cadence®

Productivity, Predictability, and Use-Model Versatility: The Three Key "Care-Abouts" of Choosing Hardware-Assisted Verification

By Frank Schirrmeister, Cadence Design Systems

Hardware-assisted verification and prototyping has become a mandatory requirement to allow design teams to gain confidence that a chip tapeout can be initiated. The choice of the right hardware-accelerated engine is driven by its productivity, predictability, and use-model versatility, all impacting the key concern of users how to remove bugs. The Cadence® Palladium® XP Platform allows design teams to get to the point at which they are confident enough to tapeout much faster, often shaving off two to four months off development cycles.

Contents

Introduction

The electronics revolution continues to accelerate at a faster and faster pace. According to industry estimates by ARM, Gartner, IDC, and SIA, by 2020 we will connect 8 billion people worldwide, with 6 billion cell phones alone. At that time, the overall unit chip shipments may reach 48 billion across the mobile, home consumer, enterprise, and embedded markets—an average of 6 chips per human on earth. At those volumes, it is mandatory that system on chips (SoCs)—from sensors combined with microcontrollers to complex multi-core application processors—work correctly within their system environments, together with the associated software executing on them. This complexity poses a huge challenge to the electronic design automation (EDA) industry.

While there are certainly differences in other application domains, let's consider prototyping in applications for mobile communications as an example to analyze challenges. When comparing the block diagrams of some of the enabling SoCs like the Qualcomm MSM8960 Snapdragon S4 processor [1], the TI OMAP4 platform [2], the NVIDIA Tegra 2 [3], and ST Ericsson's DB9500 [4], a couple of common characteristics can be derived: all of these devices have multiple processing cores, 2D/3D graphics engines, video and audio acceleration engines, complex interconnect, and peripherals to connect to the environment.

Hardware/Software Development Challenges

All these different components, shown in Figure 1, have different characteristics as they relate to processing and throughput needs, which significantly impact the ways they can be prototyped. And as the illustration clearly shows, the chip hardware and software must be considered within the system they reside in.

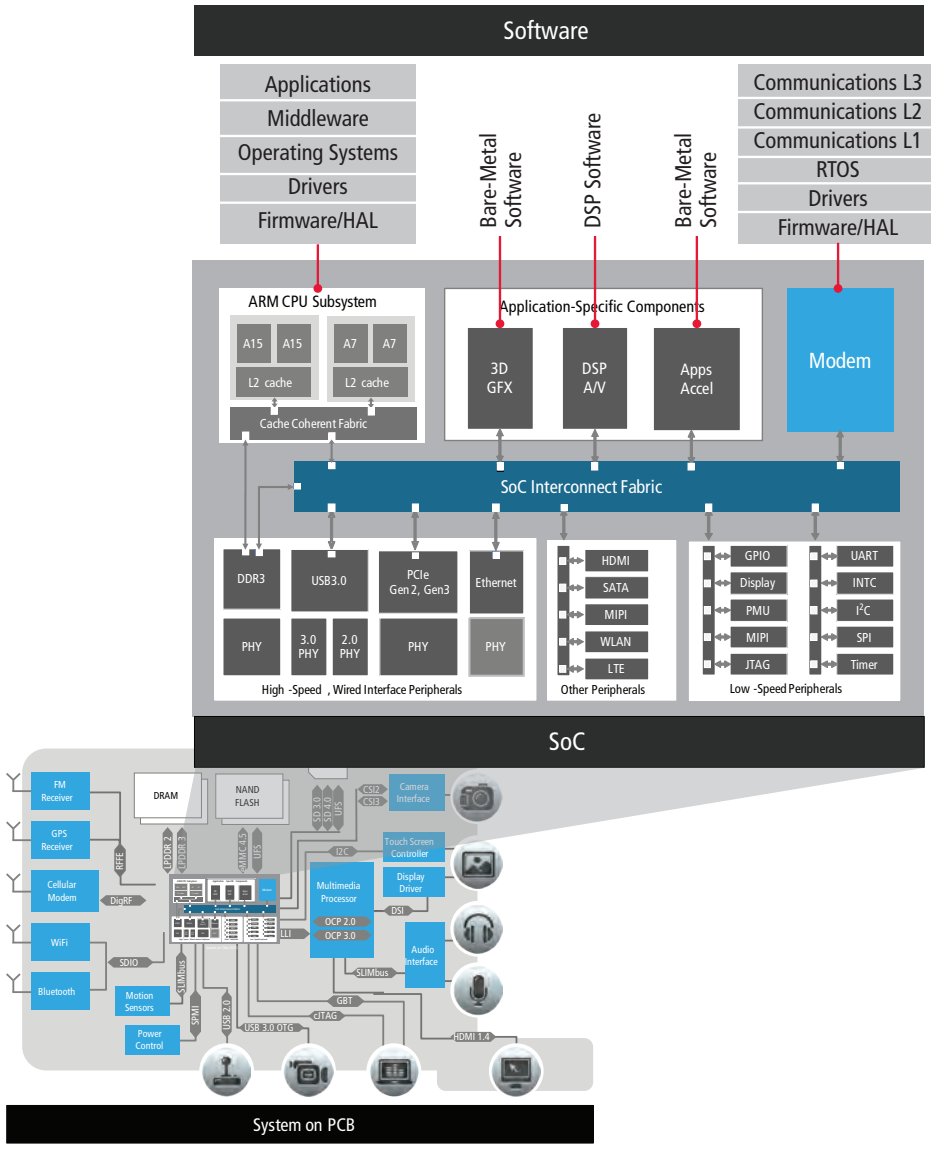

Figure 1: A typical SoC with software in its system context.

