INCISIVE VERIFICATION ARTICLE
JUNE 2005

cadence

ASSERTION-BASED ACCELERATION

MAKARAND JOSHI, CADENCE DESIGN SYSTEMS, AND KEVIN DONOVAN, CADENCE DESIGN SYSTEMS

INTRODUCTION

Assertion-based verification (ABV) has been adopted by many successful companies in the global electronics market to help them improve design quality and reduce time-to-market. ABV has evolved in response to the problem of not being able to adequately verify a design's functional operation. As designs have gotten bigger and design cycles have gotten shorter, traditional simulation has failed to deliver adequate functional verification.

Simple assertions have been used for years in both Verilog® and VHDL designs. Designers have used simple assertions like RTL constructs to monitor and report on signal activity, and thus behavior, in their designs. This has evolved over the years to where we now have specific languages, tools, and methodologies that support ABV.

There are many different aspects of this verification methodology in a design flow. This paper, however, focuses on one advanced aspect of this methodology: assertion-based acceleration (ABA). ABA provides enables users to exercise assertions in their designs and testbenches inside a hardware accelerator such as the Incisive® Palladium® system. Companies have found that integrating ABV with advanced ABA technology is a powerful combination, enabling them to enhance quality and significantly reduce the time it takes to get their products to market. We will discuss what features and capabilities make up this advanced ABA solution when using the Palladium family of hardware accelerators.

ABV OVERVIEW

Briefly stated, ABV allows design intent to be captured and gives "white box" visibility to all engineers who may work on the design. White box visibility refers to the ability to see various portions of the design, as opposed to "black box," which allows only the primary input and output to be observed or controlled. Increased visibility helps engineers understand what has already been tested and what has failed. Once ABV is implemented, users will experience a variety of benefits. The three most important are:

1. ABV detects incorrect operation (errors) directly, and close to the source. This localization simplifies debugging and fixing the unexpected behavior. (Other verification approaches either completely miss the bug or notice it at a register or primary output, which makes debugging far more difficult.)
2. ABV detects correct operation and logs exercising of the expected behavior for coverage analysis. This enables users to optimize tests to improve efficiency, and it notifies them when they are done with testing.
3. ABV documents functionality with an "executable specification." This helps facilitate IP re-use by ensuring correct usage of IP in future projects.

At this point, however, we will introduce the two major capabilities that are required to implement a basic ABV solution: a means of specifying behavior, and the ability to display and debug results.

There are two ways that assertions can be evaluated: dynamically and statically. Basic assertion analysis is performed along with simulation. Since simulation is something that nearly all designers and verification engineers already perform on their designs, assertions fit well into this methodology. Static assertion analysis, also known as formal analysis, does not require simulation.

Dynamic analysis is defined as the evaluation of assertions side-by-side with design elements by the simulator or the hardware accelerator. Assertions may be triggered in response to violation of expected behavior or simply due to completion of an expected sequence of events, and may enable some action to be taken. This action, for example, can be issuing a message or logging a coverage event. This capability easily adapts to any current verification design flow. Assertions can be added to the design or testbench. As simulations are run, assertions will help find additional bugs and report coverage data. No other changes to the design or testbench are required; neither are additional software applications required.

To learn more about the components of a complete ABV solution, please refer to Pete Johnson's paper published in the March 2005 issue of the *Incisive Verification Newsletter.*

REQUIREMENTS FOR A BASIC ABA SOLUTION

As designs continue to grow, there is more demand for simulation acceleration. While simulators continue to make incremental improvements in their efficiency, the fact is that the performance increases are much slower than the growth in design size. As such, more and more customers are turning to hardware-based acceleration. But hardware accelerators must be able to put as much of the design as possible into the accelerator.

This pertains to assertions as well, since assertions monitor the signals of interest as well as track coverage. Most accelerators get the design into the hardware through some sort of synthesis or compilation. This needs to be extended to assertion languages and libraries as well. Additionally, the user interface and debug tools must provide adequate information to debug the assertion failures while being consistent between the simulator and accelerator. Users want to be able to view results from these environments interchangeably, as well as be able to combine coverage results from each of them.

