

Interface based Hardware/Software Validation of a System-on-Chip

Debashis Panigrahi, Clark N. Taylor and Sujit Dey
Department of Electrical and Computer Engineering
University of California, San Diego
{dpani, cntaylor, dey}@ece.ucsd.edu

Abstract

The availability of reusable IP-cores, increasing time-to-market and design productivity gap, and enabling deep sub-micron technologies have led to core-based system-on-chip (SoC) design as a new paradigm in electronic system design. Validation of these complex hardware/software systems is the most time consuming task in the design flow.

In this paper, we focus on developing an efficient interface-based validation methodology for core-based SoC designs. In SoCs designed with pre-validated IP cores, the verification complexity can be significantly alleviated by concentrating on the integration of the cores in the system, rather than the complete SoC. In this paper, we investigate typical interface problems that arise in integrating cores in a SoC, and classify these problems into different categories. Based on the classification of these interface problems, we introduce an interface-based validation methodology. Finally, we demonstrate the effectiveness of the proposed methodology using an example image compression SoC that we are developing.

1. Introduction

The growing demand for hardware/software systems, together with the ability to put the entire system on a single chip using deep sub-micron technologies, has led to the evolution of complex hardware/software system-on-chips (SoCs). While the complexity of SoCs increases, so does the demand to reduce their time-to-market. Though the design time of SoCs can be greatly reduced by efficient re-use of intellectual property (IP) cores, the validation of SoCs is still a complex and time consuming task. In this paper, we focus on developing an efficient validation methodology for IP core-based SoC designs.

Any system validation framework can be characterized by three parameters, namely the validation methodology, the level of abstraction used for validation, and what needs to be validated. Figure 1 presents these parameters and the interdependency between them. For instance, traditionally

the complete system is validated using hardware simulation models of the system components at varying levels of abstraction, i.e. behavioral, register transfer, gate, and transistor.

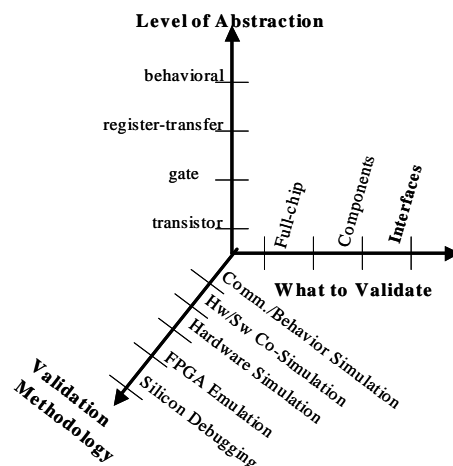


Figure 1. Different dimensions of validation

Lately, several attempts have been made to reduce the complexity of system validation by simulating at higher levels of abstraction using efficient simulation methodologies. For instance, Hw/Sw co-simulation tools [1, 3] reduce the validation time by performing simulation at higher levels of abstraction, specifically by using instruction accurate models of processors. Interface based design methodology, proposed in [2], is another example of validation at a higher level of abstraction, which allows separating communication between components from behavior of a system. In this methodology, the system can be simulated with different levels of abstraction of communication between components, initially starting with an abstract model, and using refined models of communication between components in the successive phases of the design. While previously proposed methodologies for the validation of a SoC significantly decrease the simulation time through the use of higher levels of abstraction, simulation of the entire system is still required. As the complexity of SoCs rapidly in-

creases with the use of heterogeneous and complex IP cores, simulating the entire system becomes prohibitively expensive, even with the use of abstract models for computation and communication. The problem is further compounded by the need for multiple iterations of system simulation to detect and fix design errors.

The complexity of verifying a SoC can be significantly alleviated by (1) using pre-validated IP cores and (2) concentrating on verifying the integration of the cores in the SoC, rather than attempting to verify the entire SoC. Instead of simulating all the components of a SoC together, the interface between different components can be verified independently. Independent validation of the interfaces can substantially reduce the overall time spent in SoC validation by pointing out the design problems early in the verification cycle. Moreover, by dividing the integration procedure methodically, and then simulating only the components necessary to validate the interface involved, we add another dimension for decreasing the design validation time required. In Figure 1, we illustrate how our method applies to the other methods currently used to decrease the verification time. It is important to note that the proposed interface based simulation can be performed at any level of abstraction (i.e. behavioral level, RT Level etc.) and using any simulation methodology (i.e. co-simulation, Communication/Behavior simulation [2], hardware simulation).

