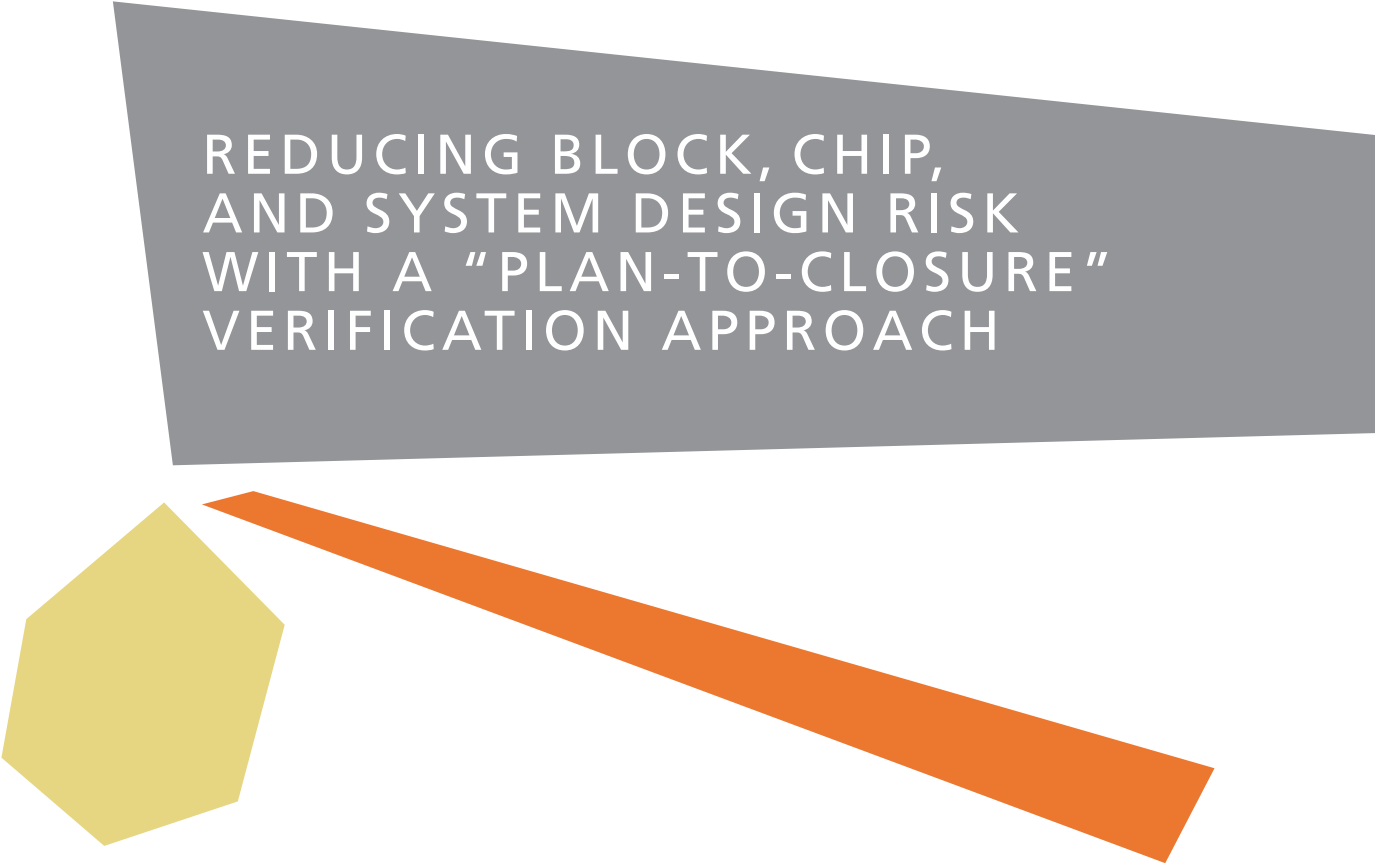




WHITE PAPER



REDUCING BLOCK, CHIP,
AND SYSTEM DESIGN RISK
WITH A “PLAN-TO-CLOSURE”
VERIFICATION APPROACH

INTRODUCTION

Verification has become a critical problem with regard to developing today's high-end digital integrated circuits (ASICs, ASSPs, and SoCs). Advanced devices like these can contain tens or hundreds of millions of logic gates, cost tens of millions of dollars to develop, and take a large engineering team years to design and verify.

This paper describes an advanced verification flow from Cadence that is scalable from block- to chip- to system-level designs, and takes the project team all the way from plan to closure. It meets the verification needs of today's high-capacity, high-complexity designs and the extreme capacity/complexity designs of tomorrow. Using this flow not only injects urgently needed predictability into project schedules (you know where you are, what you've done, and what you still have to do) but also increases productivity by optimizing engineering and verification resources. It further increases the quality of the final product while reducing overall project risk.