Assertions need to be added to the design in order for the tools to operate on it. Assertions can be specified in one of two ways: 1) using a library of pre-defined assertions and instantiating them into the design; or 2) using an assertion language to model assertions.

Design teams will do both in many cases. Designers usually want to focus on RTL, pre-defined, and standard assertions (FIFO, flip-flop) to make it easier for them to annotate their design. Verification engineers will use pre-defined verification IP (VIP) wherever possible for things such as standard protocol monitors. They will also add user-written assertions as required. Creating user-written assertions, or developing a library of re-usable assertions, requires an assertion language.

In order to support monitoring and analysis of assertions in a hardware accelerator connected to a dynamic target, it is critical that there exists support for the monitoring and recording of assertions in hardware.

A complete ABA solution requires more than just tools and language. It requires adherence to a methodology as well. In later sections of this document, we will explore this in more detail.

ABA ADHERENCE TO STANDARDS

In the past, HDL-based solutions have been used for monitoring expected design behavior. An example of using an HDL to develop basic assertions support is the user of assert (VHDL) or $display (Verilog) in the design or testbench to provide visibility into the internals of the design.

In the recent past, more design and verification engineers have adopted the use of assertion monitors from the Open Verification Library (OVL). This is a set of assertion monitors available to users from Accellera (www.eda.org/ovl/download.html). Since these assertions were developed in both Verilog and VHDL, they are easy to use in most design environments. The OVL includes basic assertion components and thus may need to be augmented with more advanced assertions embedded in the design to monitor complex design behavior.

Users typically want to check for complex behavior such as a protocol check, so developing more complex assertions is necessary for most designs. It can be difficult to develop an HDL assertion to determine that a read operation executed correctly, since it defines a temporal relationship. Assertion languages were developed to address this very limitation.

This paper will not explain the evolution of these languages — it suffices to say that there were several proprietary languages in use before the industry settled on two standard languages: Property Specification Language (PSL) and SystemVerilog Assertions (SVA). PSL is a separate language that can be used with any HDL, including Verilog, SystemVerilog, VHDL, and SystemC®. Once the assertions have been instantiated or written into the design, they can then be used for locating errors and tracking functional behavior.

The Palladium family of hardware accelerators supports OVL and complex PSL assertions today. SVA support will be available on Palladium products before the end of 2005.

PALLADIUM ABA FLOW

ABA methodologies achieve ultimate performance through synthesizing assertions so they can be loaded with the user's design into the hardware accelerator. Palladium ABA technology supports all four main usage modes: 1) emulation/ICE; 2) synthesized testbench (STB); 3) signal-based acceleration (SBA); and 4) transaction-based acceleration (TBA).

The process of including assertions and coverage constructs in the Palladium acceleration/emulation flows requires mapping the assertions and coverage constructs to the logic on the Palladium engine. This logic is responsible for implementing the functionality of the assertion and/or coverage check as well as for monitoring the violation of any assertions or completion of any coverage checks at runtime.

The compile flow with assertions is based on, and identical to, the traditional Palladium compile flow. The use model for assertions is also identical in the simulation acceleration and in-circuit emulation flows.

COMPILE FLOW WITH ASSERTIONS

The initial step in the flow is the generation of a snapshot of the design with assertions included, by running the Incisive Unified Simulator (IUS) compiler. It should be noted that this is only a compile step; no simulation is required. The user may include PSL statements embedded within the RTL or as external files. Sharing the same front end of the IUS compiler for both dynamic simulation and acceleration flows ensures consistency of semantic interpretation in the ABV and ABA flows.

Extraction and mapping of assertions is the next major step in the process. The most crucial aspect of the flow extracting the assertion information from the user's model and mapping the appropriate logic onto the Palladium engine. This step is invoked via the verification acceleration pre-processor (VAP). The user can make this completely transparent in the flow by setting a switch in the compiler. If the switch is not specified, assertions are not included in the compiled model.

The reasons why a user may not want to include assertions in the design all the time include:

- Usage of minimal gate capacity on the Palladium system: the assertion models generated are optimized; however, they do require some additional Palladium gates beyond what the design uses.
- Runtime performance impact: to let the Palladium system continue to run at maximum (hardware) speed, the user may want to disable interruptions associated with reporting assertion failures. Disabling of assertions can be done at compile time as well as at runtime.

