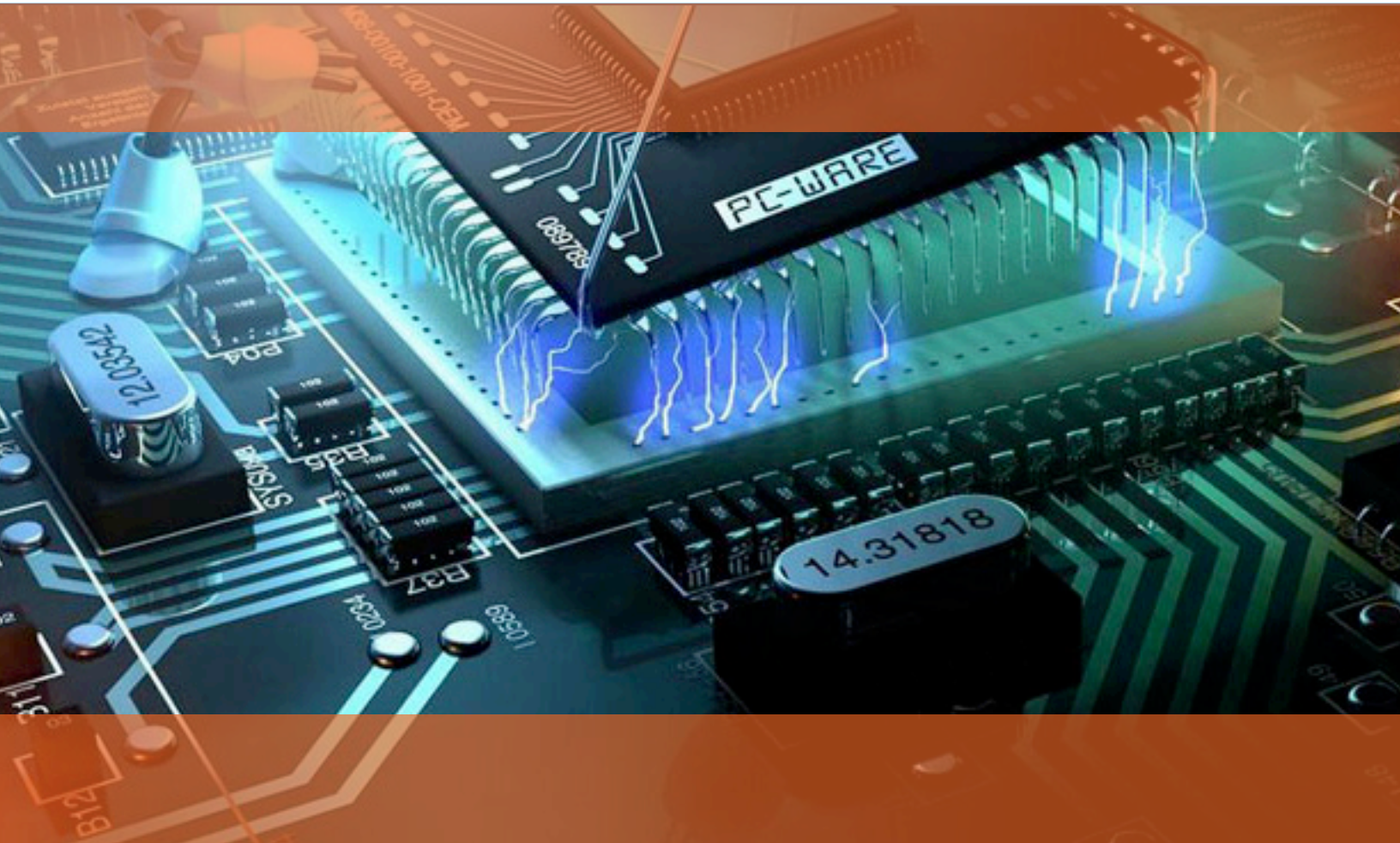
AGNISYS
ACCELERATING DESIGN VERIFICATION

How To Streamline The Universal Verification Methodology (UVM) Implementation Process