Until recently, the verification of digital integrated circuit designs was a relatively simple process. The major verification issues associated with the previous generation of designs may be summarized as follows:

- By today's standards, the vast majority of designs involved low capacity and complexity.
- The interfaces to designs were relatively simple; for example, rudimentary address and data buses combined with read and write control signals.
- The team of engineers involved in designing a device was typically also in charge of verifying the design. The disadvantage to this approach was that the same misinterpretations of the specification made during the design process were repeated during verification.
- Directed testbenches—which could be created reasonably quickly with relative ease—were typically sufficient to verify even the more complex designs.
- Testbenches were either created using a graphical waveform editor, or using a simple text-based stimulus/response language, or using the same hardware description language (HDL) with which the engineers captured the design itself and with which they were intimately familiar.
- Each design was typically unique and simple enough that verification accounted for a relatively small proportion of the entire development process. Designs also typically used little design IP. As a result, creating new testbenches from the ground up was not considered to be a problem in terms of time, effort, cost, and risk.

Now, times (and designs) have changed, and the industry is currently facing a verification crisis of the first order. The verification challenges in today's designs may be summarized as follows:

- A large proportion of designs are of extremely high capacity and complexity.
- The interfaces to the design can be incredibly sophisticated; for example, transactions requested on a bus may be deferred, re-tried, terminated, or completed out of order. This increases the difficulty of creating testbenches and tracking results.

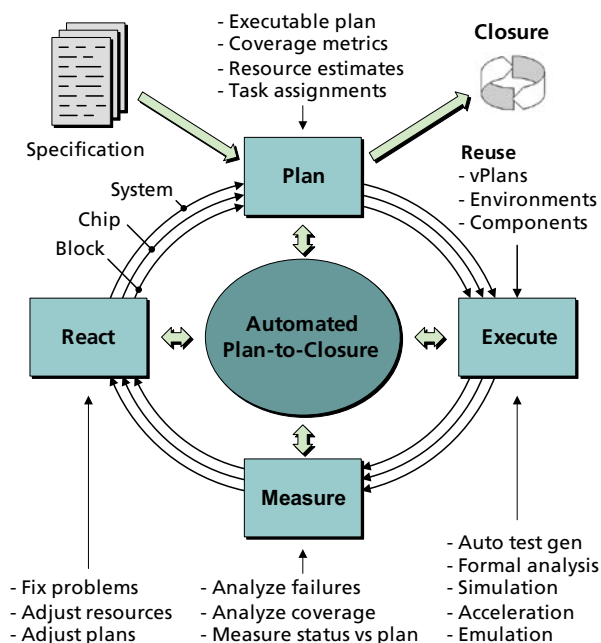
- Design engineers may have some responsibility with regard to early verification of individual functional blocks or small clusters of such blocks. However, enterprise (multi-specialist) teams are often tasked with verifying the entire design at the chip and system levels (where “system-level” refers to the co-verification of the chip hardware and any embedded software). The large numbers of people involved in the design and verification task mean that it is critical to be able to efficiently capture, communicate, and verify design decisions and intent.
- Directed testbenches are now typically sufficient to test only small portions of the design; for example, a few functions associated with an individual block. However, the complexity of today’s interfaces and protocols often requires the use of more sophisticated techniques such as constrained, random, coverage-driven testbenches, especially when verifying the design at the chip and system levels.
- Although hardware design engineers still need the ability to create small, local testbenches in the design language with which they are most familiar (SystemVerilog, for example), verification specialists often need to create advanced verification environments that significantly leverage object-oriented programming techniques (such as class libraries). Such environments often utilize the more sophisticated constructs in SystemVerilog, or high-level verification languages such as *e* and SystemC®, or a combination of all these languages.
- Modern designs make extensive use of design IP. These designs are now so complex that verification can account for as much as 70% of the entire development process, and creating new testbenches from the ground up is a huge problem in terms of time, effort, cost, and risk. In order to address this issue, project teams require the ability to create verification IP (VIP) and/or leverage existing internal or third-party verification IP that can be reused throughout the design process from the block to chip to system level and also across designs and platforms.

One problem associated with modern digital integrated circuits is that these devices are typically required to perform multiple functions. Take the case of a cell phone, for example. In addition to its role as a communications device, users also demand additional features including games, Internet access, a camera, an MP3 player and global positioning system (GPS) capability. A big consideration is that these functions are not standalone; they have to work together. If the user is using the MP3 player when a call comes in, for example, it is necessary to interrupt the MP3 function, take the call, and then resume the MP3 function. All of this adds to the complexity of the verification problem.

Cadence offers a new solution—a Plan-to-Closure verification approach that encompasses block-, chip-, and system-level verification. This approach requires much more than simply providing verification point-tools in isolation. Instead, it involves the combination of a verification methodology based on best-known principles, practices, and procedures backed by verification infrastructure and technology. The Cadence® Plan-to-Closure flow starts with the creation of an executable plan. This verification plan is executable, all of the coverage metrics can be linked to this plan, including assertion coverage, functional coverage, RTL code coverage, and software code coverage. The Plan-to-Closure process includes automatically measuring and analyzing verification results including failure and coverage data from these engines, so that all of these sources provide visibility to project managers, enabling them to make the appropriate decisions that will allow all the verifications processes to be predictably managed to reach closure. This plan-execute-measure-react cycle is repeated until verification closure is achieved. (see *Figure 1*).

