Original scientific paper

Informacije MIDEM
Journal of Microelectronics,
Electronic Components and Materials
Vol. 43, No. 3 (2013), 162 – 172

A fast simulation emulation engine

Mossaad Ben Ayed, Faouzi Bouchhima, Mohamed Abid

National Engineering School of Sfax, University of Sfax, Tunisia

Abstract: Due to the number of functionalities built on the system development, a mixed hardware software systems has increased. To solve the problem of the cost, the flexibility, the time-to-market and the resulting behaviors, many verification methods are used in the literature. This paper describes a new hardware/software co-verification method for System-On–a-Chip, based on the integration of a SystemC simulator and an FPGA accelerator. Between the SystemC simulator [1,2] and the FPGA hardware platform, a simulation / emulation engine based on different synchronization scheme was established to accelerate the verification and to control the context switch. This work presents an extension for CODIS tool and it enables not only an easy and a high verification speed, with a low cost, but also it presents the real behavior of hardware / software.

Keywords: Simulation; Emulation; SystemC ; Transaction Level Modeling; FPGA; Synchronization.

Hitro simulacijsko emulacijsko orodje

Izvleček: Z rastjo števila vgrajenih uporabnosti v sisteme naraščajo sistemi z mešano programsko in strojno opremo. Številne verifikacijske metode so uporabljene v literaturi za reševanje problemov stroškov, fleksibilnosti, časa do predstavitvena trgu in rezultirajočega obnašanja. V članku je predstavljena nova programsko/strojna metoda za verifikacijo sistema na čipu, ki temelji na integraciji SystemC simulatorja in FPGA pospeševalnika. Med simulatorjem SystemC [1,2] in FPGA platformo je postavljeno simulacijsko/emulacijsko orodje, ki temelji na različnih sinhronizacijskih shemah in pospešuje verifikacijo in nadzor nad vsebinskim stikalom. Delo predstavlja razširitev CODIS orodja in ne predstavlja le enostavno in zelo hitro verifikacijo temveč dejansko obnašanje strojne in programske opreme.

Ključne besede: simulacija, emulacija, SystemC, nivojsko modeliranje, FPGA, sinhronizacija

** Corresponding Author's e-mail: mossaad_benayed@yahoo.fr*

1 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 the 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, many synchronization schemes are used.

As mentioned bellow, this paper presents an addition work for CODIS platform. CODIS (COntinuous DIscrete Simulation) [18] is a tool which can automatically produces co-simulation instances for continuous/discrete systems simulation using SystemC and Simulink simulators. This is done by generating and providing co-simulation interfaces and the co-simulation bus. To evaluate the performances of simulation models generated in CODIS, they measured the overhead given by the simulation interfaces.

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 presents the Simulation Emulation engine and his architecture. Finally, experimental results are discussed in section 5, and concluding remarks in section 6.

2 Related work

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

Homogenous frameworks [22] [23] use a single simulator for the simulation of both HW/SW components. The main advantage of this category is the simplification of the design modeling and the good simulation performance. However, homogenous frameworks suffer from the huge time consummation and they are suitable only in a very initial phase of the design, prior to HW/SW partitioning.

Inversely, heterogeneous frameworks [19] [24] [25] warrant a more accurate tuning between HW/SW components and save much time in simulation. The major problem in this category is the communication and synchronization.

Several frameworks [4] [5] [6] are mainly focused on Multilanguage system description, that is, an HDL for hardware and a programming language for software.

All these heterogeneous co-simulations are based on solving the problems of controlling and synchronization 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. To surmount this challenge, SystemC is more adopted in hardware description. The advantage of design with SystemC [7] [8] [9] is the use of the bus as different abstraction layer to obtain more efficient co-simulation and the HW/SW process are described on 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) [20] [21].

These frameworks are based on two essential steps. The first is the Inter Process Communication (IPC) [26] [27]. 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 simulator and the ISS.

But these frameworks still suffer from some performance bottlenecks, caused by the use of the ISS. However, ISS gives the best simulation accuracy. To accelerate the 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] [11].

Other works like [17] are based on multi-ISS to accelerate the simulation. But these frameworks suffer not only from a complex synchronization scheme that increases the overhead but also from the grandiose simulation time.

3 Conventional approaches

To increase the verification speed while maintaining clock accuracy, an FPGA (Field Programmable Gate Array) such as ALTERA DE2-70 is used. The FPGA presents an easy and a fast environment for the target architecture implementation. 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 (System On Chip).

One method is based on an abstraction approach [12]. This approach uses only few FPGAs operating at about 1 MHZ. This method is similar to an accelerated C/C++ simulation. So it suffers from lack of performance estimation and no debugging.

Another method uses a transaction level C/C++ simulator to model the application program [13], and the FPGA to emulate the HW component.

