

Designing *System on a Chip* Products using Systems Engineering Tools

Graham R. Hellestrand

CEO and President, VaST Systems Technology Corporation
2700 Augustine Drive, Santa Clara, CA 95054 USA

Abstract

Systems engineering is the process which takes requirements specifications and engineers products and product families which involve hardware, software and possibly mechanical subsystems. At the front-end of this process architectural assessment and early quantification is a requirement – answering the *what-if* questions about the candidate architectures of a product. At the back-end, verification and realization of the selected architecture occurs. Tools to support systems engineering encompass architectural assessment, co-design and co-verification and feed into the synthesis and realization tools flow. Systems on silicon products are reliant on systems engineering tools to enable the concurrent design of hardware and software and their modeling and verification prior to realization. This paper describes the systems engineering process and the requirements for tools support

1. OVERVIEW

Systems on a chip are no longer hardware devices. The incorporation of wrong software in a ROM just as surely requires *respining* the silicon as a floating carry chain in an adder. The complexity of modern systems on silicon is dominated by the software complexity – device drivers, real-time operating systems, application programming interfaces (API), middle-ware, applications tasks all executing and intercommunicating. The hardware complexity is increasingly dominated by multiple processor architectures which may incorporate two or more general purpose processors and (usually programmable) signal processing devices. Such formidable complexity is typically beyond the comprehension of single engineers and the division of design groups, usually into hardware and software teams, leaves a yawning chasm with which to address system verification and integration. The 3 to 5 silicon spins, typically scheduled into project plans, attest to the methodological and technical difficulties of systems integration and verification.

The advent of systems engineering tools, which grew from the base of the predecessor co-verification and co-design

tool sets, is a response to the needs of building systems on a chip efficaciously. This paper examines the hardware-software engineering process and discusses requirements for tools to support the engineering of systems on silicon. The paper ends with a brief summary.

2. THE HARDWARE-SOFTWARE ENGINEERING PROCESS

The fundamental activities involved in hardware-software systems design, as shown in Figure 1, are: requirements derivation, specification, architectural assessment and quantification, design, verification, and realization, with the usual levels of iteration to support the predilections of human problem solving. The process steps constituting each activity are largely different, but by traversing the process steps from specification to realization a top-down methodology is evident; by reversing the traverse direction a bottom-up methodology is mapped out. The first three processes (Specification & Architectural Assessment, CoDesign and CoVerification) are the newest to be addressed by tools.

Coverification has been the work-horse of hardware-software technologies; the initiator of and developer of a young and

vigorously growing market. Coverification tools require modeling precision across the hardware-software

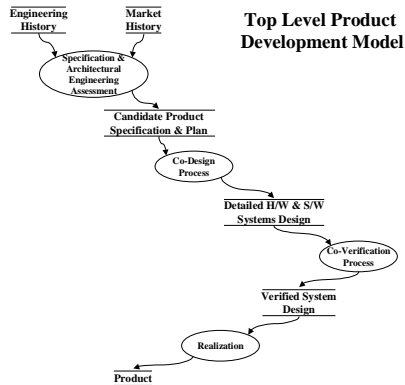


Figure 1: The System Development Process

boundary and the ability to run limited sequences of software to verify software device drivers interacting with hardware controllers. **Codesign** tools aim to push hardware-software tools into the design phase of the process but require higher level software languages than assembly, and greater speed than, and as much precision as, coverification tools. The historic limitation here has been the performance-precision tradeoff of the CPU models interpreting software: the very low performance, high complexity, yet high precision of hardware description language (HDL) models; the low to medium performance and limited flexibility of instruction set simulation (ISS) models; and the high performance but complete lack of precision of the C based software processor models. The CPU models limit the number of software cycles which can be properly verified running on a hardware-software modeling system to a few thousand per second whereas real systems execute 50 - 500 million instructions per second – a disparity of 10,000 to 100,000 in performance in which 1 second of real-time performance may require from 3 to 30 hours of simulation.