This paper presents the key concepts behind the Cadence Plan-to-Closure verification approach. First, the paper introduces the concept of creating an executable verification plan featuring executable metrics. Next, the paper considers the need for a verification manager application that can use the executable verification plan to take the design to verification closure.

Figure 1:
Graphical Representation of the
Plan-to-Closure Process



Also considered are the use of transaction-level models (TLMs) for architectural exploration and evaluation; the use of assertion-based verification (ABV) techniques throughout the design and verification process; the creation and deployment of modular, reusable, coverage-driven verification environments and testbenches; increasing system-level verification performance by means of transaction-based acceleration (TBA); and full-system validation—including hardware/software co-verification—by means of in-circuit emulation (ICE).

Using the Plan-to-Closure approach increases predictability, productivity, and quality while reducing overall project risk.

DEVELOPING AN EXECUTABLE VERIFICATION PLAN

As its name suggests, the Plan-to-Closure verification approach starts with the planning process. This may be thought of as “beginning with the end in mind.” The first phase of this process is for the whole team—system architects, system engineers, verification engineers, hardware design engineers, and software development engineers—to brainstorm together to establish top-level verification goals. The second phase involves feature capture and attribute elaboration. This is where individual items to be tested are detailed, the way in which each item will be verified is defined, and the required coverage metrics for each item are specified. The third phase involves designing and implementing the verification environment (pulling resources together such as formal engines, software simulators, hardware accelerators, and emulators) and designing and implementing an executable verification plan (see *Figure 2*).

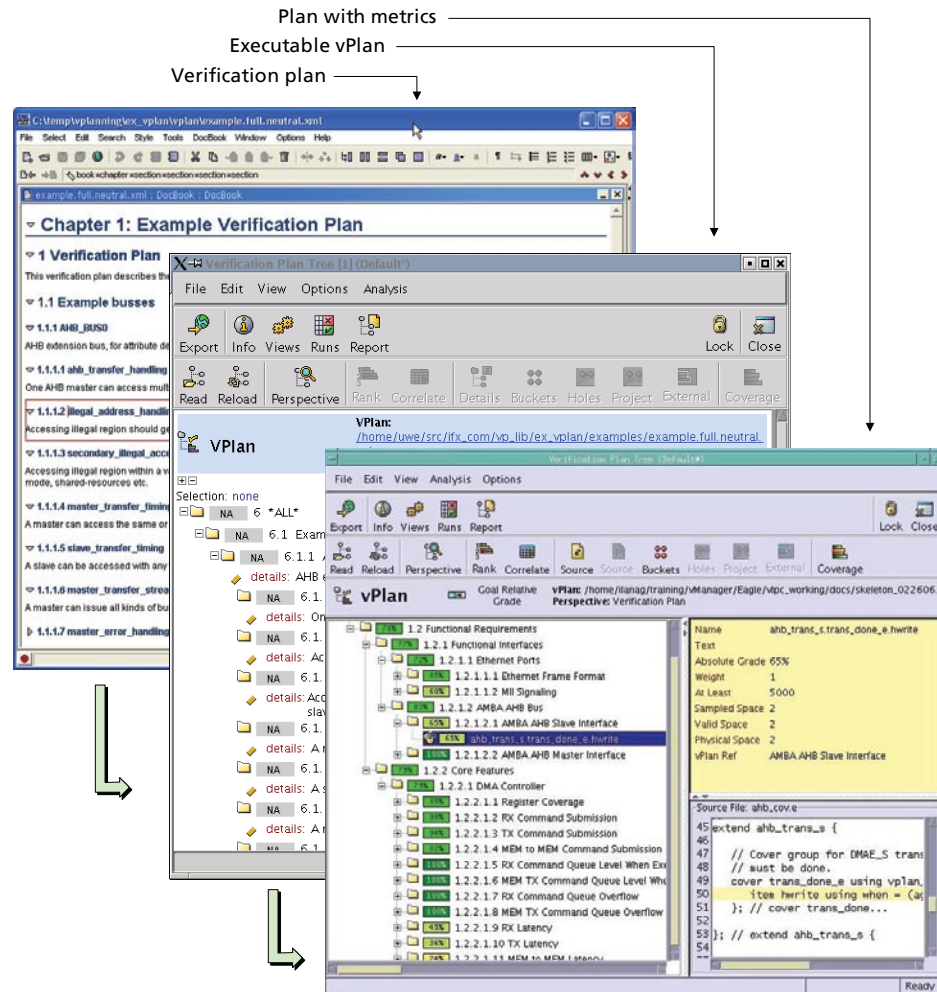
The resulting verification plan defines precisely what behaviors must be observed in order to ensure that the specification has been met. These plans also include verification milestones that define the times by which each portion of the verification process needs to be completed.

In order to facilitate the creation and reuse of verification IP, executable verification plans are hierarchical in nature; that is, one verification plan can instantiate one or more other verification plans, which can in turn call other verification plans, and so forth. This makes it possible to create individual verification plans for

different portions of the design, and then have a larger system-level verification plan that gathers all of the sub-plans together. The end result is that the sub-plans form a new type of verification IP that can be reused on future projects.

