

TLM-Driven Design and Verification – Time For a Methodology Shift

By Steve Brown, Cadence Design Systems, Inc.

Transaction level modeling (TLM) is gaining favor over register-transfer level (RTL) for design components because of its many advantages—including faster design and verification times, easier intellectual property (IP) reuse, and fewer bugs. The benefits are clear, but the transition will take time. Designers must be able to combine new TLM IP with legacy RTL IP in design and verification environments. Cadence offers methodology guidelines, high-level synthesis, TLM-aware verification and debugging, and services to ease the transition from RTL to TLM.

Contents

Introduction.....	1
Key Challenges That Demand a Change from RTL	2
The Solution That’s Needed	3
The Cadence TLM Offering	7
Conclusion.....	9

Introduction

While today’s RTL design and verification flows are a step up from the gate-level flows of two decades ago, RTL flows are straining to meet the demands of most product teams. When designs are sourced and verified at the register transfer level, IP reuse is difficult, functional verification is lengthy and cumbersome, and architectural decisions cannot be confirmed prior to RTL verification. With increasing pressure on today’s SoC and ASIC design teams to deliver more aggressive designs in less time, and the need to get designs right on the first pass, many companies are looking to move to the next level of abstraction beyond RTL to get a much-needed boost in design productivity.

That next level of abstraction is based on transaction level modeling (TLM). By creating TLM IP as their golden source, design teams can ease IP creation and reuse, spend less time and effort in functional verification, and introduce fewer bugs. Design iterations are reduced because TLM verification is much faster than RTL verification, and architectural choices can be verified well before RTL verification. Further, transaction-level models can be used for hardware/software co-verification, and can be part of a virtual platform for early software development. The net result of all these advantages will be much higher designer productivity.

Transaction-level modeling uses function calls, rather than signals or wires, for inter-module communications. It lets users analyze transactions such as reads or writes rather than worrying about the underlying logic implementation and timing. SystemC provides the best language standard for developing reusable, interoperable TLM IP. Moreover, because it’s based on C++, SystemC allows full reuse of arithmetic C language functions. The Open SystemC Initiative (OSCI) has defined several abstraction levels for TLM models, including programmer’s view (untimed), loosely timed, and approximately timed.

Cadence Design Systems now offers a comprehensive SystemC TLM-driven IP design and verification solution, including methodology guidelines, high-level synthesis, TLM-aware verification, and adoption services that help users transition to a TLM-driven design and verification flow.

This paper describes the key challenges that demand a change from RTL as the entry point for design and verification IP. It outlines the TLM-driven solution that's needed and cites its advantages. Finally, the paper provides some information about the Cadence TLM solution.

Key Challenges That Demand a Change from RTL

At the register-transfer level, the structure of finite state machines is fully described. This means that one needs to commit to micro-architectural details when writing RTL, such as the memory structures, pipelines, control states, or ALUs used in the resulting implementation. This requirement results in a longer and less reusable design and verification flow.

While TLM is sometimes used in today's flows, current RTL-based flows require the manual entry of design intent twice—once at the systems level, and again at RTL (Figure 1). This is a cumbersome and error-prone process. Architecture is not verified until RTL is generated, and retargeting IP comes with a high cost. A true TLM-driven design and verification flow (shown below at right) will offer an automated path from a single expression of design intent.