From our experiences in designing an image compression SoC, we identify common interface problems in integrating different components of a SoC. These interface problems can be classified into different categories based on their nature and scope. For example, the interface between a component and a system bus should be validated before the communication between two components using the system bus is verified. Based on the classification of these integration problems, we propose an interface based validation methodology for core-based SoCs.

For efficiently validating the interface between a hardware component and a software component, we designed a fast and accurate co-simulation environment. We use the co-simulation environment to implement the interface based validation methodology proposed.

The rest of the paper is organized as follows. In section 2, we introduce the reconfigurable image compression SoC that we have designed and will use to illustrate various issues in the rest of the paper. We introduce and classify some common interface problems of SoC integration in section 3. We also propose our interface based validation in this section. We present the co-simulation environment in section 4. The experimental results demonstrating the efficiency of the proposed methodology is presented in section 5. Section 6 concludes the paper.

2. A System-on-Chip Architecture

To fully understand the issues involved in validating a SoC, we developed a reconfigurable image compression system-on-chip (SoC) for wireless multimedia applications. Using this SoC as an example, we illustrate the validation problems and procedures outlined in this paper. A brief description of the architecture of the SoC follows.

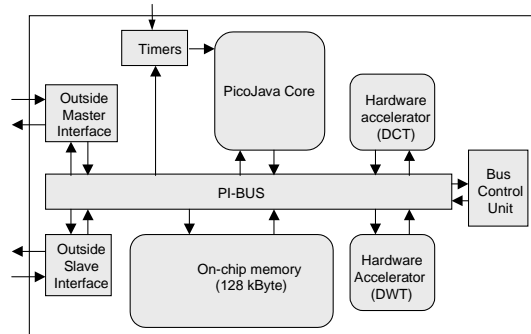


Figure 2. Architecture of an image compression System-on-Chip

The reconfigurable image compression SoC, shown in Figure 2, includes a processor core, two hardware accelerators, an on-chip memory, timers, master/slave interfaces to external components, and an internal system bus. PicoJava, a soft processor core from SUN Microsystems [4], is used as the processor, while PI-BUS [5] is used for the system bus. The hardware accelerators are designed to implement the most compute-intensive portions of image compression algorithms, while the picoJava processor implements the more control-intensive, parameterizable portions of the algorithms. We have implemented as hardware accelerators two transforms, the discrete cosine transform (DCT) [6], and the discrete wavelet transform (DWT) [7]. These transformations form the basis of image compression algorithms such as JPEG [8] and SPIHT [9]. Additionally, the chip has a 128KByte on-chip SRAM.

In the next section, we point out some common interface problems with integrating cores into a SoC. We illustrate with some examples from our experiences in the validation of the image compression SoC.

3. Problems in SoC Integration : Communication Mechanisms & Interfaces

As discussed earlier, we view the main task in validating a core-based SoC as ensuring a proper interface between the cores. In this section, we present some common interface problems, and classify the problems so that the interfaces can be verified in a methodical way.

Without loss of generality, a SoC consists of different types of components, such as hardware cores, processor

cores running software, and/or programmable logic cores, and one or more communication architectures (CA) connecting the components, as shown in Figure 3. The communication architectures may consist of system buses, interrupt lines, and other dedicated communication interfaces between the components. For example, in our image compression SoC, the main components are the picoJava processor core, the DCT component, the DWT component, and the memory; while the PI-BUS, including the bus control unit, constitute a central communication architecture.

The integration of components in a SoC involves designing proper physical interfaces and communication mechanisms between the components. The communication mechanisms consist of both synchronization and dataflow between components, and can be implemented in several ways. For example, synchronization between a HW component and a SW component can be implemented as (a) *interrupt-based*, where the HW component sends an interrupt to the SW component, or (b) *polling-based*, where the SW component polls a system memory address which is set by the HW component when the HW is ready to communicate. All physical interfaces and communication mechanisms need to be verified in the validation process of a SoC.

