

Engineering Systems on a Chip:

The Bloody Revolution in Tools, Methodologies and Power

Graham R. Hellestrand

Vast Systems Technology Corporation
1230 Oakmead Pkwy, Sunnyvale, CA, USA

Abstract

Systems engineering is not a quiet backwater. It is an arena of political and economic realignment and foment. And like any revolution in-progress it is the harbinger of great and unpredictable change bringing with it huge opportunities hand-in-hand with equally impressive threats. Nothing less than corporate power and bureaucratic structures are being reordered by the new realities of engineering systems on silicon. The 30 years of hardware dominance in silicon electronic engineering is being torn apart by the recognition that the complexity of modern systems is determined by its multi-functionality, adaptability and flexibility – attributes that, in an economic sense, are best realized in software.

The genesis of this revolution has been the stunning success of the silicon engineers, which ironically, as with the yin-yang cycle, has carried with it the seeds of its own diminishment, at least for the half-turn of the next cycle. As silicon technological progress marches through 0.18 μ minimum feature sizes to 0.15 μ to 0.13 μ in the next couple of years, and then to sub 0.10 μ , the number of transistors on a chip will approach, then exceed, one billion. The majority of these transistors will be consumed as *on chip* memory devices. Memory is most useful in programmed devices and with processors executing in excess of a billion instructions per second when implemented in 0.1 μ silicon technology, programmed devices will progressively phase out many special purpose hardware devices.

The dismantling of the power base in the semiconductor divisions of the major electronics companies to accommodate software engineering and systems integration on an equal footing with hardware engineering is a battle in full flight. Like the Iliad, where Greek's fiercely and intermittently battled Trojans for ten long years, the outcome of the systems engineering battle is certain. Unlike the complete destruction of the city of Troy, the resulting realignment will be a triumvirate between – hardware, software and mechanical designers. The sooner the battle is won the sooner the potential of the three powerful potentates will yield novel systems and architectures to dazzle the techno-energized masses. Like all power-sharing structures, it is unstable but ultimately governable by the dour and pragmatic economics of survival. Already fleet-footed start-up companies are demonstrating the fecundity of the new godhead – *carpe diem!*

1. ISSUES IN SYSTEMS ENGINEERING

Systems engineering is a process [Hellestrand 1999c]. It takes requirements specifications and generates products and product families which are realized as

an integration of electronic hardware, software and typically mechanical subsystems. Systems level products have been built largely using circuit board substrates in a traditional sequential engineering process being aided by distinct tools support for hardware, software, and mechanical subsystem development. The integration process is largely manual but with some support for debugging the software-hardware-mechanical interfaces using in-circuit emulation (ICE) equipment. Hardware, software and mechanical development usually follow each other in this process rather like the phases of the moon. And unfortunately, the iterative debugging of hardware-software-mechanical systems could readily be counted in units of lunar cycles (LC) – usually 3-5 LCs for products of medium complexity.

With the advent of system products being integrated on monolithic silicon substrate the tools support and development methodology games change forever [Hellestrand 1999a]. Systems on silicon products require systems engineering tools which enable the concurrent design, modeling, verification and integration of hardware and software and mechanical subsystems prior to fabrication. This is a radical change which is causing the complete reorganization and reorientation of systems engineering teams and processes – in short a revolution.

The economics of silicon systems manufacturing have always dictated that silicon products are to be consumed by mass markets or can command very high premiums in niche markets, in order to cover development and manufacturing costs. The technological capabilities of the dawning 21st century are driving successful niche silicon products to being components (common intellectual property) incorporated into generic silicon systems destined for consumption in mass markets.

1.1 Systems Engineering as a 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 vagaries of human problem solving. The process steps making up each activity are largely different, but by traversing them from specification to realization a top-down methodology flows; by reversing the traverse direction a bottom-up methodology is evinced. The first three processes (Specification & Architectural Engineering Assessment, CoDesign and CoVerification) are the newest to be addressed by tools. Tools to support systems engineering encompass specification and architectural assessment, co-design and co-verification and feed into the established synthesis and realization tools flow.

