Multithreading-based Coverification Technique of HW/SW Systems

Mostafa Azizi, El Mostapha Aboulhamid
Département d'Informatique et de Recherche Opérationnelle
Université de Montréal
Montreal, Qc, Canada

Sofiène Tahar
Electrical and Computer Engineering
Concordia University
Montreal, Qc, Canada

Keywords : HW/SW Systems, Specification, HW/SW Integration, Cosimulation, Coverification, Multithreading.

Abstract

An embedded system consists of a set of heterogeneous components. These components may be software or hardware blocks with a limited specification. A global verification of such systems to check the equivalence between their implementation and their specification is challenging. Number of tools such as Seamless-CVE and Eaglei are now available. These tools perform the coverification of HW/SW systems by cosimulation. This latter simulates concurrently the SW and HW components described respectively by HLL (High Level Language) and HDL (Hardware Description Language) codes. To improve the coverification process, we propose a methodology based on principles of multithreading and guided simulation. In this approach the software subsystem appears as a set of intercommunicating threads. Concurrent threads implement the communication with the hardware subsystem. Properties that should be verified can be added dynamically by creating new threads allowing their concurrent evaluation. The advantage of our methodology is that if any property is not satisfied, the result of simulation is known negative even before the computation ends, and an appropriate action may be undertaken. Furthermore, the degree and the amount of properties can be modified in a flexible way by changing the priority or the execution of some threads.

1. Introduction

The progress of microelectronics led to the development of very complex digital systems which verification and test are very difficult. During the design process, different architectures are explored in order to meet the specifications, targeted performances and cost. In this paper, we focus on an increasingly important class of designs composed of mixed HW/SW systems. Previously, hardware and software subsystems were designed separately, and problems rose at a late stage of the design when the two subsystems were integrated. This resulted in an unacceptable iterative process in terms of time-to-market and cost. To overcome these difficulties, an earlier integration of HW and SW parts is required and the coverification process has to be performed in concurrence with the co-design process. One could describe both SW and HW parts using an HDL and then use a formal verification tool to check the system correctness. However, the description of the SW part in this case is very limited and numerous processing aspects cannot be specified. For example, the VIS tool [1] accepts only a subset of Verilog and in spite of its extension by non-determinism task and symbolic variables, the different SW aspects cannot be completely represented using VIS-Verilog. Consequently, the verification of HW/SW systems will be incomplete. Hence, efficient techniques of describing and verifying this kind of systems are seriously needed.

We present in this paper a coverification technique based on the multithreading concept [2]. In this approach, the software part is described as a set of communicating threads, the interaction between hardware and software is clearly defined, additional threads are used to express specification requirements and assertions. Finally, the cosimulation of the whole code is run as a unified model using a Java compiler, or as a distributed model using a coverification environment such as CVE-Seamless [3] or Eaglei [4].

The rest of this paper is organized as follows: Section 2 reviews previous works related to coverification. Section 3 presents our coverification technique based on multithreading. An illustration example of this technique is proposed in Section 4. In Section 5, we discuss the implementation of

experimental results. Finally, Section 6 concludes the paper.

2. Related work

The coverification techniques can be divided in three categories, namely *bread-boarding-based*, *cosimulation-based* and *formal* techniques. Bread-boarding-based coverification technique (Figure 1) consists of testing a prototype to check if the requirements are fulfilled or not. This is done once both software and hardware parts are completed and the software is running on the hardware board. Cosimulation-based technique (Figure 2) simulates software and hardware parts and ensures a communication bridge between them, called virtual integration. In the case of a unified model [5], only one simulator is used to perform cosimulation but when dealing with a distributed model [6, 7, 8, 9, 10, 11], two or several simulators are required depending on how many languages are used to describe software and hardware parts of the embedded systems. Nowadays, there exist some cosimulation environments that support several simulators such as POPE [12]. By cosimulation, the integration of software and hardware starts long before the phase of prototyping, and HW/SW system requirements are verified by sequential or distributed simulation [13]. In sequel, formal coverification techniques (Figure 3) try to use formal verification methods, such as model checking, for an unified model of the HW/SW system [14, 15]. Other related papers discuss some coverification case studies such as the coverification of DPLL (Digital Phase Locked Loop) at NORTEL [16], and HW/SW coverification performance estimation of a 24 embedded RISC core design at SIEMENS [17].

