White Paper

Using Virtual Platforms for Pre-Silicon Software Development

Meeting the challenges of software-dominated design projects

Frank Schirrmeister, Director of Product Marketing, fschirr@synopsys.com
Shay Benchorin, Director of Business Development, shayb@synopsys.com
Filip Thoen, Solutions Architect, filip@synopsys.com

May 2008

Introduction

Over the last decade, a new reality has set in for semiconductor providers. In a variety of application domains like wireless, multimedia, networking and automotive, it has become more and more difficult to sell silicon without the associated software executing on the hardware. Not only has software become the key functional differentiator in many areas, its development now determines project success. If the software is late, the chip running it cannot be sold. The best architected chip design will fail in the market if its associated software is late or insufficient.

Market-research firm International Business Strategies, Inc. claims that today at 90nm the typical overall SoC-related development effort for software has already surpassed the effort for hardware. For 45nm designs in the year 2011, IBS projects that less than 40 percent of the overall development efforts will be spent on hardware. Given the never before seen levels of complexity, traditional approaches of developing embedded software for chips are running out of steam; a new era of software development has begun in which the majority of embedded software is developed with virtual platforms in contrast to traditional software development on hardware boards.

An Example Design

The transition to software dominating the development effort impacts not only the semiconductor industry traditionally delivering the chip designs. It also impacts the complete chain of IP providers, semiconductor houses, system houses and independent software vendors. To properly assess the impact of virtual platforms it is important to understand an example design and its associated design chain dynamics.

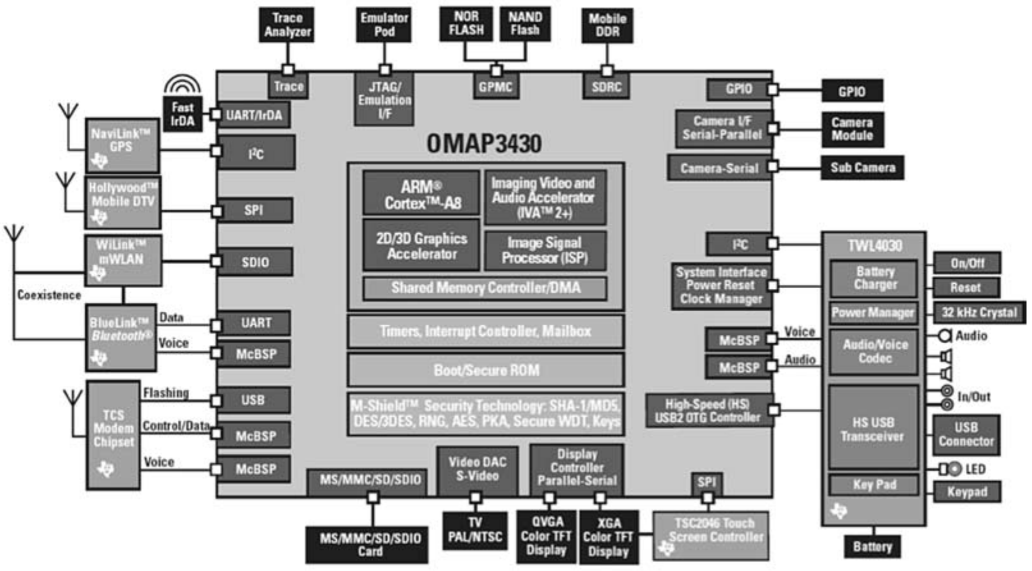


Figure 1: Texas Instruments OMAP3430 [1]

The Texas Instruments OMAP™platform was one of the pioneering early application processors subsystems that revolutionized handsets by delivering enough performance to enable functionality never before seen in mobile devices. Figure 1 shows the 3rd generation OMAP3430 platform based on the ARM Cortex® A8 processor family in its system context. Similar devices on the market - like Marvell's XScale and Freescale's i.MX architecture - are of comparable complexity. Figure 2 illustrates an example of software architecture for mobile devices, with its layered structure from kernel software and drivers close to hardware through middleware of operating system services to applications running on the operating system. With this software stack already being very complex, developers are only facing further increasing complexity as the consumers appetite for more functionality in mobile devices continues to grow.