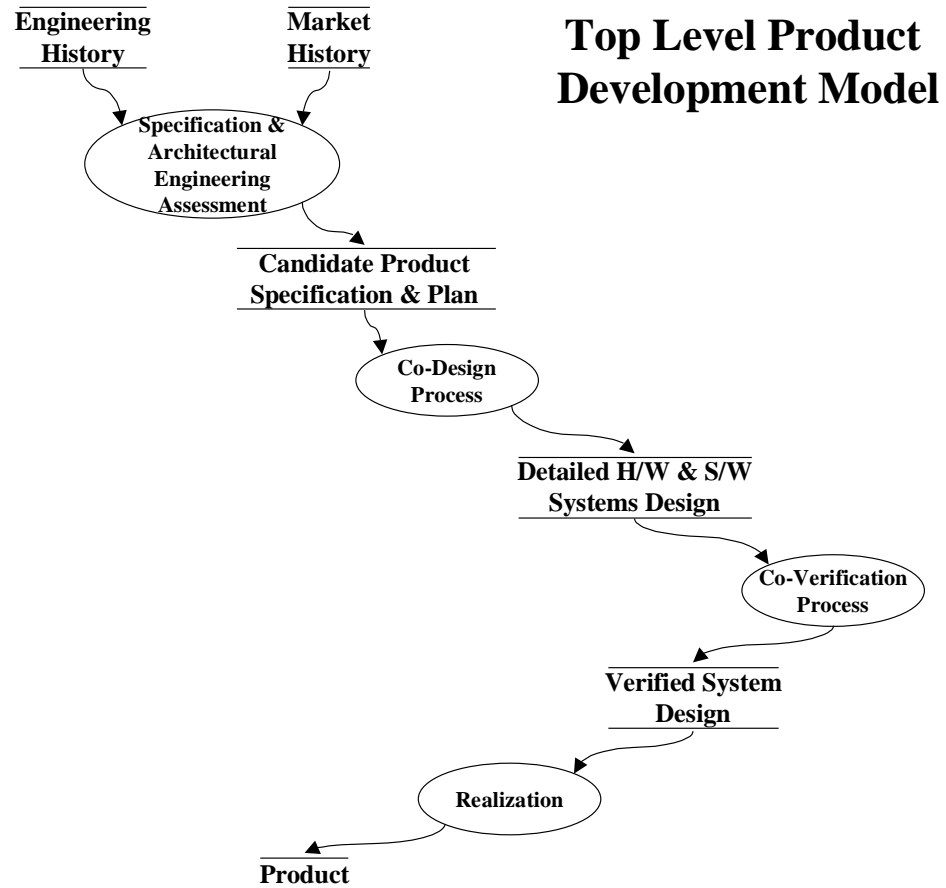


Figure 1: The System Development Process

1.2 Specification and Architectural Engineering Assessment

Figure 2 represents the true front-end of the systems engineering process. This process balances the market and the engineering input to new product and product iteration development. The ordering and iteration between these often diverging inputs is complex but fundamental to balancing the *views* brought back from the market by sales and marketing activities and the internal views accumulated through engineering and product histories in determining which a new products or product improvements to pursue. The result of the iterative contemplation of marketing, engineering and finance within a company yields a Marketing Product Requirement Specification. This is the document by which the engineering architecture selection process is driven and effectively charts the business goals and objectives and the engineering requirements of the product.

The gods of specification and architectural engineering are the system architects. Their role change is dictated by the bloody revolution. The new architects are

imbued with power through the new systems engineering tools and will need to eschew their old roles as harmless cogitators and mentors and adopt the new as political priests of the architectural covenant, the drivers of the model, and the checkers of design and implementation conformity. In this role they directly challenge the dominance of the hardware engineers. They need to bend the will of the hardware engineers to support the company and the product even while removing them from final arbitration power. As with all priests, part of this new role is to defend and modify the covenant in the face of the skeptics; unlike traditional priests, their role is to admit change and encourage skeptical attacks while building sufficient barriers to dissuade adventitious change. From the lessons of history, the transition of this priestly process to a religion must be ruthlessly guarded against by company management. The Balkanization of the design process is not an attractive outcome.

The heart of this process is the Architectural and Engineering Assessment and Quantitative Evaluation of Architecture steps. The connection between these two steps is the Technical Specification and Plan for the product which enables quantitative evaluation. The overall process yields a Candidate Product Specification and Plan which drives the CoDesign process (Figure 1). The Specification and Architectural Engineering Assessment process is highly iterative and at its interaction between marketing and engineering qualitative in nature. However, as soon as is feasible in the process of human problem solving the process makes a transition to a quantitative mode. At the back-end, verification and realization of the selected architecture occurs.

Architectural and Engineering Assessment (Figures 3 and 2) process should, after two qualitative sieve steps, produce a Technical Specification and Plan for each candidate architecture of the likely bewildering number which are thought up early in a product design cycle in response to the myriad *what-if* questions. The questions *Can it be done?* and *Will it be useful?* are the preliminary scan questions for architectural candidates. The question *How can it be done?* constitutes the technical feasibility assessment. Of course there is nothing new here. This process just embodies the *why*, *when* and *how* questions of qualitative analysis; the final implicit question *where?* determines whether the development occurs in-house or is out-sourced.

Quantitative Evaluation (Figure 2). To turn qualitative assessment into quantitative requires specifications to be in an analytical form so that modeling of the target system which yields numeric function, resource usage and performance data is enabled. This is the domain and intent of the unified systems description languages. Such tools may provide specification capabilities, transformation capabilities to help explore the design space, and partitioning support for determining whether particular objects should be realized in hardware or software. The outcome of this phase of the process is a product specification and design plan for a candidate architecture. It is this specification and plan that drives the concurrent hardware-software (co-)design process.