Next, we present some common integration problems, which can occur either in the physical interfaces or in the communication mechanisms. We have classified the problems, and propose a verification methodology based on the classification.

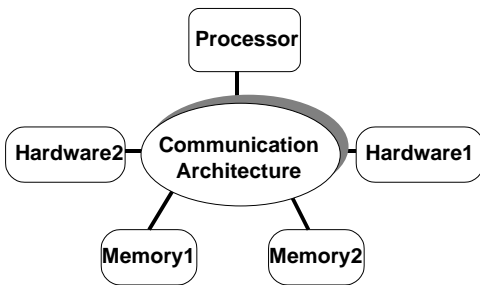


Figure 3. A Generic Architecture of Hw/Sw SoC

Interface between Component and Communication Architecture

The interface between each component and the communication architecture should be validated to ensure that the component can communicate to other components. Potential problems can arise when there is a mismatch between the interface requirement of the component and the interface supported by the communication architecture. We classify these interface problems as Component-To-Communication_Architecture (*COMP2COMM*) problems.

For example, in our image compression SoC shown in Figure 2, the PI-BUS expects the write and address signals in the same clock cycle. It is necessary to validate that the

interfaces of all blocks connected to the PI-BUS exhibit the same write and address signals timing behavior. It is important to note that these interfaces can be validated without resorting to the complete system level simulation.

Interface between Components

After the interface between each component and the communication architecture is validated, the communication between the components, using the specified communication mechanism, should be validated. Communication between two components involves the synchronization of communication transactions between the components as well as the data transfer between them. In validating the communication between two components, we need to test for potential problems in synchronization and data transfer between the components, denoted by Component-To-Component (*COMP2COMP*) problems.

For example, the DCT hardware accelerator unit in our SoC has five control registers which are written by the software (running on picoJava). The DCT component requires four control registers to have valid values when the last control register is updated. Hence, the synchronization of the communications between picoJava and the DCT is important for correctly interfacing between the two components.

Additionally, several problems can arise in transferring data from one component to another. For example, the DCT hardware accelerator is a memory-mapped component connected to the picoJava processor core through the PI-BUS. The bus arbiter of the PI-BUS is configured to channel all transactions in memory range M to the DCT hardware accelerator. The software component of the image compression algorithm, running on the picoJava processor core, should use appropriate addresses (in the range M) to communicate to the DCT hardware accelerator component. A problem we faced in validating the SoC was that the software component was using an incorrect address to interface with the DCT. Note that the interface between components can be validated without a full system simulation.

Interface of Communication Architecture with all Components

After validating the interface between each pair of components, the whole system should be validated for potential interface problems arising because of simultaneous transactions between components. Problems related to conflicts for the system bus, incorrect assignment of priorities to different components by the arbiter, latencies due to other communications, etc., are validated here by simulating the communication architecture along with the interfaces for each component. We represent these problems by a class called Communication (*COMM*).

For example, in our SoC, both the picoJava processor core and the DCT hardware accelerator unit use PI-BUS to access the main memory. The system was designed with