A verification system that uses SystemC simulator to simulate the HW component and FPGA board based on target kernel architecture to simulate the SW application is proposed. The key contribution of this work is in the interoperability between the SystemC scheduler, and the target kernel architecture deployed on the FPGA.

The main advantages of our solution are:

- Increase the speed of SW validation without any lost of accuracy since SW will be executed by the target microprocessor.
- The base architecture (processor, bus and memory) is implemented in FPGA and the specific hardware components are described and simulated using SystemC.
- This framework represents a very useful platform for software engineers to validate their code before the hardware components become available
- The replacement of the ISS by a real target processor accelerates the simulation.

– A synchronization schemes between HW and SW are described in this paper with respect of the synchronization schemes between continuous and discrete design used in CODIS tool.
– The target base architecture is implemented in FPGA board at earlier stage. This fact conserves the time to market and informs the software engineers early about many characteristics as well as the time execution of the SW components, energy consumption, etc.
– The *interface* module provides the communication and the synchronization without modifying the SystemC kernel.

Figure 1 shows the verification environment. Two steps are essential to make the environment verification: communication and synchronization model. A Simulation Emulation engine is described in the next section.

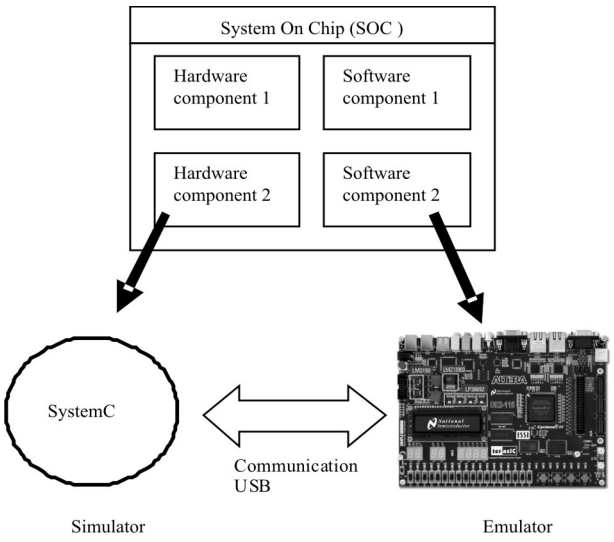


Figure 1: Combined simulator and emulator approach

4 Simulation emulation engine

Simulation engine presents the main problem of the majority of co-simulations environments. It ensures not only the communication between simulators and emulators but also the synchronization and the control. Figure 2 shows the Simulation/Emulation engine architecture. This section describes the communication and the synchronization layers with the implementation adopted.

4.1 Communication

This under section gives a brief introduction to the communication model and the associated library. A USB (Universal Serial Bus) 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 [14]. The model is divided into receiving / transmitting drivers models. Figure 3 shows the different steps of the communication.

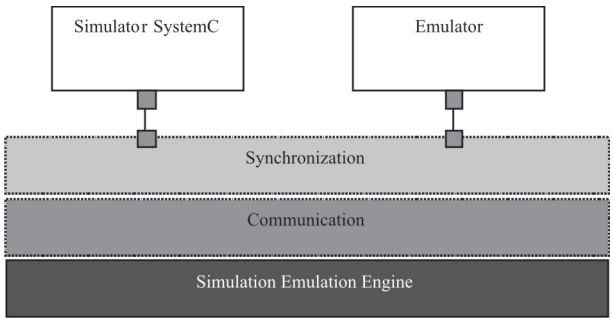


Figure 2: Simulation/Emulation engine architecture

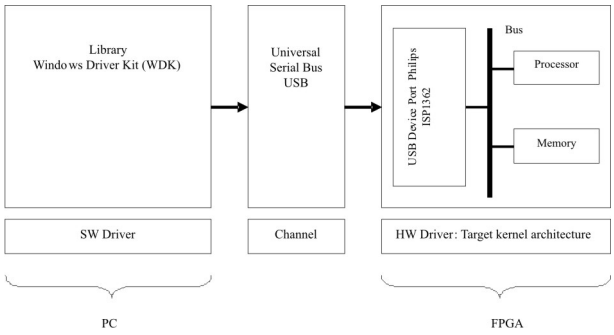


Figure 3: Communication model

- SW driver is made by Windows Driver Kit (WDK). This driver is responsible for the data transmission to and from the USB device. An operating system driver, running on the host system, responsible for receiving data from the high level application, forming the information packets as defined by the USB standard protocol, and also in charge for transmission of the data packet to the USB device. This driver will also be in charge of receiving data form the USB core (ISP1362), and deliver the received data to the high level application.

This driver contains essentially two main functions Read and Write. To achieve these two functions, four Win32 functions are used:
– CreateFile: required to connect application and USB core.
– WriteFile: required for data transfer into USB core.
– ReadFile: required for data transfer from the USB core into the application.
– ControlIODevice: requested for driver configuration.

