A Simulation Based Approach for Incorporating Virtual Components IP Cores into Multimedia Systems Design

Adel Baganne [1] , Imed Bennour [2], Mehrez Elmarzougui [2] , Eric Martin [1]

[1] L.E.S.T.E.R Lab -UBS 56325 Lorient - France

[2] E.µ.E Lab -FSM 5019 Monastir - Tunisia

ABSTRACT

Growing requirements on the correct design of high performance multimedia systems in a short time force us to use IP's blocks in many designs. However, their correct integration in a design implies more complex verification problems. In this paper, we present a C++/SystemC based simulation flow at multiple levels of abstraction. Our approach is to use SystemC to describe both application and a set of algorithmic IP cores to be incorporated throughout the design flow. Our methodology supports design refinement through four main abstraction levels, offers verification techniques at each level and allows the use of EDA co-verification tools. The use of C++/SystemC to model all parts of the system provides great flexibility and enables faster simulation compared to existing methodologies. An illustrative case study for wavelet based compression system design shows that our methodology supports efficient algorithmic specification, where IP models can be easily incorporated, modified and simulated in order to quickly evaluate alternative system implementation.

Keywords

Application modeling, Multi-level simulation, Architecture modeling, IP core reuse, Multimedia systems.

1. INTRODUCTION

High performance systems-on-chip (SoC) designs today are approaching 20 Million gates and 0.5 to 1 GHz operating frequency. To implement these systems, designers are increasingly relying on reuse of intellectual property (IP) blocks that are available in various forms ranging from soft cores to hard cores. These components represent functions of specific domains like signal processing (DCT, FFT), telecommunication (Viterbi, Turbo codes) multimedia application (MPEG2, MPEG4, JPEG) etc. The IP cores are integrated in a SOC that includes digital signal processors (DSP), shared memories, bus controller and a set of hardware IP blocks connected to the system bus through specific interfaces or wrappers.

Since IP blocks are pre-designed and pre-verified, the designer can concentrate on the complete system without having to worry about the correctness or performance of the individual components. In practice, however, assembling a SoC using IP blocks is still an error-prone, labor-intensive and time-consuming process. Actually, even if cores are pre-verified, it does not mean the whole system will work when they are put together. A successful IP integration requires the system designer to take into account the main following tasks:

- *Synchronization:* components have to be synchronized on different aspect such as global execution, data exchanges and protocols.
- *Protocol conversion:* Assure the protocol conversion between blocks that use incompatible protocols. Wrappers can be used for this purpose but introduce overhead that should be taken into account with the timing constraints.
- *I/O buffer synthesis:* data may be buffered to ensure the system behavior and to meet timing constraints.

Various interface and timing issues can cause the systems to fail even when the individual cores are correct. So, the diversity of applications coupled with the ever-diminishing time to market is creating the need for new tools and design methodologies to support rapid SoC design and verification at some levels of abstraction above the register-transfer level.

In this paper we briefly describe our co-simulation design flow for IP based design and. Our approach is to use SystemC environment to describe an entire system efficiently at different levels of abstractions for simulation and towards synthesis. Besides, the proposed design flow allows the use of industrial co-verification and simulation tools. The rest of the paper is organized as follows. In Section 2, we present our design flow. In Section 3, we give experimental results relative to image compression system design with 1D Wavelet transform IP Core. Finally, in Section 4, we state our conclusions.

2. MULTI-LEVEL SIMULATION

The methodology essentially starts with a very high, behavioral-level design of the system that can be simulated. The design process allows: (i) incorporating range of IP models at multiple level of abstraction (ii) system specification refinement and (iii) multi-level simulation. IP Cores can be generally categorized into three main flavors: Soft, Firm and Hard cores [2]. In our case, we assume that designer has -for each IP block to be integrated, a set of simulation models provided by IP creator. These models cover all abstraction levels previously described and may be relying on different Models of Computations (MOCs) [12] such as FSM, CFSM, SDF, DDF etc.