To speed the process of creating the verification plan for a complex system, a Plan-to-Closure verification environment may include access to a library of pre-defined verification IP components that are associated with industry-standard busses and protocols such as AMBA™ (AHB and AXI), Ethernet, OCP, PCI Express and USB. Each of these verification IP components will come equipped with its own predefined verification plan that can be quickly and easily incorporated into a master plan.

Figure 2:
Verification Plan Automation



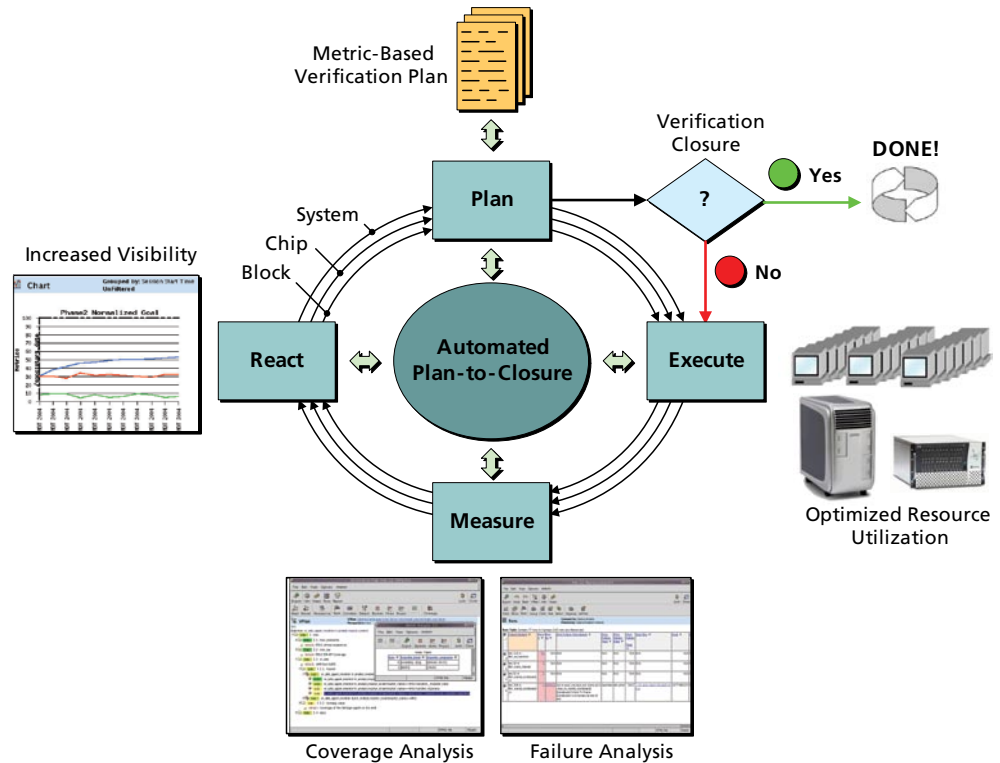
USING THE EXECUTABLE PLAN TO MANAGE THE VERIFICATION PROJECT

For the Plan-to-Closure verification environment to take full advantage of the executable verification plans presented in the previous topic, these plans must be complemented by a verification management application that automates the verification process from plan to closure (see Figure 3). This application will take the executable verification plan and automatically deploy the appropriate tools required to perform the various facets of the verification. These tools may include formal verification engines,

software simulators, hardware accelerators, and emulators. Such a verification management application will plug into workstation-farm load-balancing software to control and optimize the use of available resources.


When large numbers of engineers are creating different regression runs for different portions of a high-capacity, high-complexity design, the invariable result is a large amount of overlap. In the case of a modern design, for example, it is not unusual for 60% or more of the verification runs to be partially (or wholly) redundant. This equates to a vast waste of resources. In order to address this issue, the verification management application must locate and identify such redundant runs so that they can be eliminated from future regression tests.

Figure 3:
Verification Plan Management



Two key attributes associated with a verification management application are to bring visibility and predictability into the verification process. The management application used in a Plan-to-Closure flow must have the ability to automatically access the log files from the various verification engines, parse them, and analyze the results. It should compare the required metrics (assertion coverage, code coverage, functional coverage, etc.) with the actual results and react accordingly by re-deploying resources to address any problem areas.

The verification manager application must also separate design failures from simulation failures, sorting and grouping these failures for easy selection and action. In a Plan-to-Closure flow, it must be possible for multiple session results to be viewed together, enabling common failures to be grouped such that unique failures are emphasized and redundant work on common failures is eliminated. It must also be possible for failures to be correlated between simulation runs to determine if there is a particular bug that has a broader impact beyond the local context in which the error is first flagged. The verification manager application must also be capable of identifying the least-costly simulation to exhibit the failure and the optimal case for repeated debugging.



Users (managers and engineers) must be able to employ the verification management application to generate frequent, accurate, and concise reports in real-time. At the push of a button, a manager should be able to immediately see which portions of the design have been verified and which have not. Furthermore, as noted in the previous topic, the executable verification plan will specifically define certain verification metrics (such as coverage goals) that must be achieved by certain dates/milestones. The verification manager application must use these milestones to pace the team and to ensure that commitments to other groups and external customers are met. If any milestones appear to be in danger of slipping or have actually started to slip, the verification manager application must automatically alert the appropriate engineers, team leaders, and project managers, thereby enabling the team to quickly redeploy effort to focus on problem areas.