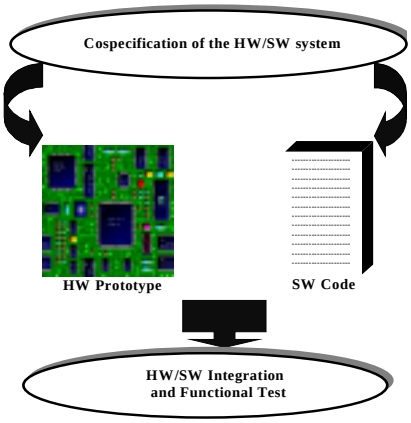


Figure 1: Bread-boarding Technique

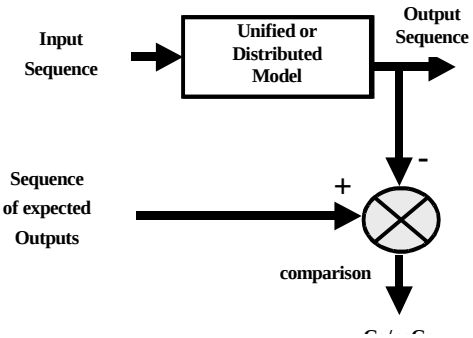


Figure 2: Cosimulation Technique

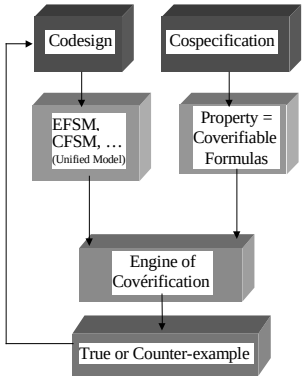


Figure 3: Formal Coverification Technique

3. Multithreading-based Coverification technique

The codesign of embedded system starts by partitioning the system into SW and HW subsystems. SW design team performs the design of the SW part and HW design team those of HW part. In parallel, an interface insuring communication between SW and HW is synthesized. When the design of all parts is approximately completed, designers integrate them to obtain the implementation of the whole system. Does this implementation satisfy the specification? In other words, are the requirements of the system satisfied by this implementation? To answer these questions, we have to perform cosimulation with a number of test cases. Therefore, the coverification of hardware/software systems consists of checking concurrently if the software implementation

satisfies the software specification, if the hardware description satisfies the hardware specification, and if the hardware/software integration respects the requirements of the global specification. Figures 5 and 6 depict the typical architecture of the systems we are interested by. The software part contains the original SW-code of the system and some data. The SW-code encompasses memory addressing, hardware interfacing, and some functions decided to be implemented by the software (after partitioning). The hardware part is a set of circuit modules, registers, IP blocks (blocks with Intellectual Properties), etc. The interface between the hardware and software parts is built around a processor that runs the software codes and manages the hardware signals. This kind of systems challenges the current methods of verification and simulation, due to their complexity and their heterogeneity. Our technique is based on a cosimulation process and on some aspects of formal verification. This mixture of simulation and formal verification philosophies is informally known as semi-formal verification [18] and it is stated as an intermediate level between their abstraction levels.

Multithreading-based Coverification technique is a coverification approach that intends to improve and accelerate coverification process by using the multithreading concept and guiding the cosimulation by checking some system properties. This technique is based on four steps: (1) thread the software part, (2) make all interesting hardware signals observable in registers, (3) manage system properties in threads, and (4) cosimulate the system added by its properties. Each of these steps is detailed in what follows and are ordered as depicted on Figure 10.

Step 1*: Threading the software part* (Figure 6)

As the HW is described by modules in Verilog or entities in VHDL which execution is parallel and event-driven, we similarly parallelize the SW by using the thread concept. We organize carefully the SW part as many threads as possible. In the worse case, the entire SW part is considered as only one thread. The use of threads increases the degree of visibility of the software code and makes its handling easy in order to extend it, to locate and correct eventual errors.

