Hardware-Software Co-Design for Embedded Systems

Software based functionality in today’s embedded products causes delays in project completion if you wait for the hardware prototype to begin software development and debugging. Concurrent design and verification of hardware and software is a trend that reduces time-to-market.

SHWETA DHADIWAL BAID

One of the most time-consuming processes in embedded systems development is hardware-software integration and redesign, which is iterative until you meet the end requirements. Waiting for the hardware prototype to begin software debugging is no more a viable option with demanding time-to-market and the cost involved in redesign. Many tool vendors are now pushing the co-design and co-verification approach to address this issue.

Traditionally, after collecting the system requirements, the design team identifies the requirements that should be met using hardware blocks and the software blocks, and then starts sequential development. “The traditional disaggregated development approach, where hardware is developed first and software later, constrains the innovation that today’s technology makes possible,” says Rahul Arya, director, marketing & technology sales, Cadence Design Systems.

“If the goal is an optimal design in terms of cost, performance, time-to-market and power consumption, the design process must thoroughly consider both hardware and software aspects together early on,” explains Ashish Gupta, founding member and R&D engineer at The Learning Labs. “Hardware/software co-design approach will help you to avoid shocks and surprises at the implementation/validation stage,” he adds. Co-verification ensures that the system meets the initial set of requirements through its hardware and software blocks put together.

Hardware-software partitioning

Embedded systems for real-time applications are implemented as a mix of hardware and software sub-systems interacting in one cohesive environment.

Moorthy Palanisamy, managing director, Oxys Technologies, explains, “The hardware part is mainly tuned for performance, while the software part implements most of the product’s features and provides flexibility to the product for customisation and easy changes. Many a times designers strive to make everything fit in software and off-load only some parts of the design to hardware in order to meet timing constraints.”

“Hardware/software partitioning is all about deciding which scheme gives you the best trade-off with respect to the known requirement metrics,” explains Gupta. Justifying this with an example, he says, “In a simple music player design, you can implement the decoding algorithm for compressed music files using software and a high-speed processor. Else, you can opt for an off-the-shelf ASIC that is capable of MP3 decoding. If the first choice makes you write lengthy code for decompression algorithms, the latter will restrict upgrading the product to support multiple compression formats.”

Indicating the challenges in independent hardware and software design approach, Moorthy shares, “Separate

Fig. 1: Traditional design flow (Courtesy: Ashish Gupta, The Learning Labs)
specifications devised and sent to hardware and software developers individually, cause extensive redesign any time when a change is required. It lacks a unified hardware-software representation, which leads to difficulties in verifying the entire system and hence incompatibilities across hardware/software boundaries. A prior definition of partitions also leads to sub-optimal designs.”

Need for co-design and co-verification

Co-design and co-verification refer to parallel or concurrent development of hardware and software for a systems-on-chip (SoC) or an embedded product. The definition varies from company to company depending on the scope of the information presentation. However, the basics remain the same, i.e., simultaneous design and verification of hardware and software blocks in a system.

“The concept of hardware and software co-design can be extended from an SoC design to entire embedded system design,” says Arya.

“In an embedded SoC system, co-design stands for concurrent design encompassing hardware portion of the design, typically implemented via register-transfer-level (RTL) code and embedded software contents. Today, there is almost no chip with only hardware content. Virtually all chips require some level of software content (drivers, operating systems and applications) to function according to specifications,” shares Moorthy.

Justifying this with an instance, Muthu Raman Iyer, key accounts manager, FTD Infocom, shares, “The common practice today is to incorporate a very simple, non-parametric model of micro-electromechanical system (MEMS) device to verify its mixed-signal behaviour with the ICs. It is tedious for MEMS engineers to manually create models that can be used by their counterparts in IC design. When a MEMS design changes, they have to repeat the model creation effort.”

Mallikarjuna B.S, senior application engineering manager from Mentor Graphics, shares, “Co-design and co-verification reduce the overall design and development cycle of your embedded system. It helps in finding the bugs at an early stage and reduces the number of errors/bugs at the system level, particularly those at the hardware-software interface level. It also helps the designers to do software validation before the hardware prototype is ready, which means accelerating the design closure time. Often, co-verification also helps in making appropriate decisions about hardware-software partitioning to optimise the throughput and efficiency of the system.”

“Semiconductor companies are under tremendous pressure to deliver a fully functional chip to the market within shorter time windows to ensure profitability. If the company misses the profitability window due to a defective chip, or one that has to go through several re-spins, it translates into a huge loss in investment for the company,” says Arya. “The advantages achieved through co-verification largely centre around maximising the productivity of designers and ensuring predictability in the designs, thus ensuring quicker time-to-market.”

Co-design flow for embedded system

In an embedded system design, after the requirements are defined, these are partitioned into hardware and software activities. “When these activities are taken up as independent processes, they yield locally optimised designs which give rise to surprises and mismatch at the integration phase,” cites Gupta. “Also, since the activities are distributed to separate and non-overlapping pure hardware and pure software teams, debugging during the integration stage becomes a nightmare with people being clueless about where the system is breaking down,” he adds.

Hardware-software co-design helps
in synchronous development of hardware and software. Though partitioning is required to define the functionality, the thought of hardware-software interface begins here itself.

Gupta shares, “The key to a good hardware-software co-design is the system partitioning scheme chosen, or rather the partitioning scheme that emerges out as a result of iterative, rapid prototyping and evaluation of various options. The next important thing to do is to rapidly prototype the design and evaluate the various schemes in terms of the requirement metrics chosen. And this needs to be an iterative process and may require multiple cycles of technology survey and prototyping across hardware and software domains.”