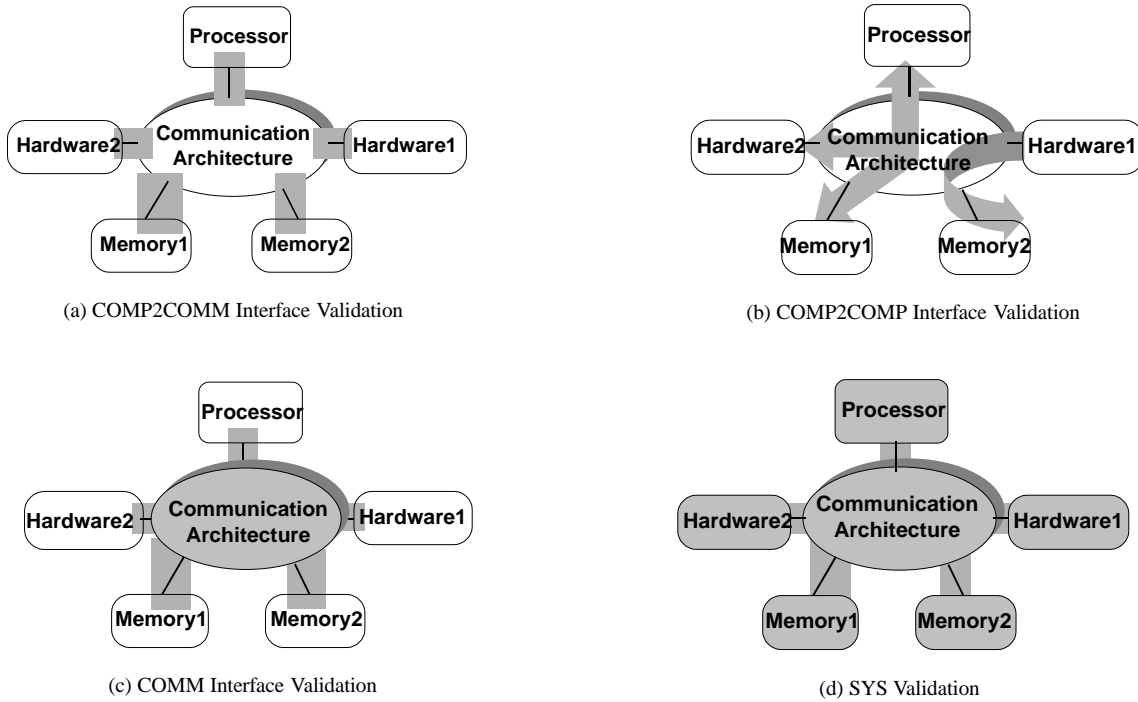


Figure 4. Interface Validation of a SoC

picoJava having higher priority than the DCT component for access to the bus. The correct assignment of priorities should be validated in this step.

Additionally, we faced some problems in validating the SoC because of an incorrect memory map in the bus control unit. As reported earlier, the DCT component as well as the DWT component are memory-mapped components to the picoJava processor through the PI-BUS. While we had validated the interface between the picoJava processor and DCT component, and the picoJava processor and DWT component, independently during the *COMP2COMP* stage, we discovered an error when the picoJava processor, the DCT component and the DWT component were simulated together as the DCT component and the DWT component had overlapping memory maps in the bus control unit.

The validation of the interfaces, as described in this section, verifies the communication between different components using relatively small test-benches. Because test-benches supply a small set of vectors and are generally designed to detect certain types of bugs, the validation steps already listed (*COMP2COMM*, *COMP2COMP*, and *COMM*) may not provide sufficient validation for a SoC design. Hence, for complete validation of the system, simulation of the entire system needs to be done, termed as *SYS* validation. However, the time consumed in the expensive step of complete system simulation can be drastically reduced, if all interfaces have been validated a-priori.

In this section, we presented some of the interface problems and classified them into different groups based on their

dependencies. Based on the classification of interface problems, we propose a validation methodology in the next subsection.

3.1. Interface Based Validation Methodology

In this subsection, we propose a verification methodology which verifies all the interfaces before the complete system level simulation is performed. We assume that the validation of all the components has already taken place (pre-validated cores).

The steps of our validation methodology are shown in Figure 4. First, the *COMP2COMM* interface needs to be verified using a high level test-bench for the communication architecture. The highlighted portion in Figure 4 (a) shows the *COMP2COMM* interface validation. The *COMP2COMP* interfaces are validated next. In this case, a pair of components with the communication architecture is simulated for potential interface problems. The *COMP2COMP* validation is shown by the highlighted portion in Figure 4 (b). A fast co-simulation environment can be used to speed up the validation of the *COMP2COMP* interface between a software component and a hardware component. The *COMM* interface is validated next. In this case, the communication architecture, the interfaces and some abstract models of components can be used to verify the potential problems in the communication architecture. Figure 4(c) shows the components involved in this validation step. After all the interfaces are verified independently the whole system is simulated, as shown in Figure 4(d). This

methodology allows the designer to validate the interfaces between components before a complete system simulation is run, drastically reducing the simulation time required for verification.

To efficiently co-validate software and hardware components, we developed a co-simulation environment for picoJava based SoCs. We describe the co-simulation environment in the next section.