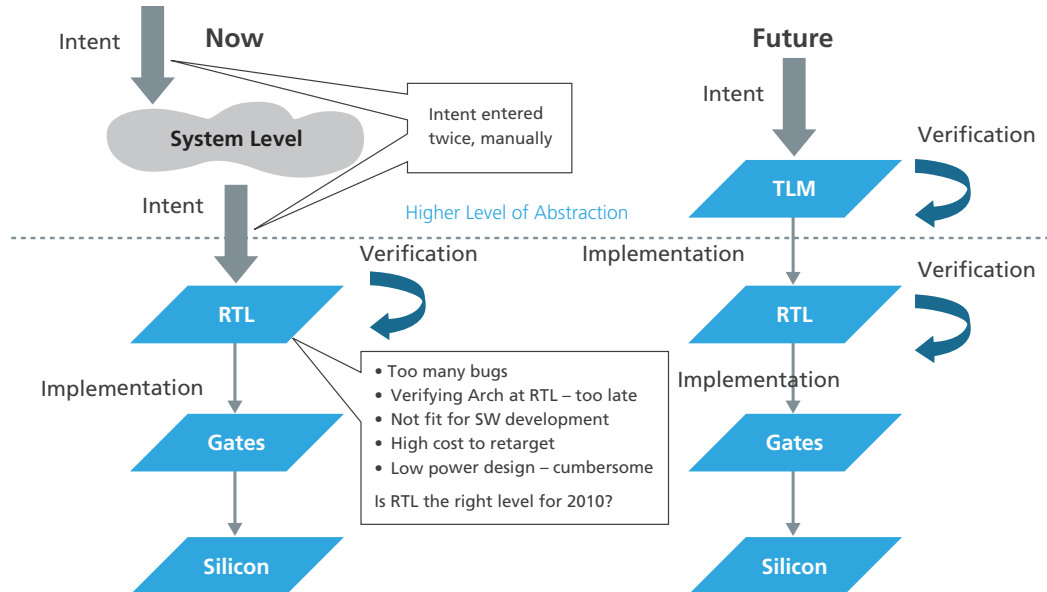


Fig. 1—An RTL-based flow requires manual design intent entry twice. A TLM-based flow, in contrast, provides an automated path to implementation.

Specific challenges with RTL-driven flows include the following points.

Detecting and fixing architectural problems starting from RTL is too lengthy and expensive

One of the big problems with an RTL-driven design methodology is that design teams don't know if an architecture can be implemented until they create RTL and run verification to determine if goals were met. Because RTL is inherently a direct representation of the architecture, most RTL designers are forced to explore functional correctness, architecture, and design goals simultaneously. The result is a lengthy cycle that begins with making an architectural decision and ends with verifying the functionality. Often, design and verification teams discover functional bugs that require fixes to the architecture, restarting the entire cycle with each bug.

Reusing design IP at RTL restricts architectural flexibility

In today's SoCs, as many as 90 percent of the IP blocks may be reused from previous projects. But reuse is difficult when the golden source for the IP is at the micro-architectural level. Retargeting the micro-architecture of RTL IP is high effort, error-prone work. The target system application may differ significantly, implying new design goals for the SoC that can only be met by re-architecting, not simple reuse. For example, RTL designers may need to re-partition the design into new RTL blocks, change the number of pipeline stages, or create a new memory architecture, because all these micro-architectural details have been fixed and pre-determined in the old IP.

RTL functional verification time is growing faster than the available throughput of today's technology

Functional verification has become the primary bottleneck in many SoC projects. Much of the verification effort invested at the system level is lost when RTL functional verification begins. While techniques such as verification planning and metric-driven verification allow design teams to handle most of today's verification challenges, time constraints and increasing gate counts are making verification much more difficult. The time required for RTL functional verification can grow exponentially with the size of designs because of the need to verify corner cases resulting from interacting modes and the many hardware and software configurations this IP needs to be tested with.

RTL logic is cycle accurate and involves far more lines of code than TLM logic. When RTL models are simulated, the simulator is examining every event or clock cycle, even if nothing significant is happening at the protocol level. The simulator thus burns a lot of computer cycles on micro-architectural detail that could be postponed until after the architecture is committed. TLM simulation is done at a higher level of abstraction, is performed earlier, and provides higher performance.

The Solution That's Needed

A TLM-driven design and verification flow makes it possible to describe IP at the functional level, and then verify the functional behavior of transactions in a fast simulation. Key advantages of a TLM flow include the following:

- Ability to create designs faster
- Reduced lines of code in golden source
- Fewer bugs
- Easier to express design intent, and it's only needed once
- Faster simulation and debugging
- Earlier power estimations
- Support for hardware/software co-verification
- Models can be incorporated into virtual platform
- Architectural verification before RTL generation
- Reuse of TLM verification IP in RTL verification
- IP reuse without micro-architectural redesign
- ECOs produce RTL with minimal changes

Used in conjunction with high-level synthesis (HLS), a TLM-based flow moves the level of abstraction up, representing the first major shift in abstraction since designers moved to RTL some 15 years ago. Experience with previous shifts in the abstraction level suggests that an order-of-magnitude improvement in designer productivity is possible (Figure 2).