Challenges for the chip development itself include:

- Multi-core software development and hardware/software verification
- SoC software and IP integration with 10s to 100s of IPs
- Configuration and verification of the more and more complex growing SoC interconnect that may include AXI Coherency Extension (ACE) protocols introducing new complexity to verify and verification of complex performance aspects
- Low-power design features spanning hardware/software, which must be verified at the SoC level.

At the system-level, users must bring up software stacks and use them for verification, and must represent the SoC environment appropriately with the right test scenarios to drive all the test cases to debug all of the hardware and software effectively. The challenges can be daunting.

Recognizing the growing importance of software and verification, the electronics industry has accepted the fact that system development without hardware-assisted prototyping is too risky, and today most companies demand the use of prototypes prior to silicon tapeout. There are, however, several different types of prototypes—both software and hardware based—that allow hardware/software integration, verification, and debugging. In addition, different users within a design team have potentially different needs for prototype capabilities. Just exactly what type of prototype to choose is not always 100% clear, and the multitude of choices can make it hard for design teams to find the right combination of prototypes to support their needs.

System Development Project Flow and Key Challenges

Analysts IBS and Semico estimate that by 2015, design teams will need to integrate on average more than 130 IP blocks, with reuse exceeding 70% of their design. More than 60% of the chip-related development effort will have moved into software, with multicore designs requiring software to be distributed across cores. Design teams will face significant low-power issues, and designs will become more and more application specific, with high analog/mixed-signal content and very complex on-chip interconnect structures.

Figure 2 shows an example chip-design flow with some of the key milestones and dependencies. The lower half indicates some of the major project steps as averaged from an analysis of 12 projects.

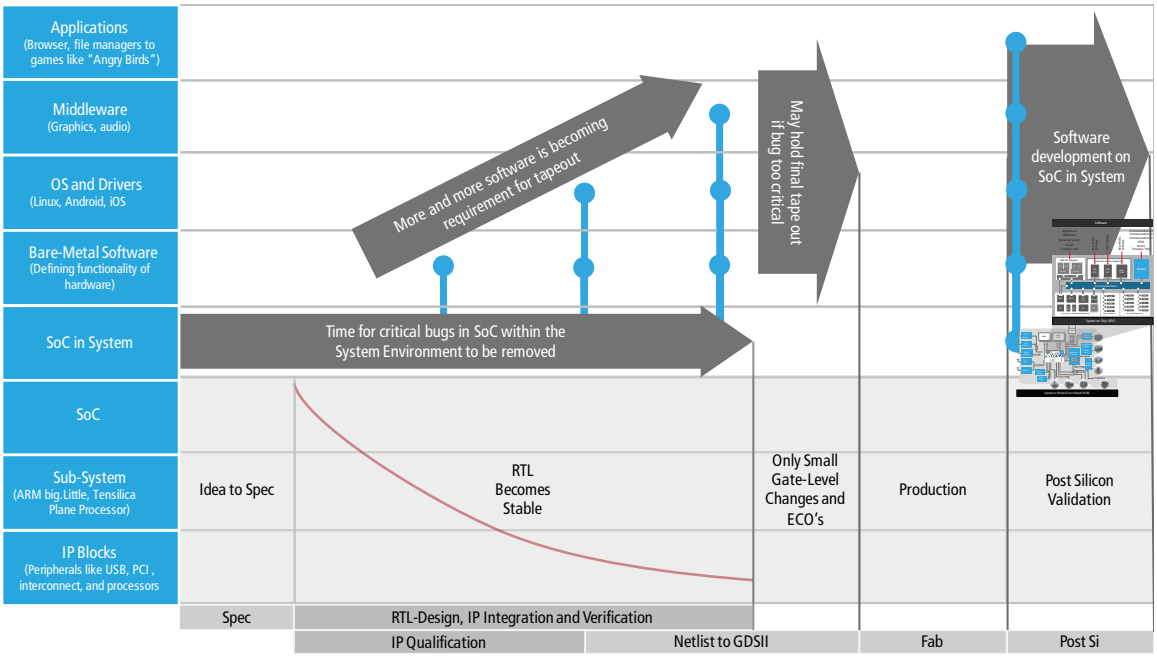

Figure 2: An example chip design flow with key milestones and dependencies.

