

# Systems Engineering: The Era of the Virtual Processor Model (VPM)

Graham R. Hellestrand

CEO, VaST Systems Technology, Santa Clara, CA, USA.

## Issues in Systems Engineering

Hardware/software systems engineering encompasses various component models: the software model, the hardware model, and the processor model that connects hardware and software. To assess the execution of system models, the dimensions of relevance are simulation performance, timing accuracy, and detail.

## Modeling Systems

Models are constructed and simulate on a *host* computer system, such as an Intel/Win NT or Sun SPARC/Solaris system. These models of hardware-software systems are usually *targeted* at embedded processors, such as the MIPS R4000 and Hitachi SH3.

**Hardware:** In systems, hardware described in a hardware description language (HDL) is simulated using an event/cycle based technique which is slow but has precise timing semantics - simulated models will be timing accurate but take considerable time to execute. This level of modeling can be made more abstract by allowing the HDL code to be replaced by high level language (HLL) code such as C/C++. The result of this is to eliminate the processing of events/cycles with the effect that hardware simulations speed-up by factors of tens to thousands.

**Software:** There are no software models as such. Software compiled to execute on a specific target processor can be executed only by a model of that processor, which may of course be a model of the host processor itself. The technologies used to model target processors are particularly important since they determine the timing and functional accuracy of the software models, the integrity of the hardware/software interface, and the overall performance of the system simulation. The input to the processor model is usually assembly language/object code, but HLL code is also used.

**Processor:** In modern embedded systems where processors may execute billions of instructions across a few tens of seconds of elapsed time, a constraint of *how long the simulation takes* comes into play, along with timing accuracy and functional integrity of the model of the system. Given that the later two conditions must be met, 10 billion software instructions (say 100 elapsed seconds on a real target processor) simulated at 5,000 instructions per second on the host computer system, would take about 24 days of simulating to complete. The processor model is the key to understanding systems simulation.

## Modeling Processors

Practical processor models contain a component that models instruction execution behavior and a component that models input/output (I/O) behavior. The first component takes input written in object, assembly or HLL code and

performs the actions prescribed by that code. This component may be built in a variety of ways, including:

- a *hardware approach* in which an HDL is used to describe internal processor details;
- a *software approach* in which an HLL program decodes and interprets the input code and handles some architectural details (this is how an instruction set simulator (ISS) works); and
- a *hybrid* approach, in which a program analyzes input code and, then using the full (or some elective set of) architectural details of the processor, builds a custom processor model. This is the basis of the static part of the virtual processor model (VPM).

The second component of a processor model, the I/O part, takes *data* transactions generated by the input decode and execution processor component and models the I/O behavior of the real computer. It is in this component that cache, virtual memory (VM), asynchronous events, and the bus are modeled. All of the processor models described so far are capable of accurately reproducing the functional, timing and architectural characteristics of the real processor. However, since ISSs are largely built as pure software models they do not model bus characteristics or processor architectural attributes such as pipelines.

Apart from the HDL, ISS and VPM approaches to processor modeling, using the HLL code designed for the target processor to, instead, execute on the host computer constitutes an approach which we will call the host software (HS) model. The HS approach is an artifice which is only capable of mimicking the target processor's gross functionings.

The attributes of each of the four approaches to modeling processors is summarized in the following Table, in order of their historical. In this table it is assumed that the host computer system is capable of executing about 350 million instructions per second.

Processor Modeling Technology	Functional Accuracy & Detail	Timing Accuracy	Simulation Performance & Notes
<b>HDL</b> <i>Hardware Description Language</i>	<b>Excellent:</b> <ul style="list-style-type: none"> <li>• Full instruction set semantics</li> <li>• Electable levels of detail</li> </ul>	<b>Excellent:</b> Proportional to architectural completeness of target model	<b>0.1-100 instr/sec</b> <ul style="list-style-type: none"> <li>• Potential to be 100% accurate</li> </ul>
<b>HS</b> <i>Host Software</i>	<b>Poor:</b> <ul style="list-style-type: none"> <li>• Models target HLL functioning using host processor running same HLL code</li> <li>• No modeling of target pipeline, cache, VM, or</li> </ul>	<b>None:</b> <ul style="list-style-type: none"> <li>• No intrinsic target processor derived timing</li> <li>• interaction with target bus model forces timing during the bus transaction</li> </ul>	<b>host speed (~250M instr/sec)</b> <ul style="list-style-type: none"> <li>• Some functional and no timing accuracy</li> </ul>