In our case, we used a set of communication principles and design guidance provided by standardization organization VSIA [3] and OSCI [4]. Thus, our design flow is built around four main abstraction levels: *Untimed Functional, Timed Functional, Bus cycle accurate and Cycle Accurate* levels [8].

- *Untimed Functional level* **:** At this level, algorithms are most easily captured and verified using untimed functional models. Only the application behavior is considered for simulation and algorithm optimization. For example communication systems can be simulated to optimize transmission and reception over simulated real world.
- *Timed Functional level* : At this level some functional models are annotated with processing or communication delay. This level of abstraction is used for analyzing latency effects system behavior and exploring system architecture in the early stages of the design process. Timing is typically expressed as a number of clock ticks relative of system clock(s).
- *Bus Cycle Accurate (BCA) level* **:** At this level the communication architecture is fixed by designer as a set of components (memory, shared bus, interrupt controller etc.). So BCA level is used to model the communication.

- *Cycle accurate (CA) level :* This is the lowest abstraction level we consider. Hardware models are described as detailed cycle accurate structures or RTL models including hardware registers, clocks, latches and combinatorial logic. Software models targeting a processor are described at instructions set level while hardware-software interactions are also specified at low level.

3. CASE STUDY : 1DDWT WAVELET IP INTEGRATION

The application of the case study is a compression application (see Fig. 1) based on 1D Discrete Wavelet Transform (1DDWT). We have chosen this application because it is not too complicated, but has enough features to illustrate the problem of IP integration and the usefulness of our multilevel simulation flow.

3.1 The DDWT IP

DDWT is a compression technique used in signal coding applications. This technique is widely used due to its perfect wavelet analysis and synthesis properties, and the absence of perceptual degradation after reconstruction (using $DDWT^{-1}$). One dimension (1_D) wavelet codes one- dimension signals, such as speech and still pictures. Two- dimensions (2_D) wavelet codes two- dimensions data, such as video sequences. The IP considered here is 1_D, with multi-stages coding, using Daubechies filters [16]. Each stage contains one filter High and one filter Low and stages are connected to each other as shown in Fig.1.c. The IP's features are:

- configurable number of stages (from 1 to 6 stages) in the wavelet tree;
- configurable filter's length: 6, 8 or 10. Two least asymmetric filters are also supported with length 8 and 10;
- configurable data bus width, varying from 8 to 16 bits;
- fixed point representation;
- 2's complement arithmetic;
- Fully synchronous.

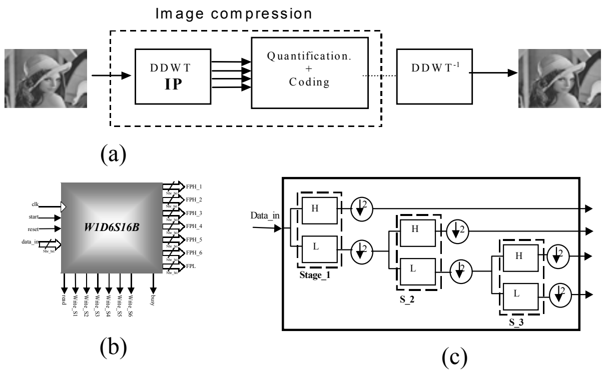

Fig.1: (a) Image compression application (b) 1D DWT IP (c) IP internal structure

3.2 Integration and simulation steps

3.1.1 Integration of the DDWT at the UT level

The model of the DDWT used at the untimed level is the synchronous data-flow (SDF) network model, a special case of Khan process networks. A SDF is a collection of functional nodes that are connected and communicate over unidirectional FIFO queues. Nodes commonly called actors perform computation that maps input data into output data, the data are divided into tokens treated as indivisible units.