Step 2*: Making all interesting hardware signals observable in registers* (Figure 7)

We consider this step in the objective to make visible to the SW part the HW parameters on which some properties are dependent. In fact, properties to be checked are defined on the software side and some of them are expressed as a function of the states of some hardware entities. Those states are stored in specific registers that are accessible from the SW part. We call them Registers For Verification (RFVs)). In some cases, these registers belong to the original design of the hardware part; but when there is a lack of registers we consider some additional registers located in extra memory in order to follow states of some signals in the hardware part.

Step 3*: Managing system properties in threads* (Figure 8)

Without making any change to the original software (software of the embedded system as it is specified), we augment it by additional threads to accomplish the Read/Write accesses of registers and checking of properties. One specific thread is responsible for accessing any register. Concurrent properties are described each by one thread with the same priority for all. Sequential properties are gathered entirely or partially in one thread. We can also handle some macro-properties expressed in function of other properties. In fact, this step makes an explicit concurrency of system properties.

Step 4*: Cosimulating the system augmented by its properties* (Figure 9)

We regroup the original software and the threads resulting from Step 3. The original hardware is eventually augmented by some registers as explained in Step 2. Thereafter, we load the software and hardware parts and we accomplish their cosimulation using an efficient coverification environment.

4. Illustration example: a Fuzzy Controller

A Fuzzy controller is a component of the control loop of an automation system. Its core executes a code based on fuzzy logic. This controller is a software/hardware system as depicted by Figure 11. The software part is the fuzzy algorithm that generates specific actions regarding the measures applied by sensors to the input of the controller. In our case, each sub-module SW_i receives temperature, pressure and moisture gap values and generates the corresponding electric current value. The hardware part (Figure 12) consists of register arrays necessary to store measures and orders of a human/machine operator, and the gaps between them (i.e. measures and orders).

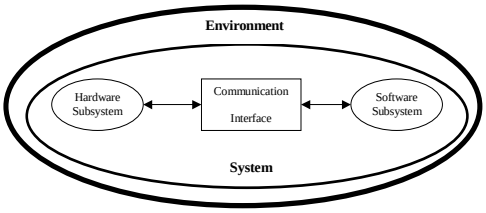


Figure 4: Behavioral Architecture of HW/SW systems

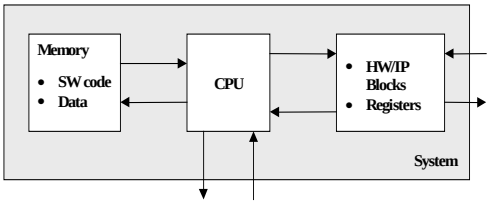
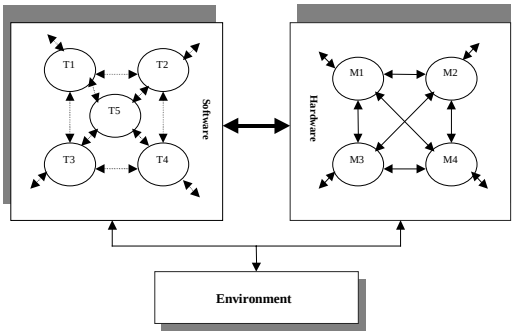