Systems Engineering tools push the codesign envelope to address the quantification of architectural attributes of a system. From Figure 1 this requires tools

to quantify system architectures. This process has rarely been explicit except as a fuzzy qualitative step. The framing and answering of the myriad *what-if* questions that qualify the selection of a few candidate architectures and result in the discarding of many, is fundamental to the process – its steps are depicted in Figure 2. This creates a yet heavier demand on tools, since CPU issues such as pipeline behavior, instruction and data cache sizing, virtual memory mapping, and interrupt handling must be capable of being modeled over hundreds of millions of instruction cycles in short periods of time. And, as well, the accurate modeling of timing and functional behavior of system components (such as buses, DMA handlers) and peripheral devices being driven by application programs interacting with the hardware through complex operating systems may require the processing and reduction of many seconds of real-time data in a single simulation run. In today's engineering environment it is unlikely that hundreds of hours of simulation time for a single data analysis would be tolerated by R&D engineers.

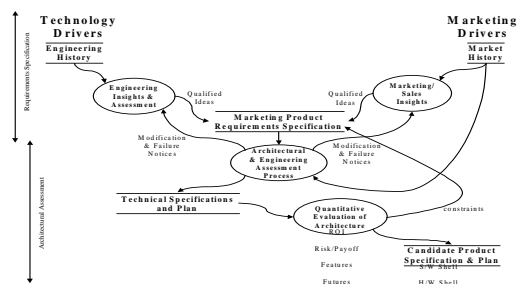


Figure 2: Specification and Architectural Engineering

The more-than trivial automatic partitioning of systems into hardware and software components requires the use of an unbiased (with regard to hardware or software) specification notation. The engineering community currently eschews the use of these tools due to their newness, lack of practicality, and unfamiliarity; this paper does not address this topic.

3. TOOLS COVERAGE OF THE SYSTEMS ENGINEERING HIERARCHY

Referring to Figure 3, Systems Engineering spans a very broad range of activities, from architectural assessment through verification. Systems engineering subsumes co-design and co-verification and pushes tools use into the arena of systems analysis (using such tools as MathLab™) and architectural assessment early in the design process and enables major structural change and re-evaluation late in the design cycle - cache sizing, the effect of incorporating different operating systems and the decision to incorporate hardware or software implementations of special processors into a product or family of products.

processors and control systems. The speed and accuracy of the co-verification tools is limited by the models used for CPUs, which are the executors of software and form the bridge between the software world and the hardware world. On the other hand, the push into hardware-software design requires a different set of tools which support the design process and feed into the co-verification process. Higher software simulation performance, than the co-verification tools have been able to deliver, is required to enable the edit-compile-modeling cycle times demanded by software engineers.

The key to simulating the hardware-software interaction is the model used for the CPU, be it general or special purpose. At the co-verification level, hardware

Figure 3: Hardware-Software Engineering

Levels of Abstraction	Actions	Point Tools	Target Deliverables	Emulation	Co-Verification	Co-Design	Systems-Engineering
Requirements Specification							
Technical & Architectural Specification	Market study Product pricing, delivery Competitive analysis Assess trends	Spreadsheet Slideware Word processor	Product requirements				
Architectural & Engineering Assessment	Partitioning Modeling, evaluation Risk assessment		Candidate architectures				
Co-Design	Use HDL language Synthesis Generators Floorplanning Version/configuration	Verilog VHDL Design compiler Floorplanner	HW/SW design planning				
Co-Verification	Compilation/execution Simulation Execution Signal integrity analysis Timing analysis Test generation	Simulator Emulator Various analysis tools ATPG Fault simulation	Design verification complete	X			
Realization	Place and route Floorplanning Build breadboard	Layout tools Floorplanner	Prototype or board chip	X			

Tools which support hardware-software design need to simulate both hardware and software and to preserve timing accuracy of the simulated hardware, the software and the interactions across the hardware-software interface. Co-verification is a term which encompasses many tools from the system level through to the circuit level. For detailed simulation the extraction of circuit characteristics is important in the determination of the maximum operating frequencies of

description language (HDL) models and instruction set simulators (ISS) models of the CPU are used. HDL CPU models execute assembly/object level instructions at $1/10^{\text{th}}$ to $\sim 1,000$ instructions per second depending on the level of detail and the complexity of the CPU being modeled. ISS models can process upwards of 100,000 assembly/object instructions per second in a cycle accurate mode but rarely provide a full architectural model of a CPU, especially complex pipelined processors. The modeling of the complex

infrastructures that interface CPUs to their system environment limits the overall speed of simulation to perhaps 5,000 instructions per second.