	primary memory		
<b>ISS</b> <i>Instruction Set Simulation</i>	<b>Excellent to Moderate:</b> <ul style="list-style-type: none"> <li>• Instruction set semantics and approximate timings</li> <li>• Pipeline modeling rare</li> </ul>	<b>Good:</b> <ul style="list-style-type: none"> <li>• Restricted by requirement to maintain target instruction history</li> </ul>	<b>2,000-100,000 instr/sec</b> <ul style="list-style-type: none"> <li>• The higher the simulation speed the lower is the detail modeled</li> </ul>
<b>VPM</b> <i>Virtual Processor Model</i>	<b>Excellent:</b> <ul style="list-style-type: none"> <li>• Models target HLL and assembly/object function using host</li> <li>• subject to underlying type representation on host</li> </ul>	<b>Excellent:</b> <ul style="list-style-type: none"> <li>• Proportional to architectural completeness of target model</li> </ul>	<ul style="list-style-type: none"> <li>• <b>~40% host speed (~150M instr/sec)</b></li> <li>• Full modeling of architecture</li> <li>• mixing of VPM and (register) R-VPM models supported</li> </ul>

Figure 1 provides a visual perspective of the various relationships between processor modeling technologies. The VPM technology dominates in the coverage of the design cycle and the speed-accuracy stakes.

### Virtual Processor Models (VPM) Unexpurgated

Since this technology is new we will spend a little time explaining it. VPMs are flexible models with the capability of exposing or suppressing levels of detail. For example, one VPM, known as an R-VPM, exposes all the details of a processor's registers, whereas a regular VPM normally suppresses such detail. The difference in terms of systems modeling is that VPMs simulate about 10 time faster than R-VPMs. Both models hold high precision in timing.

Figure 2 visualizes the structure of a VPM that is partitioned into a static and dynamic part. The static part handles the target processor's instruction set, its timing details, the pipeline behavior and fill, flush, stall, bypass and forwarding policies, exception handling, and instruction and data cache analysis. These models are built after considering code compiled for the target processor. The resulting models and their performance is impressive and is best understood in the analogous context of the models generated by static timing analyzers for circuits. The dynamic component of the model handles instruction and data cache accessing, virtual memory operation, asynchronous event and DMA control, and the processor input/output transactions. The static and dynamic parts of the VPM are *bolted* together to form the full VPM. The static VPM with virtual port I/O structure delivers 150 million instructions per second. The hardware interface is provided through the dynamic VPM and part of this is specified at the HDL level. The effort in effective processor modeling is to minimize the intrusion of the HDL code during simulation.

### Using VPMs

VPMs provide another useful capability. Elements of the static and dynamic model can be optionally included or excluded. Included elements such as

caches, memories and translation lookaside-buffers are parameterizable. This allows great flexibility in modeling processors as cores, selectable catalog components, or models customized for particular usage.

For instance, software engineers are rarely interested in the details of bus transactions when building application code. However, they do wish to see whether control bits have been set correctly in device registers and status registers. Real-time software engineers also are concerned with the reaction time of the system to events. Both of these objectives can be accommodated using a VPM with *virtual ports* which will run at 150 million instructions per second. On the other hand, hardware engineers rarely are interested in running billions of instructions, but do want to ensure that devices plugged into the bus are behaving as designed and communicate and synchronize using proper bus protocols.

Finally, the ability to parameterize VPMs means that experiments to size cache, the translation lookaside buffer of the virtual memory subsystem, and primary memory can be performed in minutes, even when simulating operating systems and large applications programs. Also the ability to rapidly change processors connected into systems via virtual ports, allows the selection of a target processor to be made quantitatively, or even, along with cache and memory sizing, to be delayed to a much later time in the project – which, of course, is the optimal time to make such integration decisions - after most of the software has been developed and the hardware subsystems determined. VPM is the only processor modeling technology capable of supporting such experimentation and *late* choice of critical system components.

### **Back to Systems Engineering and Systems Models**

Now that the *how* of modeling processors has been covered, it is reasonable to ask *so what?* for systems engineering. The *sine qua non*, the *imperative*, factors in modeling systems are: timing accuracy, functional accuracy, selectable levels of (including full) model detail, and simulation performance.