When the compiler assertion switch is set, the VAP step will walk through the design hierarchy, automatically extract the assertion and coverage objects, and map them into individual logic blocks within their original locations in the design.

The process of mapping assertions to logic involves complex analysis and mapping of temporal and logic elements of the assertions into internal representations that can then be converted into synthesizable logic for mapping to the Palladium system. The ABA flow uses a Cadence® internal proprietary technology called Common Assertion FSM Extraction (CAFE) to assist with the mapping process. The logic blocks representing assertions are stitched together with the design to establish the appropriate connectivity with the assertion driving logic as well as the assertion monitoring instrumentation. The latter is responsible for monitoring and collecting assertion and coverage information during the acceleration/emulation of the design.

Running the VAP compile step with assertions enabled also generates metadata automatically for the downstream compiler steps in the flow as well as information for the runtime environment to use.

Once the assertion mapping is complete, the design is compiled using the traditional Palladium compile flow.

TRADITIONAL PALLADIUM COMPILE FLOW

The existing Palladium compile flow for SBA and TBA includes the VAP step already. After the VAP step, synthesis and back-end compilation steps transform the RTL models into Palladium primitives and map them to the acceleration and/or emulation engine.

The Palladium system has special instrumentation for handling dynamic targets. ABA is capable of supporting assertions when an external dynamic target is connected to the Palladium system. When assertions are configured for operating in conjunction with a dynamic target, special assertion and coverage monitoring instrumentation is generated automatically. This special instrumentation allows collection of assertion failures and coverage completion information on the hardware without stopping the emulation, and it presents the information to the user at the end of the emulation run.

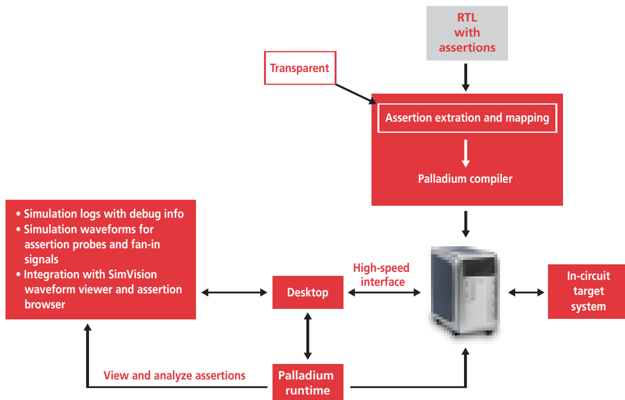

Figure 1: Compile and runtime flow in ABA

PALLADIUM RUNTIME FLOW AND OPTIONS

The first step in configuring the Palladium system at runtime for ABA requires the signal-based acceleration, transaction-based acceleration, or emulation/ICE user to select "assertion monitoring to be turned on." This can be done simply by issuing a command at the user interface or selecting it via the graphical user interface.

After running the simulation with the hardware in any of the above modes, the user can choose any or all three of the following ways to gather assertion results. A rich set of user interface commands allow ABA users to:

- Obtain detailed assertion summary reports
- Debug any unexpected results or assertion failures using assertion logs and waveforms
- Modify the testbench based on coverage results

At runtime, depending on the verification goals, the user may chose from a variety of use models:

Use model #1: In an extensive user debugging mode, the recommended option may be to stop the run on the first assertion that fires and upload assertion log data and waveforms surrounding the assertion firing. At runtime the user can select the severity level (as specified with the assertion) setting as a filter in order to best track down a problem or bug. After studying waveforms and assertion log data, they can either progress to the next assertion firing or continue to debug the existing one.

Use model #2: This mode is recommended for the user with a very long simulation acceleration or emulation run, normally taking many hours or days to complete. This mode would send real-time assertion log updates on the assertions that fired, based on the severity filter setting set by the user, back to the workstation as they occurred. This runtime use model allows constant assertion updates as they occur in real-time, so the user can decide to continue the long simulation acceleration/emulation run or stop and begin in-depth debugging.