- Channel: after the comparison research between the different kinds of communication channels, a USB communication is adopted not only because the portability but also because the speed of transaction is 480 Mbits/s; however, the speed of PCI is only 133 Mbits/s.

- **HW driver** is based on Philips ISP1362 device core that is integrated in emulation architecture with a processor, bus and memory, as shown in the Figure 3. The ISP1362 will manage the complete USB protocol from the device side, and which will also be in charge of leaving the received data in a buffer for resending when required. Note that the ISP1362 controller is used only in emulation environment and not be considered in the target architecture.

The ISP1362 provides the PIO (Programmed Input/Output) mode for the processor to access its internal control registers and buffer memory. An interruption mode is adopted to simulate the real behavior.

When an interrupt is activated, the processor halts normal operation and jump to the Interrupt Vector Table, which is a region of memory. There is one interrupt vector for each type of interrupt that can occur, and each interrupt vector is located at a unique memory address in the table. Each interrupt vector contains the address of the start of the Interrupt Service Routine (ISR) that will run for that interrupt. The ISR retrieves data from the ISP1362 Device Controller's internal FIFO (First In First Out) to NIOS II memory and sets up proper event flags to execute the program [15].

The ISR is like a subroutine and contains code that is executed once the ISR is entered. Once the code in the ISR is completed, control is passed back to the main program. If an interrupt event were to occur during this period, the processor halt the last interrupt program and jump to next interrupt program. When, it finished the processor go back to first interrupt program. This model is adopted to avoid the lost of data.

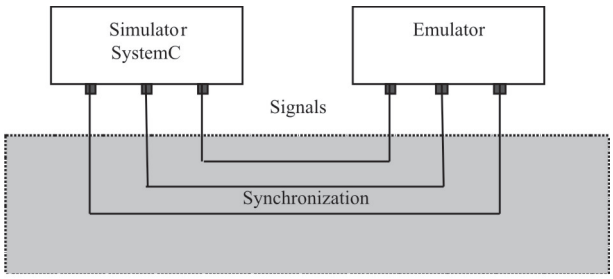
4.2 Synchronization schemes

A key part of the proposed verification is at first the synchronization schemes between the SystemC simulator and the target base architecture on FPGA board. Secondly, the model of synchronization between the different component of the Device Under Test (DUT). Note that the synchronization schemes ignored the communication overhead and focus in the process simulation.

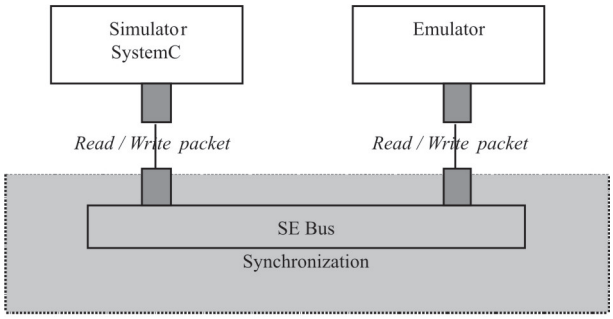
4.2.1 Simulator/Emulator Synchronization

We focus in this first part into the synchronization model between simulator and emulator. The SystemC simulator is fixed as the master of environment verification and the emulator based on target base architecture as slave.

The synchronization models are different form layer to layer. In RTL (Register Transfer Layer) layer the simulator and the emulator are interconnected with signals. While, the simulation/emulation bus ensures the transfer of data packet, in the TLM (Transaction Level Modeling) layer. Figure 4 a) shows the synchronization scheme for the RTL layer. Context switch presents the main function of the Simulation/Emulation (SE) bus. It assures the change of the control between simulator and emulator. Figure 4 b) describes the functionality of the Simulation/Emulation (SE) bus.



a) Simulation/Emulation bus in RTL



b) Simulation/Emulation bus in TLM

Figure 4: Simulation/Emulation Bus

The TLM layer is the most adopted in verification language because it is easier and faster than RTL layer.

The communication so described in the last section ensures packets forms which are constructed as interface between simulator and emulator (board).

Two forms of exchanged packets are used to perform the synchronization scheme between the simulator and the emulator, Figure 5.

Interruption packet shown in Figure 5 a) 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 [31] and it is used to execute the interrupt routine at the appropriate instant.

Data packet shown in Figure 5 b) 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.