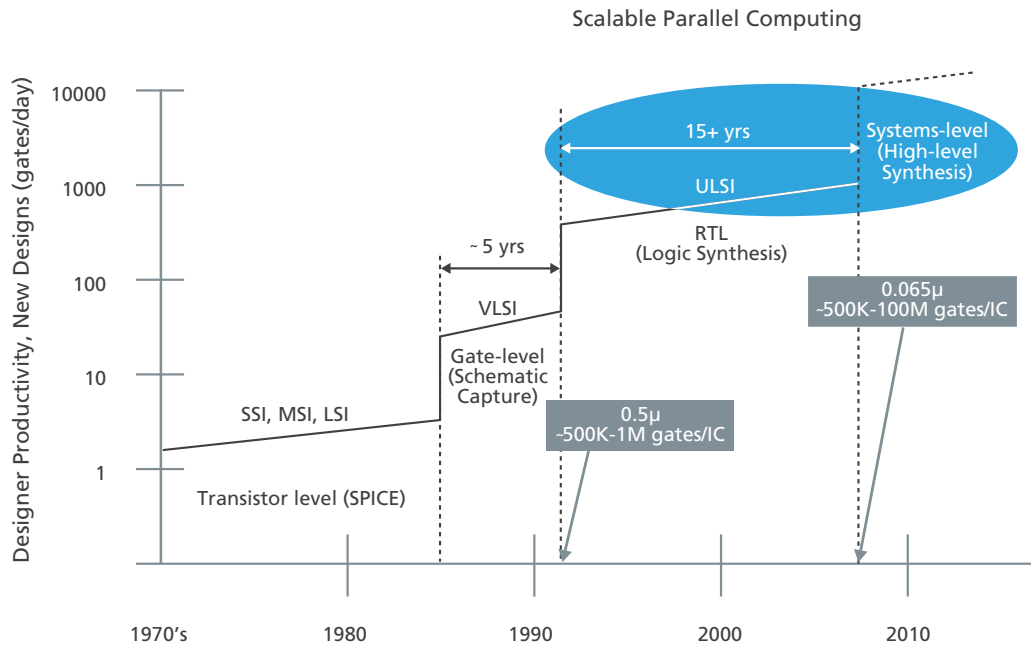


Figure 2—A TLM-based flow with HLS could bring about an order of magnitude improvement in designer productivity.

Specific attributes and advantages of a TLM-driven design and verification flow are described in the following sections.

Creating tlm as golden source - faster IP creation and design IP reuse

Unlike RTL, TLM does not describe micro-architectural details of the resulting implementation. This absence of detail greatly enhances the reusability of the TLM designs across multiple projects with different requirements, since the same TLM IP can be retargeted to RTL code with different micro-architectures. Also, because of the higher level of abstraction, it is much easier to correctly create the functionality. TLM models have far fewer lines of code than corresponding RTL models, enabling both higher coding productivity and higher quality in the resulting design.

Developing and maintaining TLM as golden source for an IP block requires synthesis and verification solutions that produce the necessary quality of results and verify the correctness, with no need to edit the RTL or the gate level design. This will enable the design team to make all decisions within the TLM environment, and to reuse the TLM source for other designs with entirely new system constraints.

With the Cadence® C-to-Silicon Compiler, for example, users can specify area, timing and power constraints in a side file separately from the golden source. It is thus possible to change micro-architectural characteristics such as the number of pipeline stages and still retain the reusability of the original TLM model. In an RTL IP block, in contrast, the number of pipeline stages would be fixed.

SystemC is the best standard for describing transaction level designs with a link to implementation, and it offers the best opportunities for reusability. It models hardware concurrency, and describes both timed and untimed behavior for processes, pins, threads, and control logic. The TLM 1.0 and 2.0 standards provide the ability to create interoperable IP models. Ultimately, there will be a need for qualified libraries of synthesizable TLM IP together with a standard (or de-facto) subset of synthesizable TLM.