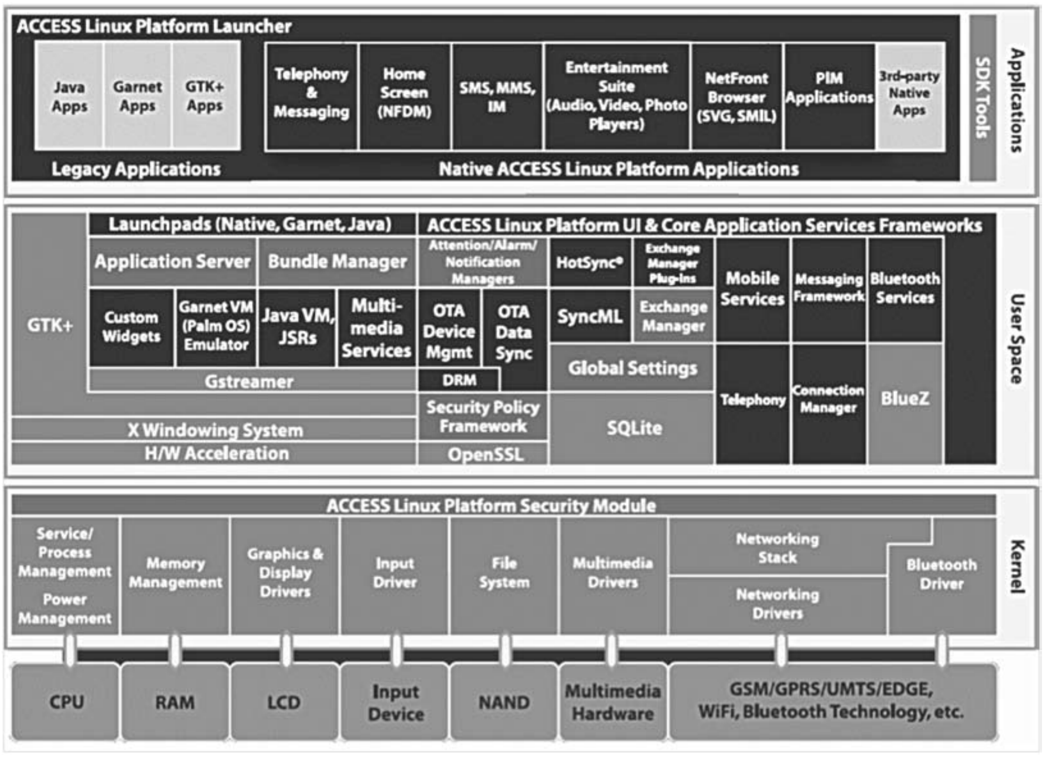


Figure 2: Example Software Architecture [2]

Hardware and Software Developers

A variety of different user types are involved on the hardware and software development side, all of which have specific needs with respect to tools supporting them. For front-end hardware development, project teams are typically comprised of members covering the following tasks:

- **System and chip architecture development** covers the assessment of the overall hardware/software architecture. Typically a full functional model of the hardware/software architecture is not required. Instead architects often ask specific questions and then build specific simulation models addressing them. A typical assessment would be to understand the required bandwidth over a bus connection given the bandwidth ranges of modules connected to the bus. Instead of a fully functional model, a model comprised of traffic generators can be used and is often more suitable because it is more general and specific scenarios can be actively specified with their corner cases and performance.
- **Hardware block development and verification** covers design and functional verification of the actual blocks of the chip. For verification, designers develop specific scenarios covering the corner cases addressing the usage of the block in question. It is often useful for to have a full model of the environment their block exists in, mostly to understand the scenarios to be covered and also to capture specific input stimulus for the blocks under development.
- **Chip integration and verification** covers the integration of re-used IP blocks and new modules of the chip. Both the connectivity within the chip and functionality within the environment of the chip need to be verified. Here a fully functional model of the chip is essential to be able to create reference outputs for given scenarios.
- **Post silicon validation** is the process during which the chip is validated for correct functionality after the silicon is available. This process is of note in the context of functional models for software rich chip designs because the development of post silicon validation tests is becoming increasingly more complex and significant time can be saved by developing them using functional models of the chip prior to silicon availability.

On the software development side, the different development tasks follow the layered software architecture indicated in Figure 2:

- **Low level driver and kernel development** covers the hardware dependent development task, often resulting in some level of hardware abstraction layer (HAL) allowing the higher level software functions to access the hardware in an abstracted fashion for ease of porting. For efficient driver development and kernel bring up, an executable model of the hardware is essential. A majority of drivers and kernel functions require only a register-accurate representation of the hardware registers and memory maps to be developed. However, for real time software development, sometimes more detailed representations of the hardware timing are needed.
- **Operating system and middleware porting** covers the adaptation of an existing operating system like Symbian OS, WinCE, Linux® or others to a given hardware architecture using the HAL developed at the lower level. To verify OS bring-up and middleware, an executable model of the hardware is required as well.
- **Application software development** covers the development of the actual applications running on the device. The objective of teams developing these applications is to run on as many operating systems as possible and to be as hardware-independent as possible. Consequently, the development teams can live with fairly abstract representations of the hardware on which they are running; often the software engineers can use more abstract OS simulators that completely "hide" the hardware.

Industry Design Chain

Relationships between stake holders in the SoC design chain can be quite complex.

Figure 3 illustrates a simplified design chain for the wireless handset segment including some company examples and the relationship to the basic user types.