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

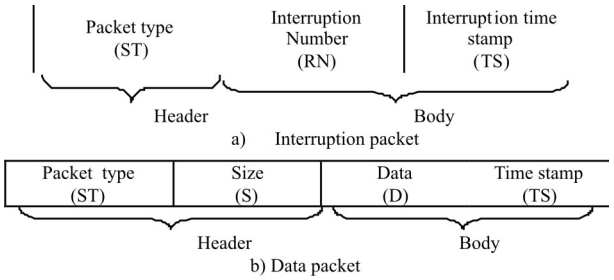


Figure 5: Synchronization forms

The verification method is based on the following synchronization schemes which respect the interaction style that can be involved between HW and SW components and the interaction style between continuous and discrete model used in CODIS tool. Note that, in the same design, HW and SW components may use different synchronization schemes. The execution time of SW applications in the target base architecture is considered as simulation time.

- 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 6). 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.

- 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 7). Once the hardware Task (Task1) is finished, the simulator sends data to the SW Task and a context witch 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 7). The SystemC and the

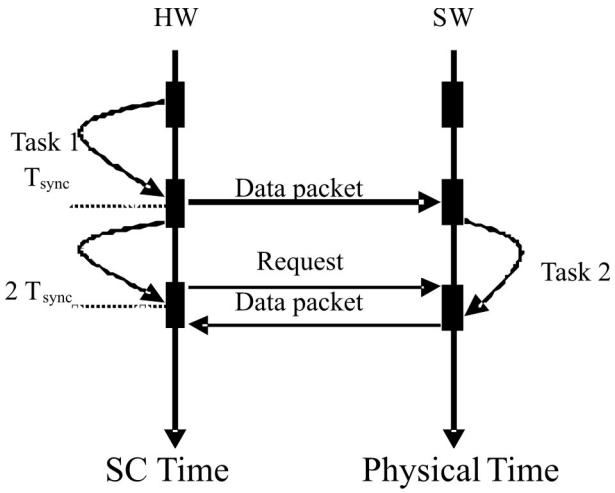


Figure 6: Synchronization model: scheme 1

emulator need to exchange the time stamp every context switch.

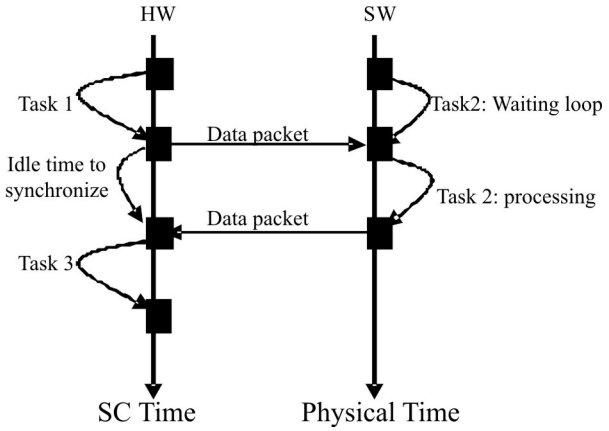


Figure 7: Synchronization model: scheme 2

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

This scheme is illustrated by the Figure 8. 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 (see Figure 14), running on the target processor, sends data to the simulation interfaces (arrow 0), which 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 (see Figure 9), 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 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 sched-

uler 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 10 shows the template of the code.