Figure 5: Structural Architecture of HW/SW systems



Ti : thread , Mi : module

Figure 6: Organization of HLL and HDL codes

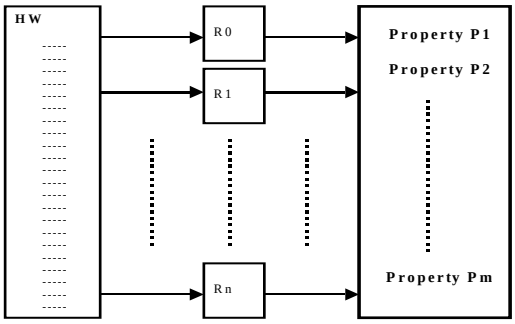


Figure 7: Auxiliary Registers for Verification

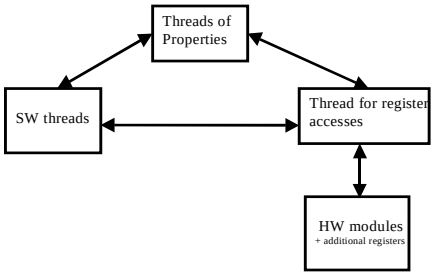


Figure 8: SW augmented by some threads

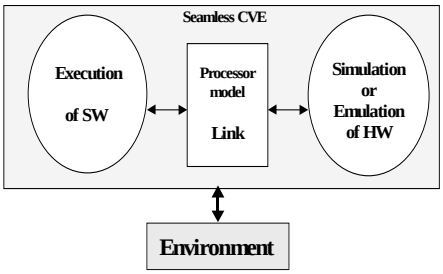


Figure 9: Behavioral View of Cosimulation

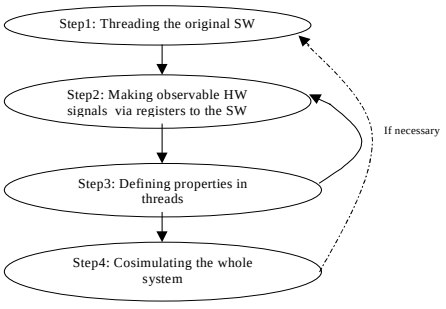


Figure 10: Multithreading-based coverification Steps

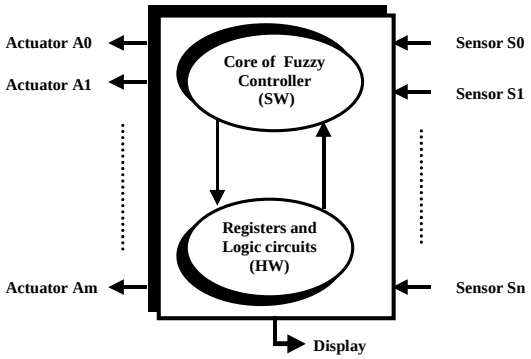


Figure 11: Fuzzy Controller

5. Implementation and results

5. 1. Specification of properties

The specification requirements are expressed as a set of properties. Each property is described using a HLL code so that it could be validated concurrently with the remaining properties. Property parameters might belong to the HW part (HW properties), to the SW part (SW properties), or to both of them (HW/SW mixed properties). For each test vector applied to the system model, all the properties are checked whether they are fulfilled or not (valid or invalid). Some of the properties related to the example of section 4 are presented in what follows.

Example of properties

P1- Each measure must be updated every Tc period
P2- Each action must be updated every Ta period
P3- Always ***not*** *(high_current* ***and*** *very_hot)*

Property expressions in HLL

```
/*  Thread Body of P1 */
{…
while Tc_spent
                    { read_measures;
                      test_update_measures;}
…}
/*  Thread Body of P2 */
{…
while Ta_spent
                    { read_actions;
                      test_update_actions;}
…}
/*  Thread Body of P3 */
{…
 assertion1=not (high_current and very_hot);
test_assertion (assertion1);
… }
```

In general, properties are classified as concurrent or sequential depending on their semantics and their parameters. For each concurrent property, we assign a corresponding thread that performs its checking concurrently with the others. Threads associated to concurrent properties are set to the same priority. In some cases, however, when some simulation goals are considered, we can attribute for each thread carrier of a concurrent property its appropriate priority. Sequential properties, all or a subset of them, are handled just by one thread which priority is defined depending on the rest of properties.

5.2. Code structure

The HW/SW code is organized as depicted on Figure 13. The extended SW part is modeled by using Java while the HW part in Verilog/VHDL. We have used a unified model for the whole system in order to perform functional simulation. In this case, the HW part is seen as a set of registers that eventually can be read or written. The case of a distributed model using an heterogeneous environment of cosimulation would be discussed in our future work. In the example presented Section 4, the SW part is divided into a number of independent sub-modules SW_i described by individual thread. The HW part is controlled by other threads. The HW/SW properties are managed in threads with respect to their dependencies.

