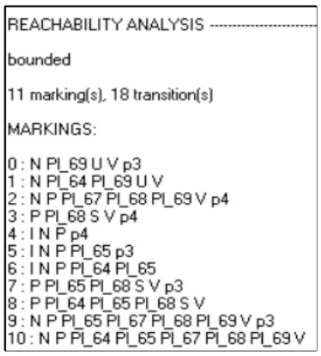

Figure 9: Accessible marking in the "Valid Key action" subsystem.

4.3 Verification by simulation

Following the verification process, the second step is to simulate the system model in order to verify the requirements which could not be verified by the formal analysis (operations sequences, temporal constraints, ...). HiLeS Designer is used to generate a virtual prototype in VHDL-AMS, transforming the system model to VHDL-AMS. This transformation is made automatically by HiLeS Designer. The tool generates the prototype and the test scenario source code in one step. The test scenario is generated according to the scenario defined in the HiLeS environment model and then SystemVision is used to simulate the virtual prototype; the results are shown in Figure 10. This scenario is related to specific requirements defined in the requirements document and the behavior is verified by the simulation.

The model formal verification helps to identify all the cases defined by the requirements.

Figure 10 shows the action to press and release a key in 2 seconds. The initial key state is "Released" and "Valid", as the Requirement 2 defined. In the first signal, the pressing key event is a pulse. At the same time, the position state changes from "Released" to "Pressed" (Requirement 3) and the status state continues being "Valid", following the

Requirement 4. Also, the key action state changes from “None” to “Selected”; this means the key pressed is selected. When the released event is done after 2 seconds (second signal), the valid position changes from “Released” to “Pressed”, and the key action from “Selected” to “None”.

The result of this simulation satisfies the system requirements. Finishing the verification process, we can conclude our model agrees to the textual requirements because all of them are correctly considered.

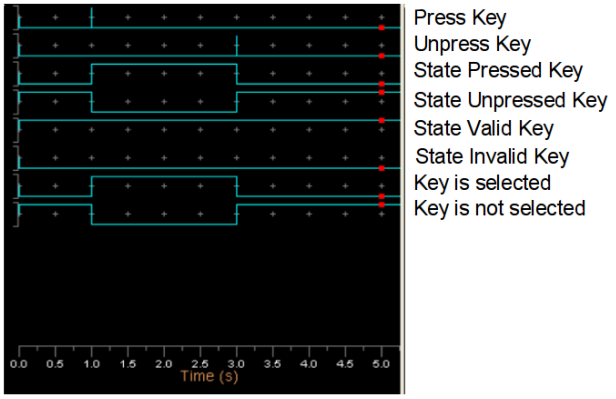

Figure 10: “Valid Key Action” Subsystem simulation based on a scenario.

5. Conclusions and future work

HiLeS Designer is a tool that follows the conception of new methodologies in ESL. We explained the formalism that this tool uses, its features and an Airbus system study case developed on HiLeS Designer. We developed an executable and verifiable model starting from the Airbus' textual requirements until we arrived to an executable virtual prototype in VHDL-AMS. We showed the practical HiLeS Designer use on verifying formally the logical structure and verifying by simulation the temporal constraints, behavior and logical sequence against each textual requirements of the SD Selector System. We also presented the possible risks (deadlock, non-liveness, non-boundedness) that a logical structure model can have if a formal verification is not applied.

In the future, we will work in the transformation from standard language such as UML and SysML to HiLeS in order to verify and to execute models described in these languages. We will use the MDA transformation concepts to achieve this objective.

6. References

[Martin07] G. Martin, B. Bailey, A. Piziali: “*ESL Design and Verification: A Prescription for Electronic System Level Methodology*”, Morgan Kaufmann, 2007.

[IEEE05] IEEE Computer Society: “*IEEE Standard SystemC Language Reference Manual*”, IEEE Computer Society, 2006.

[Fujita01] M. Fujita, H. Nakamura: "*The standard SpecC language*", System Synthesis. Proceedings. The 14th International Symposium on, pp. 81-86, 2001.

[Vanderperren08] Y. Vanderperren, W. Mueller, W. Dehaene: “*UML for electronic systems design: a comprehensive overview*”, Design Automation for Embedded Systems, Springer, Vol. 12, Num. 4, pages 261-292.

[Friedenthal08] S. Friedenthal, A. Moore, R. Steiner: “*A Practical Guide to SysML”*, Morgan Kaufmann Publishers, 2008.

[Boulet03] P. Boulet, J-L Dekeyser, C. Dumoulin, and P. Marquet: *“MDA for SoC embedded systems design, intensive signal processing experiment”,* SIVOES-MDA workshop at UML (San Francisco, USA), 2003.

[Liu01] J. Liu, ”Responsible Frameworks for Heterogeneous Modeling and Design of Embedded Systems”, Ph.D. thesis, University of California, Berkeley, 2001.

[Schorcht03] G. Schorcht, I. Troxel, K. Farhangian, P. Unger, D. Zinn, C. Mick, A. George, and H. Salzwedel, “*System-level simulation modeling with MLDesigner*,” MASCOTS, pp. 207–212, Oct. 2003.

[Ptolemy09] Ptolemy Group: “Ptolemy II”, http://ptolemy.berkeley.edu/ptolemyII/, 2009.

[Calvez93] J. Calvez “*Real-Time Systems a Specification and Design Methodology”,* John Wiley & Sons, 1993.

[MD03] M. D. Inc., “*VisualSim datasheet*”, http://www.mirabilisdesign.com/Pages/Product/mdi products.htm, 2003.

[Jimenez00] F. Jimenez: “*Spécification et conception de microsystèmes basés sur des circuits asynchrones*”, INSA and Universidad de los Andes PhD thesis, Toulouse, 2000.

[Hamon05] JC. Hamon: “*Méthodes et outils de la conception amont pour les systèmes et les microsystèmes*”, INP PhD thesis, Toulouse, 2005.

[Berthomieu06] B. Berthomieu, F. Vernadat “*Time Petri Nets Analysis with TINA*”, QEST 2006, (Riverside, USA), 2006.

[Mentor09] Mentor Graphics: “*SystemVision*”, http://www.mentor.com, 2009.

[Mathworks09] Mathworks: “Matlab”, http://www.mathworks.com/, 2009.

[Fritzson98] P. Fritzson, V. Engelson: “Modelica - A unified object-oriented language for system modeling and simulation”, ECOOP'98, (Brussels, Belgium), 1998.