Using these criteria, it is clear that HS and HDL level models cannot deliver systems engineering solutions since no timing capability exists in the former, and performance of the later is glacially slow. So the battle lines appear drawn between the first generation ISS and the new generation VPM technology.

With ISSs incorporating full architectural modeling delivering between 5,000 and 10,000 instruction executions per second, it is clear that modeling large software-hardware systems will take weeks. If ISSs which exclude architectural components such as cache, virtual memory and primary memory, are used then the weeks of simulation reduce to days but there is no longer sufficient quantitative information generated to make decisions.

The VPM technology is the only processor modeling technology capable of delivering simulation performances in the 100+ million instructions/second with accurate timing. It is this that enables full hardware-software system simulation, simulation repeatability and the ready modeling of systems

incorporating multiple heterogeneous processors each running its own operating system and set of applications programs.

In addition the VPM technology relies on standard hardware and software code (C and C++, Verilog and VHDL) to describe systems. This enables software run on a VPM to be directly ported to the real processor target just by cross-compiling, and hardware descriptions to go directly into the currently used tools flow – a necessary condition for the acceptance of this new technology.

Figure 1: **Processor Model Technologies**

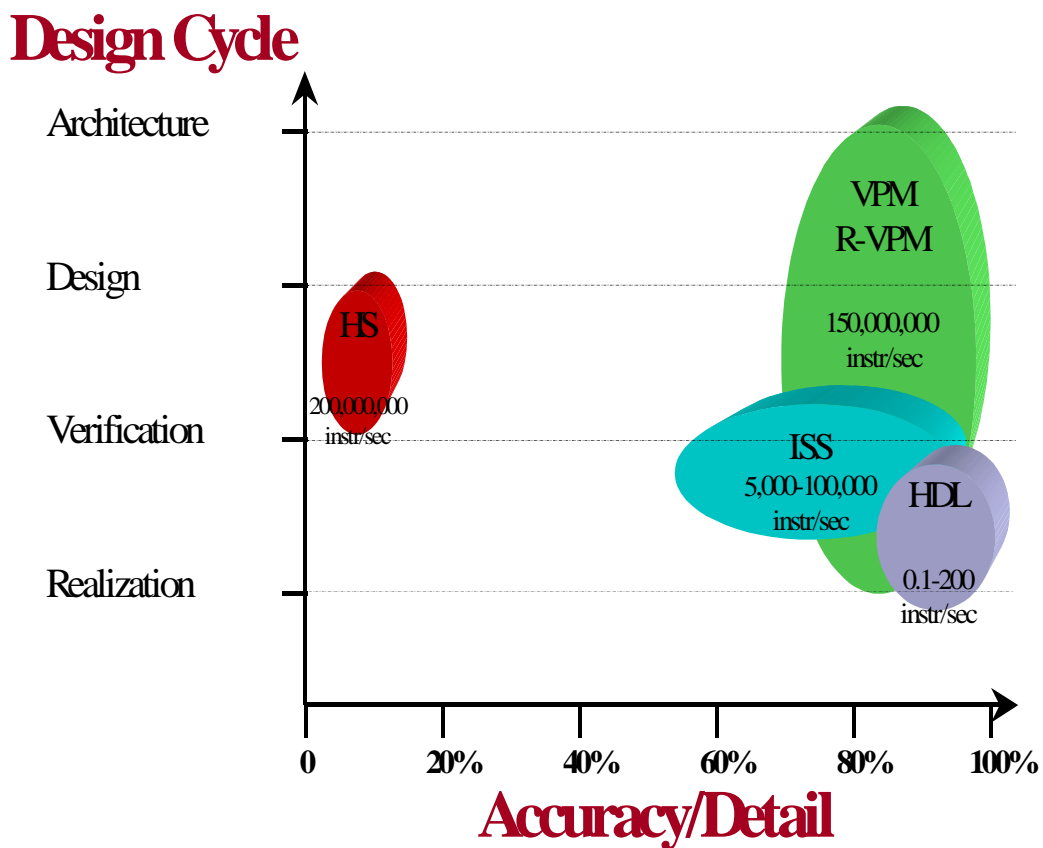


Figure 2: **Virtual Processor Model**