A specification phase of 8 to 12 weeks is followed by a phase combining register transfer level (RTL) design, integration, and verification, with a major factor these days being the qualification of IP. The overall duration from RTL to GDSII at tapeout ranges from 49 to 83 weeks. A key point is that only small gate-level changes and engineering change orders (ECOs) are allowed in the last 10 to 12 weeks, as the focus of development shifts to silicon realization. The actual tapeout is followed by an 11 to 17 week production phase and 14 to 18 weeks of post-silicon validation.

The left axis indicates the hardware/software development stack. The SoC is integrating sub-systems and IP blocks and then operates within its system environment, i.e., the PCB board and package. Different types of software execute on the SoC, from bare-metal software that, together with its associated hardware, actually defines functionality of the chip, to drivers and operating systems (OSs) like Linux, Android, iOS, Windows, or real-time OSs like OSE or vxWorks. These OSs are hosting middleware for audio, video, graphics, and networking that in turn enable the end applications that define our end-user experiences.

A couple of key dependency aspects are also indicated in Figure 2. As RTL matures during the design flow, there comes a point at which hardware functionality must be frozen as the focus shifts towards silicon implementation, and only changes at the gate level can be easily implemented. At that point, all the aspects of the chip within its system environment also must be verified, posing unique challenges to execution platforms for the RTL at that stage in the project.

The other aspect relates to software development. Usually, the interactions at the hardware/software interfaces must be validated as early as possible and today proper boot of OSs has become a de-facto requirement to allow tapeout. Again, this requirement poses unique challenges for the RTL execution engines as a large number of cycles must be executed to get OSs to boot. While the actual execution will continue during the final phase prior to tapeout, decisions to hold the tapeout at that point due to software issues must be considered very carefully. Once the chip is back from production, software development can be finalized using the actual chip prototypes.

Core Execution Engines for Verification and Software Development

During the project phases mentioned earlier, verification and software development is mainly done on four different core execution engines—"virtual prototypes," "RTL simulation," "acceleration and emulation," and "FPGA-based prototypes."

Virtual prototypes either architectural or software based. Architectural virtual prototypes are mixed-accuracy models that enable architecture decision making. The items in question—bus latency and contention, memory delays, etc.—are described in detail, maybe even as small portions of RTL. The rest of the system is abstracted as it may not exist yet. The main target users are system architects. Architecture virtual platforms are typically not functionally complete and they abstract environment functionality into traffic driving the architectural model. Specifically, the interconnect fabric of the previous examples is modeled in full detail, but the analysis is done per sub-system. Execution speed may vary greatly depending on the amount of timing accuracy, but normally is limited to 10s to low-100s of KHz.

Software-based virtual prototypes run the actual software binary without re-compilation at speeds close to real time—50s to 100s of MHz. Target users are software developers, both apps developers and hardware-aware software developers. Depending on the need of the developer, some timing of the hardware may be more accurately represented. This prototype can be also used by hardware/software validation engineers who need to see both hardware and software details. Due to the nature of "just-in-time binary translation," the code stream of a given processor can be executed very fast natively on the host. The fast execution makes virtual prototypes great for software development, but modeling other components of the example systems—like the 3D engines—would result in significant speed degradation.

RTL simulation executes the same hardware representation that is later fed into logic synthesis and implementation. This core engine is the main vehicle for hardware verification and it executes in the Hertz range, but it is fully accurate as the RTL becomes the golden model for implementation, allowing detailed debug.XVerification acceleration executes a mix of RTL simulation and hardware-assisted verification, with the testbench residing on the host and the design under test (DUT) executing in hardware. As indicated by the name, the primary use case is acceleration of simulation. This combination allows engineers to utilize the advanced verification capabilities of language-based testbenches with a faster DUT that is mapped into the hardware accelerator. Typical speed-ups over RTL simulation can reach or exceed 1000x but is typically limited to 10s of KHz.

Emulation executes the design using specialized hardware—verification computing platforms—into which the RTL is mapped automatically and for which the hardware debug is as capable as in RTL simulation. Interfaces to the outside world (Ethernet, USB, and so on) can be made using rate adapters. In-circuit emulation (ICE) takes the full design and maps it into the verification computing platform, allowing much higher speed-up into the MHz range, and enabling hardware/software co-development. While ICE typically does not consider testbenches but instead executes the design within its system environment, in-between verification acceleration and in-circuit emulation, so called synthesizable or embedded testbenches (STBs, ETBs) are mapping both the test environment and the design itself into the verification computing platform, allowing faster execution than verification acceleration itself.

FPGA-based prototyping uses an array of FPGAs into which the design is mapped directly. Due to the need to partition the design, re-map it to a different implementation technology, and re-verify that the result is still exactly what the incoming RTL represented, the re-targeting and bring-up of an FPGA-based prototype can be cumbersome and take months (as opposed to hours or minutes in emulation) and hardware debug is a difficult process. In exchange, speeds will go into the 10s of MHz range, making software development a realistic use case.