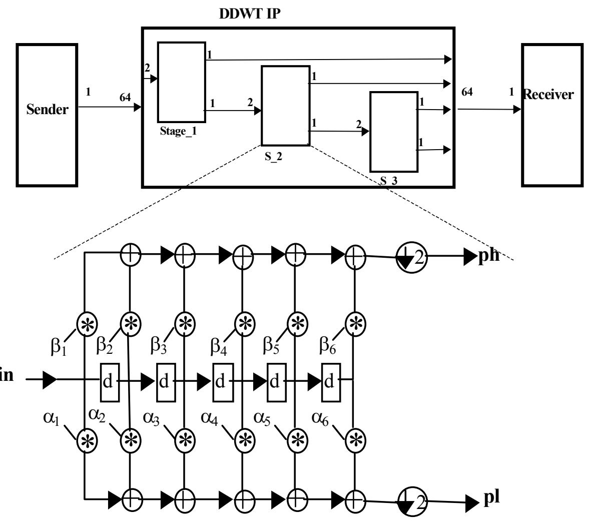

Fig.2 : SDF model

A connection between actors represents both the flow of data and the sequence of tokens, from a producer actor to a consumer one. Figure 2 shows the SDF of the DDWT. At each iteration the DDWT consumes a block of 64 pixels and produces 64 values, a stage inside the DDWT consume two tokens and produces one token on each of its outputs, while the sender and the receiver produces and consumes one token at each iteration, respectively. The number of tokens consumed and produced can be used to unambiguoudly define the minimal of firings that return the queues to their original size: in SDF models, deadlock and boundedness are decidable at compile time. The integration objectives using this model are to validate the IP functionality, its genericity and its functional performance, to select the adequate number of stages for the target application, and to define adequate filter's coefficients for the target application. To reach these objectives, which all are timing independent, we setup a simple simulation framework composed of three sequential and untimed (zero delay) processes: a transmitter process sending blocs of pixels to the IP, the functional model of the IP, and a receiver process implementing the wavelets backward transform ($DDWT^{-1}$). Processes are connected to each other through two unbounded and non-blocking fifos: a fifo_in from the producer to the IP and a fifo_out from the IP to the receiver. All processes are written in SystemC using its native abstract fifo channels for communication.

The second row of Table 1 shows the simulation speed for different image sizes. The fast simulation speed is due to the absence of synchronization between processes (no concurrency) and due to the use of abstract communication. Noting that a special attention should be done towards the non-determinism of the SystemC scheduler. For instance, when we replaced in the same design unbounded fifos by bounded and blocking fifos deadlock problems have occurred due to this non-determinism.

3.1.2 Integration at the Timed Functional Level

The timed functional model of the DDWT is a SystemC module composed of concurrent processes describing a pipelined implementation of stages with the notion of time. Prior to start the communication synthesis, the first objective was to verify the functionality and the features of the new model. This was achieved by replacing in the previous simulation design the untimed model of the IP by the timed one, while keeping the rest of the design untouched and using the same testbenches.

The third row of Table 1 shows the simulation speed, which stills fast due to the use of abstract communication.

- **Communication synthesis** : The objectives at this step are (i) to validate the synchronous communication protocol of the DDWT IP, (ii) to synthesis the communication model between the IP and its surrounded blocks, and (iii) to synthesis an initial version of the software scheduler. Based on the following design choices: the signal sampling process, the quantifier and the coder are mapped to Sw running on an ARM7_TDMI processor, the DDWT is mapped to hardware, and a point-to-point communication mode is used between the DDWT and the rest of the design. At this level, we use a network of FSMs describing the communication of the system while abstracting its behavior. Figure 3 shows this model composed of: an FSM of the DDWT describing its communication protocol while abstracting its behavior and its implementation, an FSM for the input FIFO, an FSM for the output fifo, an FSM for the controller unit that drives the FIFOs and generates an interrupt each time the DDWT produce a data - interrupts are handled by the scheduler, an FSM for the scheduler, an FSM for the sender, and an FSM for the receiver. Once the communication have been synthesized, we combine the FSMs network -describing the communication, with the SDF network -describing the computation behavior. As sequential processes are executed into zero delay under SystemC, to each sw process P_i is assigned a delay Δ_i and a $wait(\Delta_i)$ statement is added at the end of each iteration. Software processes and hardware processes are driven by different clock frequencies.