The reaction to the requirement to provide higher simulation performance for co-design was to incorporate *virtual software* CPU models that run C-code on the host processor, but at the huge cost of sacrificing hardware-software timing. Another class of tool which addresses the modeling of hardware and software are the hardware emulators (typically field programmable gate array based systems). Such tools provide fast hardware modeling (in comparison to HDL based simulators) and when used to model CPUs deliver software executions of 1-10 million instructions per second. But emulators are expensive, require large non-recoverable engineering investments to get going, and do not provide good modeling or timing of hardware circuitry. Processor emulators are essentially software development tools. The building of multiple processor systems becomes a formidable undertaking.

The last class of CPU model is the so-called Virtual Processor Model (VPM™) which uses detailed target CPU architectural analysis of C/C++ and assembly level code to build a *static* model of the target processor and then incorporates the dynamic infrastructure components required to deliver precise timing and high simulation performance - better than 150 million target CPU instructions per second. Such models provide configurability and accuracy and faithfully reproduce target CPU behavior and timing. However, without care in representing system components which are subject to event/cycle based HDL simulation, the high performance of the VPM™ can be degraded severely. Timing and function precision is always maintained. Such models enable the realistic modeling of complete systems where billions of instructions may execute in a complex hardware environment. VPM™s are not restricted to general purpose CPUs, DSP and special purpose

programmable processors are readily modeled.

System engineering tools needs to be flexible and configurable enough to support the diverse objectives of algorithmic analysis, architectural assessment, co-design and co-verification. Supporting these objectives is a tremendous challenge which is beginning to be addressed by a new generation of tools.

4. MODELING SYSTEMS

Systems engineering tools needs to be capable of architectural analysis and the ready incorporation of complex designs and algorithms into the modeling environment. A typical modern target system is depicted in Figure 4. This system supports 3 separate processors interacting through common hardware.

Within a general purpose processor system the software and software-hardware environments are themselves extremely complex.

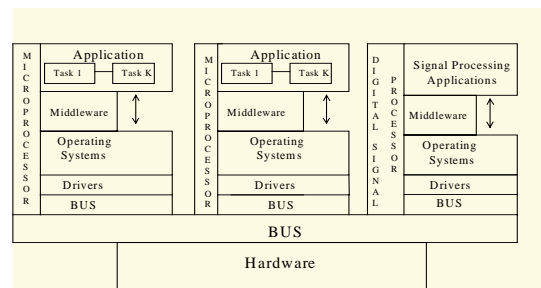


Figure 4: Modeling a multi-processor system

4.1 Choice of General and Special Processing Units (CPUs & DSPs)

An early decision made in most hardware-software projects is the choice of processor. The primary reasons for this: prototyping the hardware as a mechanism for validation, and facilitating the early development of software. The availability of virtual system prototypes, based on VPM™s enables the detailed design and validation of hardware concurrently with the development of software. A major consequence is that decisions about processors, processor infra-structure

devices (caches, virtual memory organization, memory, buses, etc.) and even operating systems can be deferred until late in the project. This is a major factor for systems on a chip (SOC) where silicon area is frequently dominated by cache and other memory elements.

The ability to model algorithms as software or hardware elements or as programs executing on a selection of programmable devices (general and special purpose, for instance DSP, processors), facilitates the accurate evaluation of various partitionings between hardware and software. Accuracy of function and timing needs to be guaranteed from all of these models. In addition, the ability to rapidly change processor models, to reconfigure cache and memory, and to rapidly simulate systems incorporating complex hardware and software accurately afford great flexibility both early and late in the engineering process. Providing a choice mechanism which can be quantified removes one of the substantial sources of risk from system architecture and design.