Use model #3: This optional mode is the fastest runtime mode for ABA when using either simulation acceleration (including TBA) or emulation/ICE modes. Using this mode creates the log file at the end of the simulation acceleration or emulation run. No real-time data is available. This mode is used for long series of tests in regression mode, where real-time updates are not as critical as getting the results of the final data in a concise assertion log file for all tests. This mode has almost zero impact on hardware acceleration speed since the run does not have to stop and record information in the log files.

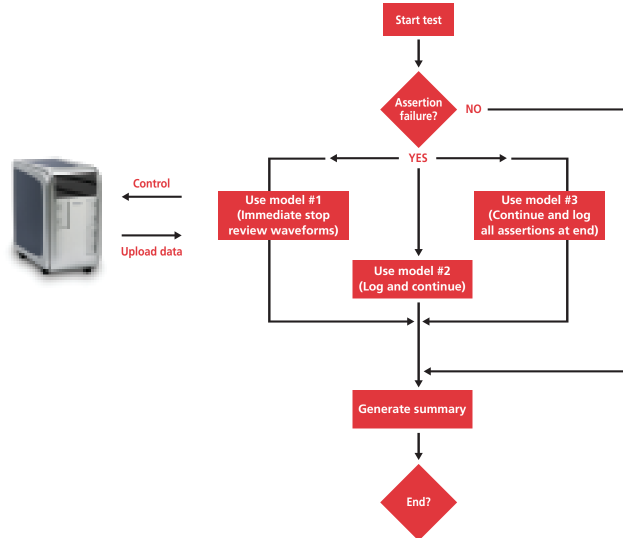

Figure 2: ABA runtime use model options

ABA DEBUG AND COVERAGE CAPABILITIES

The ABA methodology keeps the amount of user changes required to move from an ABV environment to a minimum. Since most ABV flows begin with adding assertions to the design for use with simulation, keeping the same simulation debug and analysis tools is important to minimize the learning curve. To achieve this, a complete and seamless integration of assertion debugging into the existing simulation debugging environment is required.

Accurate assertion debugging requires a variety of data, including: what signals are involved in the assertion; the current state of the assertion; and when the assertion begins, finishes, or fails. The user also needs control over the assertion language constructs. The ability to turn assertions on or off, and/or starting and stopping data recording, is important for both ABA and ABV methodologies.

Assertion coverage is one additional capability that is helpful in getting a brief overview of all the assertions in the design. Coverage information helps engineers understand the number of times the design has been evaluated, the status of every assertion, and

how many of those times it has failed. It is important in ABA that this capability has the same GUI as in ABV usage with simulation, as well as supporting inter-tool communication. This allows users to drag and drop between various windows. ABA debug and coverage capabilities achieve these objectives and enable the acceleration of assertion failure detection and remedy of incorrect behavior. The SimVision waveform viewer and assertion browser are a key component of the Incisive debug environment that facilitate easy and quick debugging of assertion failures.

When utilizing the assertion browser in ABA with the Palladium system, the contents of the assertion browser will be populated automatically to reflect the assertion and coverage objects belonging to the Palladium partition. The assertion states will reflect the status of each individual assertion whether it is in a disabled, inactive, active, finished, or failed state.

In addition to providing user access to the assertion status/state all along the run, Palladium FullVision capability allows ABA to upload the fan-in information associated with the assertion automatically in response to an assertion failure.

An ABA debug session also allows users to cross-highlight between the SimVision waveform viewer and the assertion browser. It also generates updates of assertions in the assertion browser based on moving the time cursor on the waveform viewer. Sorting and filtering of assertions based on their status at the current point in time is determined by the time cursor in the waveform viewer.

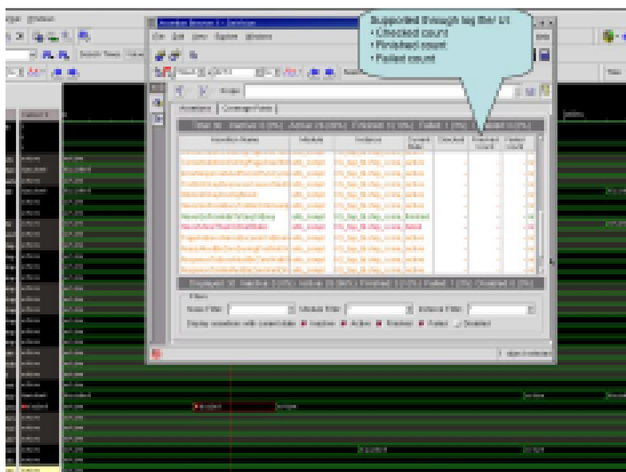

Figure 3: ABA debug and assertion browser interface

ABA LIBRARY SUPPORT

It can be a significant task for a user to write all their required assertions for a project. Whether a user needs to create a single bus interface, which can have dozens or hundreds of assertions, or multiple bus protocol and interfaces, the task can be daunting.

