

The Advent of the Virtual Processor Model

Dr. Graham Hellestrand, CEO, VaST Systems Technology

The growing use of embedded software in traditional hardware designs and the increasing prevalence of the systems-on-a-chip (SOC) technology have accentuated acutely the need to promulgate a system-level vision in which the component parts and their interactions form a coherent and functional whole. Intense time-to-market pressures combined with growing chip complexity and the profound need to communicate effectively within product design groups make the need for the support of the concurrent engineering process compelling.

Because it is the interaction between hardware and software that makes systems engineering so challenging, accurately modeling the processor—which executes the software and mediates communication between hardware and software—has become one of the factors constraining the achievement of fast, detailed and thorough system simulation and prototyping. Thinking carefully about the engineering of systems is critical, yet many engineers are only now beginning to understand the wide spectrum of technologies, design tradeoffs, and engineering and management tasks that are at play during systems engineering. These same factors affect the modeling and prototyping of systems. The advent of the virtual processor model (VPM) dramatically improves the capabilities of modeling of systems. In a future dominated by systems, the engineering of products by *trial and error*, that is physical prototyping is wasteful, inefficient and costly. The capability of rapidly and accurately modeling systems gives the means of building *soft* prototypes as direct substitutes for their physical counterparts. This then directly enables the development, building and testing of complex systems using these accurate models, yielding dramatic affects in product time-to-market, quality improvement, engineering flexibility and component choice, performance, development and production cost.

The Challenge

In the traditional engineering process, the engineering teams face daunting challenges in creating integrated hardware/software systems. The software and hardware development arenas have distinct and incompatible tools as well as markedly different mindsets. In general, hardware and software subsystems are designed not only independently, but also sequentially, with hardware designed and prototyped before serious software development and systems integration begins. In this process the critical system integration task is resolved by default - the software team becomes responsible for the system even as the product's hardware team disbands. This is technically irrational since the difficult integration and system problems invariably require the resolution of hardware and software design incompatibilities, but economically mandatory. This incongruity in the traditional system engineering process rationale can only be eliminated by the use of development tools in which an accurate model becomes the *gold* system

architecture governing the design and development of both hardware and software subsystems. This approach eliminates the communication gap between the hardware and software design groups and enables the concurrent engineering of systems from the architectural assessment phase through to sign-off for manufacture.

Currently, the traditional process is made to work for smaller, less integrated designs, but provides little flexibility to adapt to changing project size and complexity. As modern systems become large, highly integrated and dominated by software complexity, a process with inadequate communication and inherent inconsistency inevitably results in the factoring in of 5 or more physical prototypes and long development cycles for products. This is arguably tolerable for PCB based products but is catastrophic for systems-on-silicon. For the latter, the cost in money, personnel and time of producing many physical prototypes dominates the product development cost. This in turn has forced engineers to compromise designs and resulted in products reaching markets late and missing critical market windows entirely.

Processor Modeling Alternatives

A systems modeling technology must have sufficient competence to enable the deployment of *soft* system prototypes to underpin a concurrent process governing the engineering of systems. What has been conspicuously missing from the traditional modeling technology is an efficacious way of modeling the processor in a soft system prototype. Because it sits between the software and the hardware portions of the design and mediates communication across that boundary, the processor model largely determines the performance, accuracy, flexibility, adaptability and utility of the soft system prototype in modern embedded systems development. It is important then to consider the merits of the four processor modeling approaches: hardware description, instruction set interpretation, host software execution, and architectural mapping.

Hardware description of internal processor details using a hardware description language (HDL) constitutes this modeling approach which was the earliest of the processor modeling techniques. The most significant advantage is in accuracy, function and timing of internal processor details. This technique admits to levels of abstraction where internal details can be traded for simulation performance. However, the tie to event driven/cycle based simulators makes the model execution very slow. This technique provides simulation speeds of 1/10 to ~200 assembly/object instructions executed every second. The low speed limits this modeling technique to use in verifying some aspects of new processor architectures. One interesting facet of this approach is that it integrates easily with the simulation of hardware, since hardware is modeled using the same process. Software engineers and system architects do not typically employ such modeling techniques.

The second approach uses instruction set simulation to model the behavior of processors. In this mode most of the internal details of the processor are suppressed and only the gross *instruction execution* behavior is modeled. To simulate a system involving hardware and software, a sophisticated linkage needs to be made between the instruction set simulator (ISS) and the hardware simulation environment. This interface limits the interactions between the hardware and software parts of the system model, as well the time bases of the hardware and software simulators are not the same.