PERFORMING ARCHITECTURAL EXPLORATION WITH TLMs

At the beginning of the design process, there may be few (if any) functional blocks represented at the register transfer level (RTL). Even in the case where RTL representations are available, the team may elect to move to a higher level of abstraction in order to achieve the simulation performance required for architectural evaluation and exploration.

Raising the level of design abstraction can be achieved by means of transaction-level models (TLMs). Such models communicate with each other at the transaction level; for example, a TLM of a functional block may issue a transaction to a block of memory saying “I want to perform a read of the location at address xxxx.” This is much more efficient from both the performance and debugging points of view in terms of simulation time as compared to performing all of the low-level bit-twiddling operations on individual signals in RTL.

TLMs are much faster to build and verify as compared to their RTL equivalents because they contain less low-level detail than do RTL realizations of the design. In addition to facilitating architectural exploration and evaluation, working with TLMs allows system engineers to perform hardware/software integration, validation, and co-verification much earlier in the design process. In order to facilitate this process, the methodology portion of the Plan-to-Closure verification approach explains how to create these models and how to incorporate them in the system-level verification flow.

Once the architecture has been locked down, TLMs can also serve as reference models for the RTL representations that will be generated by the hardware design engineers. Thus, the methodology portion of the Plan-to-Closure verification approach also describes how to use TLMs in the role of reference models.

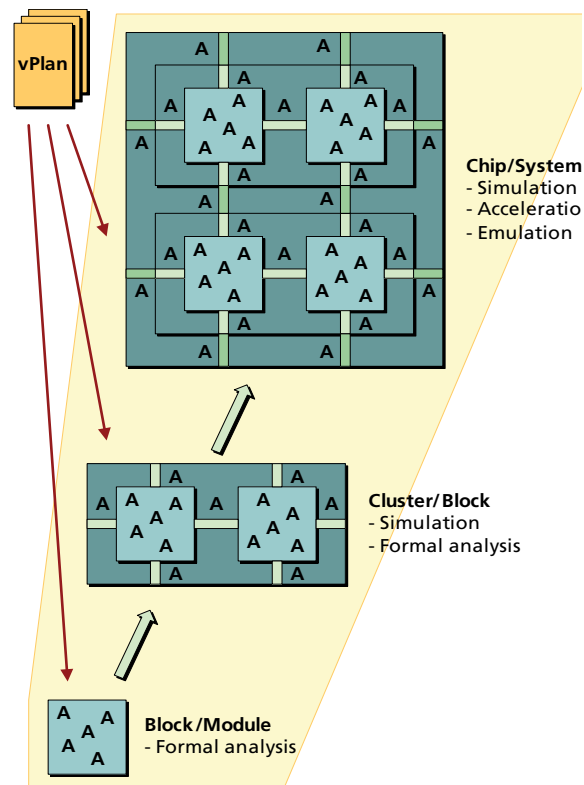
LEVERAGING ASSERTION-BASED VERIFICATION THROUGHOUT THE FLOW

Assertions are a way of specifying functional attributes or properties associated with a design or portions thereof. A simple assertion might be along the lines of “Signals A and B should never be in their active (logic 0) states at the same time.” Assertions can also extend to temporal sequences at the signal level, and even to transaction-level constructs, such as “When a memory read command is issued, an acknowledge response must be received within 6 to 18 clock cycles.”

In the case of the Plan-to-Closure verification approach, assertions can be associated with the design at any level, from individual blocks, to the interfaces linking blocks, to the entire system. The use of assertions in a modern verification environment has many different facets. The process begins when top-level specifications are captured in the verification plan. These specifications are expressed as assertions. Assertions are used to communicate requirements throughout the design and verification phases. They are also used to capture and verify interface requirements among the major functional blocks forming the design. Furthermore, they serve to document designers' assumptions and intent when implementing the various functional blocks, and they contribute to the coverage metrics in the verification plan.

The term assertion-based verification (ABV) encompasses the use of assertions throughout the design and verification process. A common starting point for hardware design engineers is formal verification, in which a formal engine is used to exhaustively verify individual blocks and remove the micro-architecture bugs. This allows a designer to start verifying the blocks weeks to months earlier before a testbench is available. As the process moves into the cluster-level (groups of blocks being verified together), both design and verification engineers can use formal verification and software simulation to verify proper integration. In addition to re-using the original block-level assertions, this typically involves writing new assertions for interface verification and coverage. Similarly, all block- and cluster-level assertions can be re-used for full-chip or system-level verification with software simulation, hardware acceleration, and emulation where they serve as excellent debug aids when bugs are encountered.

Figure 4:
Assertion-Based
Verification Flow



A key facet of the Plan-to-Closure approach is that each member of the project team must be able to employ assertions in the manner most appropriate to their portion of the design process (see Figure 4). For example, design engineers may prefer to leverage a narrow subset of SystemVerilog Assertions (SVA) or the Property Specification Language (PSL). Additionally, design engineers may decide to define assertions using a library approach, such as the Open Verification Library (OVL) or the Incisive® Assertion Library (IAL), which is a library of SVA/PSL modules that implement checks for common design structures

including datapath, control, and interface elements. By comparison, verification specialists may prefer to use more advanced SVA constructs or to take full advantage of the expressive capabilities of PSL or leverage the robust temporal capabilities of the *e* language.