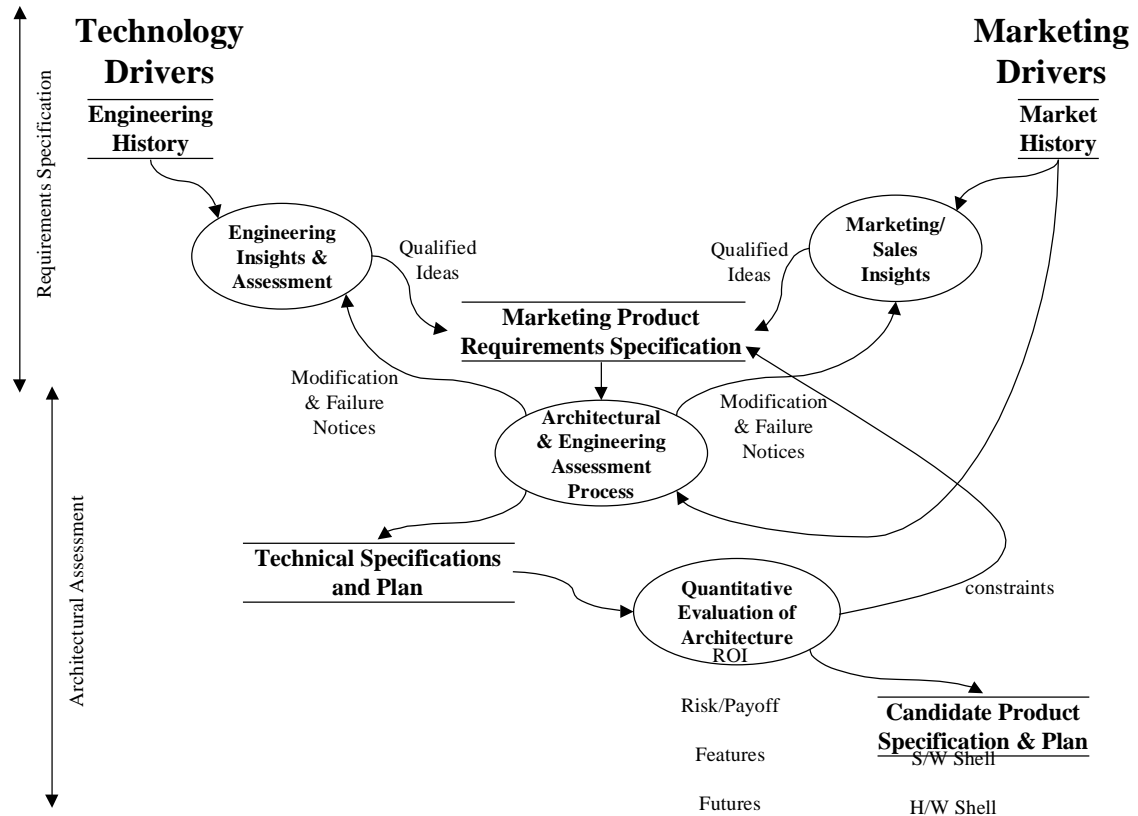


Figure 2: Specification and Architectural Engineering

The lack of a systems specification notation does not obviate the need for this step of the process. The existence of an executable specification written in C/C++ and Verilog/VHDL is sufficient for modeling and codesign but not for automatic partitioning. As yet, automatic partitioning fails the *pragmatic tool* test and until mathematically based hardware-software system specification notations are available automatic partitioning should not be on the engineering agenda.

1.3 Codesign

Codesign (Figure 1) tools support the design phase of the systems engineering process. The objectives of codesign are the concurrent design and modeling of systems involving electronics hardware, software and mechanical components. The design process for modern systems requires the support of languages used by engineers, together with environments in which evolving hardware can be tested with the software intended to run on the system, evolving software can be tested on the target hardware, and evolving mechanical subsystems can be tried

within their control environments. A proper framework for concurrent engineering requires supporting higher level languages than assembly and RTL, and providing greater modeling speed and as much precision as existing coverification tools (see below). In the event that a candidate design has been specified in an executable notation, co-design tools must work with this specification model together with the engineering languages, current C/C++ for software and Verilog and VHDL for hardware. Direct support for partitioning in the codesign process is unnecessary since it occurs as part of architectural specification and assessment.

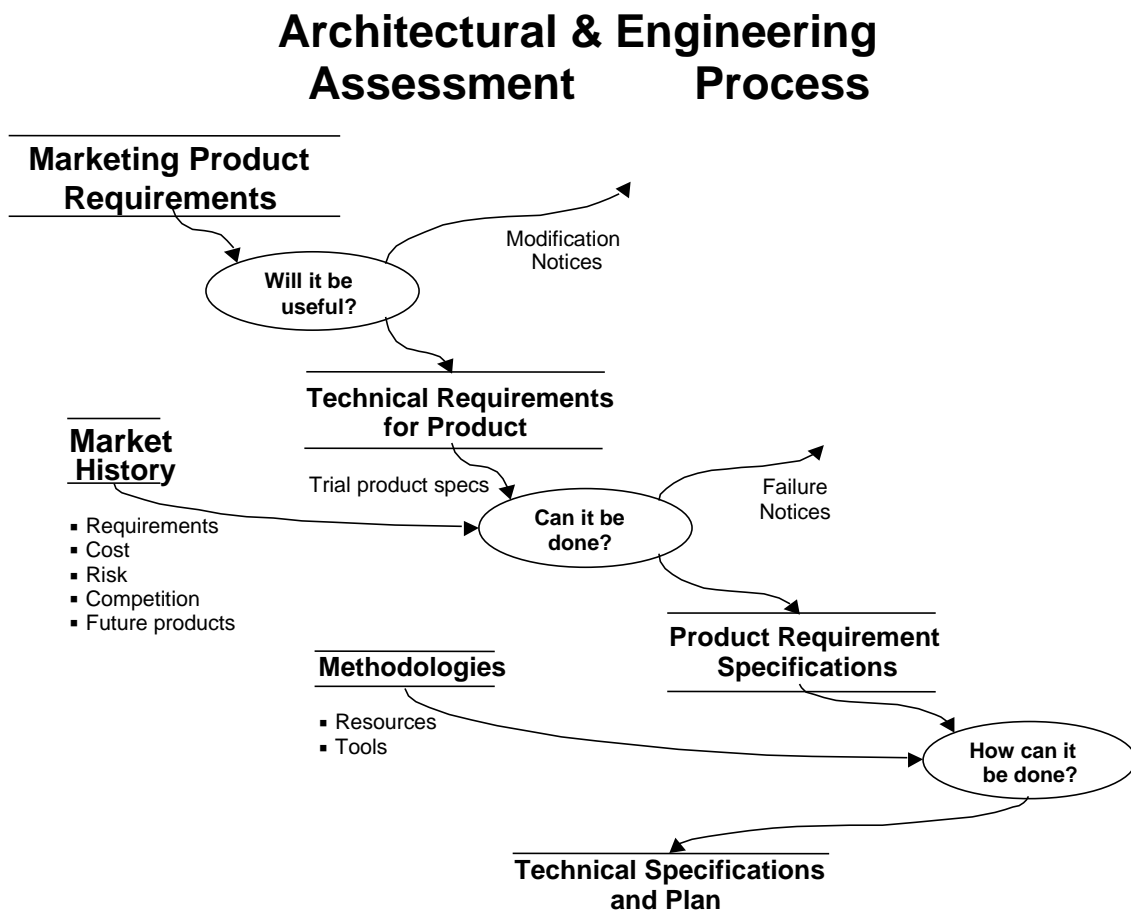


Figure 3: Architectural and Engineering Assessment

Modeling support for codesign brings with it many requirements. The development style used by software engineers is to perform many rapid *reanalysis-redesign-edit-compile-execute-debug* cycles. In the codesign environment, execution and debugging must take place on the target system – the virtual system prototype (VSP). Software tools support for the hardware engineering process, together with the near universal adoption of hardware description languages (HDLs) in digital hardware design and the requirement to test the developing hardware using target driver, operating system and applications code, has driven the hardware engineers to adopt the software engineering style. Specification and modeling tools support for mechanical subsystem design is in its infancy, but the development style will inevitably be a variant of software engineering.