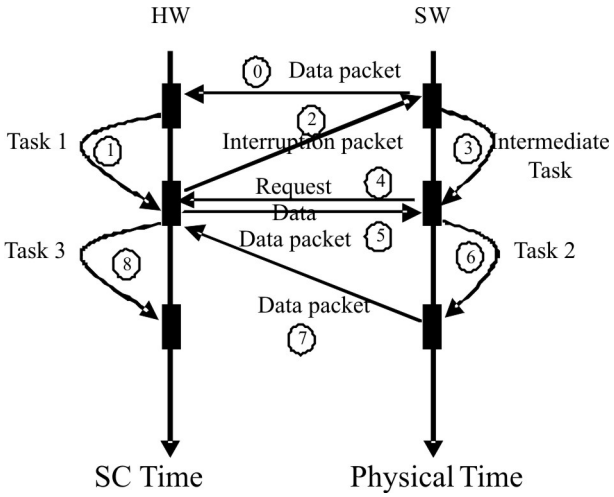


Figure 8: Synchronization model: scheme 3

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

Figure 9: 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 */
```

Figure 10: Template of synchronization code

- Scheme 4: The SW Task may receive a random interruption resulting from externally data reception

This scheme is illustrated by the Figure 11. 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).

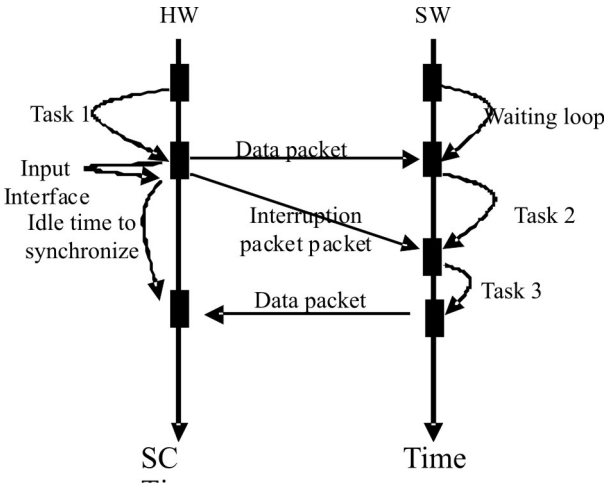


Figure 11: Synchronization model: scheme 4

To ensure communication and to save synchronization context, an array of shared registers is used. We fixed that the HW / SW partitioning is static. Also, the scheduler is at the same time static and based on data dependence. The last is used as a shared connection bus. Although this register based on bus modeling is not the same as the actual chip bus modeling, it is easy to set up. The register array is implemented on the FPGA board and can be accessed or Read / Write from the simulator using interruption services routines.

The S/E Engine ensures the verification of sequential and parallel applications. Two cases of sample are presented. The first is based on sequential Tasks and the second is based on parallel and sequential Tasks.

The Figure 12 describes the two samples based on data flow diagram and their synchronization model is shown in Figure 13.

Task 1, Task 3 and Task 5 are hardware components. Task 2 and Task 4 are software application.

For the sequential sample, the simulator begins the execution of the first Task. When Task 1 finished, the simulator sends an interrupt packet and data packet to the emulator to begin Task 2 and the simulator is blocked until he receives data packet from the emulator. At this moment, the simulator executes Task 3 and the emulator is blocked. When Task 3 finished, the simulator sends an interrupt packet to the emulator. The last runs Task 4 and returns data packet to Task 5 in the simulator.

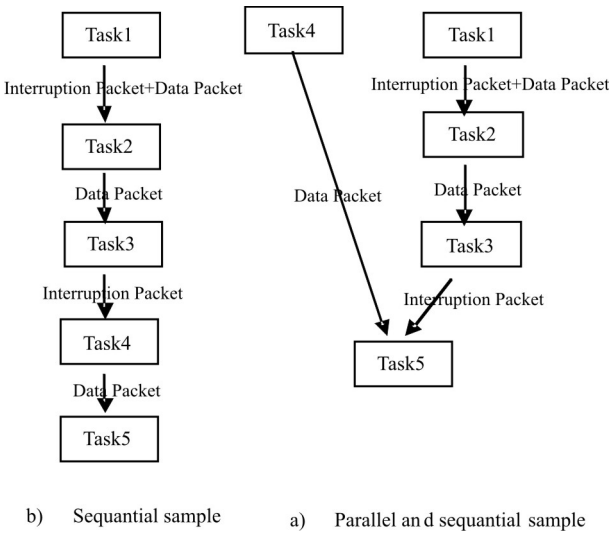


Figure 12: Data flow diagrams

For the parallel sample, the simulator begins Task1 and the emulator begins Task 4. When the simulator finished Task 1 he sends an interrupt packet to the emulator. The last stops the execution of the Task 4 and save the context then he receives the data packet and runs Task 2. The simulator is blocked. When Task 2 is finished, the emulator sends a data packet to the simulator and return to the execution of Task 4. At this moment, the simulator executes Task 3. When they finished, the simulator sends an interrupt packet. When the Task 4 is finished and the interrupt packet is sent, the emulator sends a data packet and the simulator runs Task 5.

4.2.2 Hardware/Software Synchronization interface

In this part, the hardware / software of the DUT is presented.

The SystemC is constructed using a modular approach to provide partitioning between the different functional elements of the overall controller. This facilitates the movement of functionality between different models, which proved useful during the control model design. It also simplifies the addition and/or removal of models from the system.

The hardware modules in SystemC are modeled in TLM layer and the adopted channel is FIFO. An Interface module is built to ensure the transaction between HW/SW. When a module wants to read or write data to the board, it makes connection with the TLM channel that implements a high level description of bus. The TLM channel assures the communication and the synchronization. Figure 14 describes the synchronization scheme for DUT.

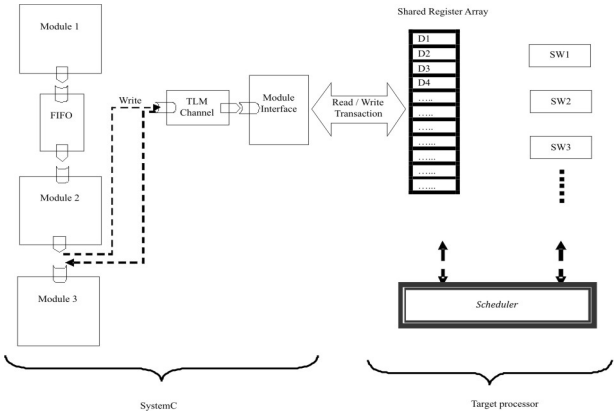


Figure 14: Synchronization interface for DUT

5 Experimental results

Three steps are essential for the Simulation/Emulation implementation and validation.

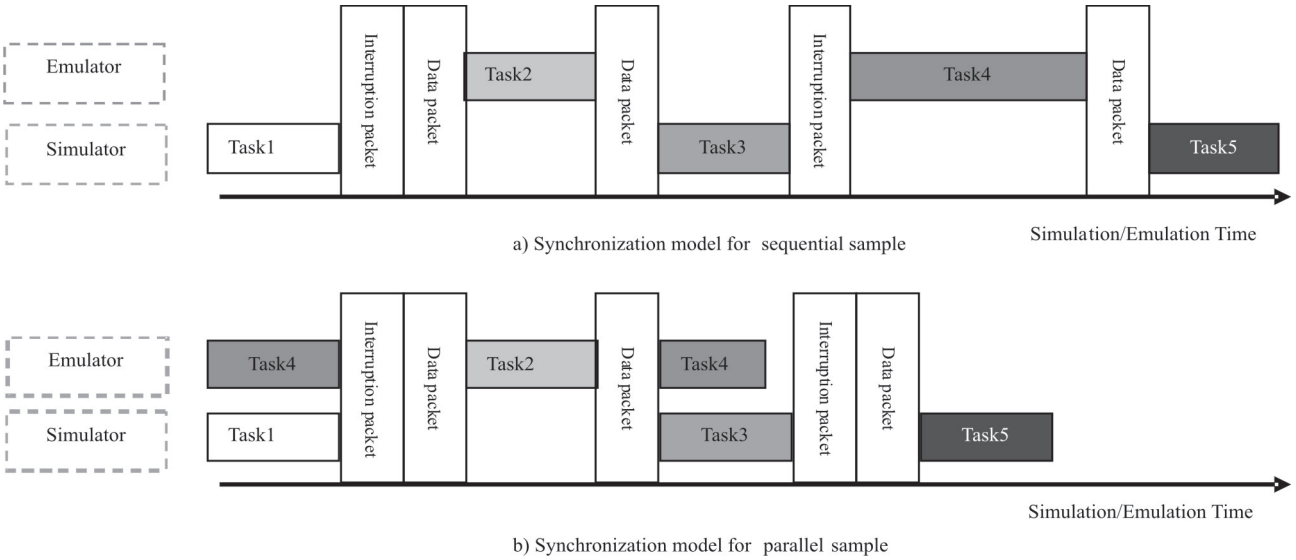


