Proceedings of The 24th International Conference of Microelectronics
ICM 2012, December 17-20, 2012, Algiers, Algeria.

A Fast Hardware/Software Co-Verification Method using a real hardware acceleration

Mossaad Ben Ayed[1], Faouzi Bouchhima[2], Mohamed Abid[3]
National Engineering School of Sfax
University of Sfax, Tunisia
[1] mossaad_benayed@yahoo.fr, [2] f_bouchhima@yahoo.fr, [3] mohamed.abid@enis.rnu.tn

***Abstract*— Due to the number and the nature of components integrated in them, Systems-On–a-Chip (SoC) have be come increasingly complex. To solve the problem of cost, flexibility and the time-to-market, systems designed with mixed hardware software systems has increased and the verification method has become a key position of the design process. This paper describes a new hardware/software co-verification methodology for SoC, based on the integration of a SystemC simulator and an FPGA accelerator. Between the SystemC simulator [1] [2] and the FPGA board, a shared communication was established to accelerate the simulation via flexible interfaces. The key issue is the synchronization between the two parts.**

Keywords-component; Co-Verification; SystemC ; Transaction Level Modeling; Synchronization.

I. INTRODUCTION

Embedded systems are mostly heterogeneous devices. Their design is based on hardware and software components. These parts cannot be developed independently, since their interaction is a key point of the system behavior. Each part needs to be aware of the characteristics of other parts, in order to provide optimized components. The best strategy adopted is co-design, since it allows us to develop HW/SW component concurrently [3].

Co-simulation is a key methodology in co-design that allows verification of the hardware, the software, and their interaction. The essential aim of the co-simulation is to validate and to cover the performance as well as the functionality. The main problem appears when the system complexity grows and the validation becomes more and more time consuming. To overcome this challenge, and speed up HW/SW co-simulation, Transaction Level Modelling (TLM) is adopted. This paper is organized as follows. Section 2 summarizes existing work on HW/SW co-simulation. Section 3 explains the proposed solution to accelerate the co-simulation. Section 4 and 5 present a detailed description of the communication model and the synchronization model between SystemC and the FPGA platform. Experimental results are discussed in section 6. Finally, concluding remarks are given in section 7.

II. RELATED WORK

Several simulation frameworks have been proposed in the literature. They can be classified into two main categories: homogenous and heterogeneous.

Homogenous frameworks use a single simulator for the simulation of both hardware (HW) and software (SW) components. The main advantage of this category is the simplification of the design modeling and the good simulation performance. However, homogenous frameworks, usually based on extended existing languages suffer from lack of libraries and synthesis tools and they are suitable only in a very initial phase of the design, prior to HW/SW partionning.

Inversely, heterogeneous frameworks, which are based on integrating existing simulators (using co-simulation), warrant a more accurate tuning between HW/SW components and benefit from the existing libraries and tools. The major problem in this category is the communication and synchronization models between the different simulators.

Several frameworks [4] [5] [6] are mainly focused on Multilanguage system description, that is, a HDL for hardware and a programming language for software. All these heterogeneous co-simulations are based on solving the problems of controlling and synchronizing several simulation engines. These frameworks are adopted because of the best simulation performance and the easiest integration but it was the only possible choice when VHDL or Verilog simulation was the highest possible level of abstraction for simulating hardware.

The advantage of design with SystemC [7] [8] [9] is the use of the bus at different abstraction layer to obtain more efficient co-simulation. HW and SW are described by using C. This approach simplifies the implementation of the initial model as well as the HW / SW partitioning. In fact, HW components are simulated by using the SystemC simulation kernel, while SW programs run on an Instruction Set Simulator (ISS). Thus, more accurate performance estimation could be obtained.

The frameworks based on the last approach use two essential steps. The first is the Inter Process Communication (IPC). It is used to make the communication between the ISS and the SystemC simulator. The second is the Bus Wrapper. It ensures synchronization between SystemC simulation and the ISS.

These frameworks still suffer from some performance bottlenecks, caused by the use of the ISS. However, ISS gives the best simulation accuracy. To accelerate simulation, in spite of the accuracy, the native SW simulation is adopted using SystemC and time annotations. Some works try to improve the performance estimation accuracy in native simulation by

modeling and simulating the OS behavior essentially the interruptions and preemption mechanism [10].

Other works [15] is based on using multi-ISS to accelerate the simulation. But this framework suffers from a complex synchronization scheme that increases the overhead.

Our solution replaces the ISS by real processor. The main advantages are:

- Our framework gives a high speed SW validation without any lost of accuracy since SW will be executed by the target microprocessor.
- The hardware part will be described and simulated using SystemC.

So, this framework represents a very useful platform for software engineers to validate their code before the hardware components become available.