- **Timing and interdependency modeling:** The transactions between sw tasks (sender and receiver) and the hw one (IP DDWT) are processed through buffered interface (fifos).

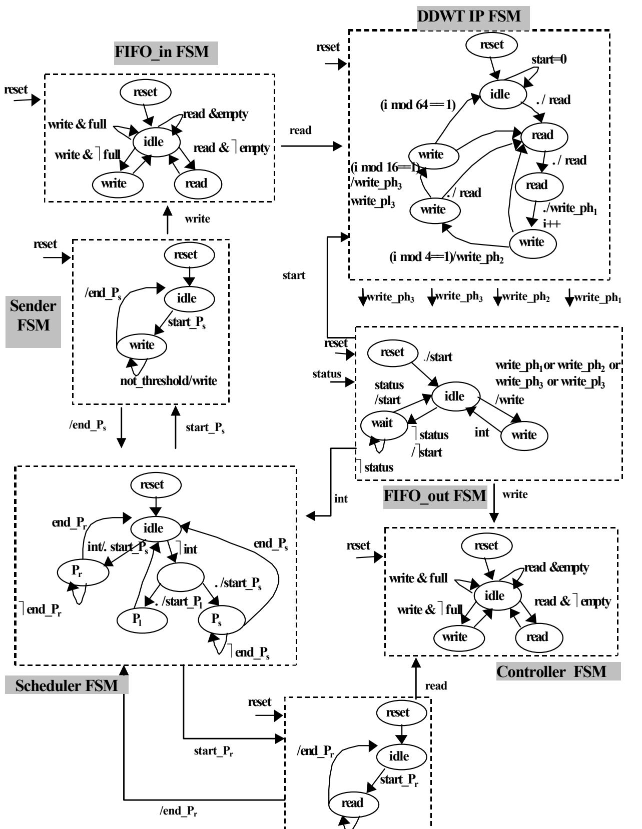

Fig.3: FSMs network model

Figure 4 illustrates timing execution of tasks with their interdependency. The DDWT task receives data from fifo_in and produces results to fifo_out at regular rate. Note that fifos empty/full states will generate waiting delays for the IP and other processes. Hence, the whole system execution performance may be affected and the IP integration can fail.

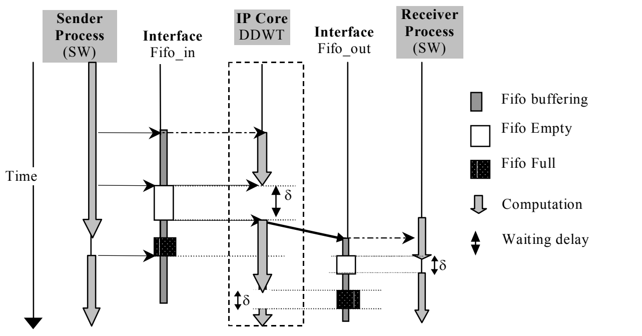

Fig. 4 : Software - IP transactions and waiting delays : an illustrative example

Our simulation environment which abstracts yet the target processor has allowed validating the timing performance of the IP (e.g. latency, throughput, etc.) for different fifos sizes, scenarios, and synthesis the control unit at a bus cycle accurate level. Simulation speed is given in the fourth row of Table 1; we can note the impact of the communication abstraction on the performance (row three and row four).