Figure 13: Synchronization model for each sample

Step 1: Target base architecture
The verification idea is based on combined tools to satisfy the HW/SW design. For the target base architecture an FPGA type ALTERA DE2-70 is used as a board and QuartusII, NIOSII IDE as tools. The first step is to set the architecture model. Figure 15 shows the architecture chosen. It contains the NIOSII processor [28], Avalon bus, memory and the ISP1362 USB controller [29].

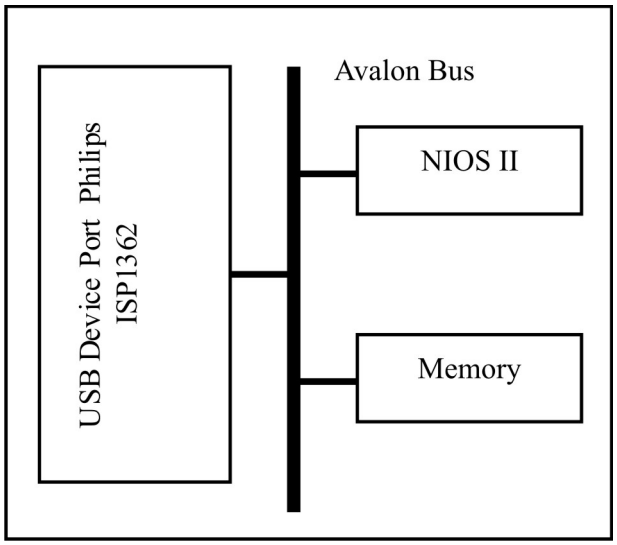


Figure 15: Target base architecture used

Step 2: Interfacing between HW/SW
This step concerns the addition of the *interface* module in SystemC. The last module is the responsible of the communication and the synchronization. When a context switch to the board is required, the *interface* module is called. The proposed interface can be added to any design described on SystemC without modify the kernel of SystemC like the work [30].

Step 3: Simulation results
In this section, two applications are proposed to validate the Simulation / Emulation environment.

*** Fingerprint recognition:**
Paper [16] presents a novel technique for fingerprint recognition based on DECOC classifier. Six steps are required for identification:
- Read the fingerprint: this phase reads the image of finger from sensor.
- Filter: A Gabor filter is used to ameliorate the contrast of ridge in the image.
- Binarization: This step converts the image from grayscale to binary system.
- Skeletonization: The neighborhood method is used.
- Minutia extraction: This step extracts the characteristics point in finger (Minutia) using the DECOC classifier.
- Matching: The final step makes correspondence between the input finger and the saved fingerprint.

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 with respect of the rule "the processes that has the more time expensive will be HW components".

The overall HW/SW configuration consists of the following entities:
– HW model of the read.
– HW model of the filter.
– HW model of the binarization.
– HW model of the matching.
– SW application of the skeletonization.
– SW application of the Minutia extraction.

Table 1 shows the simulation / emulation time of each process.

Table 1: simulation/emulation time of fingerprint

	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

To validate our simulation/emulation environment a comparison with simulation based on MIPS32 as ISS is made. Table 2 proves that the replacement of the ISS decrease by thirteen times less the time of simulation.

Table 2: Comparison Simulation Time

	Simulation (MIPS32)	Our environment
Simulation time (s)	18	0.54

*** 4 port router:**
A small 4-port router is described in SystemC, an extension of the Multicast Helix Packet Switch example distributed with SystemC 2.0.1.

This router receives data packets on its input ports and forwards them to the proper output port according to

a routing table embedded into the router. Whenever a new packet arrives on one of the input ports, it is stored into an internal buffer. If the buffer is full, the packet is dropped. Each packet is then read from the buffer by the main process of the router, and checked for errors by a checksum algorithm.

If the checksum is correct the destination address stored in the packet is used to find the right output port using the routing table; otherwise the packet is dropped. The packets consist of the following fields:
– Source address: the address of the producer.
– Destination address: the address of the consumer to which the packet must be sent.
– Packet identifier: an integer value used for debugging purposes only.
– Data field.
– Checksum: a 16 bit field used for error detection.

The overall HW/SW configuration consists of the following entities:
- HW model of the router.