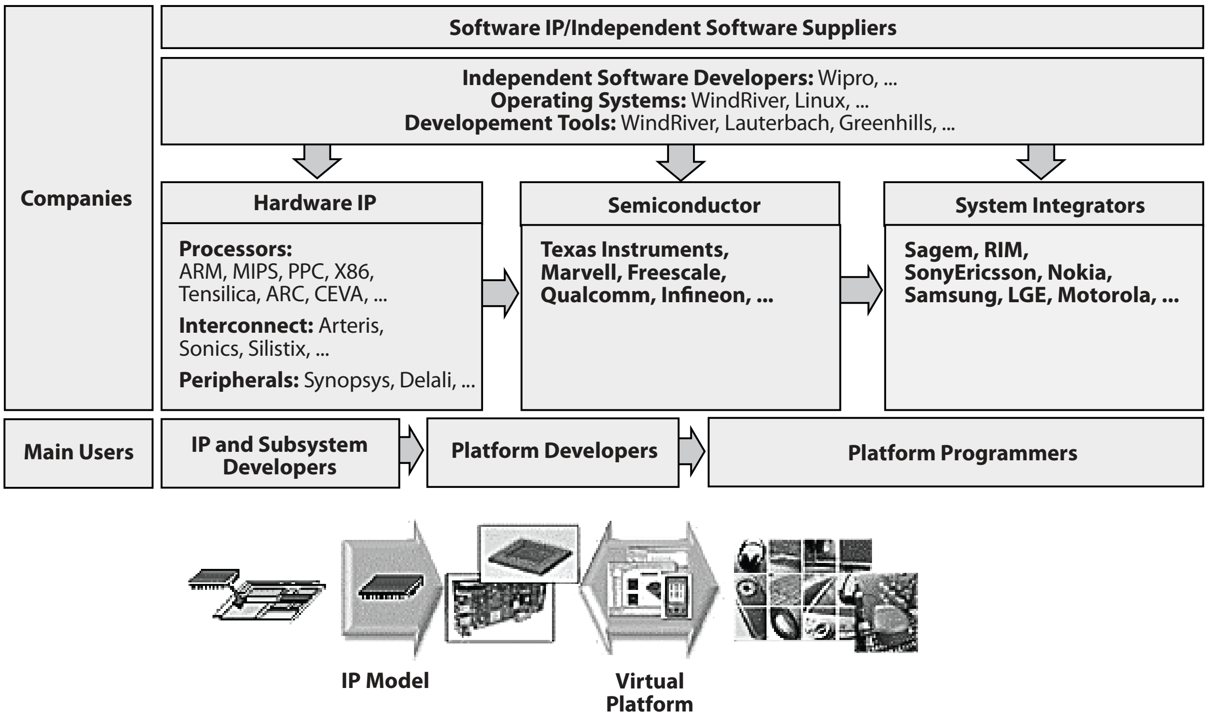

Companies
Software IP/Independent Software Suppliers
Independent Software Developers: Wipro, ...
Operating Systems: WindRiver, Linux, ...
Developement Tools: WindRiver, Lauterbach, Greenhills, ...
Hardware IP
Processors: ARM, MIPS, PPC, X86, Tensilica, ARC, CEVA, ...
Interconnect: Arteris, Sonics, Silistix, ...
Peripherals: Synopsys, Delali, ...
Semiconductor
Texas Instruments, Marvell, Freescale, Qualcomm, Infineon, ...
System Integrators
Sagem, RIM, SonyEricsson, Nokia, Samsung, LGE, Motorola, ...
Main Users
IP and Subsystem Developers
Platform Developers
Platform Programmers
IP Model
Virtual Platform

Figure 3: Wireless Design Chain and Users

System OEMs are the actual interface to the consumer, using the network providers like Vodafone and T-Mobile as their channel. Semiconductor houses provide the silicon, including reference design kits to the system houses. Given the significant amount of IP reuse in today's chip designs, the semiconductor vendors often license IP from external hardware IP providers. Semiconductor and system houses interact with software IP providers – the independent software vendors (ISVs), which include both operating systems (e.g., Symbian OS, WinCE, Android™ and Linux) and application software developers like those who develop multimedia and game applications.

With respect to users, the software programmers are split among semiconductor houses, system houses and ISVs. Programmers in the semiconductor houses and IP providers are traditionally more focused on lower-level drivers and the OS kernel, while the ISVs and system houses actively use middleware and application software as their differentiator.

Traditional Methods Running out of Steam

Figure 4 shows a semiconductor case study from the wireless application domain. The design, including volume production, had a lifecycle of over 3 ½ years and required two re-spins before volume ramped up after 2 ½ years.

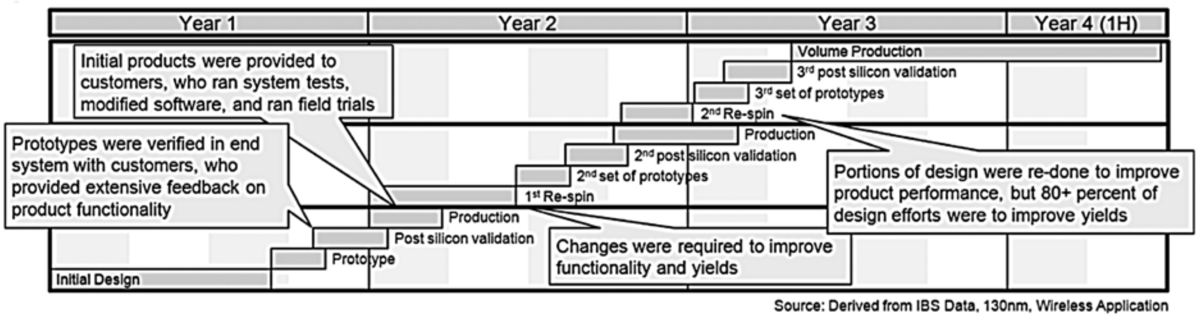

Year 1
Year 2
Year 3
Year 4 (1H)
Initial products were provided to customers, who ran system tests, modified software, and ran field trials
Prototypes were verified in end system with customers, who provided extensive feedback on product functionality
Volume Production
3rd post silicon validation
3rd set of prototypes
2nd Re-spin
Production
2nd post silicon validation
2nd set of prototypes
1st Re-spin
Portions of design were re-done to improve product performance, but 80+ percent of design efforts were to improve yields
Production
Post silicon validation
Prototype
Initial Design
Changes were required to improve functionality and yields
Source: Derived from IBS Data, 130nm, Wireless Application

Figure 4: Example Semiconductor Case Study

First hardware prototypes were available after nine months of the initial design and were actively used in customer interaction to get feedback. The first small production run was used for system tests, field trials and software development only. After feedback on functionality was received, a re-spin was performed, which also improved yield issues. Even after 1 ¾ years, some performance and yield improvements were required, so a second re-spin was performed prior to ramping up volume. The product had a mainstream lifetime of about 2 ½ years before successors took over. At about $200M overall development and production costs, profits of about 25% in total were achieved.

While hardware issues played a central role in delaying the actual time to revenue for this project, late start of software development also was a key issue for delays. Traditionally, several techniques have been employed to enable or even accelerate software development and verification:

- **Software Prototypes** using C/C++/ SystemC had always been intended to support the actual embedded software development, but until recently, SystemC has not possessed all the necessary technology components to fully enable it. For starters, within the Transaction-Level Modeling (TLM) working group of OSCI, several different abstraction levels were introduced without clear guidance about how to use the models. While experts understood the differences, the varying abstraction levels left some users confused, forcing them to create custom derivatives to address specific needs. In addition, when it came to processor models, the level of abstraction was typically dictated by hardware designers and often included more detail than software programmers really needed. In turn, execution speeds were in the single-MIPS or even sub-MIPS range, which is far too slow to serve the needs of most software programmers.