Figure 3 illustrates the key upsides and downsides of the four core engines together with two additional flanking engines, software development kits (SDKs) in the front, and the actual silicon-based prototype after the four core engines.

SDKs typically do not run the actual software binary but require re-compilation of the software. The main target users are application software developers who do not need to look into hardware details. SDKs offer the best speed but lack accuracy. The software executing on the processors, as in the SoC examples given earlier, runs natively on the host first or executes on abstraction layers like Java. Complex computation, as used in graphics and video engines, is abstracted using high-level APIs that map those functions to the capabilities of the development workstation.

Silicon-based prototypes come in two incarnations. First, like the SDK in the pre-RTL case, the chip from the last project can still be used, especially for apps development. However, the latest features of the development for the new chip are not available until the appropriate drivers, OS ports, and middleware become available. Second, there is the actual silicon prototype once the chip is back from fabrication. Now users can run at real speeds, with all connections, but debug becomes harder as execution control is not trivial. At that level, the execution is also hard to control. Starting, stopping, and pausing execution at specific breakpoints is not as easy as in software-based execution, FPGA-based prototyping, and acceleration and emulation.

SDK OS Sim	Virtual Platform	RTL Simulation	Acceleration Emulation	FPGA Prototype	Prototyping Board
• Highest speed • Earliest in the flow • Ignore hardware	• Almost at speed • Less accurate (or slower) • Before RTL • Great to debug (but less detail) • Easy replication	• KHz range • Accurate • Excellent hardware debug • Little software execution	• MHz range • RTL accurate • After RTL is available • Good to debug with full detail • Expensive to replicate	• 10s of MHz • RTL accurate • After stable RTL is available • OK to debug • More expensive than software to replicate	• Real-time speed • Fully accurate • Post silicon • Difficult to debug • Sometimes hard to replicate

Figure 3: Key characteristics of hardware/software development engines.

Key Users and Development Tasks Drive Sweet Spots of the Core Execution Engines

Let's compare the main users, use models, and care-abouts that can be satisfied by the different characteristics of core execution engines.

Application software developers need a representation of the hardware as early as possible. It must execute as fast as possible and must be functionally accurate. This type of software developer would like to be as independent from the hardware as possible, and specifically does not need full timing detail. For example, detailed memory latency and bus delays are usually not of concern.

Similarly, hardware-aware software developers, i.e., developers of low-level software like firmware, also would like representations of the hardware to be available as early as possible. However, they must see the details of the register interfaces and they expect the prototype to look exactly like the target hardware will look. Depending on their task, timing information may be required. In exchange, this type of developer is likely to compromise on speed to gain the appropriate accuracy.

System architects care about early availability of the prototype, as they must make decisions even before all the characteristics of the hardware are defined. They must be able to trade off hardware versus software and make decisions about resource usage. For them, the actual functionality counts less than some of the details. For example, functionality can be abstracted into representations of the traffic it creates, but for items like the inter-connect fabric and the memory architecture, very accurate models are desirable. In exchange, this type of user is willing to compromise on speed and does not typically require complete functionality as the decisions are often made at a sub-system level.

Hardware verification engineers typically do need precise timing accuracy of the hardware, at least on a clock cycle basis for the digital domain. Depending on the scope of their verification assignment, they must be able to model the impact of software as it interacts with the hardware. Accuracy definitely trumps speed, but the faster the prototype executes, the better the verification efficiency. This type of user also cares about being able to reuse testbenches once they have been developed.

Hardware/software validation engineers make sure the integration of hardware and software works as specified, and they need a balance of speed and accuracy to execute tests of significant length to pinpoint defects if they occur. This type of user especially must be able to connect to the environment of the chip and system to verify functionality in the system context.

The resulting key tasks to be performed by those users during the development include:

- System modeling and tradeoffs
- Early software development
- IP selection and design verification
- SoC and sub-system verification
- Gate-level timing and power signoff
- Hardware/software validation for SoC and bare-metal software
- Software integration and QA
- System and silicon validation

Figure 4 outlines the different engine sweet spots as an overlay on the main user tasks.