3.1.3 *Integration at the Cycle Accurate Level*

The goal at this level of integration is to refine the results obtained in the previous step by using a synthesizable RTL model of the IP and an Instruction Set Simulator (ISS) of the ARM7. We setup a hw-sw co-simulation environment composed of the Seamless_CVE hw/sw co-simulation tool, an ARM7-ISS and the VHDL Modelsim simulator. The SeamlessCVE tool allows to link the hardware simulator to the software running on the ISS through the bus interface model (BIM) of the ARM7 (see Fig 5). The BIM is modeled in HDL and connected from one side to the hw code and to the sw code from another side. Fifo_in is connected to the system bus, and each time the sw executes a write instruction to fifo_in there will be simulation of the equivalent bus cycle by the BIM. Fifo_out is connected to the ARM through an I/O port and data transmission is performed by an interrupt mechanism. The processor clock frequency is 33 MHz, and the DDWT clock frequency is 10 MHz.

In order to identify possible deadlocks and to evaluate the impact of DWT IP integration, we parameterized the software execution with a stochastic model.

$$R = \frac{\text{Emission transactions (from processor to IP)}}{\text{Processor processing}}$$

This ratio that varies from 0 to 100% allows us to identify and to explore critical situations. Typically, the DDWT IP -that is supposed to work intensively may accuse waiting delays when few data are sent from the processor and vice-versa. Too low sizing of fifos has important effect -on both IP and the whole system execution, in terms of resources access. Figures 7 and 8 show the waiting delays – obtained by co-simulation, relative to empty or full fifos states (fifo-in, fifo-out) and for size ranging from 8 to 256 points. First, we observe that for fifo_in = 128 the IP waiting delay due to input data is reduced significantly for all values of parameter R.

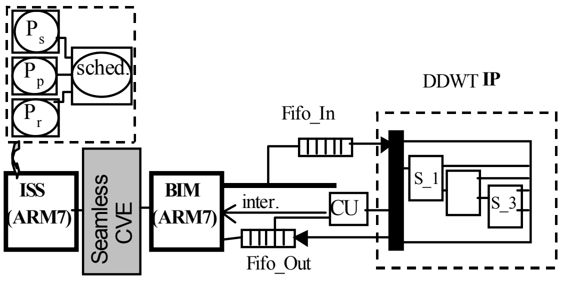

Fig. 5: Cycle Accurate IP integration with Seamless-CVE tool

Second, simulation results shows that –for different fifo-out sizes and R values, there is always waiting delay due to output fifo (see Fig. 8). Execution rates difference between the ARM7 and the DDWT IP is the origin of this overhead. In our experiments, co-simulation has allowed: (i) to determine the size, threshold min and threshold max values of fifos, and the maximum processor load for this application (ii) the verification of functional and cycle accurate DDWT IP integration.

Co-simulation speed depends on both the hw simulator performance and the sw simulator performance. In our design, the slow co-simulation speed for the CA level (fig 6) is due mainly to the cycle by cycle coordination between the ISS and the HDL simulator. Noting that the instruction fetch cycles have not been simulated.

Table 1. Multilevel Simulation time

Abstraction Level	Image size (octets)			
	16385	**100 K**	**500 K**	**1 M**
Untimed Functional (UT).	0.1 s	0.6 s	2.9 s	5.8 s
Timed Functional (TF)	2.4 s	15.0 s	70 s	146 s
Bus cycle Accurate (BCA)	52.7 s	312 s	1581 s	3192 s
Cycle Accurate CA.	1H23:2	3H23	10H50	17H20

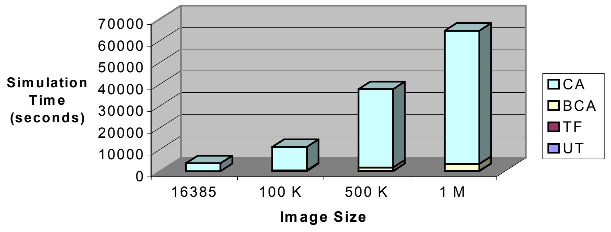

Fig. 6: Simulation time

4. CONCLUSIONS

In this paper, we described a multi-level system modeling and simulation flow for SoCs that allows incorporating IP cores at different levels of abstraction. This flow allows the use of a set of design and simulation techniques for system verification and Hw/Sw co-simulation. Integration overhead and timing features relative to IP core execution constraints can be simulated and reduced by optimizing some design parameters.. Multi abstraction levels are also a key solution for the simulation speed bottleneck occurring in complex systems. In our case study, even though it is not so complex, the simulation speed is reduced by at least a factor of twenty from one level of abstraction to another (see Table 1).