- **FPGA Prototypes** can be created once RTL is available and hardware verification has progressed reasonably. Their advantage is that they are available prior to silicon, albeit typically only by weeks or perhaps a couple of months, and are close to real time, providing a good platform for the actual software development. Unfortunately they are typically available too late to impact architecture and performance issues in the current design. It can be a tedious process to hook up debuggers via JTAG ports, and control and visibility into the design are sometimes limited. After being used as an FPGA prototype, custom boards are often simply thrown away. Given that the FPGA is like a second target technology, the actual development can require either an independent, additional team, or can take away resources from the actual chip RTL implementation teams. In complex designs, partitioning issues can become quite difficult.

- **Emulation** is a technique also used, but after the RTL is available and verification has progressed reasonably. Emulation typically is slower than FPGA prototypes and is hence used mostly for low-level software validation, not actual software development. The setup time for emulation – while improved in recent years – still is often a matter of weeks, but in exchange control and visibility into the design are excellent and the actual partitioning of the design is largely an automated process. However, traditionally high capacity emulation has been prohibitively expensive.

- **Hardware Development Kits** are made available after silicon is available. They run at real time and most of the disadvantages of FPGA prototypes apply here as well. Their costs are typically in the low $1000's and the process of deployment to users, and keeping them updated with any design changes, is logistically non-trivial.

The Need to Change the Design Flow

Traditional techniques to enable software development have been too slow, too late, too expensive, too difficult to use or any other combination. With software development determining project success and failure, new techniques are required. The objectives for a virtual platform based design flow are indicated in Figure 5.

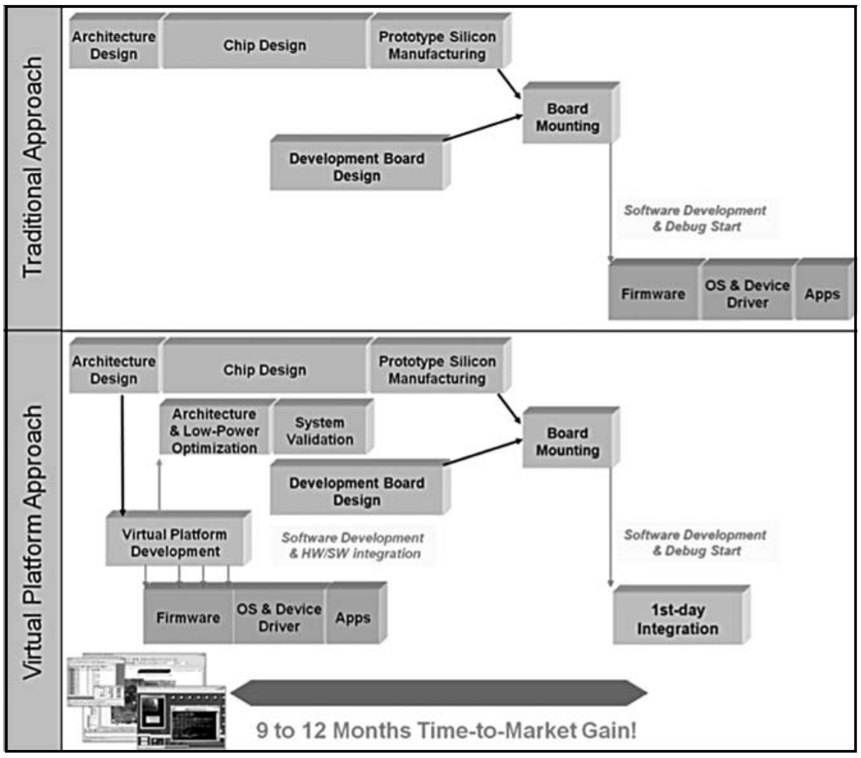

Traditional Approach
Architecture Design
Chip Design
Prototype Silicon Manufacturing
Board Mounting
Development Board Design
Software Development & Debug Start
Firmware
OS & Device Driver
Apps
Virtual Platform Approach
Architecture Design
Chip Design
Prototype Silicon Manufacturing
Architecture & Low-Power Optimization
System Validation
Board Mounting
Development Board Design
Virtual Platform Development
Software Development & HW/SW integration
Software Development & Debug Start
Firmware
OS & Device Driver
Apps
1st-day Integration
9 to 12 Months Time-to-Market Gain!

Figure 5: Time to market savings using virtual platforms

As indicated above, traditional product development approaches led to a sequential process which ended with a "big bang" integration of hardware and software, after hardware prototypes or development boards became available. A virtual platform approach allows delivery of a software development platform much earlier, as part of the architecture design phase. A staged delivery allows instruction accurate models in as little as a few weeks, which enables firmware, OS and driver development often before the first line of RTL is coded or re-used IP is qualified. Pre-silicon software development is the main use model for virtual platforms today. However, timing approximation can be added with reasonable incremental effort to the virtual platform models in later pre-implementation stages, which in turn allows feedback on the actual chip design, often for the same revision of the design still in development.

Introducing Virtual Platforms

A virtual platform is a fully functional software representation of a hardware design that encompasses a single- or multi-core SoC, peripheral devices, I/O and even the user interface. The virtual platform runs on a general-purpose PC or workstation and is detailed enough to execute unmodified production code, including drivers, the OS and applications at reasonable simulation speed. Users have articulated the need for virtual platforms to not be slower than 1 tenth of real time to be effective for embedded software development. The achievable simulation speed depends on the level of model abstraction, which also determines the platform's accuracy.

To be most effective at improving productivity, a virtual platform is available as early as possible, sometimes within weeks of the architecture being specified. Even at the beginning of an SoC design flow, the system architecture can be defined to a level of detail that enables an unambiguous executable specification — a virtual prototype of the system or virtual platform. This platform can be used to develop and integrate the software, which can in turn be used to refine the hardware architecture in an iterative process. It is sufficient to allow a "just-in-time" delivery in phases, i.e. starting with an instruction accurate version and delivering timed versions later in time.