III. CONVENTIONAL APPROACHES

As known, if all modeled blocks are implemented in hardware emulation, the system cost, as well as the running and debugging cost, will become expensive. Therefore, a combined method using an emulator and a simulator is the most adopted to model SOCs.

To increase the verification speed while maintaining clock accuracy, an FPGA type ALTERA DE2-70 is used. Thus the verification framework uses SystemC simulator to simulate the hardware components and the NIOS II processor to execute the software applications. The main idea is based on the replacement of the ISS by a real processor, which complicates the synchronization task between the HW and SW. Two issues are essential: the communication and the synchronization models. The next section describes the communication model.

IV. COMMUNICATION MODEL

This section gives a brief introduction to the communication model. A USB link is used in the communication between PC and FPGA because this kind of communication has better speed than PCI which it adopted in emulation [11]. This communication is based on packets which are constructed by the communication interface between simulator and emulator.

Two forms of exchanged packets are used to perform the synchronization scheme between the simulator and the emulator, figure 1.

Interruption packet is the first form. It consists of two parts: a header and a body. The last one contains the routine number and the interruption time stamp. The header of this form presents the type of synchronization and the routine number indicates the routine task to be executed. The time stamp represents a synchronization point and it is used to execute the interrupt routine at the appropriate instant.

Data packet is the second form. It comprises a header and the data. The header in this case contains the synchronization type, the size of data to send and the time stamp to synchronize when it is necessary.

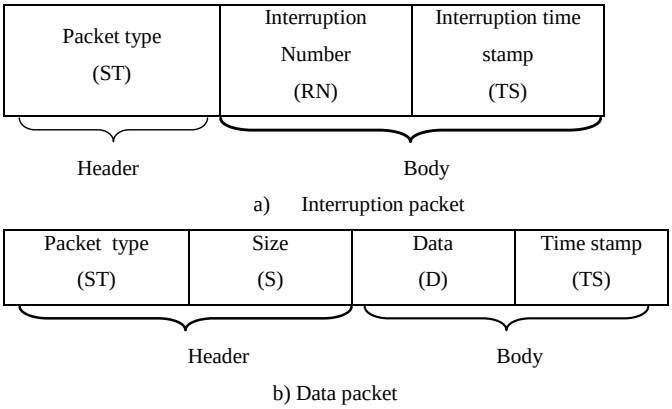

Figure 1: Synchronization forms

Note that any packet received by the NIOS II side generates an USB interruption that can be exploit in the implementation phase to interrupt the NIOS processor each time a packet is received.

V. SYNCHRONIZATION MODEL

A key issue of the proposed verification framework is the time synchronization between the SystemC simulator and the NIOS II processor emulated on the FPGA board. The verification method is based on the following synchronization schemes which respect the interaction style that can be involved between HW and SW components. Note that, in the same design, HW and SW components may use different synchronization schemes

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

This scheme is based on FIFO memory between SW task and HW task. The main idea consists on fixed synchronization time between simulator and emulator (see figure 2). Because of the difference of speed, the HW imposes a synchronization Time (T_{sync}). This T_{sync} must be more than HW or SW tasks time.

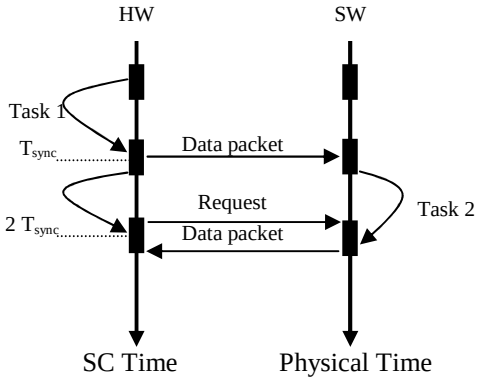

Figure 2: Synchronization model: scheme 1

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

When a hardware component is simulated by SystemC, the SW task uses a waiting loop for data (see figure 3). Once the

hardware task (task1) is finished, the simulator sends data to the SW task and a switch context from SystemC to board is taken. At this time, the SW task receives data and resumes the execution. Here, 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 (see figure 3). Note that the SystemC and the emulator need to usually exchange information about the time.