- HW model of the packet generator (producer), which is attached to an input port of the router, and generates packets with a random destination address.
- HW model of the packet destination (consumer), which is attached to an output port of the router, and analyzes the integrity of the received packet.
- SW application computing the checksum, executing on NIOS II processor.

Table 3 shows the simulation / emulation time with 10 exchanged packets.

Table 3: Simulation/Emulation Time of 4-port router

	Module	Time (s)
HW Components	Router	0.03
	Packet generator	
	Packet destination	
Interface	Interface	0.12
SW Applications	Checksum	0.006
All Modules		0.156

To validate our simulation/emulation environment a comparison with simulation based on MIPS32 as ISS is made. Table 4 proves that the replacement of the ISS decrease by fifty times less the time of simulation.

Table 4: Comparison Simulation Time

	Simulation (MIPS32)	Our environment
Simulation time (s)	8	0.156

Discussion

These two samples highlight the performance of our environment with comparison with MIPS32 simulator. The simulation time is decreased by averaging 38 times.
The difference of the interface time between the fingerprint application and the 4-port router application is justified by the quantity of data transferred in each context switch. For the first sample, the interface consumes much time because the totality of the image must be transferred from/to the board.

6 Conclusion

The co-verification environment is based on synchronization between SystemC simulator and FPGA board emulator. The essential aim is to accelerate the simulation with the replacement of an ISS with target base architecture implemented in the board. The main features of the proposed synchronization schemes are the adaptation with CODIS tool. Experiments with real-life examples proved the effectiveness of the proposed system.

The resulting simulation/emulation time allows fast functional validation of HW devices before actually designing them and including onto a board, and, in particular, without changing the software executed on the board. The results shows that the simulation/emulation environment decreases the time simulation forty times less than the simulation environment based on ISS.

Acknowledgments

We are grateful to Mr. Hatem Bentaher and Mme. Massira Ben Said for helpful suggestions and for English correction.

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", book published by Kluer Academic Publishers.
3. De Micheli, D., Ernst, R., and Wolf, W. Eds. Readings in Hardware/Software Co-design, Morgan Kaufmann, 2001.
4. Liem, C., Nacabal, F., Valderrama, C., Paulin, P., and Jerraya, A. 1997. System-on-chip cosimulation

and compilation. *IEEE Design and Test of Comput. 14,* 2, 16–25.
5. Valderrama, C.,Nacabal, F., Paulin, P., and Jerraya, A. 1998. Automatic VHDL-C interface generation for distributed cosimulation: Application to large design examples. *Design Autom. Embed. Syst. 3,* 2/3, 199–217.
6. Coste, P., Hessel, F., Marrec, P. l., Sugar, Z., Romdhani, M., Suescun, R., Zergainoh, N., and Jerraya, A. 1999. Multilanguage design of heterogeneous systems. In Proceedings of IEEE International Workshop on Hardware-Software Codesign. 54–58.
7. Liu, J., Lajolo, M., and Sangiovanni-Vincentelli, A. 1998. 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.
8. Fummi, F., Martini, S., Perbellini, G., and Poncino, M. 2004. 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.
9. Moussa, I., Grellier, T., and Nguyen, G. 2003. Exploring SW performance using SoC transactionlevel modelling. In Proceedings of the IEEE Conference on Design Automation and Test in Europe. 120–125.
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. Ziyi Jin, "A remote controlled embedded system implemented in FPGA", Master of Science Thesis in the Programme of Integrated Electronic System Design, Department of Computer Science and Engineering CHALMERS UNIVERSITY OF TECHNOLOGY UNIVERSITY OF GOTHENBURG Göteborg, Sweden ,2009.