For hardware designers, virtual platforms support the SoC design flow with mixed-mode simulation and a path to RTL design and verification. A virtual platform can integrate with simulation acceleration technologies and even actual hardware in the loop. Virtual platforms can be particularly useful for refining high-level power-reduction strategies, because these strategies depend on architectural choices and the evaluation of software effects on the chip's power consumption.

Rather than creating an entirely new methodology, a virtual platform supports existing best-practice design, debug and verification processes for both hardware and software developers. The platform interfaces with a range of hardware and software tools, including standard languages for hardware and software development. For software engineers, for example, the virtual platform enables the use of development environments such as Texas Instruments' Code Composer Studio, ARM's RealView, the GNU tool chain and debug environments from other vendors.

Most importantly, the platform ensures full binary compatibility with production software. To get this level of compatibility, it is not necessary to use cycle-accurate models. Such models provide a high level of detail and timing metrics, but are more difficult to validate, require additional development effort and typically execute software at only about 500 kilo-instructions per second. This performance level is too slow for porting an operating system, integrating middleware and developing applications.

In the past decade, proprietary solutions for virtual platforms designed to accelerate early software development have been introduced by several companies, including Virtio (now Synopsys). These virtual platforms had to include several key technology components to enable the fast simulation speeds expected by programmers. In the absence of standards, these have originally been implemented using proprietary technology component. As a result of these key technologies, virtual platforms were able to run above 50 MIPS and, sometimes, even reach several 100 MIPS of execution speed.

Over the last two years in particular, users and vendors have realized that "it's all about the models," and that model interoperability is key to maintaining and growing a healthy virtual prototyping industry. Important donations to OSCI's TLM working group have evolved into the draft TLM-2.0 standard, which now includes the transaction abstractions so that all virtual platform components can communicate and be interoperable. The draft standard has passed public review and is on track to be ratified mid-year 2008 and subsequently contributed to IEEE. At a recent North American SystemC User Group meeting, TLM-2.0 early adopters reported processor-based platforms running at 250MIPS utilizing SystemC TLM-2.0.

The magnitude of this standardization is equivalent to the introduction of Verilog in the late 1980s, leading to the eventual demise of proprietary languages like HiLo, DABL, UDL / I and n dot. SystemC TLM-2.0 is the key enabler for a standards-based virtual platform ecosystem with model interoperability and, going forward, will serve as the bridge between hardware and software design.

There is a growing eco-system based on the concept of transaction-level modeling using SystemC, including training, tools, and models. For example, Synopsys' DesignWare® System Level Library consists of over 80 transaction-level models including models of DesignWare Cores connectivity IP, AMBA and CoreConnect bus components, as well as IBM, MIPS and ARM processors.

SystemC TLM-2.0 Abstraction Levels

As indicated above, the value of virtual platforms within the design flow is tightly linked to two main aspects: first, the early availability to enable software development long before first silicon is available, and second, high simulation performance to allow the execution of software with almost real-time performance. Both aspects call for abstracting from unnecessary details. Abstraction allows for faster model creation, and achieves higher simulation performance.

The need for abstraction results in the concept of transaction-level modeling, making transaction level models the building blocks of virtual platforms. TLMs come in different flavors, categorized by the timing accuracy: these range from cycle-accurate to approximately timed [AT] to loosely timed [LT]. What is common to all of them is the way to model the communication among them: representing transactions flowing within the system instead of individual bus cycles, or specific pins and wires, and thus achieving higher simulation speeds.

Figure 6 illustrates the different abstraction levels at which SystemC TLM-2.0 virtual platforms can be used. While actual application development often can be done even without instruction accurate representation of the processor models (using what is indicated as Application View), the pre-silicon integration of firmware, OS and drivers at least requires instruction and register accurate representation of the hardware. Loosely timed virtual platforms focus on the hardware details that are exposed to software running on the systems. Processors in a virtual platform execute target program binaries on an instruction-by-instruction basis and are thus binary compatible with the actual hardware. Functional models of the memory management unit (MMU) and caches provide run-time statistics on cache hits and misses. To maximize speed, however, the platform avoids processor details that are not exposed to the programming model, such as the processor's pipeline.