4. Co-simulation Environment

The interface and communication mechanism between a software component and a hardware component can be validated by simulating the software component using a hardware simulation model of the processor. However, simulation using a processor hardware model can be prohibitively slow. As the processor core is pre-validated, hardware simulation can be avoided by using an abstract, Instruction Accurate Simulator (IAS) model of the processor. In this section, we compare the simulation performance of an RTL hardware model of the picoJava processor with an IAS model. We also present a fast and efficient co-simulation environment, using the IAS model of the picoJava processor core, for validating the Hw/Sw interface problems of picoJava based SoCs.

Table 1 shows the time taken (ms) in simulating several test programs provided in the picoJava validation test suite [10], using RTL simulation as well as IAS based simulation. From the results in Table 1, it can be seen that the IAS simulation is several orders of magnitude faster than the RTL simulation.

Table 1. Comparison of IAS model with RTL model

Program	RTL Simulation Time(ms)	IAS Simulation Time(ms)
arithm_int	4460	1.20
dcu_2_7	3960	0.99
fadd_00	109340	46.00
icu_3_1_1	3990	1.10
push_pop_1	11200	7.50

To efficiently use the IAS model of the picoJava processor to validate the interface between software and hardware components, we developed a Hw/Sw co-simulation framework shown in Figure 5. In this framework, the software simulation uses the IAS model of the picoJava processor. For hardware simulation, rather than using a RTL model of the processor, a Fast Interface Model (FIM) is used for simulating the interface of the processor with other components such as the bus interface, interrupts, and timers. However, the FIM hardware simulation must be synchronized with the IAS software simulation for correct Hw/Sw system simulation.

An efficient co-simulation interface has been developed to synchronize between the software simulation (the IAS

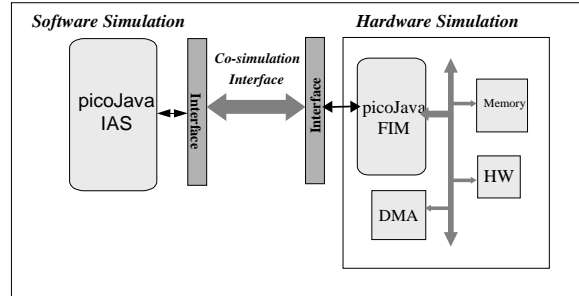


Figure 5. Co-validation Environment for a SoC

model) and the hardware simulation (the FIM model). Whenever external memory is accessed in IAS, IAS sends the request to FIM using the co-simulation interface, which in turn initiates the read/write cycle on the bus and acknowledges back to IAS with the fetched data. Additionally, whenever the FIM gets an interrupt from other hardware components, it is communicated to IAS in software so that the software can execute the appropriate trap handler.

The interfaces between a software component and other components in a picoJava based SoC can be verified very fast using this co-simulation environment as it avoids the time consuming hardware model simulation, and instead simulates only the interface of picoJava with other components in hardware. To show the effectiveness of the proposed co-simulation, we simulated the interface between the picoJava processor, the DCT component and the memory component, using both RTL simulation as well as the co-simulation environment proposed. For a particular image (COWWEI) of size 64x64 pixels, the RTL simulation took 12975 seconds while the system simulation using the co-simulation environment took only 138 seconds. From this and other experimental results that we are getting, co-simulation environment gives nearly 100X improvement in system simulation speed.

5. Experimental Results

In this section, we demonstrate the effectiveness of the proposed verification methodology based on our experiments with the image compression chip presented in section 2. Some of the interface problems described earlier are introduced in the SoC design and the time required to validate the SoC is compared with other methodologies.

We introduce in the SoC one error from each class of interface problems (called ERR1 to ERR3). A possible way of validating the design would be to simulate the complete SoC using some input data (an image in our case). The row SYS in Table 2 corresponds to simulating the SoC with a particular image (CUTWEBEAST) of size 128x128 pixels using RTL simulation as well as co-simulation. Though co-simulation is significantly faster than RTL simulation, over-

all validation time can be prohibitive since detecting an error may need multiple such simulations. The rows ERR1 to ERR3 in Table 2 correspond to validation for detection the errors using interface-specific test-benches. Before commenting on the results, we give a brief description of the errors considered in the design.