Functional verification of TLM IP offsets the verification throughput explosion

Verification of TLM IP has many advantages over RTL verification. First, simulation runs faster—perhaps an order of magnitude over RTL simulation. This allows many more functional use cases to be verified. Also, debugging at the TLM level of abstraction is easier and faster than RTL debugging.

By coding at a higher level, TLM IP requires fewer lines of code, and thus has fewer bugs. Functional bugs are detected and resolved earlier in the design cycle. The total verification effort can thus be greatly reduced.

It is easier to identify and understand bugs at the abstract TLM level, and it is easier to fix bugs at this level, because there are fewer details to deal with. A TLM flow makes it possible to verify each concern at the level of abstraction that’s most appropriate—for example, TLM for functionality, and signal-level verification for interfaces.

The TLM verification flow begins with algorithmic functional verification, allows functional validation with software, and moves on to TLM functional verification (Figure 3 on the following page). Following compilation with the C-to-Silicon Compiler, users can move on to micro-architectural RTL verification and RTL-to-gates equivalence checking. In addition to supporting untimed modeling, which simulates very fast, TLM also allows users to make refinements that progressively include micro-architectural details and refine timing accuracy.

Hardware/software co-verification and early software development

TLM models are at a sufficiently high level of abstraction, and execute quickly enough, to make hardware/software co-simulation practical. Designers can co-simulate embedded software with TLM hardware models to check for hardware/software dependencies, and to begin early debugging of hardware-dependent software. They can potentially apply such technologies as randomized stimulus and coverage to hardware/software interactions.

Virtual platforms used for early software development and debugging may include subsystems comprised of SystemC TLM models. Because of their fast execution, models developed for hardware design creation can also be used to accelerate software design.