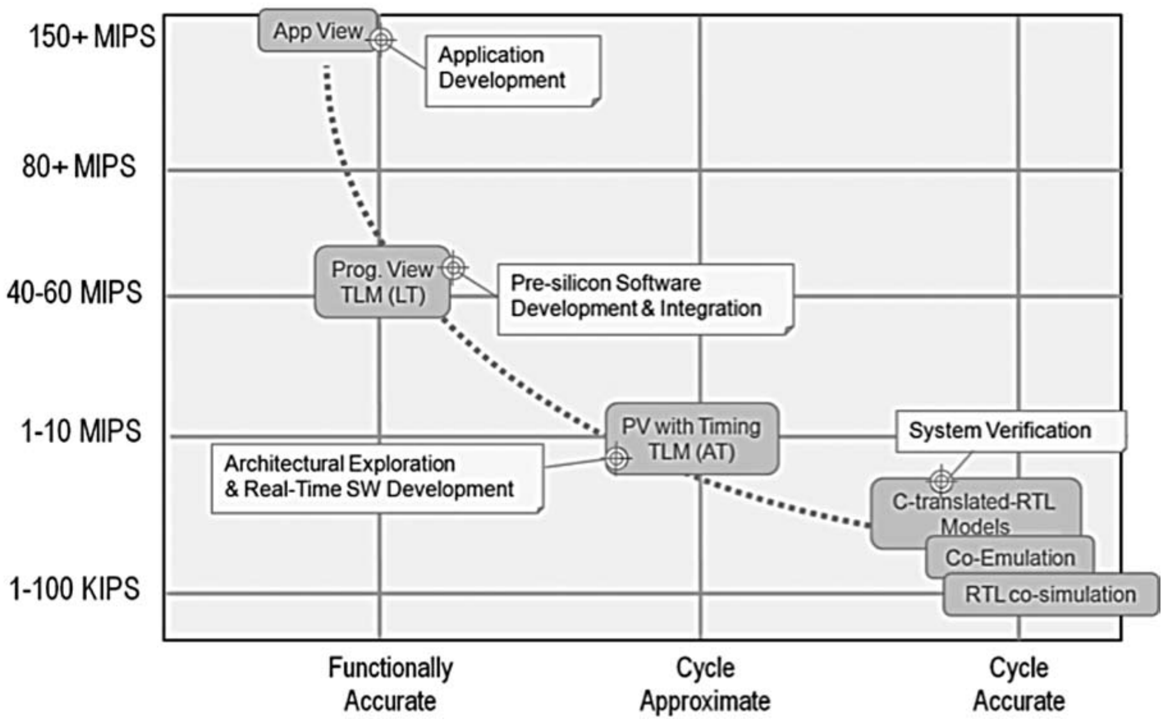


Figure 6: TLM Abstraction Levels and Use Models

Bus models in loosely timed platforms use a similar approach, reducing complex bus traffic to simple read and write transactions. These transactions include representations of address decoding and control registers for elements such as bus bridges and arbiters. However, the bus models avoid detailed bus phasing, arbitration and timing beyond simple access delays.

This abstraction level is the most commonly used one and ideal for early embedded software development of drivers, kernel and OS bring-up.

The next level of abstraction, adding yet more accuracy, is the approximately timed (AT) view, which is used for architectural exploration and real time software development. Approximately timed virtual platforms offer more timing detail. For example, these platforms might include cycle-approximate bus models that add arbitration, pipelining and concurrency details. Access timing for on- and off-chip peripherals can also be included. This additional timing information allows the platform to better support detailed software partitioning, performance tradeoffs and debugging of time-dependent problems such as race conditions. This is the level of abstraction at which annotation of timing and power to the virtual platform real hardware software trade-offs can be performed.

The remaining stages indicated in Figure 6 – C translated RTL models, co-emulation and RTL co-verification – are often not only too slow for real hardware/software trade-offs, they also require the actual implementation to have progressed to a stage at which changes to the current design are often no longer feasible. They are, however, useful for verification of low-level software in the context of hardware simulation.

Virtual Platforms for Pre-Silicon Software Development

For any software debug task, a virtual platform provides superb visibility and control. In addition to virtually unlimited signal trace and logging capability, the system exposes details such as memory management states and the intermediate states of specialized accelerators, without the wait for a JTAG dump.

Additionally, any debugger can freeze all system clocks at a breakpoint — a capability that is especially useful for multicore SoCs. In fact, virtual platforms solve a major problem with debugging software on the actual multicore hardware, where some clocks continue to run after a breakpoint, and some system states may be altered. By stopping all activity simultaneously, including inter-processor-communication hardware and peripheral clocks, a virtual platform preserves all the system states that led to the breakpoint.

With this visibility and the ability to run software long before actual hardware becomes available, virtual platforms have the potential to revolutionize the way software is ported to SoCs, beginning with OS bring-up and continuing to the porting of application software. To see how this process works, it is useful to consider the example of an SoC whose main processor runs Linux. To offer this SoC with a development board, the software team must create sample applications for the board as well as the basic software for the chip.

When working with a new physical processor, one of the first tasks is to develop a boot-loading routine that installs Linux in memory so that the OS can begin executing. At this point, dealing with problems is difficult because the OS resources needed to resolve problems are not yet in place.

A virtual platform bypasses this difficulty by providing direct access to the model's memory. The OS can be loaded directly into the model, and the boot sequence can be verified when the processor's basic operations have been proven out. By allowing developers to work with the OS using only minimal processor resources, the virtual platform limits the number of variables that have to be considered when debugging.

For any test run, developers can load the OS instantly, rather than waiting for the simulation model to boot each time. Similarly, flash and ROM contents can be updated instantaneously, rather than waiting the several minutes required to flash a physical development board. A virtual platform also saves time by allowing other system states to be set as desired through backdoor access.

After verifying that the OS and the core processor work together, developers can add additional hardware, and debug in manageable steps. When creating peripheral drivers, for example, developers can enable the peripherals as needed, making sure each peripheral works before adding another.

With the SoC's drivers and basic software in place, the development team can work on board-level drivers and application examples. The development board may not be as clearly defined as the SoC, but the team can still make headway by verifying software against different platform variants. If the choice between NAND or NOR flash memory has not been made, for example, the software team can use models for each memory type to make sure that the software works with both.

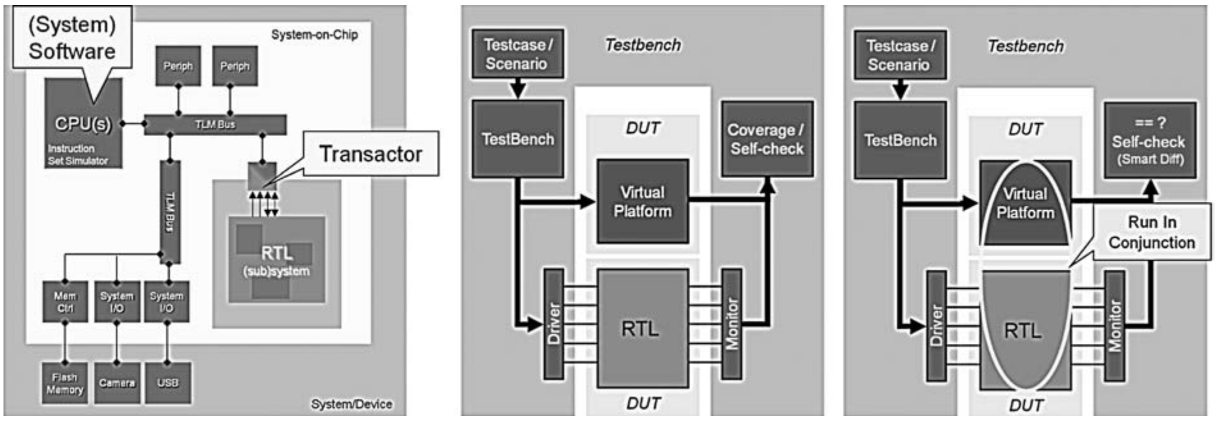
In any development project using a virtual platform, the software team will probably have to deal with uncertainties about details such as flash memory types, the address mapping of peripheral registers, and interrupt assignments. Some rework is inevitable as these details are resolved, and good regression testing is needed to ensure that the software stays in sync with the actual silicon. Still, it is reasonable to expect that about 85 percent of the software will run unchanged on the final hardware.

