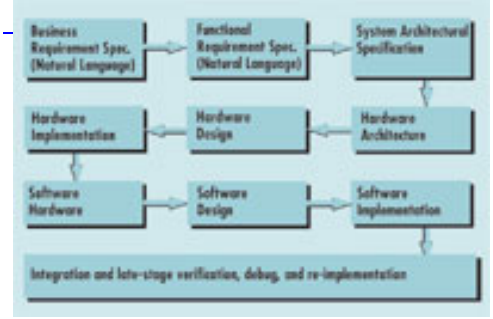
Architecture-driven Design for Complex Electronic Systems

Architecture-driven design, using a virtual system prototype, allows the hardware and software portions of the design of next generation of mobile wireless to be developed concurrently, significantly reducing the overall design cycle and subsequently, costs.|By Graham Hellestrand

In spite of the bumpy road high technology has been on of late, the wireless industry is still at the forefront of electronics systems development. Complex and sophisticated engineering processes yield its products. Acute competition in the high-volume mobile communications consumer markets drives down development time and consumer cost for increasingly rich integrated capabilities. In turn, the foremost technical drivers in "wireless space" become innovations in RF technology, semiconductor technology, multimedia technology, and software technology and methodology.

Users of mobile wireless platforms are demanding progressively more sophisticated feature sets that require massive computational resources. Simultaneously, these products must be small and lightweight, with extremely low power consumption. One key attribute of today's wireless products is that they feature tightly integrated combinations of hardware and embedded software. The software components may include system initialization routines, a hardware abstraction layer, a real-time operating system and associated device drivers, all the way up to embedded application code.

To achieve size, weight, and power consumption goals, the digital portions of systems — including multiple computational and data-processing engines, memory subsystems, and peripherals — are being constructed mainly on a single semiconductor SoC platform. Any design errors requiring re-spins of such a device result in huge costs (typically $1 million or more in the case of a 90 nm chip technology) and significant project delays, usually measured in months and often fatal from a competitive standpoint.



[1]

Unfortunately, traditional design flows and the traditional EDA tools that automate their design and verification are sequential in nature. First, the overall architecture of the system is determined. Then, the hardware is designed and a hardware prototype is constructed. Next, the operating system and/or middleware is installed and tested, and only then can the bulk of the software be developed, ported, integrated, and debugged. The end result of this sequential chain is that the majority of wireless systems miss their targeted ship date, and the late stage verification inherent in this sequence means that many require a chip re-spin.

An emerging solution to this problem is the concept of architecture-driven design using a virtual system prototype. Among other benefits, this allows the hardware and software portions of the design to be developed concurrently, which significantly reduces the overall design cycle.

Oh, How the World has Changed

A little more than three decades ago, in 1973, the world's first cell phone made its début. Reminiscent in size to a house brick at 9" x 5" x 1.75" — and weighing in at a strapping 2.5 lbs. — the Motorola Dyna-Tac contained 30 circuit boards. The Dyna-Tac's only supported features were to dial numbers, talk, and listen; it provided a talk time of only 35 minutes before the batteries died; and recharging those hefty power cells took ten hours (on a good day).

This type of device was typical throughout the late 1970's and early 1980's. Cell phones of that era had limited range; their connectivity (such as it was) was to an almost non-existent wireless network; and their price tag was around $1,000, which meant that they were owned by a select few whose income was beyond the dream of most American families.

In those ancient days, the majority of this type of product's functionality was implemented in hardware. The process began when the system architects designed and captured the system's architecture using pencils, paper, erasers, slide rules, and other 1970's-era tools. Once the initial architecture had been defined, the rest of the design and implementation process flowed sequentially. First the hardware portion of the design was established; then a hardware prototype was constructed; and only then could the bulk of the software portion of the design be addressed.

The point is that this style of design worked well — and was well suited — for that time, because the architecture and hardware portions of the system were simple (by today's standards) and the software content relatively small. The small size of the software was reflected in the amount of time required to develop it, and when coupled with the simplicity of the hardware architecture, this served to limit the number and scope of any hardware/software integration issues.

In addition to the fact that designs were much less complex through the early 80's, time-to-market windows were larger, cost issues were less sensitive, and any potential design problems were mitigated by the fact that most design companies and system houses were vertically integrated. Confused about some portion of the

design or a reused part or subsystem? Just wander down the hall and chat with its designer(s). Today's functionality is only achievable using sophisticated technology, such as SoCs featuring embedded microprocessor and DSP cores with large amounts of associated software, blocks of third-party intellectual property, real-time operating systems, and a host of other advanced and mixed-technology solutions.