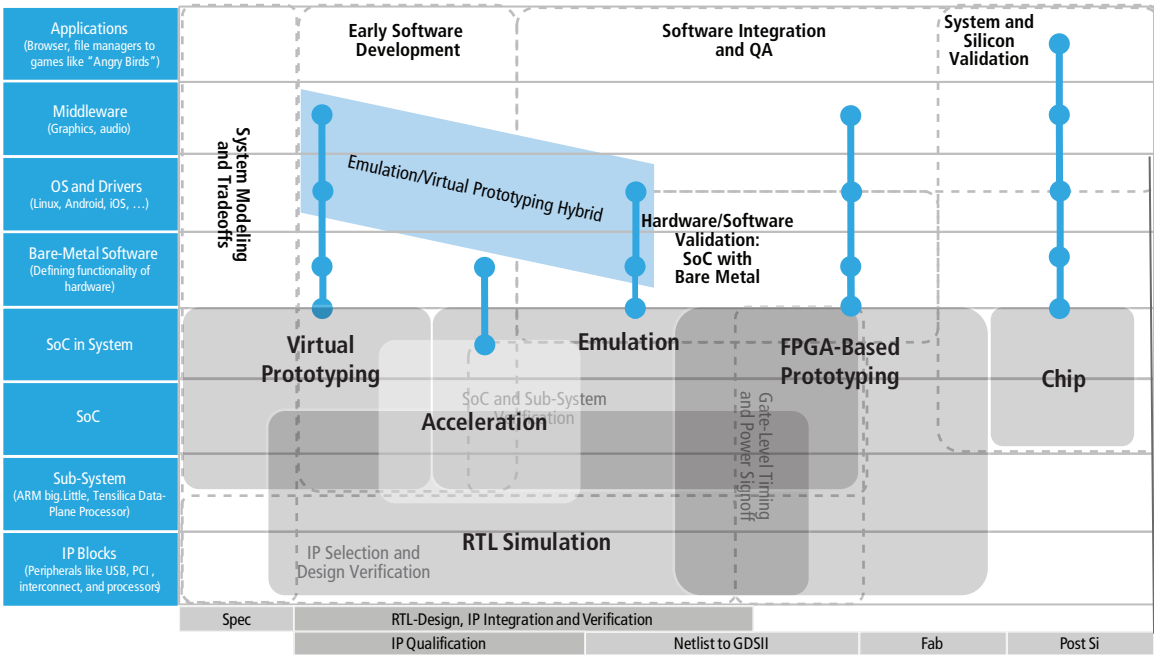

Figure 4: Sweet spots for execution engines to run verification and software development.

As indicated earlier, depending on whether models are available, virtual prototyping can enable software development as early as a couple of weeks after the spec is available. It is fast, allows good software debug insight and execution control, and is typically the quickest way to bring up software on a new design. By itself, it does not allow detailed hardware debug, which is the strength of RTL simulation. Used initially for RTL development, IP integration, and design verification, RTL simulation can extend to the complexity of sub-systems and certainly is a sign-off criterion for gate-level simulation and timing sign-off. It allows the fastest turnaround time for new RTL and offers excellent hardware debug, but is typically too slow to execute meaningful amounts of software.

To better extend to sub-systems and the full SoC, simulation acceleration moves the DUT into hardware and can allow enough speedup for bare-metal software development. With its in-circuit capabilities, emulation extends the verification to the full chip and chip-in-system level by enabling connections to real system environments like PCI, USB, and Ethernet. The main advantage of processor-based emulation is fast turnaround time for bring-up, which makes it ideal for the project phase in which RTL is not quite yet mature. In addition, it allows multi-user access and excellent hardware debug insight in the context of real software that can be executed at MHz speeds, resulting in very efficient hardware/software debug cycles. Standard software debuggers can be attached using JTAG adaptors or virtual connections. FPGA-based emulators are typically weaker with respect to debug efficiency and turnaround time.

FPGA-based prototyping allows the extension of the speed range into the 10s of MHz range and often offers the best cost per gate per MHz for software development and hardware regressions. In the project phase, the RTL is stable enough that fast turnaround time and hardware debug matter less. The downside to standard FPGA-based prototyping is the capacity limitations as well as longer bring-up due to the changes that must be made to map the RTL to FPGAs.

Productivity, Predictability, and Use-Model Versatility

To summarize, the key issues to consider when choosing a development platform fall into three main categories: productivity, preditability, and use-model versatility

Productivity

Productivity is a more general aspect not limited to execution speed. It includes other aspects like emulation capacity, debug visibility, the number of users that can use the emulator in parallel, available memory capacity, and bring-up time. Features like debug trace depth and the number of parallel possible users can mean that an offering with a buffer not big enough or that is accessible only by 1/16th of the users must run multiple times to get the same amount of debug information, very quickly eliminating any alleged advantages in power consumption as they must run multiple times to yield the same results, which also takes much longer. Specifically, our user feedback shows that the key care-about is the ability to remove bugs as fast and efficiently as possible, getting the design to a point that the confidence for its correctness is high enough that tapeout can take place. It all boils down to the loop of: (1) design bring-up, (2) execution of test, (3) efficient debug, (4) bug fix, and back to (1) to confirm that the bug has indeed been removed. To enable best-in-class productivity, the following care-abouts influence the user's choice:

- Speed—How fast does the execution engine execute? Previous-generation chips and actual samples are executing at actual target speed. Software virtual prototypes without timing annotation are next in line, followed by FPGA-based prototypes and in-circuit emulation and acceleration. Speed is often seen as the main concern, but it must be balanced with accuracy, bring-up efforts, and debug insight.
- Accuracy—How detailed is the hardware that is represented compared to the actual implementation? Software virtual prototypes based on transaction-level models (TLMs), with their register accuracy, are sufficient for a number of software development tasks including driver development. However, with significant timing annotation, speed slows down so much that RTL in hardware-based prototypes often is actually faster.
- Capacity—How big can the executed design be? Here the different hardware-based execution engines differ greatly. Emulation is available in standard configurations of up to multiple billion gates, and standard products for FPGA-based prototyping are in the range of up to about 100 million gates, but often suffer speed degradation with larger designs when multiple boards are connected to enable higher capacity. Software-based techniques for RTL simulation and virtual prototypes are only limited by the memory capabilities of the executing host. Hybrid connections to software-based virtual platforms allow additional capacity extensions.
- Development cost and bring-up time—How much effort must be spent to build the execution engine on top of the traditional development flow? Virtual prototypes are still expensive because they are not yet part of the standard flow and often TLM models must be developed. Emulation is well understood and bring-up is very predictable, in the order of days/weeks. FPGA-based prototyping from scratch is still a much bigger effort, often taking several months. Significant acceleration is possible when the software front-end of emulation can be shared.

- **Software debug, hardware debug, and execution control**—How easily can software debuggers be attached for hardware/software analysis and how easily can the execution be controlled? Debugger attachment to software-based techniques is straightforward using standard interfaces, and execution control is excellent as simulation can be controlled easily. However, the lack of speed in RTL simulation makes software debug feasible only for very low-level software. For hardware debug, the different hardware-based engines again vary greatly—hardware debug in emulation is very powerful and comparable to RTL simulation, while in FPGA-based prototyping it is very limited. Hardware insight into software-based techniques is great, but the lack of accuracy in TLMs limits what can be observed. With respect to execution control, software-based execution allows users to efficiently start and stop the design, and users can selectively run only a subset of processors enabling unique multi-core debug capabilities.
- **System connections**—How can the system environment be included? In hardware execution engines, rate adapters enable speed conversions and a large number of connections are available as standard add-ons. RTL simulation is typically too slow to connect to the actual environment. TLM-based virtual prototypes execute fast enough and have virtual I/O to connect to real-world interfaces like USB, Ethernet, and PCI, which has become a standard feature of commercial virtual prototyping environments.

Predictability

Predictability is important and relates to the partitioning aspects and compile-time required. Processor-based emulation is very predictable with a compiler, while FPGA-based prototyping and FPGA-based emulation have challenges due to complex FPGA routing and interconnect issues.

- **Development cost and bring-up time**—From a development cost and bring-up time perspective, the key question is how much effort must be spent to build the execution engine on top of the traditional development flow? Here virtual prototypes are still expensive because they are not yet part of the standard flow and often TLM models must be developed. Emulation is well understood and bring-up is very predictable, in the order of weeks. FPGA-based prototyping from scratch is still a much bigger effort, often taking several months. Significant acceleration is possible when the software front-end of emulation can be shared.
- **Time of availability during a project**—When can I get the development engine after project start? Software virtual prototypes win here as the loosely timed TLM development effort is much lower than RTL development. When models are not easily available, then hybrid execution with a hardware-based engine alleviates re-modeling concerns for legacy IP that does not exist yet as a TLM. This is why hybrid use models with both acceleration and emulation and FPGA-based prototyping have gained popularity in the last couple of years.

Versatility

Versatility relates to the tasks that can be executed with hardware-assisted verification, which includes the choice of target designs (SoCs and CPUs), APIs to RTL and TLM simulation, availability of transactors and rate adaptors to connect to real-world interfaces, support for advanced verification-like coverage support, and native support for verification languages. In addition, hardware-assisted verification includes support for acceleration use models (transaction based, signal based, in-circuit acceleration, and gate-level acceleration), low-power verification, and emulation use models (in-circuit, synthesizable testbenches, dynamic power analysis, and hardware/software co-verification).

The three traditional use models for in-circuit emulation and verification acceleration are:

- Functional verification
- Performance validation
- System emulation, including software development/validation for drivers, OS bring-up, and middleware

There are less conventional, but no-less-proven use models available:

- Regression runs
- Post-silicon debug
- Test pattern preparation
- Failure reproduction and analysis

- Virtual silicon support making the "chip-to be" available to customers for early access

Two use models are very unique to the Cadence Palladium XP Platform due to its processor-based architecture and truly make Palladium's versatility unmatched by alternative FPGA-based emulators:

- **Gate-level acceleration**—Something FPGA-based emulators cannot support due to the explosion in complexity in re-mapping the target technology to FPGA gates
- **Dynamic low-power analysis**—Power has become a crucial issue for most application. Can users run power analysis on the execution engine? How accurate is the power analysis? With accurate switching information at the RTL level, power consumption can be analyzed fairly accurately. Emulation adds the appropriate speed to execute long-enough sequences to understand the impact of software. At the TLM level, annotation of power information allows early power-aware software development, but the results are not nearly as accurate as at the RTL level.