Links to Functional Verification

While the main use-model for virtual platforms is pre-silicon software development, as the SoC design cycle progresses, a virtual platform can evolve to meet different needs. There are three main use models of software driven verification which utilize the integration of virtual platforms with signal-level simulation at the register transfer level:

- When an RTL block becomes available, for example, it can be substituted for its transaction-level model in the virtual platform. Software can then be verified on this version of the platform as a way to validate both hardware and software.
- The virtual platform can also provide a head start towards testbench development and post silicon validation test by acting as a testbench component running actual system software. The virtual platform can be used to generate system stimuli to test RTL, and then verify that the virtual platform and RTL function in the same way.
- Additionally, as portions of the virtual platform are verified as equivalent to their corresponding RTL, the virtual platform can become a golden or reference executable specification.

Figure 7 illustrates these three use models bridging the TLM abstraction level to signal-level RTL simulation.



Using TLM model to generate stimulus for RTL verification

Early testbench creation and post Silicon validation test development

Verifying TLM platform and RTL equivalence

Figure 7: Use Models for Links to Verification

The interface from virtual platforms to traditional RTL simulation can be done using transaction-level interfaces, allowing the actual transactor to be written in SystemVerilog and the bus functional model to be synthesizable in order to allow co-execution with hardware based environments. Alternatively, the transactor can be written in SystemC and the interface to RTL simulation can be at the signal-level.

The first use model – using TLM models to generate stimulus for RTL verification – allows early software validation on the actual hardware implementation, as well as efficient scalable system validation, for which software becomes part of the verification setup. Knowing that real system scenarios are used, does increase verification confidence. Furthermore, simulation used in verification is faster, given that as much of the system as possible is simulated at the transaction-level.

The second use model – early test-bench creation and development of post silicon validation tests and development of post silicon validation tests on a virtual platform – allows development of all the test-bench infrastructure on a virtual platform, as well as early scenario and test-case creation. Users can efficiently develop on the TLM model "embedded directed software" tests, which can be used for system integration testing. As a result productivity of verification test case development increases.

The third use model – verifying TLM platform and RTL equivalence – helps to validate the correctness of the software development platform against the RTL once it becomes available. As a result users gain a single golden test-bench for the transaction-level and the RT level.

Managing power

Though power management is generally seen as a hardware issue, software has a great deal to do with how much power a SoC consumes. Certainly, hardware designers cannot know a design's actual power consumption until the software is available to drive the hardware in realistic usage scenarios.

Since virtual platforms enable early software development, they can also provide early insight into power consumption at an architectural and system level. With information about power consumption on an application-by-application basis, system architects can make realistic tradeoffs. For example, looking at execution frequency versus application runtime allows architects to make intelligent choices that can achieve significant power savings.

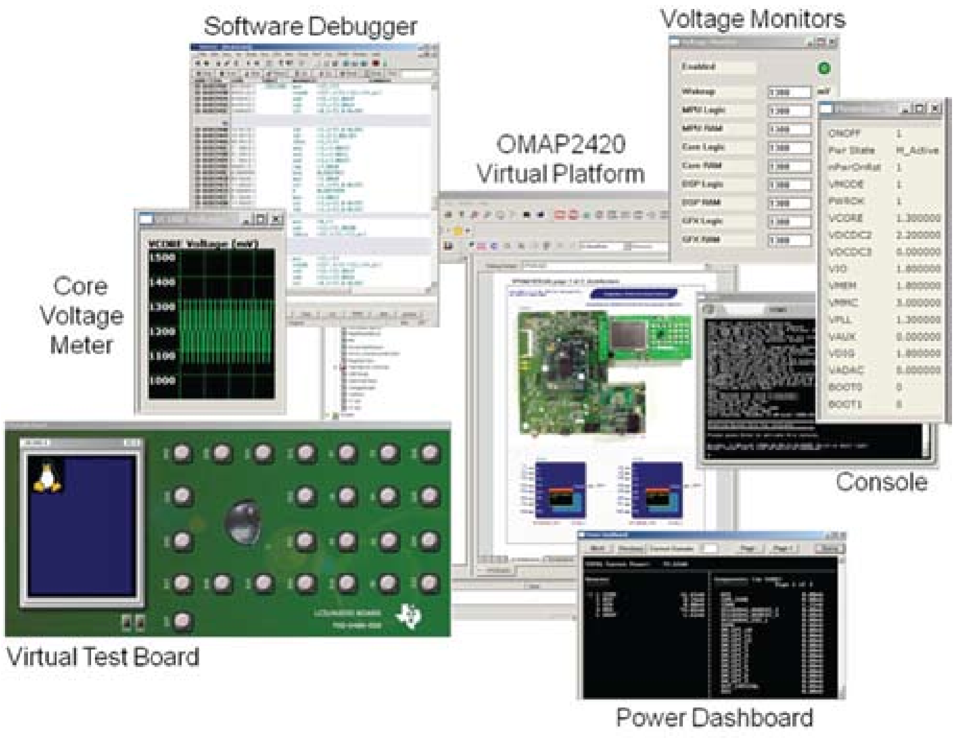


Figure 8: Example TI OMAP Platform in Synopsys Innovator with Power Dashboard

Further, some power management schemes use a software management layer or power-aware OS to manage a hardware controllear that handles settings for frequency scaling and various power domains. A virtual platform enables validation and refinement of this interlocking hardware/software scheme. Using a virtual platform that includes an ARM1136 processor core, for example, a system architect can get immediate visual feedback about the results of frequency scaling applied to the processor while running actual applications. The system can then be optimized accordingly.

