The power of virtual prototyping: From SoC design to software development

arm

White Paper

As embedded SoCs continue to become more powerful and complex, beating the market may very well rely on non-traditional approaches to product design and development. Software-based methodologies involving virtual prototypes are helping to prove out designs earlier and enable companies to parallelize hardware and software development.

Virtual prototypes and hardware design

More powerful and complex integrated circuits and System-on-Chip (SoC) designers have a daunting task at both the hardware and software level. SoC architects need a method for early evaluation of hardware components, known as Intellectual Property (IP) blocks, that will have direct impact on the commercial success of the SoC. There are a range of complex questions that need answers:

- Which CPU should be selected to balance performance, power and area?
- What caching hierarchy, associated sizes, and algorithms are appropriate for the target workloads?
- Which memory controller best fits the design criteria?
- Which interrupt controller offers the right latency for a real-time design?
- Which architecture is best suited to deliver a secure system?
- Should the team develop key algorithms in software or map those algorithms to hardware accelerators?

These are typical questions that arise early in the design process and there can be catastrophic impact if answered incorrectly. Architects need the ability to examine various IP configurations to mitigate technical risks, assess design trade-offs, and understand what works best for the given system-level requirements. Design performance is oftentimes impacted by the software running on the device, but how do you predict performance before silicon?

“Getting the design right for performance aspects of the SoC right can make or break a product, so accuracy is key in early stages of the design.”

SoC designers need to model the system. One approach that has been employed is a spreadsheet comprised of estimated performance deltas over previous generation components, but this makes it very difficult to estimate performance of the running system when external factors based on the environment or software interaction aren’t easily predictable. What the SoC designer really needs is a system-level representation of both hardware and software and a means to estimate performance of the interacting components.

Virtual prototypes are a powerful method of testing elements at both the component and system level. Virtual prototypes typically consist of connected component models written in a special purpose language such as SystemC. Using a standard language permits component models of IP from different vendors to be stitched together into a virtual prototype suitable for running real world software workloads.

Getting the design right for performance aspects of the SoC right can make or break a product, so accuracy is key in early stages of the design. EDA tools enable running the actual RTL design in simulation. An alternative approach is to convert the RTL design into a C model and then simulate. There are several advantages to this approach:

1. **Execution performance.** For software performance analysis, only the number of clock cycles are of interest. C models can be more abstract than the RTL itself, without losing the desired level of accuracy. The more abstract the model, the faster it can simulate, resulting in faster turnaround when running benchmarks.

2. **Software models are easy to reconfigure.** Design layout on hardware can take hours to synthesize whereas software models can be reconfigured and rebuilt in a matter of minutes. This is important when a designer is looking for the best system-level result as components in the system such as CPU, interconnect and memory will have multiple configuration options that can impact the desired performance profile.

3. **Better visibility and control.** Traditional RTL simulators target hardware design and oftentimes don’t cater to the architect who is studying effects of software on the system. C models can be easily instrumented to provide additional information not found in traditional RTL simulators. For example, bringing out register and memory in views that are consistent with traditional software debuggers, enabling the setting of software breakpoints to easily halt running software at points of interest and bringing out performance counter data into a software analysis view.

Cycle accurate C models are a great alternative to RTL simulation, particularly when it comes to analyzing effects of software on the system. Easier reconfigurability, better performance and better software analysis capabilities leads to a faster **time to result** when working with C models.

Cycle accurate C models are provided by most IP vendors, or, can be derived using tools such as Verilog-to-C compilers.

"Proving out the hardware/ software interface early on can save costly hardware re-spins and associated schedule delays."

Early software development

Given the complexity of modern embedded software, product teams can no longer afford to wait for the hardware to arrive to begin developing software. There are two distinct advantages to starting the software development early:

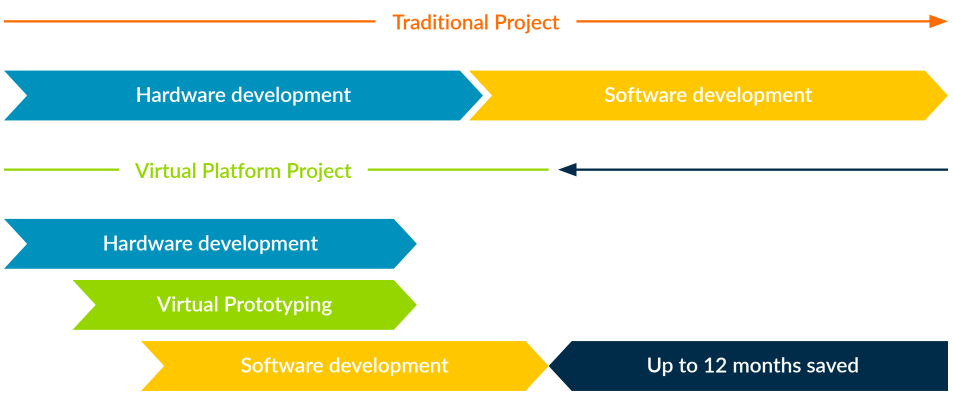
The first is proving out the hardware/software interfaces. The integration of software with the hardware platform oftentimes turns up bugs, not just in the software, but also in the hardware design. Proving out the hardware/software interface early on can save costly hardware re-spins and associated schedule delays. This is where virtual prototypes can help. Fast, functional models of the hardware design have enough detail to emulate the hardware from a software programmer's view, but abstract enough to run at speeds required for large scale software development and debug. Modern functional models can boot Linux in a matter of seconds and step through code at speeds equivalent to a debugger connected to hardware.

Functional models are derived from the hardware specification and can be developed into virtual prototypes iteratively as the hardware specification stabilizes. There's no need to wait for the final design. As portions of the design are stabilized, the modelling can start. This allows software teams to iteratively develop the software well before the design is complete. Inevitably this leads to higher quality software as the hardware and software teams are collaborating during design of each aspect of the product and the parallel development leads to earlier and longer validation cycles.

Virtual prototypes are also a great way to learn how new pieces of IP are configured and work together under software control. Programmer's view models that come from the vendor with software examples enable software engineers to learn the system architecture. Many vendor systems have a lot of code reuse and models are a great tool to help software engineers get familiar with the code and how it interacts with the hardware.

The second advantage to starting the software development early is time to market. Software development for complex embedded systems now exceeds the time and cost of the hardware design and starting the software development process after the hardware is present and stable can add 12 months or more to your product development timeline.

Fast, functional models which are accurate from a software programmer's view are ideal for early development of low-level software, e.g. device drivers and firmware, which are critical to bringing up a new hardware platform. The models can also be used for porting or developing higher levels of the software stack such as middleware and the operating system. It is not uncommon to see an entire Linux or Android software stack being developed on a virtual prototype and then ported to hardware within 1-2 weeks of hardware availability.



Save time with parallel software development

Virtual platforms for software validation

The larger and more complex the software stack, the more difficult it is to develop and test. No longer can large software teams afford to work in isolation for weeks or months and then attempt to integrate and test the finished product. Modern software development methodology such as continuous integration helps identify defects very early in the software development cycle, making it easier to identify where and when the defect was introduced. Finding defects sooner results in faster and cheaper remedies versus trying to unwind hundreds or thousands of software commits over weeks or months.

Continuous integration of software requires robust and scalable test infrastructure as tests must complete overnight to avoid delays the following day. Traditional hardware targets for embedded applications are not well suited for continuous integration flows as they require dedicated infrastructure such as development boards, JTAG connection devices, special purpose racks and cooling and remote reset hardware. Virtual platforms are ideal for continuous integration and validation farms as they are deployable on off-the-shelf server hardware and can utilize the same server infrastructure that already exists at most companies. It is not uncommon to see companies running thousands of parallel test vectors on a server farm overnight, leaving software developers confident that the previous day's commits have not broken the product build. Price reduction for server hardware and the availability of low-cost, scalable capacity in the cloud has made software validation farms the state-of-the-art methodology for validation of large software platforms.

Improving system performance by compilers

Another topic which doesn't get too much attention but is critical for software development is compiler performance. Typically, software engineers are given a fixed hardware system and are asked to optimize system performance by modifying software and utilizing the compiler to generate the best fit for code size and performance. The performance vs. code size trade-off is closely monitored throughout the software development process and there can be dramatic swings between compilers and compiler optimization flags. It's important to choose the right compiler and utilize models to compare compilation results. Virtual prototypes can be used to estimate software performance and monitor for performance regressions in a continuous integration environment.

Comparing strategies

Whilst looking at the benefits of virtual prototypes there are other approaches to hardware/software co-development and early software development. Two of the most popular are hardware emulation and FPGA prototyping.

Hardware emulation

Hardware emulation involves compiling the RTL design onto a hardware emulator, mainly for functional verification of the system. Emulators are orders of magnitude faster than running the RTL in an RTL simulator. They are fast enough to run real software workloads, although not fast enough for large scale software development. Visibility of the design is superb as virtually all signals can be exposed, making debugging of the hardware/software interface possible.