In addition, if the user must add in all the assertion elements for the design as well, the user may be looking at a significant time requirement to complete the effort.

There are two main types of assertion-based verification IP (VIP) that can help: 1) function monitors, and 2) protocol monitors.

Function monitors are modules that can be used on the logic structures of the design. Some of these are available in the OVL; more advanced libraries may include checks for constructs like FIFOs, state machines, and arbiters. They consist of one or more assertions, based on the complexity of the functionality being checked, and they are bundled together in a module such that they can be easily instantiated into the design. Cadence supports a rich function monitor library called the Incisive Assertion Library (IAL).

Protocol monitors are essentially a set of assertions that cover the various behaviors of a standard protocol, such as AMBA or PCI Express. These types of protocols have numerous operating modes, and each mode has dozens of different things that need to be checked: simple things like the “read” and “write” signals never being on at the same time, and complex things such as the proper sequence of signals to initiate and complete a burst operation.

Building and testing such a complex monitor is a time-consuming task. It is preferably done only once, and then re-used across many designs. Protocol monitors therefore are developed by a variety of sources. Design IP companies are building them to help their customers use IP blocks more effectively and easily. Verification IP companies build them to sell to customers that use a variety of different interfaces and protocols in their designs. EDA companies build them to offer along with their tool sets. And finally, end users build them for proprietary or custom interfaces and protocols that they plan to use across multiple projects.

The most useful forms of VIP will contain not just assertions for error checking, but coverage monitors as well. This allows the monitor to log various types of coverage data automatically, without the user having to add it manually. Both types of VIP can be used in a single design. For instance, a design engineer may use a variety of function monitors to check the inner workings of a block or module, while a verification engineer may instantiate protocol monitors on the major interfaces of the design to make sure they are properly exercised during system-level simulation. Cadence supports a library of assertion protocol monitors as part of its Incisive Verification Intellectual Property (VIP) library.

Currently, Palladium ABA supports the Incisive Assertion Library (IAL), which is made up of a library of components that can check common design blocks,

such as FIFOs, arbiters, memories, and CRC generators, as well as check buses for one-hot, grey-code, or other legal values. In addition, there is support for assertion monitors for complex protocols like AMBA AHB and PCI Express within the Incisive VIP library. Both the IAL and the assertion monitor library components can be accelerated to full Palladium performance levels since they are fully synthesized into the hardware accelerator.

SUMMARY OF ABA

We have presented the methodology and the support in the Incisive Palladium environment to implement an assertion-based acceleration verification infrastructure. Assertions allow design and verification engineers to leverage design knowledge captured within the design to identify bugs much earlier and with enhanced visibility in comparison with traditional verification methods. The use of assertions along with the raw performance of the Palladium acceleration and emulation engine is a powerful combination. It enables users to harness up to 100,000 times the performance over simulation in their verification efforts, thus allowing them to hone in on and fix bugs quickly.

By implementing an ABA flow, designers gain a variety of benefits that include catching hard-to-find bugs embedded deep in the design by exercising assertions that would not be exercised in realistic verification cycles in a simulation environment alone. With ABA, designers can debug their assertions in real-time even when connected to a live target in an in-circuit emulation mode. The use model for ABA is simple and very similar to the familiar Incisive simulator environment, so debugging commands and scripts can be re-used in many cases. All these capabilities allow users to find bugs earlier in the design cycle, thereby making verification more productive and isolating bugs more precisely, thus increasing debug efficiency.

The net result: the Palladium ABA user will find more bugs more quickly and get to market faster with a thoroughly tested and more reliable product.