ISS's are useful for hardware engineers writing and verifying device drivers and can be used by software engineers in much the same way as an *in-circuit emulator*, albeit with considerably less performance. ISS's execute between 2,000 and 500,000 assembly/object level instructions per second – the former when heavily interacting with the hardware model. For software engineers developing applications to run on embedded systems which execute 100's of millions of instructions per second, being constrained by an ISS model of a processor means that 60 seconds of the software executing on the real target system will take between 1.5 million seconds (36 days) and 12,000 seconds (3 hours) to model. Neither delay is acceptable to software engineers and the reality is that the 36 days of simulation is closer to what can be expected. As with all processor models, ISS may model architectural elements such as cache, pipeline, pipeline hazards, and virtual memory, but the incorporation of these elements further degrades performance considerably. ISS modeling clearly creates a software execution “choke point” in the hardware/software co-simulation process.

The third processor modeling technique, called “host software execution” (HSE), is perhaps the least attractive of the alternatives. The HSE approach uses the high level language (HLL) code that has been designed for the target processor and simply executes it on the host computer instead. This is an easy solution to implement, but it is inadequate. Most significantly because the software is not executing on the target processor, the model can only mimic gross functioning, and it cannot provide timing for the simulation. It also has no ability to model the target pipeline, cache, virtual memory, or primary memory. However, such processor models can be made to communicate with hardware system elements via, so called, *bus functional models* described using an HDL. When the bus functional model is executing the transactions it performs are timing accurate. The HSE advantage is that software will execute at host processor speeds – maybe 250 million instructions per second. However, since no timing or architectural element modeling (cache, virtual memory, etc.) is possible, this technique is unacceptable as a systems engineering tool.

Mapping Architectures: Virtual Processor Modeling

This brings us to the fourth approach which maps the target processor's architecture into an executable model – the virtual processor model – which in turn executes the target code.

The VPM is fast primarily because it is composed of two parts. In the first, which models the instruction execution behavior, an analyzer builds a custom virtual processor model (VPM) based on all or some elective subset of the architectural elements required in the processor, from the target code. This static analysis is analogous to 'static timing analysis' in circuit simulation and the resulting model runs very fast. The code executed by a VPM may be HLL C/C++ code or assembly/object level code.

The second portion models the dynamic parts of the processor, those portions whose function cannot be determined prior to simulation. This includes the I/O parts of the processor that communicate with the hardware: cache, virtual memory, interrupts, bus signals, and the like. For obvious reasons, the simulation speed on this portion is limited by the level of detail modeled, and, where communication with hardware occurs, the speed of the hardware simulator during that communication.

With a VPM, it is also possible to select the architectural elements and the level of detail modeled in both the dynamic and static portions of the design. In this way, processors can be customized for a particular use, or modeled as cores or selectable catalog components. This feature is especially helpful given the differing concerns of engineers. Software engineers on a typical project do not care about the details of bus transactions when building application code, but they do want to know if control bits have been set correctly in device and status registers. At the same time, real-time software engineers need extremely high timing accuracy, but care less about function. Sometimes the two come together, as when a complex, concurrent system will lose functional accuracy if the timing is flawed. Finally, hardware engineers come with a different perspective altogether. They are rarely interested in running billions of instructions, but they do want to ensure that devices plugged into the bus behave as designed and communicate and synchronize using proper bus protocols.

A VPM allows modeling flexibility for hardware, software and architectural engineers, providing accuracy where it is needed, and trading detail for simulation speed at the election of the design engineers. A VPM with *virtual port* (memory mapped) input-output typically executes 150 million instructions per second on a 400MHz host, without sacrificing accuracy in function or timing. The VPM is by far the fastest, most accurate, and most malleable approach to processor modeling, and as such effectively addresses the requirement in the concurrent engineering process to execute software at sufficiently high speed and precision to enable soft system prototypes to stand in stead of physical prototypes. It is the only processor modeling technology that truly makes sense for complex hardware-software system tools.

Concurrent Engineering

With the availability of flexible VPMs, system engineers are able to bring a great deal of the innovative system design technology to bear very early in the design cycle. As mentioned earlier, one of the major drawbacks of the traditional design approach is the inability of different engineering groups to communicate with one another and make trade-offs between different design possibilities as the project moves through its various phases. This may result in sub-optimal designs based on inadequate investigation of alternative architectures, poor configuration analysis, and little knowledge of critical performance constraints that might affect the later stages of the design process. The consequences—technical and fiscal—can often be negative and severe.