Thus, the methodology portion of the Plan-to-Closure approach documents which aspects of the various languages and libraries are best suited for different tasks. Moreover, the methodology provides guidelines that describe how to develop reusable assertion-based verification IP components for such things as in-house protocols so as to improve verification efficiency.

In order to reduce risk and shorten time-to-market (and time-to-revenue) the Plan-to-Closure verification approach may employ pre-verified assertion-based verification IP products that completely implement commercial protocol specifications such as the AMBA bus from ARM. In this case, the assertions define a set of rules—or properties—that completely represent the behavior of the protocol and that can also be applied as constraints for formal verification to limit the “state space” being verified. These assertions are capable of being applied to multiple engines of verification, such as formal verification, dynamic simulation, or hardware-based acceleration and emulation. The advantage of such pre-defined and verified assertion-based verification IP is that users do not have to invest resources or cycles in developing these assertions, thereby shortening their verification cycle, increasing their product quality, and reducing their overall verification risk.

CREATING AND REUSING AUTOMATED TESTBENCH ENVIRONMENTS

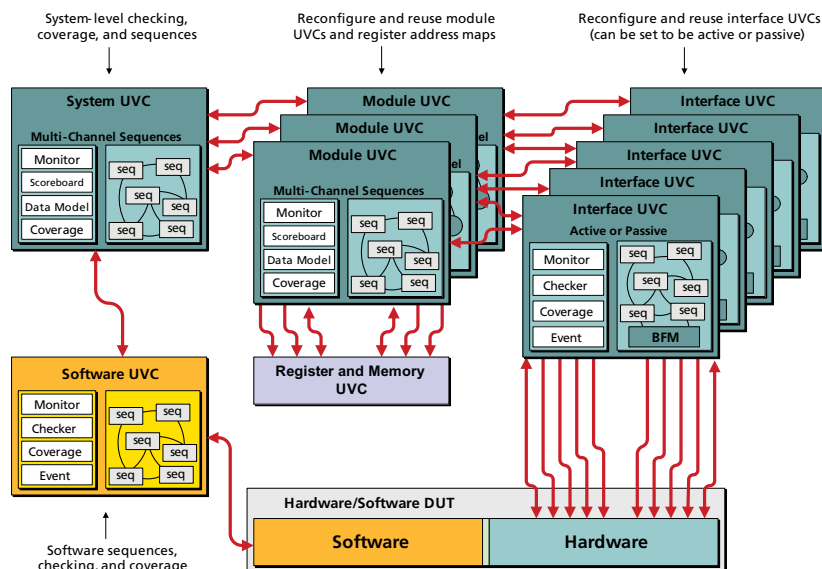
In conventional verification environments, testbenches created to verify individual functional blocks have to be discarded when the verification process moves to the cluster level. Similarly, testbenches created for use at the cluster level are discarded at the chip level, and so forth. This results in a tremendous waste of time and resources.

In order to address this issue, the Plan-to-Closure verification approach includes a methodology that defines how to construct verification environments in the most efficient manner. Using this methodology, it is possible to create block-level interface verification components in such a way that they can be used independently or controlled from a higher level. This modular, layered approach makes it possible to add to existing environments rather than being forced to create new environments from the ground up. When a cluster of blocks are being tested together, for example, a higher-level verification component can be created to direct and control a group of individual block-level testbenches. Similarly, cluster-level verification environments can be created in such a way as to facilitate their being controlled from the chip level, and so on. This layered approach to verification is a key aspect to creating complex system-level scenarios in the most efficient manner (see *Figure 5*).

Such a modular, layered approach to creating verification components also forms the basis for reuse. In addition to allowing the creation of plug-and-play verification components that can be reused—from block to cluster to chip to system—these components can also be reused across multiple projects and platforms.

It is important to remember that the various members of the design and verification teams involved in the development of a large digital integrated circuit will typically have different backgrounds and different ways of looking at things. For example, hardware design engineers working at the RTL level will tend to create directed, non-object-oriented testbenches using a language with which they are most familiar, such as SystemVerilog. By comparison, verification specialists may create advanced verification components and environments that significantly leverage object-oriented programming techniques (for example, class libraries) and use constrained, random, coverage-driven techniques. These components and

Figure 5:
Testbench Automation and
Reuse at the Block, Chip, and
System Levels



environments may be developed in SystemVerilog, or in a high-level verification language such as *e*, or in SystemC, or as a combination of these languages. Thus, the Plan-to-Closure verification approach supports all of these languages and techniques.

In order to speed the process of creating the verification plan for a complex system, a Plan-to-Closure verification environment should include access to a library of pre-defined verification IP components that are associated with industry-standard busses and protocols such as AMBA (AHB and AXI), Ethernet, OCP, PCI Express, and USB. Each of these verification IP (VIP) components should come equipped with its own predefined verification plan that can be quickly and easily incorporated into a master plan.