The historic constraint on virtual systems prototype modeling has been the performance-precision tradeoff of the CPU models interpreting software and mediating interactions between the software and the hardware. To be useful VSPs need to have high performance and to be able to model timing and function accurately. This requires attention to be paid to HDL simulation and to CPU modeling.

HDL simulation using the two popular HDLs, Verilog and VHDL, is glacially slow due to the number of events generated and processed during the simulation of even modest models. The advent of the higher-level system specification notations may address this problem in hardware modeling but, in the large interim, accepted engineering practice is to use a mixture of HDL skeletons to model concurrency and timing and C/C++ language descriptions within the modules of the skeleton to produce function. The C/C++ descriptions do not directly generate events. Reported hardware simulation speed up is factors between 10 to 2,000.

The modeling of CPUs dominates co-simulation. Modern systems increasingly incorporate large amounts of software – operating systems, middleware and applications code. It is not unusual to execute millions of instructions during the routing of packets on a communications controller and the VSP for such a controller must do the same. CPU models come in a variety of flavors: very low speed ($1/10^{\text{th}}$ – 100 instructions per second), high complexity, yet high precision models specified using HDLs; low to medium performance models (1,000 to 50,000 instructions per second) with reasonable function and timing accuracy but limited flexibility implemented using the instruction set simulation (ISS) where both function and sequencing of the CPU is mimicked in software; and high performance (100+ million instructions per second), reasonable function but zero timing precision host software models where application code is run on the development processor (rather than the target) and linked to the HDL described hardware models. These 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 - requiring from 3 to 30 hours of simulation to model 1 second of real-time performance.

Recently, new technologies have been developed for modeling processors. One such technology, the Virtual Processor Model (VPM) [Hellestrand 1999x] is capable of simulating software at speeds in excess of 150 million instructions per second, maintaining timing accuracy, and of trading detail for simulation performance. This model has been constructed to support software engineer, hardware engineering and/or hardware-software engineering.

1.4 Coverification

Coverification (Figure 1) has been the work-horse of hardware-software technologies; the initiator of and developer of a young and vigorously growing market. Coverification tools have required modeling precision across the hardware-software boundary and the ability to run limited sequences of software to verify software device drivers interacting with hardware controllers. The range of coverification technologies is large and includes: in-circuit emulation, hardware emulation (usually FPGA based), melanges of CPU models, software and hardware simulation models.

In reality, coverification is a step in the codesign process. With the coming availability of competent codesign and architectural assessment tools the limited tools of the coverification process should fade as honored ghosts into history. It is probably true that the *in-circuit emulator (ICE)* and physical prototypes will persist as part of a distinct coverification process for years, partly due to history and partly as a result of the need for engineers to have a tangible system as the ultimate proof of concept. The question which has dangled from the dawn of simulation dangles still: *Is simulation as good as the real thing?*

The demise of coverification will not be a bloodless revolution. There are companies battling in the marketplace, big and small, who will fight to live another day. It is not always the best technologies that prevail, but it is just as sure that inadequate technologies will decline and disappear.

2. STATE OF THE INDUSTRY AND UNDERLYING RESEARCH

Systems engineering holds different meanings in industry and academe. Industry is largely driven by the realization that systems involving complex hardware and software need to be fabricated in silicon and that verification dominates this activity. In research, the interesting problem is how to classify system tasks as hardware, software, and (possibly) mechanical.

2.1 A View from Research

2.1.1 Co-Simulation

Co-simulation is the connected, concurrent simulation of systems built using more than one technology. This is typically, digital hardware being simulated with software executing on some processor model. However, the co-simulation of abstract subsystems with other hardware and software subsystems, or DSP/mechanical subsystems with hardware and software subsystems is covered by this topic.

2.1.2 Co-Verification

Co-verification has come to mean, in the vernacular, the verification of digital hardware systems incorporating a processor running a couple of hundred assembler instructions to check that hardware operates correctly. The practical rationale for this limited use of the term has been that since the digital simulator (Verilog or VHDL) completely dominates the simulation, running more than a few instructions per simulation is impractical.

More meaningfully, co-verification means the ability to run a complete software system (GUI, application code, middleware, operating system, device drivers) on a processor model connected into a large amount of digital hardware, which itself might be connected to DSP and mechanical subsystems. Now the billions of cycles needed to run the software systems tends to completely dominate the simulation when using typical instruction set simulator (ISS) models of processors. With the advent of fast and cycle accurate processor models (Virtual Processor Models), the co-verification of a complete product is now a reality. This is true co-verification.

2.1.3 Automatic Partitioning

The recent work on automatic partitioning of systems has a focus on hardware-software partitioning. This has been the meaning of co-design research for more than a decade. This problem is regarded as an emerging issue in the systems-on-a-chip (SoC) engineering community. For the past twenty years, following the wide availability of cheap microprocessors, the partitioning of tasks into hardware or software has followed the old recipe that largely software is for control and hardware does the heavy lifting.

The advent of 20 mm² processor cores embedded in a sea of 300 mm² silicon and capable of running at 100MHz today and 400MHz in 4 years time has changed the game irrevocably. Now there is real choice between implementing sophisticated algorithms in hardware or software and the tradeoff is usually unclear – technically, economically and managerially.

The partitioning of systems is not as narrowly defined as the hardware-software partitioning research might imply. In a modern SoC system, there may be as many as 12 partitioning boundaries that can potentially have much more impact on system performance than the hardware-software interface. For example, the bi-directional interface between the I/O subsystem of a real-time operating

system and its Device Drivers is likely to have a significant impact on overall system performance.

2.1.4 System Level Description

2.2 A View from the EDA Vendors

A snapshot of the coverification/codesign electronic design automation (EDA) industry was obtained recently from a paper/panel session held at the 1999 International Symposium on Circuits and Systems. Here 5 spokespersons for the leading companies in the industry delivered papers and interacted in an open forum panel. As may be imagined there are a number of common issues, expressed with different emphases. Four quotations neatly capture the state of the industry and the urgent problems that need to be solved.