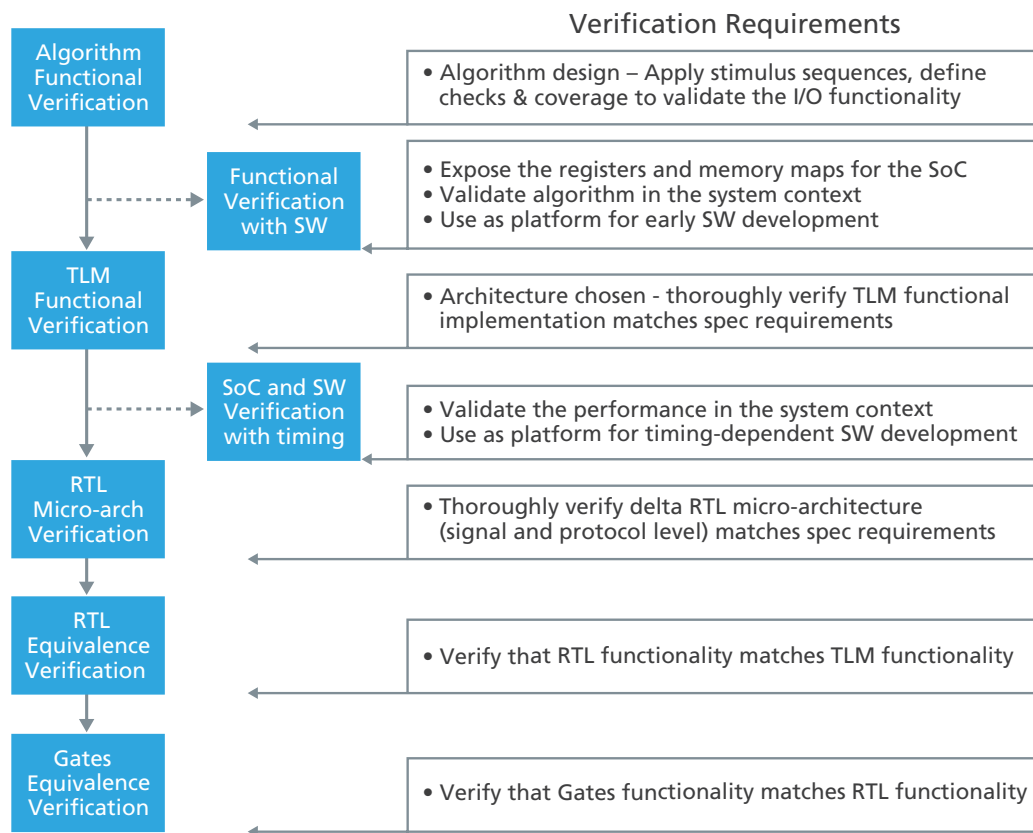


Figure 3—TLM verification flow and requirements

Supporting combined TLM and RTL verification

At the SoC level, mixed TLM and RTL functional verification is required because of the large amount of legacy RTL IP that will be reused, and because it will still be necessary to perform detailed RTL functional verification for portions of the design. Some verification tasks will still be done only at RTL. This includes micro-architectural structural verification for such attributes as memory access sequences or state transition coverage.

Mixed TLM/RTL verification is made easier by the fact that most verification artifacts including verification plans (vPlans), Open Verification Methodology (OVM) Verification Components, testbenches, sequences, tests, checks, and coverage are reusable across abstraction levels. Functional verification planning and management across both TLM and RTL let teams track and control the verification at each level in a mixed level design, and combine the results as needed, ensuring overall quality.

OVM for SystemVerilog has been extended to support multiple languages including e and SystemC. The OVM library also supports TLM. Currently, the OVM methodology description is being extended to show how TLM and RTL models can be combined in a comprehensive regression solution. This will facilitate the creation of verification IP (VIP) that works in a multi-language, mixed TLM/RTL verification environment.

A multi-level functional verification testbench is based on transactions, and when it is connected to RTL-based IP, busses, or interfaces, a transactor is needed to translate between the transaction-level domain and the pin-accurate RTL domain. Similarly, transactors are needed to connect TLM IP blocks to busses or interfaces on RTL IP blocks. A TLM-based methodology must consider how these transactors operate in order to gain the maximum benefits of mixed TLM/RTL verification. Some transactors may be commercially available, while others are proprietary, created by the project team and managed as verification library components.

Many projects are implementing TLM for new IP only, gradually building up a library of TLM IP, whereas other teams are embracing a TLM methodology for all major IP blocks in their mission-critical project. Eventually, SoCs will emerge in which all IP golden source is at the TLM level. In these cases benefits of quality, productivity, and ease of debugging will be more pronounced than with mixed TLM/RTL projects. SoC TLM functional verification, including SoC level architectural analysis and optimization, will be achievable.

Reusing vip from TLM through RTL verification

VIP reuse is now mainstream because the time to create high quality verification environments often exceeds the time to create the design IP itself. The wide employment of standard protocols has driven the development of a fast-growing commercial VIP market. Today, most of this VIP is at the register-transfer level. TLM-enabled VIP will also be needed, but it must be reusable for mixed TLM/RTL functional verification.

RTL functional verification is dominated by advanced testbenches that use constrained-random stimulus generation. TLM-enabled VIP should be operable in testbenches for TLM, mixed TLM/RTL, and RTL functional verification. That same VIP needs to allow the application of metric-driven verification as customers apply coverage metrics at all levels of verification abstraction. Finally, supporting embedded software and directed tests is a necessity for verification teams who work closely with the architects and software engineering teams.

Incremental design refinement from algorithm to micro-architecture

The TLM IP design and verification flow has several distinct phases, including algorithm verification, architecture verification, and micro-architecture verification (Figure 4). The first step, algorithm design and verification, may involve C++ or a product such as Matlab or Simulink. Users define a vPlan for key algorithm features, verify I/O functionality, and apply stimulus sequences for key use cases.