12. N. Kim, H. Choi, S. Lee, S. Lee, I-C. Park, C-M. Kyung, „Virtual chip: making functional models work on real target systems", Proceedings of ACM/IEEE Design Automation Conference (DAC 98), pp.170-173, 1998.
13. 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, february 2005.
14. Jan Exalson, "USB COMPLETE Everything You Need to Develop Custom USB Peripherals" book, third edition, 2005.
15. ISP1362 Embedded Programming Guide Version 9 June 2002.
16. 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. Volume 4 (2012) pp. 546-553
17. S. Cordibella, F. Fummi, G. Perbellini, D. Quaglia, "A HW/SW Co-Simulation Framework for the Verification of Multi-CPU systems", IEEE transactions, 2008
18. F. Bouchhima, M. Brière, G. Nicolescu, M. Abid, E. M. Aboulhamid, "A SystemC/Simulink Co-Simulation Framework for Continuous/Discrete-Events Simulation", Behavioral Modeling and Simulation Workshop, Proceedings of the 2006 IEEE International.
19. Primož Puhar, Andrej Žemva, "Hybrid functional verification of a USB host controller", Informacije MIDEM – Journal of Microelectronics, Electronic components and materials, Vol. 38, Iss. 2 (2008), p94-102.
20. Xunying Zhang, Fei Hui, Qiang Wang, Xubang Shen, "Integrated ISS and FPGA SoC HW/SW Coverification Environment Design", 12th International Conference on Computer Supported Cooperative Work in Design , 2008, p 1071-1075.
21. Mehrdad Reshadi, Nikil Dutt, "Hybrid-Compiled Simulation: An Efficient Technique for Instruction-Set Architecture Simulation", ACM Transactions on Embedded Computing Systems, Vol. 8, No. 3, Article 20, April 2009.
22. Wei Qin, Joseph D'Errico, Xinping Zhu, "A Multiprocessing Approach to Accelerate Retargetable and Portable Dynamic-compiled Instruction-set Simulation", The International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS'06, 2006.
23. Carsten Gremzow, « Compiled Low-Level Virtual Instruction Set Simulation and Profiling for Code Partitioning and ASIP-Synthesis in Hardware/ Software Co-Design", Proceedings of the 2007 Summer Computer Simulation Conference, SCSC 2007.
24. Hoeseok Yang, Youngmin Yi, Soonhoi Ha, "A Timed HW/SW Coemulation Technique for Fast Yet Accurate System Verification", Proceedings of the 9th international conference on Systems, architectures, modeling and simulation,2009, Pages 74-81.
25. Y.B. Liao, P. Li, A.W. Ruan, Y.W. Wang, W.C. Li, W. Li, "Hierarchy Communication Channel in Transaction-Level Hardware/Software Co-Emulation System", Ninth International Workshop on Microprocessor Test and Verification, 2008.
26. Franco Fummi, Giovanni Perbellini, Mirko Loghi, Massimo Poncino, « ISS-Centric Modular HW/SW

Co-Simulation », Proceedings of the 2006 ACM Great Lakes Symposium on VLSI GLSVLSI'06, April 30–May 2, 2006.
27. Luca Formaggio, Franco Fummi, Graziano Pravadelli, « A TimingAccurate HW/SW Cosimulation of an ISS with SystemC", The International Conference on Hardware/Software Codesign and System Synthesis CODES+ISSS'04, September 8–10, 2004.
28. NIOS II Processor Reference. (2013) [Online]. Available: http://www.altera.com
29. ISP1362 Single-chip Universal Serial Bus On-The-Go controller. (2013) [Online]. Available: http://www.cs.columbia.edu
30. Franco Fummi, Mirko Loghi, Stefano Martini, Marco Monguzzi, Giovanni Perbellini, Massimo Poncino, «Virtual Hardware Prototyping through Timed Hardware-Software Co-simulation », Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05), 2005.
31. J. Tratnik, P. Lemut, M. Vidmar, "Time-transfer and synchronization equipment for high-performance particle accelerators", Informacije MIDEM – Journal of Microelectronics, Electronic Components and Materials, Vol 42, No 2 (2012), p115-122.

Arrived: 22. 01. 2013
Accepted: 19. 08. 2013