2.2.1 Virtual Prototyping – A Requirement

[Bunza 1999] (Eaglei, Synopsys) expresses the need for virtual prototyping: *Virtual prototypes facilitated by fast hardware/software co-simulation tools offer the best hope for control, visibility, debugging, and integration in ever more complex embedded systems. Effective use of co-simulation tools has repeatedly delivered twenty to thirty percent (20-30%) reductions in development schedules, by eliminating ASIC re-spins and facilitating parallel, early hardware/software integration. Dramatic successes in product quality and time to market are evident, but these dramatic results need the support of methodology, process and management changes to yield the best returns.*

2.2.2 Assessing Coverification/Codesign Tools

[Kenney 1999] (Seamless, Mentor Graphics) comments on the requirements for assessing coverification tools in the context of engineering real products: *When evaluating co-verification solutions, make a careful assessment of what functions of your design are supported by the tool and its associated CPU model. At what level of hardware detail can these functions be simulated? What is required to shift between detailed hardware simulation and high-speed software execution? Will you be required to make changes to your embedded hardware or software? You want to co-verify your design, not some distant derivative of it.*

These factors as well as how well the tool fits with your current design environment will determine how much value you derive from the adoption of co-verification.

Incurring the risk of introducing a new processor into your project should be accompanied by an appropriate level of risk mitigation. Co-verification has proven to be an effective approach to comprehensive system simulation. Design teams faced with the prospect of incorporating a new processor should seriously think about incorporating this powerful tool into their embedded system design flow.

2.2.3 The Importance of Simulation Speed in Coverification

[Jain 1999] comments on the requirement for simulation speed: *SOC (systems on a Chip) design verification requires that the design is simulated under various non-deterministic external behaviors to gain high confidence in the design. Each of these simulations require millions of cycles of simulation. HDL simulators take weeks to complete each simulation. Although each of these non-deterministic behaviors can be executed in parallel, this requires multiple simulator licenses, increasing the cost of verification. The low speed and high cost of HDL simulators also limit design optimization and do not allow design error detection before manufacturing, increasing the cost of fixing an error, and delaying product introduction. An improved methodology and supporting tools are needed to improve verification quality. This methodology must provide enough simulation speed to let designers try out many alternative architectures, run the necessary iterations to analyze them, and be affordable.*

2.2.4 Some Requirements for Tools Supporting Systems Engineering

Finally, [Kroliloski 1999] (Felix, Cadence) expresses eloquently the need for a paradigm shift in the tools to support systems engineering: *Recently there has been a rapid increase in the integration level of embedded systems for use in communications and multimedia products. This higher level of integration has focussed attention on significant gaps in the methodology and the technology for design of complex system-chips and chipsets including problems with design environments.*

The design of consumer products, e.g., in the communications and multimedia domains, is rapidly changing. Significant changes in the marketplace are demanding commensurate changes in design methodologies and tool sets:

- *System-level decisions made very early in the design cycle determine the cost, performance and viability of the product.*
- *Creating a “virtual prototype” of the product is vital to guarantee acceptance at type approval or product qualification*
- *Engineers must evaluate, combine, integrate and verify pre-designed virtual components in order to meet design deadlines. Virtual components are needed in both the software and hardware domain.*
- *Effective HW-SW partitioning and implementation decisions require explicit definitions of system function and architectural realization, and the exploration and analysis of a number of alternative mappings between the two domains.*

3. TOOLS ARCHITECTURES AND FLOWS

3.1 Tools Flows

In this Section, we will examine the architecture and flows for one of the new breed of systems engineering tool sets. Architectural quantification and codesign/coverification must fit into existing tool flows which take designs through synthesis and realization. The flow of the new engineering process, as depicted

in Figure 1, embodies physical realization. From this view, synthesis is a preliminary phase of the realization process.

Currently, logic synthesis tools take in low-level specifications of systems and produce *equivalent* gate-level descriptions of the hardware and C/C++ or assembly code of the software. The hardware and software tools are typically separate as is the synthesis and realization activity for hardware and software. Mechanical system modeling and synthesis is not on the radar screen of most tools companies, but remains an ever present problem to the designers of products.

The key to integration into the existing engineering tools flow is the languages used in the modeling and specification of products. In the near term, the new systems engineering tools must deal with existing languages C/C++ and Verilog/VHDL both for importing existing component designs into new products and for integration into existing tool flows.

3.2 System Level Modeling Tool Architecture

Even though we will use a particular tool, CoMET, to describe the architecture of a systems engineering modeling system, the intent of the coverification tools mentioned in Section 2 is similar enough to be subsumed by this discussion (for further elaboration see [Goering 1999]). The differentiation is in the models used to mediate the execution of software and the interactions between software and hardware – the CPU, the modeling of time and causality, and the ability to trade-off architectural detail for simulation performance.

In order to maintain compatibility with existing hardware design systems and expected hardware design and simulation environments, the systems level tools incorporate one of the many commercially available Verilog or VHDL compilers/simulators. This simulator now needs to be linked to the CPU model which executes software and mediates hardware/software interactions. These models differ widely as discussed in Section 1.3.

3.2.1 A New Fast and Accurate Processor Model

The model used in CoMET is the so called Virtual Processor Model (VPM) [Hellestrand 1999d] which 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 part models the dynamic parts of the processor, those portions those 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. A comparison of VPM, HDL, ISS and host-software based processor models may be found in [Hellestrand 1999b].

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. 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 fast, accurate, and malleable and as such effectively addresses the requirement in the concurrent engineering process to execute software at sufficiently high speed and precision to enable VSPs to stand in stead of physical prototypes.

3.2.2 The CoMET Simulation Framework

In CoMET (Figure 4), many VPMs are supported and a single HDL simulator is executing. The hardware and VPM modeling domains are clear in the diagram. The *back-plane kernel* mediates the maintenance of causality between the domains (VPMs and HDL simulator) that define their own relative frames of space and time. Models describing subsystems implemented in various technologies may be included in this open-ended system, including: mechanics and continuous processing (such as in chemical engineering).