To speed the process of creating testbenches, the Plan-to-Closure verification approach includes a special verification component for verifying registers and memory; this would be especially useful at the chip and system levels. The verification environment should include access to a library of pre-defined VIP components that are associated with industry-standard busses and protocols such as AMBA (AHB and AXI), Ethernet, OCP, PCI Express and USB. In addition to supporting multiple languages, each of these components should come equipped with a compliance management system to achieve protocol compliance and with the ability to generate sequences of control and data stimulus. Furthermore, it should be possible to specify whether this stimulus is to be generated at the transaction level for use with TLMs or at the signal level for use with RTL representations. Also, it should be possible for these components to be used to create testbenches at the block, cluster, chip and system levels.

PERFORMING FULL-SYSTEM VERIFICATION

In order to achieve first silicon and first software success, it is necessary to perform full-system hardware/software co-verification. In addition to a plan- and metric-driven approach—coupled with the use of high-quality verification IP—it is necessary to raise the level of design abstraction and to increase system performance using hardware acceleration and/or emulation.

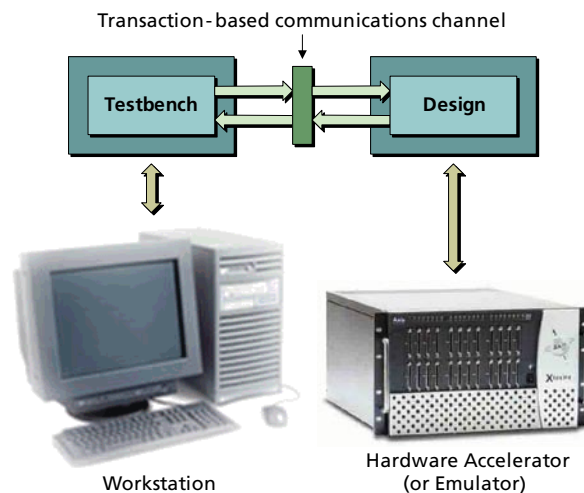
As discussed earlier in this paper, one technique for raising the level of design abstraction is to use transaction-level models (TLMs), which are much more efficient from both the performance and debugging points of view in terms of simulation time as compared to performing all of the low-level bit-twiddling operations on individual signals in RTL.

And, as discussed in the previous topic, the advanced testbenches associated with a Plan-to-Closure verification approach will have the ability to generate sequences of control and data stimulus. It should be possible to implement this stimulus at the transaction level for use with TLMs or at the signal level for use with RTL representations. Furthermore, a Plan-to-Closure environment must support the mix-and-match (or plug-and-play) of RTL and TLM blocks.

At some stage the main logic being tested will be represented at the RTL level while the surrounding functions may be represented as TLMs. When working with RTL, software simulation cannot provide the speeds necessary to perform tasks such as hardware/software co-verification. At this point it becomes necessary to move up the performance curve to hardware acceleration and/or emulation.

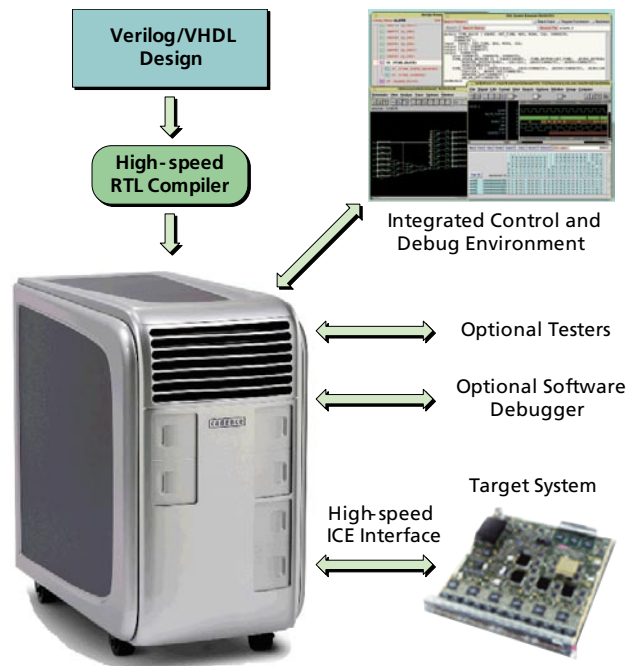
In the case of hardware acceleration, any performance advantages gained from moving the RTL into the accelerator will be severely degraded if the stimulus from—and responses to—the testbench running on the host workstation's software simulator is handled at the bit-twiddling signal level. Thus, the Plan-to-Closure approach supports transaction-based acceleration (TBA). In this case, communications between the testbench running on the host workstation and the RTL running inside the accelerator are handled using transactions, which dramatically increases the performance of the system as a whole (see Figure 6).

Figure 6:
Transaction-Based Acceleration



The highest level of performance is achieved by in-circuit emulation (see Figure 7). In this case, the entire chip design is loaded into an emulator, which communicates with the surrounding system at hardware speeds. However, even emulators cannot achieve the extreme real-time speeds required by today's high-end designs. Thus, in order to reduce time-to-emulation for design applications in wireless, multimedia, and networking markets, a Plan-to-Closure verification environment may include special SpeedBridge® rate adapters. These rate adapters—for example, Advanced Graphics Port (AGP), Multi-Ethernet, PCI/X, PCI Express, Audio/Video, and RGB adapters—must be capable of interfacing an in-circuit emulation system to a real-world environment that is running at tens to hundreds of MHz.