4.2 Application Specific Hardware

With very fast software simulation being available via VPMTMs, careful attention must be paid to the modeling of hardware since the performance of even the best event/cycle based hardware simulators is comparatively glacial. The selective re-expression of hardware designed using HDLs into some high level general purpose language (HLL) helps to defeat the slowness of these simulators. Simulation speed improvements of up to 2,000 times have been reported for moderate (~300k gate) designs.

Two issues arise from this approach to hardware modeling: does the re-expressed description faithfully reproduce the functional and timing behavior of the original, and how much effort is required to perform the re-expression. From theoretical reasoning, the answer to the first question is *yes* providing the critical concurrency of the hardware description remains in the HDL and the HLL has a set

of library routines which facilitate suspension and resumption of the software when delays and synchronization are required. The modeling of data dependent path-delays requires some effort. The answer to the second question depends on the design being re-expressed. For the moderately complex design cited above, the re-expression and its validation required less than 5% of the original synthesizable RT HDL design time.

In general, the selected expression of hardware in a mixture of HDL and HLL code enables the flexibility to trade simulation performance for modeling detail. This is a valuable approach, when after validation of the detailed design, the abstraction leaves an accurate kernel description of the design and a large boost in simulation performance. Programmed interfaces to hardware can be verified using this approach, even when the inherent disparity between hardware and software simulation performances is 4-5 orders of magnitude.

4.3 Operating Systems & Middleware

The choice of operating systems in embedded and real-time systems is usually made early, along with the processor. This choice locks software design and system performance in very early – a risky strategy in a world in which technology rapidly changes and corporate histories are measured in short years. This decision can be delayed provided some standards in regard to application programming interfaces (APIs), operating system services, and device driver interfaces are observed, and these are consistent with the offerings of the candidate which need to be available for the set of candidate processors.

The ability to regard the operating system as any other piece of selectable intellectual property (IP), which can be quantitatively evaluated provides an unusual but valuable option in the building of complex systems.

Middleware is that layer of software sitting above the operating system that

provides additional services - either user or system. Examples of middleware are networking services (for example TCP/IP stacks, packet switching services), graphical and user interface services, and application specific services. Middleware often assumes that multiple systems will be interconnected placing an even heavier demand on the verification process, which may require the transmission, analysis and modeling of thousands of transactions containing thousands of bytes of data.

It is necessary to be able to accurately model both operating systems and middleware service. There are several approaches to such modeling. The first is emulation by which a target system is modeled through an existing system, and the emulated characteristics approach the behavior and timing of the target system. A second approach is the simulation of behavior, which largely precludes the accurate modeling of timing. A third approach is direct VPM™ execution of the actual operating system and middleware on the selection of target processors. The last approach guarantees accuracy of both timing and behavior.

4.4 Applications Programs

Application code requires similar support to middle-ware. Multiple tasking, inter-task communication and synchronization and the use of substantial libraries and APIs demand the existence of such support in a modeling system. With applications programs interacting with middleware and operating systems, and driving systems resources (such as hardware devices) is application driven ways, the requirement for fast and timing accurate modeling for reactive, real-time and embedded systems is clear. Here all levels of the system must be modeled properly in regard to function and timing to ensure correct, robust and satisfactory operation of the realized product.

5. SUMMARY

In this investigation the overall hardware-software engineering process was discussed, from specification and architectural assessment to verification

and realization. In modern, hardware-software systems the requirement to run full virtual hardware-software prototypes at high speed with full function and timing accuracy in order to assess architectures early in the design cycle, to size and configure critical components late in the design cycle (when *real* hardware and software designs exist), and to co-design and co-verify the system iteratively throughout the engineering process is mandatory.

The products of the next millenium which facilitate transport, energy generation and distribution, communication, medical prosthetics and general control, will have demanding functional and performance requirements, and unprecedented need for correct, safe and reliable operations. Such formidable technical requirements require a new approach to hardware-software modeling, design and architectural assessment. It is a requirement which must be met and a new generation of systems engineering tools is becoming available to fulfil the dream.