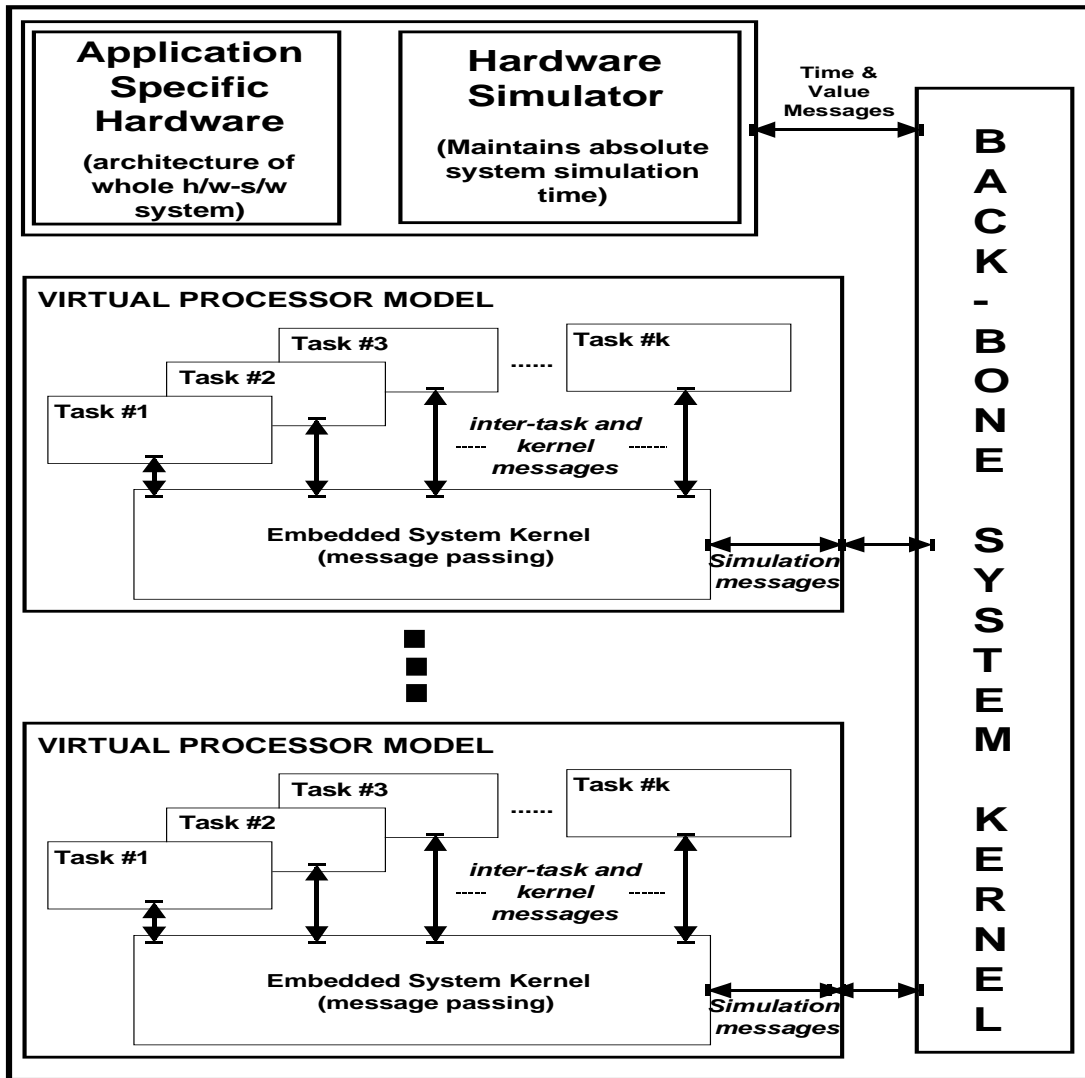


Figure 4. A Hardware-Software System Simulator Model

4. MODELING HARDWARE, SOFTWARE AND PROCESSORS

The complexity of systems increases algebraically with the number of component type layers present. In a modern system there are 8 types of layers: mechanical hardware, analog electronics, digital electronics, processors, device drivers, operating system/kernels, middleware, and application software. It has been estimated that in determining system components, methodologies and tools some 100,000 discrete choices are available [Hellestrand 1999c].

One key to building efficacious VSPs lies in the capability of the CPU model. Another key is the maintenance of causality and relative time across simulation domains – digital hardware, CPU model, software - where the expected time relationship between systems elements, at any particular instant, may not be maintained but causality must be. A final key is the modeling of signals (or more generally symbols) and symbol transitions across domain boundaries. Effective modeling requires the ability to model failure and this triggers the requirement for sub-symbol interpretation. The proliferation of symbols representing values of signals, is anathema to fast simulation but is a requirement to accurately model failure.

4.1 Modeling Single Processor VSPs

How is a system modeled? Figure 5 shows the various parts of a typical system incorporating a single processor. Modeling requires yet further choices. On the hardware side there is the intellectual property (IP) (purchased or in-house created) required: to model hardware components (processors, controllers, memories, I/O devices, FPGAs, custom chips and ASICs); to institute effective design methodologies; to describe designs at a high level using hardware description languages (Verilog, VHDL, C); and to analyze, verify and realize the design with tools. On the software side the Operating System (OS), device drivers and middleware (such as TCP/IP stacks) are likely to be purchased but occasionally custom developed. The remaining software infrastructure to support modeling is a combination of design methodologies; programming languages (C, C++, Java); tools (editors, compilers, debuggers, software synthesis) and whatever applications software (accounting and costing, etc.) might be built or bought.

The VSP modeled using these software and hardware elements, needs to simulate software at millions of instructions per second and hardware 10-1,000 times faster than existing HDL based simulations. Currently these requirements can be satisfied using VPM technology and the extensive incorporation of C/C++ functional hardware descriptions within the HDL skeletons describing systems.