Figure 7:
In-Circuit Emulation



SUMMARY

Verification has become a critical problem with regard to developing today's high-end digital integrated circuits (ASICs, ASSPs, and SoCs), which may contain large numbers of logic gates, be costly to develop, and take large numbers of engineers several years to design and verify.

Today's designs require a verification approach that encompasses block-, chip-, and system-level verification, from plan to closure. The Plan-to-Closure approach to verification from Cadence provides much more than simply verification point-tools in isolation. Instead, it involves the combination of a verification methodology based on best-known principles, practices, and procedures backed by verification infrastructure and technology.

The Plan-to-Closure flow starts with the creation of an executable plan. This plan is subsequently executed in a reusable way across all engines from formal verification to emulation. The Plan-to-Closure process includes automatically measuring and analyzing verification results and reacting appropriately. This plan-execute-measure-react cycle is repeated until verification closure is achieved. The core concepts associated with the Plan-to-Closure verification approach are as follows:

- **Verification planning and management:** This involves the ability to capture a detailed executable verification plan that is both human-and machine-readable. Since the verification plan is executable, all of the coverage metrics can be linked to this plan, including assertion coverage, functional coverage, RTL code coverage, and software code coverage.


The Plan-to-Closure approach also includes a verification management application that deploys and controls the various verification engines, including formal, simulation, and hardware acceleration/emulation; it extracts failure and coverage data from these engines; it provides analysis capabilities from all of these sources; and it also provides visibility to the project manager so that appropriate decisions can be made to predictably drive all the verifications processes to reach closure.

- **Assertion-based verification (ABV):** Using the Plan-to-Closure approach, it is possible to associate assertions with the design at any level, from individual blocks, to the interfaces linking blocks, to the entire chip or system. ABV may be employed at every one of these levels, thereby enabling design engineers to achieve more efficient block bring-up and helping verification engineers to ensure proper functional behavior as captured in the verification plan. The Plan-to-Closure approach also supports ABV using a combination of technologies, including formal verification, simulation, hardware acceleration, and emulation.
- **Testbench automation and reuse:** The Plan-to-Closure verification approach features a methodology that defines how to construct verification environments in the most efficient manner. Plug-and-play testbenches—ranging from simple directed tests to constrained, random, coverage-driven testbenches—can be created in a mixture of languages, including SystemVerilog, *e*, and SystemC. Furthermore, testbenches created to verify individual functional blocks and clusters of blocks can be reused at the chip and system level and across multiple designs and platforms.
- **Full-system verification:** The Plan-to-Closure verification approach supports multiple levels of abstraction, including transaction-level models (TLMs) and RTL. It also supports multiple verification engines, including formal verification, software simulation (TLM and RTL), hardware acceleration, and emulation. These representations and engines enable system verification and validation engineers to verify the entire system, including hardware, embedded software, and external interfaces.

It is important to remember that the various members of the design and verification teams involved in the development of a large digital integrated circuit will typically have different backgrounds and different ways of looking at things. For example, hardware design engineers working at the RTL level will tend to create directed, non-object-oriented testbenches using a language with which they are most familiar, such as SystemVerilog. By comparison, verification specialists may create advanced verification components and environments that significantly leverage object-oriented programming techniques (for example, class libraries) and use constrained, random, coverage-driven techniques. These components and environments may be developed in SystemVerilog, or in a high-level verification language such as *e*, or in SystemC, or as a combination of these languages. Thus, the Plan-to-Closure verification approach supports all of these languages and techniques.

The Plan-to-Closure verification approach spans the entire verification domain, including design teams (small groups of logic design engineers working with RTL) and enterprise (multi-specialist) teams comprising system architects, system engineers, design engineers, verification engineers, and software engineers.

The Plan-to-Closure approach to verification from Cadence delivers a number of benefits. It increases predictability (you know where you are, what you've done, and what you still have to do). It improves productivity by optimizing engineering and verification resources. It increases the quality of the final product. And it reduces overall project risk.



This whitepaper provides only an overview of the Plan-to-Closure approach. Cadence has developed a complete Incisive Plan-to-Closure Methodology that includes documented best practices, golden example, and libraries and utilities. For more information, go to:
www.cadence.com/products/functional_ver/incisive_plan_to_closure_methodology.aspx



Cadence Design Systems, Inc.

Corporate Headquarters
2655 Seely Avenue San Jose, CA 95134
800.746.6223 / 408.943.1234
www.cadence.com

© 2006 Cadence Design Systems, Inc. All rights reserved. Cadence, Incisive, and SpeedBridge are registered trademarks and the Cadence logo is a trademark of Cadence Design Systems, Inc. ARM is a registered trademark and AMBA is a trademark of ARM, Ltd. SystemC is a registered trademark of Open SystemC Initiative, Inc. in the U.S. and other countries and is used with permission. All others are properties of their respective holders.

7204 10/06 KM/FL/JMR/PDF