Despite the huge upswing in technology since the 1970's, most organizations still try to use 1970's/80's sequential development and verification processes, supported by traditional EDA tools. The problem is that these processes are massively insufficient with regard to today's design challenges. The result is a bankrupt situation in which:
• 50% of embedded system designs (such as cell phones) are late.
• 20% of embedded system designs are cancelled.[1]
• 40% of chip designs must be re-spun (re-designed, tested, and manufactured).[2]
As system houses are painfully aware, late delivery reduces revenue opportunities by 25%, compounded each quarter.

Traditional System Development Flows

As previously noted, conventional system development flows are sequential in nature (see Figure 1). The result from the architectural design phase is typically a natural language specification a few thousand pages long. Using this specification as a basis, the system architect's work has to be reinterpreted and disambiguated to implement the hardware portions of the design and then reinterpreted and disambiguated again for the software development phase.

This sequence of reworks has the undesirable side effect of significantly increasing the risk of introducing errors in each, predominantly manual, translation step. And, any definitive system level tests with regards to such aspects as system performance and throughput cannot commence until the software portion of the design has been completed.

In many organizations using these sequential design flows, at most, 5% of the total development effort is devoted to system-level architectural design and exploration; 20% is spent on hardware and software design and implementation; and as much as 70% is consumed by late-stage verification, debugging, and re-implementation. As a result, development teams are encouraged to rush through the early phases of architectural design and exploration to "save" enough time to undergo the lengthy debugging cycles.

Conventional Verification Solutions

More than 30 years have gone into developing EDA tools to automate various stages of the sequential development process described above. Most of these tools are at, or below, the register transfer level of abstraction and are intended primarily to facilitate the detailed hardware design. Even the attempt to create electronic system level design tools has inherited a hardware-centric view of the world. If the hardware is specified using an application programming language like C or C++, and then constrained by limiting it to a register-transfer level-compatible subset, then there is little gain compared to using Verilog or VHDL. A summary of the most

widely used "alternatives" for verification illustrates just how deeply EDA has failed to be relevant to the needs of today's complex design processes.

Physical prototypes — In the context of a wireless system such as a cell phone, a physical prototype typically involves the use of a circuit board and the SoC in the form of working silicon. However, this means that the hardware portion of the system must be almost 100% tied down (with the exception of any field-programmable fabric incorporated into the SoC) before verification. Thus, this approach is essentially useless with regard to exploring and evaluating alternative architectures. Furthermore, because the physical prototype requires that the hardware portion of the design be largely completed, any software development is pushed downstream in the design cycle.

FPGA-based emulation — This also requires the hardware portion of the design to be largely completed, which pushes software development downstream in the design cycle. This approach also requires a complex (and often lengthy) mapping of the design into the emulation environment, an activity that usually must be performed by specialists. Furthermore, the cost of these special-purpose, massively parallel computers typically renders them too expensive for use in a mass-deployment scenario.

Software simulation — In this case, a model of the whole system is developed in software and run on the workstation. The most common approach is to use an instruction set simulator for each processing core in the design. This technique offers many advantages, such as allowing for shorter model design time, high flexibility with regard to model changes, and full observability and controllability of the design. This model-before-silicon approach potentially allows software development to commence long before the actual chip is produced, but traditional simulation solutions are limited in terms of speed and model accuracy. The execution speed of a standard instruction set simulator is in the order of tens of thousands to a few hundred thousand instructions-per-second on a 2 GHz host. With models running at these speeds, the verification cycle is measured in weeks or months.

Processor-based Designs and the Growth of Embedded Software

Our world population is now home to more than one billion mobile super computers, in the form of cell phones, that offer voice, text messaging, data (Internet connectivity), and digital photography. What drives the cell phone's explosive growth in functionality is the combination of embedded processors and embedded software.

The SoC in a modern cell phone typically contains one or more processor cores and one or more DSP cores. The processor core(s) may control 20 to 40 peripherals (many in the form of third-party intellectual property blocks) for such tasks as multimedia functions, digital camera interfaces, cryptographic functions, 2D and 3D graphics processing and acceleration, and interfaces such as WiFi, USB, and UART. Meanwhile, the DSP core(s) typically have a number of associated accelerator peripherals for tasks such as modulation, baseband filtering, channel decoding, and so forth.

These processing and DSP cores require access to a variety of tiered memory and peripheral devices. Some devices will be tightly coupled to a single processing engine by means of a dedicated bus. Other devices may be shared among a group of processing engines. To further complicate the issue, the various device subsystems may have different speed requirements, bus widths, and clock domains.