Power consumption has become a key issue in the design of SoCs, whether the chips are battery powered or not, and power management is just one of the ways in which hardware and software are interdependent. By enabling early software development and hardware/software co-design, virtual platforms supply a critical part of the SoC puzzle that has been missing. As SoC complexity continues to grow, this solution will become increasingly vital for meeting both schedule and functionality goals.

Figure 8 shows the top-level of a virtual platform of Texas Instruments OMAP 2420 in its development environment. It has been developed in Synopsys Innovator using transaction-level models and has been instrumented with characterized power information. Several control and visualization elements are shown, including a software debugger attached to the OMAP 2420 ARM processor core executing the actual application software while providing associated power information. Various power related windows show the core voltage, the current voltage in the different power domains, a power dashboard for more detailed analysis and a virtual representation of the actual test board.

Summary

Given the growing importance of software, virtual platforms are a key component for reducing time to market in high software content chip designs. The most cleverly architected chip will not sell if the associated software is not ready. Hence the main use model for virtual platforms is the enablement of pre-silicon software development to allow as much software development as possible to happen pre-silicon. Once virtual platforms are available, they can be used as part of the verification flows as well.

The return on investment into virtual platforms has several components:

- By using a virtual platform based on the architecture specification, the actual implementation cost for the phase from specification to RTL and net list can be reduced using early design validation. An executable reference virtual platform especially helps to avoid communication issues when implementing the RTL from specification.
- Getting a virtual platform into the hand of software developers early avoids "lost integration days" by virtually advancing the integration phase to a much earlier time in the project. No hardware development kits need to be shared, every software developer can be equipped with a virtual platform exactly when needed.
- Virtual platforms help augment hardware based verification based on FPGA prototypes, emulation or hardware acceleration. Only the tests absolutely essential need to be executed on hardware and co-execution with virtual platforms increases the utilization of hardware based verification.
- Given the "electronic" nature of virtual platforms they can be distributed to users easily and at low cost. In some instances and for some use models they can replace hardware development kits, which significantly reduces cost.
- Finally, given the parallel software development virtual platforms enable, users completely avoid the penalties of arriving late at market. Figure 9 shows the cost and revenue structure of the example project already introduced in Figure 4. A three month delay in a fast moving market for this particular project would have meant, according to IBS estimates, a 50% loss in profit – almost $30M in this particular case.

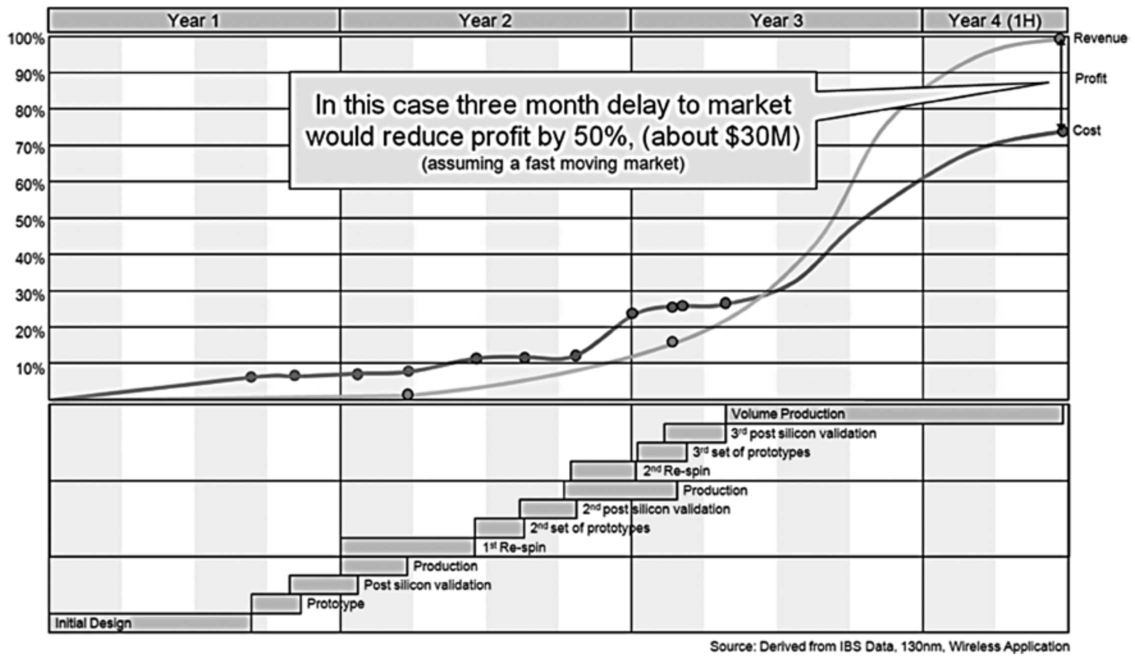


Figure 9: Example Chip Project Cost Structure

After a decade of proprietary virtual platform solutions, SystemC TLM-2.0 standardization has now provided the appropriate interfaces to enable efficient model interoperability. SystemC based virtual platforms are ready for mainstream adoption and are becoming a key component for software-rich chip design projects.

Sources

[1] Graphic source:
http://focus.ti.com/pdfs/wtbu/ti_omap3430.pdf
[2] Graphic source:
http://www.palminfocenter.com/news/9246/new-alp-details-and-screenshots/

SYNOPSYS®
Predictable Success