Simulation-Based Verification in HW/SW Codesign: An FPGA Approach* *

Tony Ewing and Jerzy W. Rozenblit
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, Arizona 85721-0104, USA
{tceljr@ece.arizona.edu}

Abstract

This paper develops design principles for a reconfigurable system to assist in verification of HW/SW systems developed using model-based codesign [1]. One of the key issues in codesign is the verification of specifications and requirements prior to system implementation and deployment. Simulation is used as a means for gathering data necessary to assess if the specifications are met by a proposed design solution. However, one must ensure that hardware, software, and interfaces are all simulated correctly. We argue that the use of an FPGA will enable us to simulate hardware and its interfaces more accurately and efficiently. Our goal is to demonstrate that the use of an in-system FPGA will be much more efficient than using software simulation of hardware to detect timing and other design errors.

1. Introduction and Motivation

The design method of model-based codesign views a system as a source of observable data. If a real system exists, this would be measurements of outputs for the system for a given set of inputs. When designing a new system the input to output relationship is determined by the specifications of the system. A model is a set of instructions that capture the behavior or a subset of the behavior of the desired real system.

When using model-based codesign, it is important to be able to correctly and efficiently simulate the model of the system being designed. A simulator runs the model and allows the model to interact with an experimental frame.[1] The experimental frame allows measurement of the behavior of the model. It obtains the answers to questions about the system being designed. This separation of the model from measurements made on the model allows the same testing apparatus to be used for any proposed model. This means that the measurement of properties of the system is independent of the model used to generate the data. This allows for many different proposed systems to be simulated before partitioning of the system into its components. The experimental frame also allows verification that a refined model (or the real system) behave the same as the specification of the proposed system. The design comes from the iterative process of refining a current model, then determining if the new version meets more of the requirements. During this process different decisions can be compared to determine which solutions are good. When the model is refined enough to be implemented, a system is needed to convert the system into a usable prototype. When this prototype is built, new constraints on the design may be found. These new constraints should be used to improve the model, and the experiment

* This research has been supported by the National Science Foundation under grant No. 9554561 "Hardware/Software Codesign for High Performance Systems"

base. When the final system is deployed, the model used to build it should be available. This model can be placed into a library to help with designing future systems. A user can test the system by designing their own experiments to see if it also meets their needs.

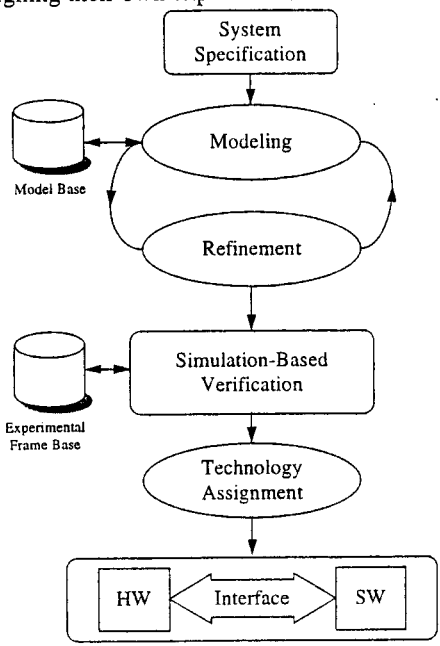

Figure 1: Model-Based Codesign

In the specific field of embedded systems there are some requirements not present in the simulation of software only or hardware only systems. Some of the properties of embedded systems that make simulation difficult are real-time I/O, custom hardware interfaces, and software on custom hardware.

The first property of embedded systems that makes simulation difficult is real-time I/O. This I/O can be both analog and digital, and can be used to control mechanical systems. In a pure software simulation system, the I/O must be simulated along with its effects on the mechanical systems. This forces the experimental frame to be complex. It is required that the experimental frame be an accurate representation of how the real mechanical system responds.

Another property of embedded systems that causes problems is the use of custom hardware that interfaces with I/O signals. Most simulations ignore the interfacing because the virtual signals are easy to interface with. The sensitivity of the interfacing to exact values is not generally tested, because it would make the simulation have multiple copies of the model with slightly different parameters. However, when the real system is built, tolerances on components exist so each real system is slightly different.

Software running on custom hardware forces a simulator to run a virtual machine inside a virtual simulation of a real system. This causes simulations to be slow and not able to interact under the real-time constraints. The interactions between the custom hardware and

the devices it controls are highly dependent on the timing. The simulation must have enough time resolution to capture these interfaces.