Again, at every stage, it is important to keep the hardware and software teams synced with each other and ensure information flow and regular updates about changes during the design phase.

**Use of simulators**

Simulation is a process of verifying the abstract model of the design (described in hardware description languages like VHDL, Verilog and SystemC) on a computer using a software tool called ‘simulator.’ In the context of embedded systems and co-verification, one of the simulation-based approaches that the designers can take is “to have both the hardware and processor models written at RTL and have the application software written in ‘C’ or similar language and run the entire system simulation using a hardware simulation environment,” shares Mallikarjuna. However, this approach could be cumbersome because it takes lots of simulation cycles and debugging the system could be cumbersome. He suggests that a better approach is to use an instruction set simulator (ISS) for the processor part together with a hardware simulator for the hardware part to co-verify the system-level environment using tools like Questa Codelink.”

Some of the tools like Cadence’s Incisive Enterprise Simulator (IES) allow engineers to co-verify hardware and software in the system context, performing both high-speed hardware verification and hardware-software co-verification, thereby simulating system-level behaviour to reduce the risk.

There is no guarantee that your verified RTL will function as desired after tape-out once the embedded software is loaded onto the hardware. Neither can you find an easy solution when you hit a bug after you download the algorithm onto the actual hardware.

An emulator allows software debugging at the RTL stage, which is the earliest stage at which co-verification can happen. Fast emulation has become a critically important verification component for large and complex SoCs. Emulation is today adopted not only to test the hardware aspects of an SoC design but also to verify integration of the hardware and embedded software.

“Logic simulation is excellent in debugging RTL code but it is not up
to the task due to its intrinsic slow execution time. Electronic-system-level (ESL) simulation is of great help in starting embedded software validation before silicon but it does not offer the hardware design accuracy to trace hardware bugs,” says Moorthy. Neither can you test integration of the hardware with software. This is where hardware-assisted verification comes into play.

“Platforms like Zebu map hardware part of the design into dedicated and reprogrammable hardware resources and store the software portion into local memories. It acts as an early prototype of the complete SoC design and offers a powerful debugging means into the hardware and software parts to help hardware and software teams to quickly zero-in and pinpoint design bugs,” shares Moorthy.

Such tools can help you start RTL verification at the block level and move on to the system level as the entire design solidifies. These can also co-verify interaction of hardware and software once the full RTL design is complete.

**Transaction-based verification**

SoC design teams have started adopting high-level design languages like System C and Verilog and transaction-level modeling for test benches to speed up functional verification. It is essential that the test bench and hardware emulator communicate in real time with maximum efficiency. Transaction-based modeling helps in achieving this.

According to Moorthy, there are three specific challenges to the creation of a successful transaction-based emulation environment: transactor creation, test environment creation (or stimulus generation) and performance. “Creating a transactor, especially for a complex protocol, can be a difficult task. Some tools provide large catalogues of transactors including most popular protocols (PCIe, USB, Ethernet, AXI and Video),” he adds. You can also create custom transactors using a SystemVerilog behavioural compiler such as ZEMI3.

Transactors connect the emulated design to the test environment and the test environment generates stimulus for the design. ZeBu transactors are easily integrated with hardware verification and software development environments in a variety of ways, making it easy to reuse your existing test environment or quickly create a new one. For hardware verification environments, transactors can be integrated with existing high-level verification language (HVL) and C/C++ test benches via the SystemVerilog direct programming interface (SV DPI) or SystemVerilog classes. For software development environments, transactors can be integrated with C, C++ and SystemC ESL environments either via DPI, API or TLM-2.0 transactor adaptor.

Another tool from Mentor Graphics, TestBench Xpress (TBX), supports SV DPI to provide building blocks that can create transaction-based communication link between HVL and HDL. The basic transaction constructs in TBX are exported and imported constructs/tasks from SV DPI. TBX automatically generates a direct communication interface between the C, C++ or System C environment on a host and the SoC device-under-test (DUT) in the Veloce emulator.

ESL tools like QEMU can also be used to provide a virtual version of the ICE environment, providing real-world stimulus to the design. Cadence Incisive Palladium series leverages advanced RTL and ESL verification automation features, such as assertion-based acceleration and transaction-based acceleration. Arya cites, “In such tools, you can provide real-world stimuli by external equipment.”

**Parametric hardware blocks for modeling**

Going back to the MEMS example, one approach for co-design is to do finite element modeling. Raman explains, “You can extract the so-called reduced-order models that can extract accurate MEMS behaviour with finite element modeling. These models can be implemented in simulation languages that are used in IC simulators, such as Verilog-A.”

The limitation of this approach is that the design is not parametric, so in a way it doesn’t support 100 per cent co-design. Raman says, “Another approach is to include a library of parametric MEMS building blocks, like the ones available in Coventor MEMS+. Most types of MEMS can be modelled by combining these blocks, which can be directly simulated using IC simulators such as Cadence Spectre or MATLAB Simulink. MEMS+ produces models that can be automatically imported to the Cadence Virtuoso and MATLAB Simulink environments accurately.”

**To conclude**

The ultimate goal of any designer is to meet the initial requirement within the given time-frame. Adoption of development methodologies like concurrent development helps in reaching this goal. Hardware-software co-design and co-verification is a great way to augment the independent development process, thereby increasing the design productivity. The methodology of hardware modeling has advanced to a stage where we can sense the developed software on the models of hardware, making co-simulation part of a new paradigm. The result is reduced debug cycles and accelerated time-to-market.

The author is a senior technology journalist at EFY