Developing a SoC is a risky business in terms of getting it right considering the technical complexity involved, managing the mixture of hardware and software design disciplines, and finding an optimal trade-off between design performance and power. One way to reduce these risks is to use a design and verification flow that is scalable enough to handle the complexity and is flexible enough to explore architectural alternatives early in the design cycle before implementation starts.

Mentor’s System Design and Verification flow encapsulates a range of design and verification disciplines such as: embedded software; SystemC platform design and validation tools; and HDL simulation tools that can be used in various combinations using common interfaces. The flow originally came out of pilot work for the TSMC ESL reference flow and has since matured into a methodology that allows Mentor’s Sourcery™ CodeBench, the Vista™ ESL platform, and the Questa® platform to be used together.

One way to view the flow is to consider it as a three step process, with each step meeting its own set of objectives, but where each step can be used sequentially, concurrently, and iteratively until all design and verification objectives are met.

- Step 1 – ESL Platform Design and Verification
- Step 2 – RTL Design and Verification Using ESL Reference Model
- Step 3 – Hybrid Verification of RTL in ESL Platform Environment

ESL PLATFORM DESIGN AND VERIFICATION

One of the immutable laws of design is that it is easier and less costly to make corrections to a project early on in its life than later. For instance, the cost of changing a sentence in a specification at the system modeling stage is a few minutes of engineering time; whereas making the same change post-silicon could involve a respin or a product recall — with all the associated complexity, economic costs, and business impacts that are massive by comparison.

The ESL Platform Design and Verification step reduces the risk of having to make painful design changes during the later implementation phase, due to misinterpretation of the design requirements or flawed architectural decisions. It increases predictability during the downstream design phase and increases productivity during the RTL design and verification phase.

The Mentor System Design and Verification flow uses Vista for building the ESL Design and Verification platform. This platform is used for developing and verifying that the system-level requirements of a SoC design are met early in the development cycle, so that project risk can be brought under control.

Vista allows SystemC models to be developed into a library of transaction level models (TLMs). Raw C/C++ algorithmic code, or an abstract SystemC model, can be imported and wrapped in a layer of TLM 2.0 interfaces that allow it to be integrated with other models. The TLMs support both loosely timed (LT) modeling, which is used for fast untimed functional simulation, and approximately timed (AT) modeling, which is used for the analysis of system performance and power consumption. The values for the timing and power parameters used in AT modeling...
are entered and updated as “policies” through the Vista GUI, and the AT portion of the TLM code is produced automatically. Similarly, the key attributes of the TLM interface are captured, and the TLM 2.0 interface code is automatically created. Once produced, the individual models become library components that can be assembled into a system-level TLM platform using Vista’s block diagram editor that automatically generates the entire ESL representation of the SoC design.

Vista comes with a library of pre-packaged system-level components that includes instruction set simulation (ISS) models of leading edge processors capable of running the same software that would run on the final SoC device. The level of abstraction of the TLM platform allows it to boot an operating system, such as embedded Linux, in a few seconds in LT mode. This means that it is now practical to develop embedded software applications using the platform as a fast virtual model of the final device. This allows the hardware-software interfaces of the system to be proven early in the development cycle, and for the impact of specification updates to be rapidly assessed, removing considerable project risk, uncertainty, and cost. Using the Sourcery CodeBench IDE connected to the Vista ISS TLM models, the user can debug software in conjunction with the TLM platform code within Vista’s comprehensive SystemC/TLM2.0 debug environment.

Vista also provides performance and power analysis tools which allow the system to be validated against its power and performance requirements. A system-level scenario can be simulated in AT mode within Vista, and the transactional latencies and throughputs on different interfaces can be viewed and analyzed. Power profiles that provide peak and average power for each TLM and the entire system can be viewed on Vista power plots. This enables performance bottle-necks to be identified and corrected and for power-performance trade-offs to be made at the architectural level. This is one aspect of system design which has to be done early, since it is very difficult or costly to make corrections in the down-stream implementation if a gross architectural error is made at the system level. It also sets the functional behavior, timing, and power constraints that drive the subsequent RTL design and verification steps.

**Figure 2. ESL TLM Platform**

**Figure 3. Power consumption in AT simulation mode.**
Once the system-level design process has reached a suitable level of maturity, the RTL part of the SoC design can start using, as a reference, the design intent captured in the system-level model.

**RTL DESIGN AND VERIFICATION USING AN ESL REFERENCE MODEL**

Most of the design content in an SoC is design IP that has already seen use in another design. This should mean that there is less risk associated with using it and that there is associated verification IP immediately available. However, there are times when a block needs to be designed from scratch or modified to meet new requirements, in which case the Mentor flow makes it straightforward to use a SystemC TLM as a reference model of the block in a UVM testbench. This is done using the UVM Connect package.

UVM Connect is one of the enabling technologies developed for the Mentor System Design and Verification flow, and it enables transactions to be passed between a SystemC TLM model and a SystemVerilog/UVM TLM model. The package is open source and uses native C++ and SystemVerilog code to allow the user to connect TLM sockets across the language barrier. Once the connections are made, transactions can be sent in either direction between models implemented in the different languages, using the normal TLM API. Most SystemC TLM models are implemented with TLM 2.0 socket interfaces that use the generic payload transaction defined in the SystemC IEEE 1666™-2011 standard. Using UVM Connect, these can be directly connected to UVM TLM 2.0 sockets. UVM Connect also supports other types of transactions and the various TLM 1.0 ports, exports, and analysis ports and exports.