There are many commercial simulation tools available including Arena[12], Statemate[13], SES/workbench[14], and Modsim III [15]. These tools are relatively easy to use and can generate models of a wide variety of systems. They are good for testing a concept or determining feasibility of a system. However, they lack in the ability to separate the experiment from the model that is running. It is difficult to rapidly test multiple models and compare the results. Another problem with these tools is that they are hard to interface to a real system.

There are also design environments such as Cadence[7], Mentor Graphics[8], Altera[9], and Xilinx[10]. These environments limit themselves to hardware design. These tools are designed to aid in ASIC VLSI design, or for designing systems to be placed on their (FPGA) hardware. The simulation tools are tightly coupled to the product used. A change from one design tool to another means the entire model needs to be redone in the new design environment. These tools are very good for designing specific hardware, but the testing of software to run on the hardware is difficult and time consuming.

Other systems developed at universities such as the Ptolemy/Gabriel System [2] from UC Berkeley can handle Hardware/Software Codesign. The problem is that the system is specialized to handle DSP systems. This limits the scope of designs to specific hardware before the specifications are fully known for the system. If the system grows beyond the capabilities of a DSP system, another design tool must be used to implement it. Ptolemy does a good job of separating the model from the experiment, but it forces an early partitioning of hardware and software in the model. The Ptolemy system also works with only one level of the design. It would be difficult to start with a conceptual model, then refine it to a working model.

The simulation of mechatronic systems is more difficult due to the real-time demands. The mechanical system must be accurately modeled, or the simulation must be able to handle real-time interactions with a mechanical system. The I/O of mechatronic systems is usually tested only in a virtual simulation, but the I/O is a major problem in implementing the real system.

2. Cosimulation Methods

There are a few different methods of cosimulation of a model. One method is to limit the design to known hardware. Another one is to use a hardware accelerator to perform computations for the experimental frame. Another method is to choose the transducers based on known requirements and interface directly to them. Finally, a unified representation scheme can be used to aid in simulation.

The first method of limiting the design to built hardware limits the design space. It forces the software to handle any design changes that might occur. If a design change occurs that the software cannot handle, the system requirements must be changed. This method really limits the problem to software design.

Using hardware accelerators for the simulation system can be very useful. It allows for more complex designs to be simulated in real-time, or for more complex experimental

frames. The Kress machine [3] is an example of a hardware accelerator that could be used in a simulation system. The Kress machine could be configured to represent a variety of hardware systems, then used to run the software for the custom hardware. This eliminates the virtual machine inside a virtual machine slowdown problem. The physical hardware is reconfigured to match functions of the model.

The method of choosing the transducers for the system, then designing the system limits the amount of change allowed in the original specifications. This method works well when updating or replacing an already existing system. It limits solutions to the same style as the previous solutions for the problem reducing the amount of innovation allowed in finding a better solution.

The final method for assisting cosimulation is using a unified representation scheme. This is a formal method that allows the simulation system to handle only one representation for all systems. The problem is designing a language that can represent both hardware and software efficiently. Many attempts [4,5,6] have been made to describe hardware/software systems using a formal scheme. These attempts have chosen Petri nets, finite state machines, and functional mathematics to try to represent complex systems. These systems are good for proving that the system is correct, but designing a complicated system is very difficult to accomplish. The representation scheme chosen may not easily map onto a real system that can be built. These systems usually have a bias towards one side of hardware/software making it easier to represent certain designs. A good unified modeling schema is not easy to develop.

3. Towards a Cosimulation System

A good cosimulation system would have the following properties:

- *Easy revision or replacement of models*
- *Reconfigurable hardware*
- *Access to digital and analog I/O*
- *Common representation scheme for models*
- *Compiler/synthesis tool that converts model to software/hardware*

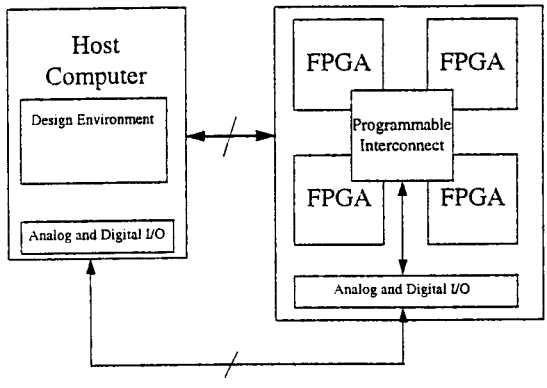

Figure 2: Diagram of Proposed System

We propose that such a system should start with a general purpose computer that has in system programmable FPGA(s) to represent hardware. These FPGAs would be connected using a programmable interconnect. The programmable interconnect allows for hardware module interfaces to be accurately modeled. If the timing is incorrect, the operation of the modeled system will not be correct. The simulation system could get state information from the model by using boundary scan access to the FPGAs. The FPGAs allow custom hardware to be used to accelerate the design. A major problem with software simulations of hardware is the amount of time to run a simulation. This leads to the use of very simple hardware models that do not accurately model the behavior of the system. The use of FPGAs allows the hardware model to be very close to the final implemented model. A conversion from the partitioning onto an FPGA to an ASIC design could readily be accomplished. The use of FPGAs will allow for more accurate simulations that execute faster than a software only model.