5.3. Simulation and Test

To perform the simulation of our controller system, test vectors are generated randomly at the end of each predefined period Tc and consequently each property is checked to be valid (1) or invalid (0). More than hundred test vectors have been applied on the properties mentioned in the subsection 5.1 (P1, P2 and P3), as depicted on Figure 14 ((a), (b) and (c)).

6. Conclusions

In this paper, we have presented a coverification technique based on the multithreading concept and using cosimulation technology. This technique allows a good structuring of the software code and a particular flexibility to define and to control hardware/software properties. The handling of these properties is made easy and all of them are tested in the software part, even those related to the hardware part. Also, by applying this technique, we can guide the cosimulation process and elaborate fault tolerance cosimulation. This technique ensures an important speed up of the coverification task if the machine host is a multiprocessor system with shared memory, due to the fact that the performances of concurrent programming are not efficient with sequential machines (Von Newmann's model). Our coverification technique of HW/SW system is relevant to the cosimulation process. We believe that it forms an important contribution in the HW/SW coverification domain in the sense that currently there are no efficient formal coverification methods specific to this kind of systems.

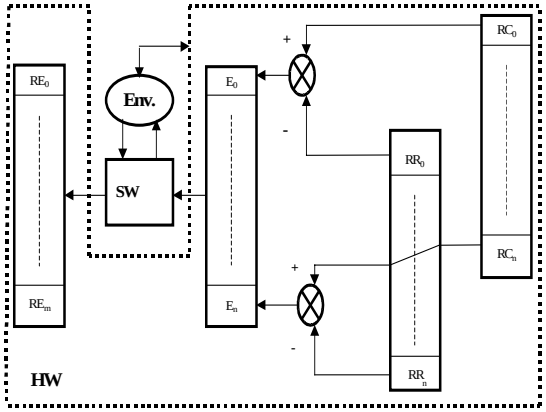


Figure 12: Description of HW part

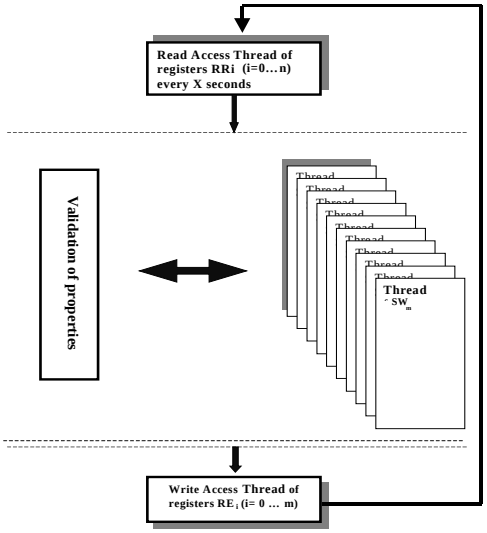
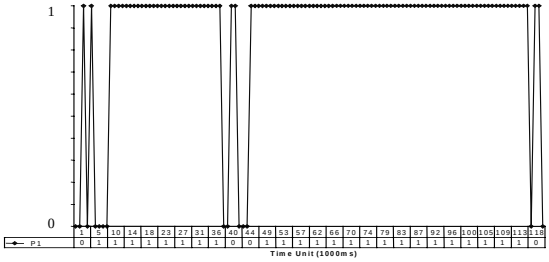
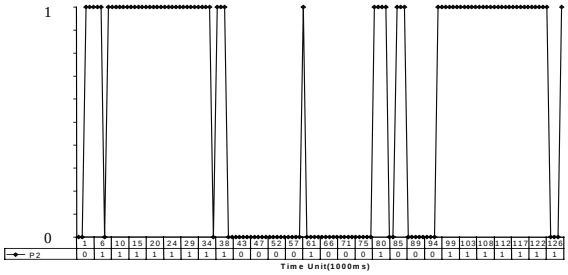