System Model

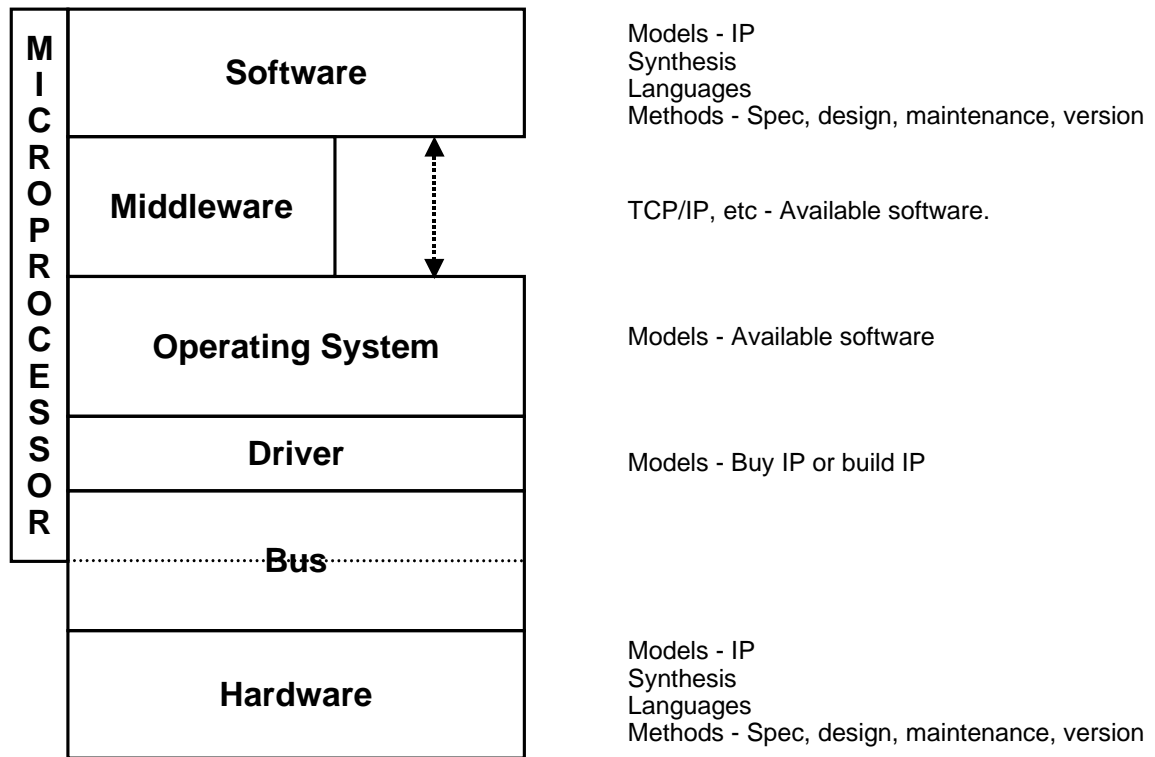


Figure 5. Components of a Typical System

4.2 Modeling Multiple Processor VSPs

Modern systems are likely to incorporate multiple CPUs whether general purpose processors or Digital Signal Processors (DSPs). The system model is commensurately more complex, as shown in Figure 6.

With the base of a comprehensive single processor model, multiple processor modeling presents issues of scalability and separability. The use of different operating systems, middleware and applications code on each of the interacting processors is a challenge for concurrent systems development, debugging and simulation. The incorporation of DSP processing elements adds requirements for handling data sizes and types which are potentially incompatible with related structures (such as type size and representation) on general purpose processor processors. The solutions to such problems are similar in modeling as in realizing physical systems.

The advantage of modeling, rather than physical prototyping systems, some of which are amongst the most complex systems devised by human beings, lies in the observability, manipulability, early availability, speed and accuracy of the VSP. This is especially so for multiple processor systems destined to be realised on silicon, where extensive debugging and iterative fixing of problems, requiring the respinning of silicon, simply pushes time to market and cost beyond practical business limits.

Modeling a Multi-Processor System

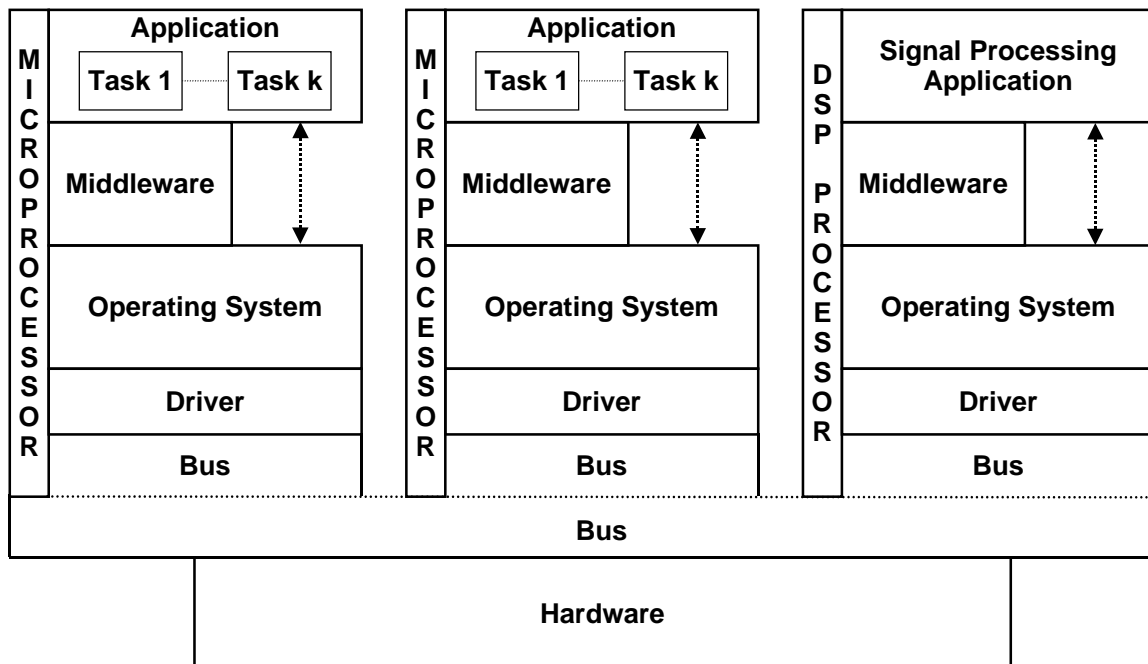


Figure 6. Multi-processor System Model

5. THE BRILLIANT FUTURE

The many facets of systems and models described above form a basis for developments in systems modeling that will transpire over the next decade. The first requirement for advancement in systems engineering tools support is the existence of a comprehensive development framework which supports fast, accurate virtual prototyping of hardware, software and mechanical subsystems. As described above, this became feasible in late 1998 with the advent of the VPM processor model. Such frameworks will be the basis of future engineering workbenches, upon which will be built higher layers of modeling tools.