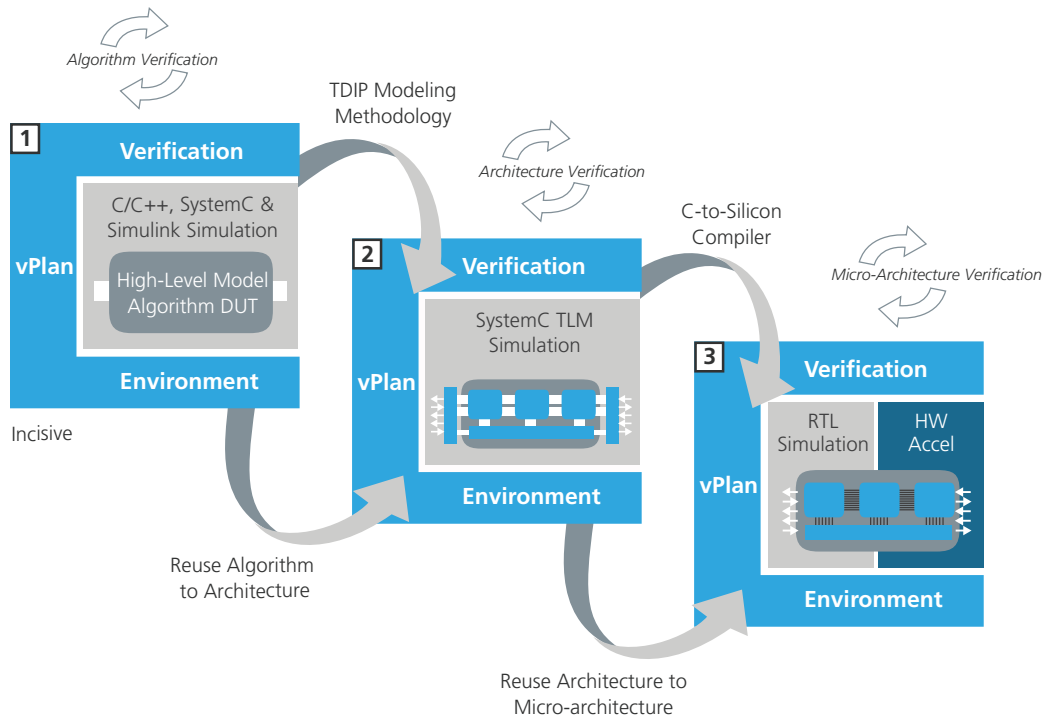


Figure 4—A TLM-based IP design and verification flow encompasses algorithmic, architectural, and micro-architectural verification.

In step two (architecture verification), designers use a TLM-driven IP modeling (TDIP) methodology to define the architecture and the interface protocols. They reuse the algorithm vPlan and apply additional stimuli, checks, assertions, and coverage. They also define a vPlan for the key architecture and interface protocol features. In step three (micro-architecture verification), following synthesis with the C-to-Silicon Compiler, designers reuse the algorithm and architecture vPlans and expand to micro-architectural detail in the stimuli, checks, assertions and coverage.

The Cadence TLM Offering

The Cadence TLM-driven IP design and verification solution includes methodology guidelines, the C-to-Silicon Compiler, the Cadence® Incisive® functional verification platform, and TLM-driven IP design and verification adoption services.

Unified TLM-driven IP design, verification, and reuse methodology and coding guidelines

Cadence will offer methodology guidelines for TLM-driven IP design and verification to help teams launch and complete their initial TLM projects in the shortest time with the highest productivity, and avoiding typical pitfalls of adopting a new methodology. Starting with a TLM IP design coding style, modeling guidelines, and synthesis subset, users can create TLM IP with an architecture that leverages the power provided by high level synthesis. The reuse of both design and verification IP is considered throughout the TLM-driven IP methodology.

C-to-silicon compiler creates high quality RTL using TLM golden source

The C-to-Silicon Compiler is a high level synthesis product that takes TLM SystemC IP descriptions and constraints, and creates RTL that can be used with the standard RTL implementation flow. To ensure the quality of results, it uses the Cadence® Encounter® RTL Compiler technology to create logic, and uses the timing and power information of this logic to extract architecture details of the resulting RTL.

The C-to-Silicon Compiler GUI shows the correspondence between the original SystemC and the lines of RTL code generated from it. This unique cross-referencing capability encourages communications between the systems designer and the RTL designer, and helps maintain the SystemC TLM as the golden source. It also brings debugging to a higher level of abstraction, and lets designers evaluate how changes to the SystemC source impact the generated RTL.