VPM's add two things to this process. First, they provide modeling flexibility, to permit changes as problems are discovered. This stems primarily from the speed, flexibility, and accuracy of VPM's. Without having to commit to silicon, engineers can make critical decisions in hardware/software partitioning, processor configuration, and choice of operating system quantitatively and much earlier in the design cycle, when trade-offs can be without incurring severe re-engineering penalties.

Soft prototyping using VPMs affords particular advantage to the software group, which has typically had to wait to test software alternatives until very late in the design cycle, after a physical prototype has been created. Fixing problems in the physical prototype is difficult not only because the prototype itself is expensive to respin, but also because the change occurs so far along in the process that it may readily produce a cascade of consequent changing. Furthermore, as more and more software is embedded in the chip itself, it becomes just as difficult to correct as hardware. VPM's give the software engineers a model fast and flexible enough to target for simulation, thus helping to avoid these late design changes.

Second, VPM's provide a means of quantifying systems decisions, which gives the system architect a precise mechanism for communicating engineering change decisions to the software and hardware groups. In contrast to the manual system that has held sway until relatively recently, this quantifiability lends the design process a rigor it has lacked, and eliminates errors that can easily become very costly.

Conclusion

Concurrent engineering—the simultaneous design of hardware and software systems—is unavoidable as time to market shrinks, the amount of software embedded in systems explodes, and system complexity multiplies. Since the processor executes software and mediates the communication between the hardware and the software, concurrent engineering is only possible if the processor model is fast, accurate and flexible enough to permit a soft prototype to efficaciously stand in stead of a hardware prototype. Only virtual processor models can provide the features required to support concurrent engineering via soft prototypes. To those working with leading-edge designs, such as

system on a chip, VPM's will become a standard part of the modeling arsenal and a central element governing the systems design process.

Sidebar 1: Choosing Hardware Software Systems Engineering Tools

Not every designer needs to be concerned with processor modeling. Designs that will benefit most are those that have one or more processors, a high level of application or middleware code driving the processor, a complex ordering of operations for the application code by the middleware, and (for real-time systems) a timing responsiveness requirement to an external system.

When choosing a tool, the primary criterion is the tool's functional and timing accuracy. Each is critical in certain circumstances. An engineer verifying real-time systems requires a great deal of timing accuracy, while someone verifying a system on chip needs to have a great deal of functional accuracy. Where these two needs intersect is in a complex, concurrent system, when sacrificing timing accuracy means sacrificing functional accuracy.

Another criterion is the tool's performance. For systems in areas like telecommunications or embedded graphics (where a large amount of data is processed), sheer speed of simulation becomes very important. For example, in a telecommunication system it is not uncommon for errors to occur after processing 1,000 packets of data. If it takes a week for designers to simulate such a system, then a systems engineering tool's usefulness is limited.

Other important criteria to consider include: compatibility with existing tool flows; operation with standard HDL and software HLL languages; and the level of inter-operation with other in the tool flow.

Sidebar 2: Integrating a Systems Engineering Tool into a Design Methodology

From a methodology point of view, integrating a hardware-software systems engineering tool requires the system architect to produce and justify a quantifiable model of the system that then becomes a sort of *gold* standard for the hardware and software design groups. All modifications to that system must be reflected in that standard. An integration along these lines gives the benefits of enabling the hardware and software groups to work concurrently on a design whose architecture has been selected quantitatively. And binding the architects of the system into a continuing commitment to actively control modifications to it and its family of derivative systems.

Organizationally, integrating a systems engineering tool will ideally start at the architecture level, because system experimentation provides the highest payoff in a

systems engineering and verification strategy. Systems engineering tools permits testing the feasibility and complexity of candidate systems rapidly, including choices of processors, sizing of caches, and estimation of the power required to drive devices. After deciding the system architecture, the design and implementation must be driven as much by the hardware group as by the software group. For systems containing a significant amount of software, the software components dominate the complexity of the system, while the hardware components dominate the detail. Consequently, the software engineers dominates the system personnel budget, but the hardware engineers control the purse strings for historical reasons. More equitable allocation of resources and decision making amongst systems architects, software engineers and hardware engineers will lead to better tools purchasing decisions and improved efficiency in the design process and hence the organization.