The verification of an RTL block-level design has to be comprehensive to ensure that it has the right level of quality before it is integrated with the rest of the SoC. Often, the block will be developed as a design IP with unknown future applications, requiring the verification to be even more stringent. Using SystemVerilog and the UVM, a testbench can usually be architected and constructed rapidly from existing verification IP and some custom components. Having a SystemC functional model of the block available can help with testbench development by allowing the testbench and the test stimulus to be developed before the RTL is available and by providing a reference model against which the RTL behavior can be compared when the RTL becomes available.

**JUMP-STARTING UVM TESTBENCH DEVELOPMENT**

A UVM testbench is interface centric; that is to say, at its core is a set of verification components, called agents, whose purpose is to drive the various signal-level interfaces on the RTL DUT. UVM stimulus generation is abstracted and layered on top of the agents. At the lowest layer, transactions are sent to drivers embedded within the agents to be converted to signal-level activity on the RTL DUT interfaces. A TLM platform SystemC model can be substituted for the RTL DUT, either by introducing proxy agents that take the UVM stimulus transactions and send them to the SystemC model or, as is the case with the Questa VIP library, using the agent in TLM mode.

![Figure 4. A UVM testbench using a SystemC reference model as the DUT.](image-url)
The SystemC model of the DUT will have an equivalent set of interfaces to the RTL signal-level interfaces, except that they will be implemented using TLM. In most cases, these will be TLM 2.0 generic payload interfaces which can be connected to UVM TLM 2.0 initiator or target sockets, using UVM Connect. The code example shown in Figure 5 illustrates how a proxy agent is put in place to convert UVM APB sequence_items into TLM 2.0 generic payload transactions that are then sent to the SystemC model via an initiator socket.

The analysis side of the testbench, such as scoreboards and functional coverage monitors, can be constructed on top of the proxy or TLM agents, since the interfaces are transactional. This means that development of the testbench and test cases can get underway before the RTL is available. When the RTL is available, the testbench can be reconfigured so that the RTL is driven directly without having to change the stimulus or the analysis components.

The TLM abstractions used in the SystemC model will mean that it simulates faster than the RTL, this means that even when the RTL is available, it may be useful to use the SystemC configuration of the testbench to develop and debug stimulus before applying it to the RTL.

**USING A SYSTEMC**

**TLM AS A REFERENCE MODEL**

One goal of the Mentor System Design and Verification flow is to ensure that all components have a consistent definition.
by comparing them against a reference. The SystemC model used and validated in the TLM platform design is reused as a reference model in the UVM RTL testbench to achieve this consistency. The TLM model is used as a sub-component within the UVM scoreboard for the RTL DUT, making use of the UVM Connect package to send input transactions to the reference model and receive response transactions back from it to compare against the behavior of the RTL DUT. This principle is illustrated in Figure 6.

Using the SystemC reference model will also ensure consistency with the TLM platform, so that software being developed on that platform should run unchanged with the end product of the RTL development. Having a ready-made reference model makes a dramatic difference to the development effort involved in putting together a scoreboard for even a moderately complex RTL DUT.

There will be some minor differences between the RTL and the SystemC TLM. These will come about because of abstractions and timing simplifications, but designing the scoreboard as a transactional component allows these differences to be easily reconciled.

HYBRID VERIFICATION OF RTL IN AN ESL PLATFORM ENVIRONMENT

Although the TLM platform design and the RTL verification process have both used the same SystemC TLM as a reference model to ensure behavioral and interface consistency, there is still a need to close the loop and to ensure that there are no system-level interactions that may have been overlooked in the definition process. To mitigate this risk, the Mentor System Design and Verification flow allows RTL blocks or sub-systems to be verified within the context of the ESL TLM platform.

Again, TLM interfaces are used to make the connections between the RTL and the TLM platform using UVM Connect. The hardware design is wrapped with a thin adaptation layer which converts the transactions from the TLM platform into signal-level activity and vice versa.

Software based scenarios can be run on the TLM platform so that the hardware-software interface can be double checked at the logical level. This type of environment allows any abstraction or interpretation assumptions about timing
behavior to be identified and tracked down. It also allows system-level assumptions about the behavior of the RTL to be validated.

If a problem is identified, it can be quickly analyzed at either the platform-level by reconfiguring the TLM platform to include the TLM representation of the RTL, or at the RTL level using the UVM environment. Once the issue is corrected, the platform-level configuration can be reconfigured.

CONCLUSION
The Mentor System Design and Verification flow reduces project risk, increases predictability, and improves verification productivity by allowing important system-level behavior, performance, and power to be validated early; allowing software to be developed in parallel with the hardware design; and enabling a consistent flow between SystemC and SystemVerilog/UVM using TLM 2.0 and UVM Connect as the enabling technology. Using software-driven verification of RTL at the block and subsystem levels allows early development and validation of drivers with the RTL. Although described here as a three step process, these steps are frequently overlapping and allow for an iterative process among the three main steps to resolve platform-level issues.

The Mentor System Design and Verification flow is available now, ready for deployment to Mentor customers.

REFERENCES:
2. Introducing UVM Connect, A Erikson – Verification Horizons Volume 8, Issue 1

Figure 7. Validation of RTL within a TLM platform.