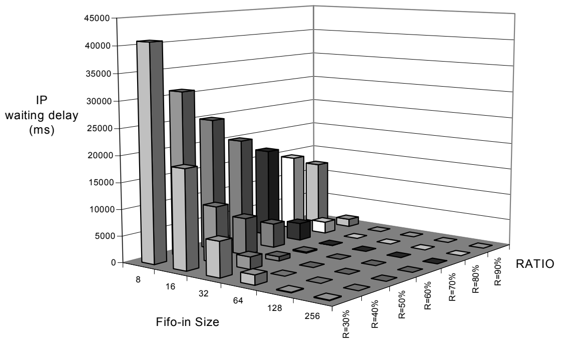

Fig. 7: IP waiting delay due to fifo-in sizing (Ratio)

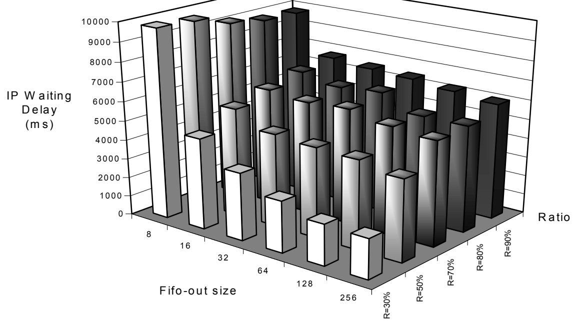

Fig. 8: IP waiting delay due to fifo-out sizing (Ratio)

5. REFERENCES

[1] H. Chang, L. Cooke, al., "Surviving the SOC revolution, A guide to Platform-Based Design", ", KAP publishers, 1999

[2] M Keating, P Bricaud, "Reuse Methodology Manual for System-On-A-Chip Designs" KAP, Boston, 1998

[3] Virtual Socket Interface Alliance, http://www.vsi.org

[4] Open SystemC Initiative, http://www.osci.org

[5] Sonics Inc, "Sonics μnetworks Technical Overview", June 2000

[6] K. Van Rompaey, D. Verkest, al. "CoWare A design environment for heterogeneous hw/sw systems", in Proc of EURODAC, 1996.

[7] Cadence VCC 2001, http://www.cadence.com/datasheets/vcc.html.

[8] Cocentric System Studio, http://www.synopsys.com/

[9] Seamless CVE , http://www.mentor.com/seamless/

[10] G. Cyr, al., "Synthesis of communication Interfaces for SOC using VSIA recommendation", in Proc. of DATE, 2001,

[11] G. Nicolescu, S. Yoo, and A. A. Jerraya "Mixed-Level Cosimulation for Fine Gradual Refinement of Communication in SoC Design" Proc. DATE 2001.

[12] Stephen Edwards, Luciano Lavagno, al. "Design of Embedded Systems: Formal Models, Validation, and Synthesis" Proc. of the IEEE , 1997

[13] R. Gupta, "Co-Synthesis of Hardware ond Software for Digital Embedded Systems", KAP, 1995.

[14] P. Coussy, A. Baganne, E. Martin, " A Design Methodology For IP Integration ", in Proc. of ISCAS 02, Phoenix. USA.

[15] J. Staunstrup, W. Wolf "Hardware/software Co-design Principles and practice", Kluwer academic publishers 1997.

[16] I. Daubechies, W. Sweldens "Factoring Wavelet Transforms into Lifting Steps", in Jour. of Fourier Analysis and App., Vol 4, Nr 3, 1999