C-to-Silicon Compiler offers an incremental synthesis capability that can greatly ease the engineering change order (ECO) process and minimize changes to RTL code. Most other HLS tools require the re-synthesis of entire algorithms, meaning that small changes in the source code could result in totally different RTL. In such cases, logic synthesis and RTL verification must be redone. This makes it hard to maintain the SystemC code as the golden source. C-to-Silicon Compiler, in contrast, generates RTL code only for the portions of the algorithm that are changed, leaving the rest of the design untouched.

C-to-Silicon Compiler users can retarget TLM design IP to new micro-architectures by applying new constraints and generating new RTL. By specifying different timing, area, and power constraints, or different micro-architectural guidance such as pipeline stages, designers can generate new RTL. Design teams are thus able to reuse their IP with less effort, higher quality RTL, and shorter timeframes. Designers can also run “what-if” experiments by trying different micro-architectures.

Finally, C-to-Silicon Compiler automatically generates cycle-accurate SystemC Fast Hardware Models (FHMs) that execute at 80 to 90 percent of the speed of an untimed TLM model. These SystemC models permit early and fast verification and hardware/software co-development. The FHM is instrumented with extensions available in the Cadence Incisive environment to provide extra visibility of variables and signals for analysis and debugging.

Incisive functional metric-driven verification solution from tlm to closure

The Cadence Incisive functional verification platform is a fully integrated multi-language, multi-level functional verification solution. The Cadence Incisive Enterprise Simulator fully verifies OSCI TLM 2.0 compliant design IP using metric-driven verification, hardware-focused directed tests, software directed tests, or hardware/software co-verification.

Specially designed transactional analysis and unified debugging features aid the creation and verification of TLM IP, whether the design is a full TLM SoC, or an IP block in context of a legacy RTL SoC. The Incisive Enterprise Simulator natively recognizes TLM 2.0 constructs in its debugging environment, and provides save/restart and reset capabilities with extensions for SystemC/C++. The simulator infers transactional information and offers TLM-aware control, visibility, and debugging features. Users can debug all the interacting elements of their SystemC TLM 2.0 designs using control and debug operations at the transaction level.

With Cadence® Incisive® Software Extensions, designers can co-simulate processor models running embedded software with TLM hardware models. Incisive Software Extensions gives the verification testbench access to software executing on processor models, and brings capabilities such as metric-driven verification, pseudo-random test generation, and verification coverage to combined hardware and software simulation.

Cadence Incisive Enterprise Manager provides technology to mix TLM, TLM/RTL, and RTL functional verification to achieve successful closure. For SoCs that have large RTL legacy content, TLM simulation can be complemented with fast RTL validation using Cadence Incisive Palladium® or Cadence Incisive Xtreme®. These hardware platforms allow cycle-accurate verification at speeds that permit low-level software validation.

Services to help plan and implement key changes in projects

Approaching the transition to TLM-driven design and verification one IP block at a time mitigates some of the risk and costs. However, some projects must further mitigate risk and bring in experienced help to plan, execute, and proliferate best practices. Cadence offers globally available service experts in TLM-driven design and verification to maximize success and reduce adoption time, effort, and risk.

Conclusion

TLM-driven design and verification will ultimately enable TLM to replace RTL as the golden source for most design components. The advantages are compelling—much faster design and verification times, easier IP reuse, and fewer bugs. Designer productivity will take the biggest jump since the advent of RTL design. But the transition will not occur overnight. The adoption of TLM-driven design and verification will occur one IP block at a time, as new IP is created. And some design components will be best designed directly in RTL. Thus, it must be possible to combine new TLM IP with legacy RTL IP in design and verification environments.

By offering methodology guidelines, high-level synthesis, TLM-aware verification and debugging, and adoption services, Cadence can help ease the transition to a TLM-driven design and verification methodology.



Cadence is transforming the global electronics industry through a vision called EDA360. With an application-driven approach to design, our software, hardware, IP, and services help customers realize silicon, SoCs, and complete systems efficiently and profitably. www.cadence.com