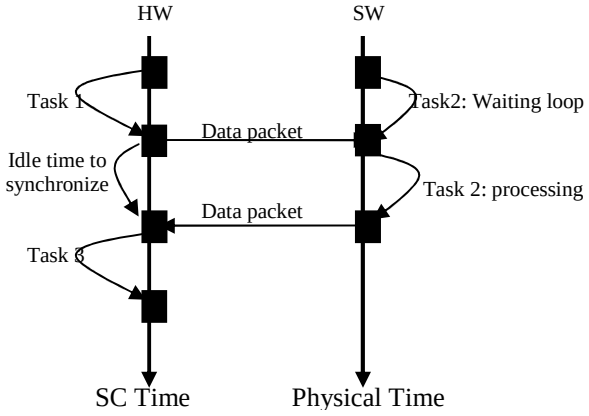

Figure 3 : Synchronization model: scheme 2

- **Scheme 3: The SW task receives an interruption to indicate the end of the hardware task**

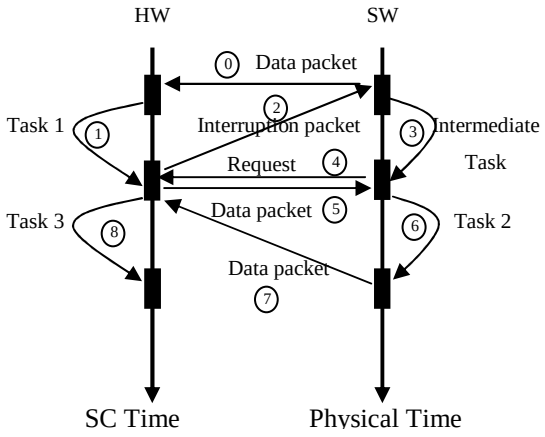

Figure 4 : Synchronization model: scheme 3

This scheme is illustrated by the figure 4. In this case, the software does not use a waiting loop but the end of the task is indicated by interruption, so the software can execute the task instead of waiting. The Simulation scheduler, running on NIOS II, sends data to the simulation interfaces (arrow 0), which activates the hardware task1. At the end of task1 process, and before sending data to SW task, the *wait_for_interrupt(sc_time)* function is called (see figure 5), so the simulator advances its time (arrow 1) and sends an interrupt packet to inform emulator for the next time stamp (arrow 2). At this time, the simulation scheduler activates a NIOS II timer with a period that coincides with the received interruption time stamp and begins the execution of an intermediate task (an eventual user background task). When the timer is reached, it interrupts the background task. Thus the simulation scheduler activates the task 2 (the number of the interruption is received with the interrupt packet). The last one may request data, thus the task 1 resumes execution and sends data packet (arrow 4), which activates task 2. Figure 6 shows the template of the code.

```
void wait_for_interrupt(sc_time t)
{
wait(t);
send_interruption_packet(…..) ;
}
```

Figure 5 : Wait_for_interrupt code

Where t is an estimation of the task1 duration

```
            /* Task1 code */
Instructions
….
….
Wait_for_interrupt (t);
Switch_context();/* switch context to SC*/
```

Figure 6 : Template of synchronization code

- **Scheme 4: The SW task may receives a random interruption resulting from externally data reception**

This scheme is illustrated by the figure 7. The SystemC begins the execution of the task 1 and, when finished, sends a data packet to the SW task. The task 2 starts and the SystemC executes the *Hardware_Input_Interface* : a process that models the input interfaces of the hardware subsystem (its execution do not advances the SystemC local time). The process may generate a random interrupt packet which informs of the reception of a new data. The sent packet via USB generates an USB interruption which will interrupt the task 2. Thus, the USB interruption plays the same role as the hardware interruption. Once the interruption is occurred, we need only to know, thanks to the received interrupt packet, the interruption routine to execute (here is task 3).

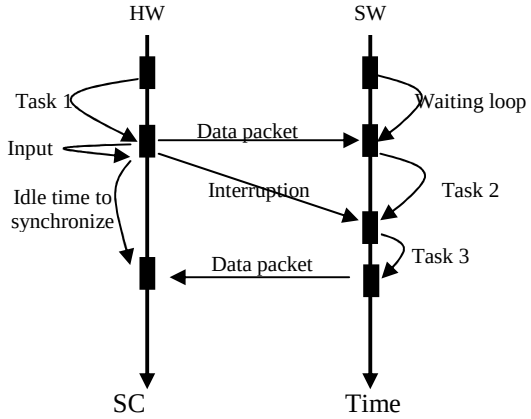

Figure 7 : Synchronization model: scheme 4

To ensure communication and to save synchronization context, an array of shared registers is used.

VI. EXPERIMENTAL RESULTS

In this section, we propose a fingerprint recognition system [14] to validate our co-verification framework. At first, a communication model is implemented. The model is divided into receiving/ transmitting drivers. Figure 10 shows the different components of the model.

- SW driver is made by Windows Driver Kit (WDK). This driver contains two main functions Read and Write.

- Channel: the USB communication offers a transaction speed of 480 Mbits/s; unlike PCI which offers only 133 Mbits/s [12] [13].

- HW driver is based on Philips ISP1362 controller.

Then, Based on the native execution of the fingerprint recognition on a 2 GB RAM, 1.66 GHz Intel Core 2 Duo processor with Windows XP operating system, we notice that the time execution of the minutia extraction is the minimum. We divided our system on hardware components and software applications based on time execution (see figure 8).