All these care-abouts within the three basic categories of productivity, predictability and use-model versatility then must be compared to the actual cost, the cost of both replicating and operating the execution engine (not counting the bring-up cost and time). Pricing for RTL simulation has been under competitive pressure and is well understood. TLM execution is in a similar price range, but the hardware-based techniques of emulation and FPGA-based prototyping require more significant capital investment.

Different Configurations of Hardware Acceleration and Simulation

As the description above clearly shows, there is no one-size-fits-all engine. Users must carefully select the right engine for the task at hand. Not surprisingly, the engines are used more and more in hybrid combinations to make use of the combined advantages. For example, combinations between virtual prototypes and emulation can speed-up the OS bring-up significantly, and in exchange, enabling designers to get to tests intended for software-driven verification much faster.

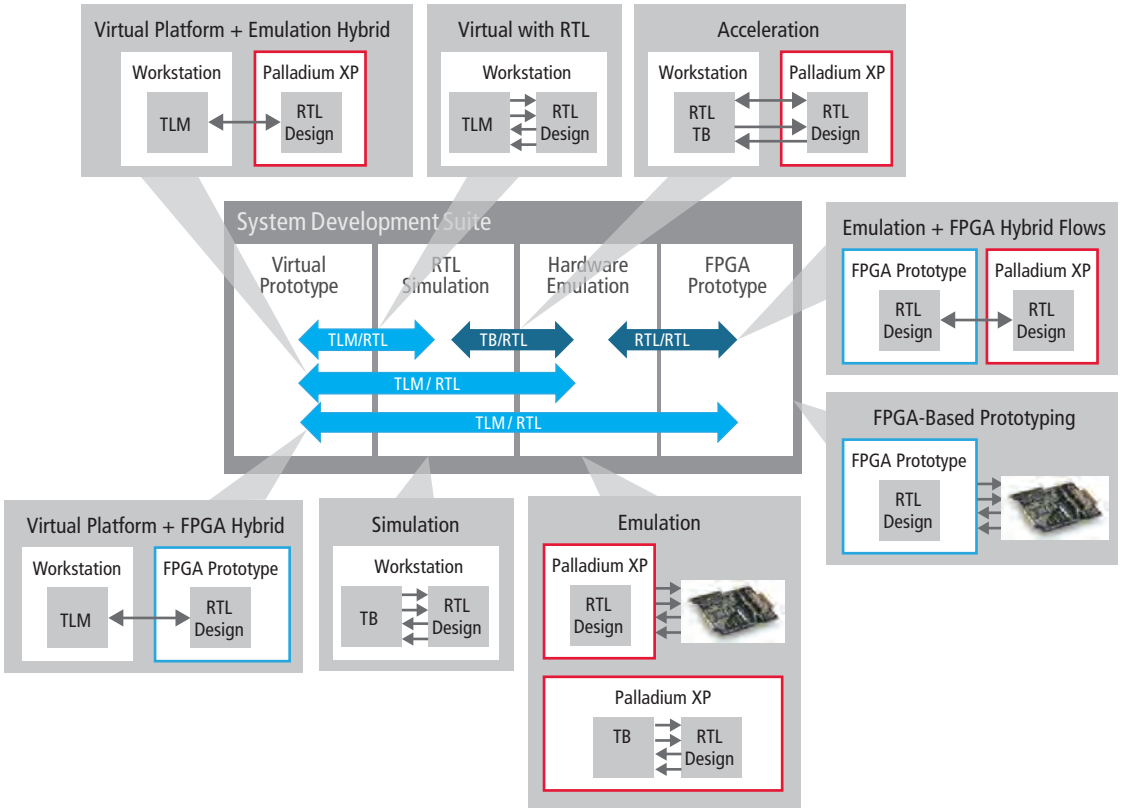

Figure 5: Configurations of execution engines extend to hybrid-use models.

As the different execution engines grow closer together, efficient transfer of designs from engine to engine with efficient verification reuse as well as hybrid execution of engines will gain further importance. Figure 5 shows some of the combinations possible today and a couple of reference examples can be found in [5].

Starting with RTL simulation in Figure 5 and going counter-clockwise, the different configurations can be summarized as follows:

- RTL simulation is primarily used for IP and sub-system verification and extends to full chip and gate-level sign-off simulation. Given the need for accuracy for analysis of interconnect, it is also used as engine performance analysis. It runs in the Hz to KHz range
- Emulation is used for in-circuit emulation in which the full design is run in the emulator and connected to its actual system environment using rate-adaptors called Speed bridges to adjust for speed. Emulation is also used with synthesizable test-benches representing either actual tests for the hardware or representing the peripherals of the system environment mapped into the emulator as embedded test-bench. Emulation runs in the MHz range and is primarily used for hardware verification, early software development, OS and above software bring-up, and system validation on accurate, still maturing, and no-stable RTL.
- FPGA-based prototyping typically runs in the 10s of MHz range, which makes it suitable for software development with the design mapped into the prototype and software debugging using JTAG-attached software debuggers. The higher execution speed allows FPGA-based prototypes to directly interface to the system environment of the design. Given the longer time required to re-map the hardware into FPGAs, it is not used that much for hardware verification, given the emergence of automated front-end flows that can be shared between emulation and FPGA-based prototyping (like with the Cadence Palladium XP Series and Cadence RPP). FPGA-based prototypes find additional use as an adjacency for emulation to run hardware regressions. Its sweet spot is lower cost and faster pre-silicon software development and regressions and subsystem validation on mature and stable hardware with real-world interfaces.
- Emulation and FPGA-based prototypes can be combined, with the less mature portion of the RTL being used in the emulator utilizing the fast bring-up and re-used IP together with the already stable RTL in the FPGA-based prototype enjoying the higher speed range.
- Acceleration is the combination of RTL simulation and emulation, keeping the test-bench in RTL simulation on the host, enabling advanced debug and non-synthesizable test bench usage in combination with a higher speed for the DUT in emulation. Acceleration, as the other hybrids of hardware acceleration with simulation, is enabled using transactors as part of an Accelerated Verification IP (AVIP) portfolio. Acceleration is measured in speed differential to pure RTL simulation and users typically see between 25X and 1000X speed improvement, depending on their design.
- Virtual prototypes using TLMs find their sweet spot in pre-RTL software development and system validation. Together with RTL simulation, enabled by connections through AVIP, they can be used for driver verification with accurate peripherals represented in RTL. Typical speeds depend on the RTL/TLM partitioning, but can reach speeds of 10s of KHz or even faster if most of the design runs as TLM in the virtual prototype.
- Hybrids of virtual prototypes and emulation enable OS and above software bring-up as well as software-driven verification on accurate, still maturing, and non-stable RTL. Primary motivation is speed-up, which is measured as the differential of pure emulation and users typically see up to 60X speed improvement for OS bring-up and up to 10X speed-up for software-driven verification executing on top of an OS that is booted.
- Hybrids of virtual and FPGA-based prototypes are used for lower cost and faster pre-silicon software development and regressions and subsystem validation on mature and stable hardware. In contrast to hybrids with emulation, the primary motivation here is not speed but a better balance of capacity, which often poses a limitation to FPGA-based systems. Moving significant portions, such as a processor sub-system into a virtual prototype at the transaction-level, can save significant capacity and allow the overall design to fit into a hybrid of virtual prototype and FPGA-based prototypes.

Conclusion

As shown above, a specific return on investment of the different verification engines is hard to calculate, but as indicated earlier, the risk of a fatal bug slipping into final silicon has simply become unmanageable as the cost per bug simply bears too much risk. As a result, users require to run more and more verification cycles and make issues

like low-power optimization and software bring-up of OSs like Android, iOS, and Linux a gating requirement for tapeout. Hardware-assisted verification and prototyping has become a mandatory requirement to allow design teams to gain confidence that a chip tapeout can be initiated.

The choice of the right hardware-accelerated engine is driven by its productivity, predictability, and use-model versatility, all impacting the key concern of users how to remove bugs. In a typical bug removal cycle, users first compile the design into the accelerator, then execute it until a bug has been identified, debug the root-cause issues for the bug, fix it and then start from the beginning again. The Palladium XP Platform's best-in-class compile time, execution speed, software debug, hardware debug trace depth, and data transfer speed allow about 2.5X faster bug finding than with the closest competitor, allowing design teams to get to the point at which they are confident enough to tapeout much faster, often shaving of two to four months off development cycles.

Further Information

1. Qualcomm white paper—"Snapdragon S4 Processors: System on Chip Solutions for a New Mobile Age": www.qualcomm.com/media/documents/files/snapdragon-s4-processors-system-on-chip-solutions-for-a-new-mobile-age.pdf
2. Texas Instruments—OMAP 5 Processors: www.ti.com/lsds/ti/omap-applications-processors/omap-5-processors-products.page
3. AnandTech—"NVIDIA Introduces Dual Cortex-A9-Based Tegra 2," by Anand Lal Shimpi, 7 January 2010: www.anandtech.com/show/2911
4. AnandTech—"ST-Ericsson Announces Inclusion in Future Nokia Windows Phones," by Brian Klug, 2 November 2011: www.anandtech.com/show/5038/stericsson-announces-inclusion-in-future-windows-phones
5. "System to Silicon Verification—CDNLive Gives a Reality Check on How Hardware and Software Meet": www.cadence.com/Community/blogs/sd/archive/2013/03/08/system-to-silicon-verification-a-reality-check-on-how-hardware-and-software-meet.aspx?postID=1321185

cadence®

Cadence Design Systems enables global electronic design innovation and plays an essential role in the creation of today's electronics. Customers use Cadence software, hardware, IP, and expertise to design and verify today's mobile, cloud and connectivity applications. www.cadence.com www.cadence.com