FPGA Prototypes

FPGA prototyping involves synthesizing the RTL design onto Field Programmable Gate Arrays. FPGAs are fast and relatively low in cost when compared to emulation but can be difficult to configure and debug. Oftentimes the SoC design is larger than the largest FPGA, requiring the design to be split across multiple FPGAs, introducing speed and timing challenges. FPGAs run a couple of orders of magnitude faster than emulators, making them far more suitable for software development. Turnaround times for a new layout can take several hours, so FPGA are best suited for relatively stable designs. Debugging of FPGA systems can be a challenge as visibility of internal signals is limited relative to emulation or virtual prototypes.

Virtual prototypes

For an FPGA or emulation solution to be employed, the RTL must be nearly complete to load and run software. Virtual prototypes offer an alternative for software teams to perform many of the software tasks. In fact, virtual prototypes are not reliant on the RTL and can be delivered months before the RTL is made available.

While not as detailed as hardware-based solutions, simulation models provide high performance, flexibility to easily reconfigure designs and excellent software debug and analysis capabilities. Virtual prototypes are also well suited for continuous software integration tasks due to their speed and ease of deployment in compute farms, ensuring better software quality during development.

Generally, testing software for functional correctness does not require cycle accuracy. The most important factor initially is that it works without programming errors. Especially in complex multi-core situations it's best to run the software with a fast, flexible model first to eliminate many of the functional issues.

"The biggest challenge for design teams is to understand all the available options and make the best use them."

Hybrid techniques

Emulation, FPGA prototyping and virtual prototyping all have their advantages and disadvantages and with most projects, a combination of techniques will be employed.

A hybrid **combination of fast, functional virtual prototypes with emulation** delivers the best of both worlds, performance that is faster than the emulator alone and increased visibility and flexibility of debugging software on a virtual model.

Hybrid virtual prototypes and FPGA can also be deployed, where the regularly changing portion of the design would reside in virtual prototype whereas the stable portions of the design would reside in FPGA.

Another type of hybrid solution involves **pure virtual prototypes**, using fast, functional models to run to interesting points in the software stack at very high speeds and then transitioning to slower, more accurate models for performance analysis. This involves running the software on the faster model until a point of interest is reached, then capturing the architectural state, a technique commonly referred to as checkpointing. The checkpoints can be restored into a cycle accurate simulation from the point of interest and run forward. This provides a fast-forward solution when accurate models are too slow and the start of the workload is not important, for example during Linux operating system boot. This can be used to study interaction of software with hardware in a situation such as a performance critical Linux device driver.

Choosing the right development path

The biggest challenge for design teams is to understand all the available options and make the best use them. Here are a few things to consider:

- Models can be run without RTL access during early evaluation phases, while emulation and FPGA techniques need RTL access.
- Models can provide nearly unlimited capacity for large software teams whereas emulation and FPGA prototypes are more difficult to scale due to the incremental cost and physical infrastructure requirements.
- Emulation runs faster than cycle accurate simulation models by orders of magnitude.
- Virtual prototypes run faster than emulation but lack details of the actual CPU implementation.
- Virtual prototypes provide better software debugging capabilities and analysis than hardware-based approaches and are better suited for continuous integration environments.

All of this is not meant to convey that the problem is impossible to solve, but it is a challenge to decide which paths to take for early development and one path is probably not enough for most projects.

An obvious question is “where do I get a virtual prototype?” In most cases the IP vendor will develop models alongside the IP design. This enables them to prove out hardware/software interfaces early on as well as begin OS porting and driver development earlier. These virtual prototypes might then flow downstream to their end customer to act as a learning tool and platform for software development. Alternatively, software service providers may create models of silicon vendor IP or integrate IP from multiple vendors into a virtual prototype for sale to anyone adopting that IP.

Conclusion

Challenges related to early software development in SoC projects are not new, but like hardware complexity, have been steadily increasing. Projects need to utilize a mix of techniques and solutions to complete all the required tasks. Software also plays a more important role in early system design, especially in IP selection and configuration. Software has also become more complex as virtualization and hypervisors are being added to embedded products, and security is just as important as a functioning software stack.

IP providers are putting more effort into providing both cycle accurate models for performance analysis and fast, functional models for development of a full software stack. Enabling technologies such as emulation and FPGA prototyping as well as hybrid use modes are another way IP providers can aid software development.

There is no one-size-fits-all solution, so project teams should keep up-to-date on the latest techniques and utilize all possible solutions to have the best chance of achieving a complete, successful delivery of an SoC with software.

For further reading or additional resources, please use the links below:

- Fast Models
- Cycle Models
- Virtual prototypes: 5 considerations when choosing software development platforms

arm