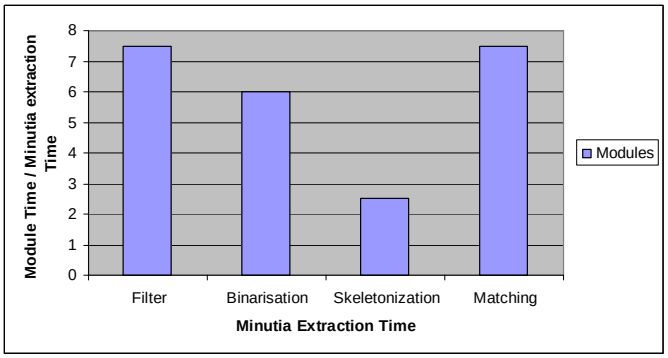

Figure 8: Time execution of each modules as a function of minutia extraction time

The co-verification time of each module is given in table 1.

TABLE I : SIMULATION TIME

	Module	Time (s)
HW Components	Read of fingerprint	**0.03**
	Filter	
	Binarization	
	Matching	
Interface	Interface	**0.5**
SW Applications	Skeletonization	**0.01**
	Minutia extraction	
All Modules		**0.54**

Result shows that our simulation/emulation environment reduces the time simulation.

VII. CONCLUSION

A methodology to perform early design stage validation of hardware/software systems is proposed in the paper. The co-verification framework is based on synchronization between SystemC simulator and FPGA board emulator. The main idea is to accelerate the simulation by replacing the ISS with NIOS II as a real processor. Experiments with a real example proved the effectiveness of the proposed framework. We need others type of examples to validate all synchronization aspects. As a future work, the adding of OS support by the software part will be considered.

VIII. REFERENCES

[1] OSCI; "Functional Specification for SystemC 2.0", available at www.systemc.org

[2] T. Grotker, S Liao, G. Martin, S Swan; "System Design with SystemC"; Kluer Academic Publishers, ISBN 1-4020-7072-1

[3] D. Micheli, D.Ernst, R. Wolf, W. Eds. Readings in Hardware/Software Co-design, Morgan Kaufmann, 2001

[4] Liem C., Nacabal F., Valderrama C., Paulin P., And Jerraya A.. System-on-chip cosimulation and compilation. IEEE Design and Test of Comput. 14, 2, 16–25.1997.

[5] Valderrama C., Nacabal F., Paulin P., And Jerraya A. Automatic VHDL-C interface generation for distributed cosimulation: Application to large design examples. Design Autom. Embed. Syst. 3, 2/3, 199–217.1998.

[6] Coste, P., Hessel, F., Marrec, P. L., Sugar, Z., Romdhani, M., Suescun, R., Zergainoh, N., AND Jerraya, A. Multilanguage design of heterogeneous systems. In Proceedings of IEEE International Workshop on Hardware-Software Codesign. 54–58.1999.

[7] Liu, J., Lajolo, M., AND, A. Software timing analysis using HW/SW cosimulation and instruction set simulator. In Proceedings of the IEEE International Workshop on Hardware/Software Co-design. 65–69.1998.

[8] Fummi, F., Martini, S., Perbellini, G., AND Poncino, M. Native ISS-SystemC integration for the cosimulation of multi-processors SoC. In Proceedings of the IEEE Conference on Design Automation and Test in Europe. 564–569. 2004.

[9] Moussa, I., Grellier, T., AND Nguyen, G. Exploring SW performance using SoC transactionlevel modelling. In Proceedings of the IEEE Conference on Design Automation and Test in Europe. 120–125. 2003.

[10] Bouchhima, A. Yoo, S. Jarraya A.., "Fast and accurate timed execution of high level embedded software using HW/SW interface simulation model", Design Automation Conference: ASP-DAC, pp. 469 – 474, 2004.

[11] Soha Hassoun, Senior Member, IEEE, Murali Kudlugi, Duaine Pryor, and Charles Selvidge "A Transaction-Based Unified Architecture for Simulation and Emulation" IEEE transactions on very large scale integration (VLSI) systems, vol. 13, no 2, 2005.

[12] Jan Exalson, "USB COMPLETE Everything You Need to Develop Custom USB Peripherals" book, third edition, 2005

[13] ISP1362 Embedded Programming Guide Version 9 June 2002

[14] Mossaad Ben Ayed, Faouzi Bouchhima and Mohamed Abid, "Automated Fingerprint Recognition Using the DECOC Classifier", International Journal of Computer Information Systems and Industrial Management Applications. ISSN 2150-7988 Volume 4 pp. 546-553. 2012.

[15] S. Cordibella, F. Fummi, G. Perbellini, D. Quaglia, "A HW/SW Co-Simulation Framework for the Verification of Multi-CPU systems", IEEE transactions, 2008