ERR1 is an interface problem between the PI-Bus and the memory unit, which is an example of a *COMP2COMM* problem. The PI-Bus expects the data and the data-ready signal at the same clock cycle. However, in the design with ERR1, an error was introduced in the physical interface between the memory and the PI-Bus, causing timing mismatch between the data and the data-ready signal.

Another error, ERR2, is an interface problem of class *COMP2COMP*, between picoJava and the DCT unit. In the SoC design, the communication between picoJava and DCT is performed by a polling based mechanism where picoJava writes to an internal register to start the DCT processing. In ERR2 the polling address of DCT and the address picoJava writes to are different, resulting in wrong functionality.

As described earlier, the DCT and DWT components are memory mapped in the address space of picoJava. In ERR3, we consider an interface problem where there is an overlap in the address space between the DCT component and the external memory. This is an interface problem of class *COMM*.

We simulated each of the above errors using three different simulation methodologies: (1) complete system simulation at RT Level(RTL), (2) system simulation using co-simulation methodology described in section 4 (Cosim), and (3) the proposed interface based simulation (Interface). We report the CPU time (secs) required for validation using above simulation schemes in Table 2.

Table 2. Comparison of Validation Time

Case	Class	Validation Time in sec		
		RTL	Cosim	Interface
ERR1	COMP2COMM	170.1	19.2	3.1
ERR2	COMP2COMP	185.3	25.3	12.0
ERR3	COMM	197.0	29.7	15.2
SYS	SYS	32781.0	518.0	-

It can be seen from Table 2 that the proposed interface based validation methodology is very efficient in finding interface errors in the integration of components in a SoC. Firstly, detecting the interface errors by specifically targeting them with interface specific test-benches (rows ERR1 to ERR3) takes significantly less time compared to simulating the system with arbitrary image data (SYS). This shows that system validation time can be significantly reduced by initially targeting possible interface problems, like the ones presented in this paper.

Secondly, while detecting any interface error, significant improvement in validation time can be achieved by just simulating the required components instead of performing the complete system simulation.

For example, to detect the error ERR1, complete RTL simulation takes 170.1 secs, co-simulation takes 19.2 secs, where as the proposed interface based simulation takes 3.1 secs only, a performance improvement of 6X over co-simulation and 55X over RTL simulation. Similarly, in case of ERR2 and ERR3, the interface based simulation is 2X faster than the co-simulation and 12X faster than RTL simulation. These improvements in simulation time become even more significant when the user has to perform the simulation multiple times to find the errors.

6. Conclusion

In this paper, we address the issue of validating complex core-based SoCs. From practical experience in designing a SoC, we identify several common problems in integration of components of a SoC, and classify them into general interface validation problems. We proposed a methodology to validate the communication mechanisms and physical interfaces between the components before the complete validation of the system is performed. Our experimental results demonstrate the significant reduction in validation time that can be achieved using our proposed methodology.

References

- [1] Synopsys Eagle, <http://www.synopsys.com/products/hwsw/hwsw.html>.
- [2] J. A. Rowson and A. Sangiovanni-Vincentelli, "Interface-Based Design", Proc. of Design Automation Conference, 1997
- [3] Mentor Graphics Seamless, <http://www.mentorgraphics.com/codesign>.
- [4] PicoJava Microprocessor Core, Sun Microsystems Inc, <http://www.sun.com/microelectronics/picoJava/>.
- [5] PI-Bus Toolkit, <http://www.sussex.ac.uk/engg/research/vlsi-Jan97/projects/pibus/>.
- [6] N. Ahmed, T. Natrajan and K. R. Rao, "Discrete cosine transformation", IEEE Trans. on Computers, vol. 23, pp. 90-93, Jan 1974.
- [7] A. Averbuch, D. Lazzar and M. Israli, "Image compression using wavelet transform and memtiresolution decomposition", IEEE Trans. on Image Processing, vol. 5, pp. 4-15, Jan 1966.
- [8] G.K.Wallace "The JPEG still picture compression standard", Communications of the ACM, vol 34, no 4, pp 30-44, 1991.
- [9] Amir Sair and W. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees", IEEE Transactions on Circuits and Systems for Video Technology, vol.6, (no.3), pp. 243-50, June 1996.
- [10] PicoJava-II Verification Guide, Sun Microsystems, March 1999