Hardware/Software Co-Simulation

James A. Rowson

Redwood Design Automation Inc.
97 South 2nd Street, San Jose, CA 95113
jimr@redwood.com

Abstract - Hardware-software co-simulation refers to verifying that hardware and software function correctly together. This has traditionally been a task performed after the prototype hardware is available using in-circuit emulators and other techniques. With hardware-software co-design and embedded processors within large single ICs, it is more necessary to verify correct functionality before the hardware is built. This paper surveys the techniques available for co-simulation with an eye toward the strengths and weaknesses of each.

Overview

The available techniques for hardware-software co-simulation trade off between a number of factors, including:

- raw performance (generally of simulation, but sometimes of turnaround time)
- timing accuracy
- model availability
- visibility of internal state for debugging purposes

Raw performance is often at odds with timing accuracy. Many of the co-simulation techniques listed here represent different modelling styles that provide a different accuracy/performance trade off. Performance numbers provided in this paper are based on anecdotal numbers and not on rigorous benchmarking, so your mileage may vary. In all cases, the performance numbers are heavily dependent on how fast the hardware that surrounds the processor simulates.

Model availability is a completely different kind of problem and can often dictate the choice of technique. There are ways to avoid model availability problems using both hardware and software techniques.

Techniques Requiring Models

The most accurate software model, but with the slowest performance, is to use a processor model that has nano-second accurate timing for all the pins plus complete functionality. Because each pin is changing at potentially unique times, many events must be propagated, slowing down the simulation. However, for ringing out the hardware, this is by far the most accurate method. Typical performances for these types of models are in the 1 to 100 instructions per second.

The next most accurate software model provides the correct transitions at each clock edge but without regard for timing (a “zero-delay” or “cycle accurate” model). Internally, this model can be simpler since it is not attempting to schedule each pin separately. Also, there are fewer unique event times in the system, making the simulator run faster. Typical performance for this type of model is in the 50 to 1000 instructions per second.

Faster yet is a model of the processor that only guarantees to emulate the instruction set accurately, which means that the values in registers and memory are correctly modelled. Inaccuracies here revolve around superscalar ordering effects and pipeline stalls that are not modelled. Speed is again improved by ignoring the internal pipelines, hazards, and interlocks. These kind of instruction emulation models can run from 2000 to 20,000 instructions per second.

Techniques Requiring No Model

If the software and hardware communicate through asynchronous communication methods such that time between communications has no effect on functionality, then an even faster method of simulation is available. At this level, there is no need for a processor model. The software is compiled on the host machine and linked with the simulator. The detailed communication between hardware and software is then replaced with a synchronizing handshake. With the software running at the native speed of the workstation, you get the fastest possible version of the software running with the simulation of the hardware. Using this synchronizing handshake technique, the software can run at workstation speeds, measured in MIPS. The overall speed will be dominated by the hardware simulator performance.

One of the fastest, but with the least pretense of accuracy, is to have no hardware model at all, but instead to create a virtual operating system and machine in pure software that has no relation to the real hardware - disk I/O is just mapped through the native operating system, etc. At this level, the hardware is not being debugged at all, just the software. Here again, the software runs at workstation speed, but now the operating system and hardware is also running at that speed.

At the other end of the spectrum, bus functional models of

Table 1: Comparison of Hardware/Software Co-Simulation Techniques

	speed	debug	model	turn-around	sw	hw
nano-second accurate	1-100	best	hardest	fast	ok	yes
cycle accurate	50-1000	excellent	hard	fast	ok	yes
instruction level	2000-20,000	ok	medium	fast	yes	ok
synchronized handshake	limited by hardware sim	no processor state	none	fast	yes	ok
virtual hardware	fast	no processor or hw state	none	fast	yes	no
bus functional	limited by hardware sim	no processor state	easier	fast	no	yes
hardware modeler	10-50	no processor state	timing only	fast	ok	yes
emulation	fast	limited	none	slow	ok	ok

the processor allow the simulation of the hardware, but no simulation of the software. Here, the bus functional model allows the user to create test benches that make sure interfaces are correct. Very elaborate test benches are possible that emulate the traces that will be generated by real software. Because the processor is not fully modelled, it is difficult to measure instructions per second for this technique, but it should be limited more by disk I/O than processing, so might be in the 1000 to 10,000 range.

The traditional way to avoid needing a hand generated software model is to use a hardware modeler. Hardware modelers use an actual part as the model, called by the simulator. Systems based around complex processors often use hardware modelers to emulate the CPU. Current implementations of hardware modelers have the most accurate models from a functionality point of view, but have only modest performance because of the network round trips necessary to integrate the modeler into the simulator. Long simulation runs can also be a problem since the hardware modeler has to reapply the entire stimulus history to get the next vector. One drawback to hardware modelling is the lack of visibility into the processor state during simulation. Hardware modelers typically run in the 10 to 50 instructions per second range, with network round trips making it unlikely to see more than 500.

Perhaps the fastest raw performance comes from doing emulation. Emulators map the hardware down onto programmable hardware that runs only slightly slower than the real hardware (perhaps 1/10th the speed). Most emulators allow the designer to add customization boards for processors, memory, and other standard products. Again, visibility into the internal states of the add-on standard products and limitations on debug access into the emulated hardware can make debugging here approximately the same difficulty as the true prototype. In addition, the turn-around time to make a change to the hardware can be very slow. However, emulation provides the closest to a real prototype that is possible.

Conclusions

Most companies today still use in-circuit emulation to integrate their hardware and software. Model availability dominates those who try to co-simulate. Bus functional models are the least expensive and most readily available, and are the most widely used technique to debug hardware. When a model is available, the nano-second accurate technique is by far the most popular, although only limited diagnostics can be run in software, not entire algorithms. Where models are not readily available, hardware modelers are often used Table 1 summarizes some of the most important characteristics of each technique.

Co-simulation is a relatively unexplored topic. There are a variety of techniques available, each with advantages and disadvantages. Timely model availability will continue to dominate the choice of techniques, until processor vendors release models before silicon (which is starting to happen). Full functionality models will be predominately cycle accurate and instruction level. The synchronized handshake will be used in isolated cases only if the software architecture allows it.