The next big advance in the quest for improving the productivity, efficiency and effectiveness of the systems engineering process will be the specification and adoption of mathematically based system description notations to specify and model digital hardware, software and mechanical systems [Hellestrand 1994]. Such notations must have synthesis paths, to appropriate target realization technologies, which are observable, predictable and controllable. This will constitute a radical departure from current practice in the use of behavioral/RT level Verilog or VHDL for hardware engineers and will contribute further to the blood to be shed in completing the systems engineering revolution. These attributes imply that mathematically well formed, explicit syntactic and semantic constructors for concurrent and sequential control and data flows will be the foundation for such languages. Since the notion of time is inherently embedded in concurrent and sequential constructors, synchronization control constructors will be built from the more fundamental constructors. These languages are likely to be based on the higher order functional mathematics, which will constitute a major departure in engineering practice for both software and hardware engineers.

With the presence of competent system description notations, the ability to transform a system specification using mathematics opens dramatic capabilities. One such capability will be to formally construct a family of products from a *golden* architectural specification by provable transformations [Cheung 1997a, 1997b, 1997c], where each member of the family can be defined by constraints that determine performance, resource usage, power consumption and hence cost. The members of the product family have provably the same functionality ab initio and bypass the requirement for formal verification tools. By incorporating predictions of improvements over time in realization technologies, such as silicon structures, such product families can be tagged with likely lifetimes in competitive markets.

Another benefit of systems description notations is the ability to automatically partition designs into hardware and software components, based on some cost function. Given the prior requirement of direct synthesis being a fundamental requirement of the notation, together with the transform capability described

above, the support of extensive and systematic exploration of the design space is enabled.

The final benefit, is the capability of driving physical floorplanning [Saheb-Zamani 1995] directly from such specifications. This close connection between specification and realization can be used to huge advantage in systems-on-silicon to optimize silicon area, power, performance and hence cost.

The future awaits the curious and bold.

6. REFERENCES

- Bunza, G.J. (1999): *Towards System Integration in a Virtual Environment: Small Steps, Big Results, and Complications to Come for Embedded Systems Engineering in the Next Millenium*. Proc. ISCAS, May 1999, Orlando, Florida.
- Cheung T.K.-Y., Hellestrand G.R. (1997a): Form: A Functional System Specification Notation, The Fourth Asia-Pacific Conference on Hardware Description Languages (APCHDL'97), APCHDL'97, HsinChu, Taiwan, 18-20, August 1997, pp10-15.
- Cheung T.K.-Y., Hellestrand G.R., Kanthamanon P. (1997b): System Level Analysis of a Coprocessor Architecture for Block Matching Motion Estimation Computation, IEEE International Symposium on Circuits and Systems. 1997, IEEE Society, Hong Kong June 9-12, 1997.
- Cheung T.K.-Y., Hellestrand G.R., Kanthamanon P. (1997c): A Transformational Codesign Methodology, Asia and South Pacific Design Automation Conference, ACM/SIGDA CDROM, IEEE, Chiba, Japan, 28-31, Jan. 1997.
- Goering, R. (1999): *VaST Systems releases high-level codesign tool*. EE Times, CMP Media Inc., 5 Apr 1999.
- Hellestrand, G.R. (1994): Events, Causality, Uncertainty and Control. Proc. 2nd IEEE Asia Pacific Conference on Hardware Description Languages, pp 221-227, Toyohashi, Japan, October.
- Hellestrand, G.R. (1998): *The Engineering of Mixed Technology Systems*. IEEE Circuits and Systems Society Newsletter, Vol. 9, No.2, June 1999, pp 1-9.
- Hellestrand, G.R. (1999a): *Systems Engineering: It is not a luxury?* Electronic Systems Technology and Design, May 1999.
- Hellestrand, G.R. (1999b): *Systems Engineering: The Era of the Virtual Processor Model (VPM)*. Electronic Component News, 15 May 1999.
- Hellestrand, G.R. (1999c): *Designing Systems-on-a-Chip using Systems Engineering Tools*. Proc. ISCAS, May 1999, Orlando, Florida.
- Hellestrand, G.R. (1999d): The Advent of the Virtual Processor Model. EE Times, CMP Media Inc., 14 June 1999.
- Jain, P. (1999): *Cost-effective Co-verification using RTL-accurate C Models*. Proc. ISCAS, May 1999, Orlando, Florida.

- Kenney, J. (1999): Co-verification as Risk Management: Minimizing the Risk of Incorporating a New Processor in Your Next Embedded System Design. Panel Session, ISCAS, May 1999, Orlando, Florida.
- Krolikoski, S.J, Schirrmeister, F., Salefski, B., Rowson, J. and Martin, G. (1999): *Methodology and Technology for Virtual Component Driven Hardware/Software Co-Design on the System-Level*. Proc. ISCAS, May 1999, Orlando, Florida.
- [Saheb-Zamani 1995] Saheb-Zamani M., Hellestrand G.R. (1995): A Stepwise Refinement Algorithm for Integrated Floorplanning, Placement and Routing of Hierarchical Designs. IEEE International Symposium on Circuits and Systems, IEEE, Seattle, Washington, USA, May, 1995, pp49-52.