Access to digital and analog I/O is important for capturing real-time aspects of the design. When the system is close to being deployed, it could be tested using the actual chosen transducers. This I/O would also be helpful in choosing which transducers are necessary for the system. By using real I/O it is easier to separate the experimental frame from the system being tested. Instead of having perfect signals that exist only inside a simulation, real tools could be used to measure the outputs and compare them to the expected values in the specifications. Using real I/O is closer to the environment of the real system. Early in the design process simulated I/O is useful to determine that the system responds according to the specification. As the design progresses, it is important to validate that the transducers used match the model of them used early in the design process. A system with real I/O can be treated as an experimental frame for the transducers. This means that the same model-based design can be used for making a wider variety of the design decisions. Better specifications can be determined for required transducers, based on a simulation of the system. This will help to limit possibly expensive purchases of transducers that will not work in the final system.

A common representation scheme would need to be able to capture the interface aspects of the system. Previous attempts were good at representing hardware and software, but not the interface or timing that is necessary to build the system. The representation needs to hierarchical to allow for complex designs. Being able to associate multiple models with a single component is useful for different types of simulations. If this part of the model has been tested and is known to be correct, a simpler version could be used so that resources could be concentrated on the unverified parts of the system.

In conjunction with a common representation scheme, compilers or synthesis tools are necessary. These tools should be able to convert from the representation scheme into a variety of software or hardware configurations. A front end that partitions the model for a specific simulation, then converts software to C and hardware to VHDL could be used. The goal is to not force the design into a specific configuration by the choice of design tool. The front end may be able to convert a high level model to a pure software representation, then as the design is refined portions are converted to hardware as acceleration becomes necessary.

References

[1] J. Rozenblit and K. Buchenrieder. "Codesign: An Overview." in *Codesign: Computer-Aided Software/Hardware Engineering*. IEEE Press. 1995 New York. pp. 1-15.

[2] A. Kalavade and E. A. Lee. "Hardware/Software Codesign Using Ptolemy: A Case Study." in *Codesign:Computer-Aided Software/Hardware Engineering*. IEEE Press. 1995 New York. pp. 397-413.

[3] R. Kress. *A Fast Reconfigurable ALU for Xputers*. Universität Kaiserslautern. 1996.

[4] G. Dittrich. "Modeling of Complex Systems Using Hierarchical Petri Nets." in *Codesign:Computer-Aided Software/Hardware Engineering*. IEEE Press. 1995 New York. pp. 128-144.

[5] J. Staunstrup. "Towards a Common Model of Software and Hardware Components." in *Codesign:Computer-Aided Software/Hardware Engineering*. IEEE Press. 1995 New York. pp. 117-127.

[6] R. Boute. "A Declarative Formalism Supporting Hardware/Software Codesign." in *Codesign:Computer-Aided Software/Hardware Engineering*. IEEE Press. 1995 New York. pp. 41-66

[7] Cadence Design Systems, Inc. *Concept: Mixed Level Design Entry System Product Description*. 1997

[8] Mentor Graphics Corporation. *Seamless CVE ™ Product Description*. 1996.

[9] Altera Corp. *Max+PLUS II Getting Started*.1996.

[10] Xilinx, Inc. *Development Systems Product Overview*. 1996.

[11] S. Kumar. *A Unified Representation for Hardware/Software Codesign*. University of Virginia. May 1995.

[12] N.A. Markovitch and D.M. Profozich. "ARENA Software Tutorial." in *Proc. 1996 Winter Simulation Conference*, San Diego, pp. 437-440. 1996.

[13] i-Logix, Inc. *Statemate Technical Overview*. 1995.

[14] Scientific and Engineering Software, Inc. *SES/workbench Introductory Overview*. 1996.

[15] A. Mullarney. "MODSIM III —A Tutorial." in *Proc. 1996 Winter Simulation Conference*, San Diego, pp. 542-546. 1996.