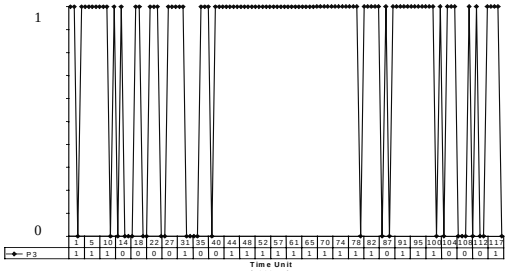
Figure 13: Global View of Extended SW



(a) : Checking of property P1 vs different test vectors



(b) : Checking of property P2 vs different test vectors



(c) : Checking of property P3 vs different test vectors

Figure 14: Checking of Properties vs Test Vectors

References

[1]. R. Brayton et al., "VIS: A system for verification and synthesis", Technical report UCB/ERL M95, Electronics Research Laboratory, University of California, Berkeley, December 1995

[2]. A. Burns and G. Davies, "Concurrent Programming", International Computer Science Series, Addison-Wesley, 1993

[3]. Mentor Graphics Corporation, "CVE-Seamless", http://www. meng.com/, 1998

[4]. View Logic, "Eaglei", http://www.Viewlogic .com/, 1998

[5]. R. K. Gupta, C.N. Coelho Jr. and G. De Mecheli, "Synthesis and Simulation of Digital Systems containing interacting hardware and software components", 29th Design Automation Conference (DAC'92), June 1992

[6]. D. Becker, R. K. Singh and S. G. Tell, "An engineering environment for HW/SW cosimulation", 29th Design Automation Conference (DAC'92), June 1992

[7]. D.E. Thomas, J.K. Adams and H. Schmit, "A model and methodology for HW/SW codesign", IEEE Design and Test of Computers, September 1993

[8]. K. Ten Hagen and H. Meyer, "Timed and untimed HW/SW cosimulation : application and efficient implementation", International Workshop on HW/SW Codesign, October 1993

[9]. J. Rowson, "HW/SW cosimulation", 31st Design Automation Conference (DAC'94), San Diego, June 1994

[10]. J. Wilson, "HW/SW selected cycle solution", International Workshop on HW/SW Codesign, 1994

[11]. C. A. Valderrama, F. Nacabal, P. Paulain and A. A. Jerraya, "Automatic generation of interface for distributed C-VHDL cosimulation of embedded systems: an industrial experience", proceedings of the 7th IEEE International Workshop on Rapid systems prototyping, June 1996

[12]. H. De Man, I. Bolsens, B. Lin, K. Van Rompaey, S. Vercauteren and D. Verkest, "Codesign of DSP Systems", Kluwer Academic Publishers, 1997

[13]. W. D. Bishop and W. M. Loucks, "A heterogeneous environment for HW/SW cosimulation", Proceedings of the 30th Annual Simulation Symposium, pp. 14-22, Atlanta, April 1997

[14]. L. Lavagno, A. Sangiovanni-Vincentelli and H. Hsieh, "Emdedded Systems Co-Design : Synthesis and Verification", Kluwer Academic Publishers, 1997

[15]. R. Kurshan, V. Levin, M. Minea, D. Peled and H. Yenigun, "Verifying HW in its SW Context", IEEE/ACM International Conference on Computer Aided Design (ICCAD'97), San Jose, November 1997

[16]. E. Hunnell and M. Lyons, "Coverification Goes From Cutting Edge to Mainstream: DPLL Design Demonstrates the Viability of Today's Tools", Electronic Design, June 22, 1998

[17]. T. W. Albrecht, J. Notbauer and S. Rohringer, "HW/SW CoVerification Performance Estimation & Benchmark for a 24 Embedded RISC Core Design", Proceedings of Design Automation Conference (DAC'98), pp. 808-811, San Francisco (California), 1998

[18]. D.L. Dill, "What's between simulation and formal verification?", Design Automation Conference (DAC'98), San Francisco, June 1998