Meanwhile, the software content of wireless products such as cell phones is increasing at a phenomenal rate, to the extent that software development and testing now dominates the costs, timelines, and risks associated with such designs. For example, a representative GSM phone circa 2005 may contain approximately two million lines of code. This requires an extreme amount of time and resource use to develop, integrate, and debug. The risks are dramatically intensified when software development teams remain on the critical path in the sequential design process. Sequential development will be impossible when software content of the typical cell phone rises the expected tenfold to 20 million lines of code by 2007 to 2008.

As technologies evolve, software algorithms take over from hardware implementations, and the need to evaluate the tradeoffs between hardware and software implementations grows, as does the need to perform verification across the hardware/software interface. Thus, today's complex wireless systems require a new paradigm for design and verification.

In an SoC, the very nature of integration on a single chip reduces the ability to observe and control events in the implementation. It is, for example, practically expensive and difficult to connect the probes of a "logic analyzer" to the signals interfacing the various functional blocks comprising the design. Such problems mushroom as the number of processing engines embedded on an SoC increase. Mobile wireless "super computers" live in a complex, real-time environment, so their design and verification methods and tools must evolve to reflect their needs.

Architecture-driven Design and Virtual System Prototyping

Given the functional and business requirements for the operational interface and the various subsystems forming a wireless product, systems architects must be able to explore and analyze a large number of potential architectures. Bounding this work and quickly isolating any viable candidates for more intense evaluation is an enormous problem.

If a set of architectures that satisfy the requirements can be reasonably bounded, the problem then becomes one of selecting an optimal subset of those architectures. This process requires the efficient discarding of all but a relatively small set of real candidate architectures. At this stage, the only effective and efficient approach is a quantitatively driven assessment whereby the architects can optimize for communications bandwidths, cost, performance, power, time-to-market, and other critical factors. At this point, architects need a powerful and automated way to perform a statistical analysis of the measurements gathered from the various candidate architectures so as to drive the setting of critical parameters.

A virtual system prototype is a software-simulation-based, architectural-level model of the electronic system. This model can include one or more processors or networks of processors, buses, hardware peripheral components, and even models of mechanical subsystems that are part of the overall system.

A true virtual system prototype is cycle-, register-, and timing-accurate, which allows the system under design to be modeled and verified for real-time requirements. Most notably, a virtual system prototype runs the same compiled and linked target code as the real hardware, which means that it can accurately predict the system's real-world behavior.

Furthermore, a true virtual system prototype is fast. A state-of-the-art virtual system prototype based on third-generation software architecture can achieve 50 to 200 MIPS for a single-core simulation, while multi-core systems with tiered device structures and multi-level buses can be simulated at 10 to 100 MIPS, depending on the configuration. This is sufficiently fast enough to support real-world tasks such as:
• Booting a real-time operating system in a matter of seconds
• Running applications code calling databases
• Sending/receiving messages via communications stacks using services from the operating systems
• Interacting with devices using device drivers

The use of a virtual system prototype facilitates architectural exploration and evaluation by allowing hardware and software components to be optimally matched. The system architects can analyze performance considerations and effects such as cache sizing, processor capability, and bus bandwidths; and they can also detect potential resource-sharing contentions and synchronization problems. Accurate measurements that model real-world behavior under real-world data processing and software workloads allow system architects to make accurate hardware/software tradeoffs early in the development process.

One significant advantage of this architecture-driven design approach is that the virtual system prototype becomes an executable specification for use by both the hardware and software development teams. But perhaps the most significant advantage is that, once the optimal architecture has been determined, the software development teams can start creating and running their application code on an actual model of the hardware that is concurrently being realized. Because the virtual system prototype can be made available as much as nine to 18 months earlier than physical hardware, this has a dramatic effect on the product's time to market.

Summary

Existing design and verification strategies — such as informal architecture design and the use of traditional automated methods like physical prototypes and FPGA-based emulation — hamper the ability of architects to optimize their architecture. Existing strategies keep the design process sequential, assuring that the hardware must become available before the main software effort can begin.

The dominance of software content in today's wireless products, coupled with shrinking development times and shorter product life cycles, mandates optimization of the architecture and concurrent development of the hardware and software portions of the design.

Architecture-driven design using virtual system prototypes offers the ideal combination of performance and accuracy. These prototypes facilitate sophisticated architectural exploration and evaluation; they allow hardware and software development to take place concurrently; and they provide executable specifications and golden reference models that can be used throughout this and future product development cycles. Overall, the use of software virtual prototypes results in the development of better products with shorter development times for less money with less risk.

WD&D

About the Author

Graham Hellestrand is the founder of and chief technology and strategy officer at VaST Systems Technology Corp.

Footnotes

1. Source: Venture Development Corporation, 2003
2. Source: Collett International

Links:
[1] http://www.wirelessdesignmag.com/sites/wirelessdesignmag.com/files/legacyimages/0504/wd54feature1_a_lrg.jpg