Introduction

The increasing prevalence of electronic systems results in complexly that make designing such integrated circuits challenging, but also turns the verification process into a rigorous and time-consuming step. Originally, verification of the integrated circuit used to be performed manually, however with the exponential growth rate of the circuit's complexity the process needed to be automated and streamlined as much as possible. Since there are countless methods to define and automatically generate, check and report test vectors (stimulus); the verification engineering community gathered their skills and expertise and defined a Universal Verification Methodology (UVM) as a standardized solution for such tasks. The current reference implementation of the UVM standard is in the SystemVerilog language. SystemVerilog and UVM provide mechanisms to create company-wide consistency and reusable verification components for checking, coverage collection, and stimulus generation, and to modify the behavior of those components for specific tests. However, SystemVerilog and UVM provide more than this, so much more, in fact, that the learning curve can be discouraging for novice engineers.

While much of the research and development in functional verification is dedicated to experienced users who solve the toughest verification problems, we find that the majority of mainstream users are focused on how to become productive with SystemVerilog and UVM faster and with minimum programming skills.

If you are already an experienced verification engineer familiar with SystemVerilog you probably face less trouble understanding and adapting your test benches with UVM. In the next section, we review how to get started with understanding and learning UVM faster and in a more efficient manner.

Get started with Basics of UVM

To adopt your test bench with UVM, you should first know which materials and topics you should learn first. Then you should answer a rather more important question. Where are the resources that can help accelerate your learning process?

Where to begin?

The very first step to get familiar with UVM is to find the equivalent representative for familiar VHDL or Verilog language terminologies like design entities, modules, processes, ports in UVM. For example, it shortens the learning time for the novice verification engineer to know that UVM component is the equivalent of design entity in VHDL or module in Verilog. The second step towards fast adoption of UVM in your design is to limit the numerous features and protocols that are provided in the UVM library and stick to the most practical and well-behaved subsets of the UVM library.

As module is the most important part of any Verilog design, component (uvm_component) is the central part of any System Verilog design. UVM library has several alternative classes for uvm_component suitable for different applications and designs. Each of these classes inherits all methods and properties of their parent uvm_ component class.

Among these components, the following components are the most important ones:

- **uvm_env:** The base class that contains all other components and the complete testbench.
- **uvm_agent:** Standard architecture for verification component provided by UVM.
- **uvm_driver:** Base class that initiate requests for new transactions.
- **uvm_monitor**: It is a base class for user-defined monitors.
- **uvm_scoreboard:** It is a base class that let you define your customizable scoreboard object and methods. The scoreboard objects hold and determine the results of each test.
- **uvm_sequencer:** is the base class for the sequencer object that is to generate a stream of transactions to be consumed by the driver object.
- **uvm_subscriber:** Provides an analysis export for receiving transactions from a connected analysis export.

In addition to the component classes there are three significant other classes that will be found in any UVM design.

1. **uvm_object**: which is the parent class of all components and transactions. uvm_object defines an interface of core class-based methods.
2. **uvm_transaction:** The uvm_transaction class unlike uvm_components, is transient in nature. It inherits all of the features of uvm_object and adds timing and recording interface.
3. **uvm_root:** The uvm_root class is special uvm_component that serves as the top- level component for all UVM components provides phasing control for all UVM components, and other global services.

Constrained-Random Stimulus

Aside from the complexity of the test bench, when the design under the test is relatively complex and has many inputs and outputs, generating all possible inputs conditions can be practically impossible. For example, consider a relatively complex digital system with 45 inputs that will have approximately 35 trillion possible input conditions (test vectors). Assuming a computational resource that can assert 1,000,000 test vectors per second, it will take more than a year to test and verify all possible input combinations of that system! Of course, that cannot be the solution for verifying designs with that level of complexities. One way to overcome this issue is to apply random stimulus (test vectors).

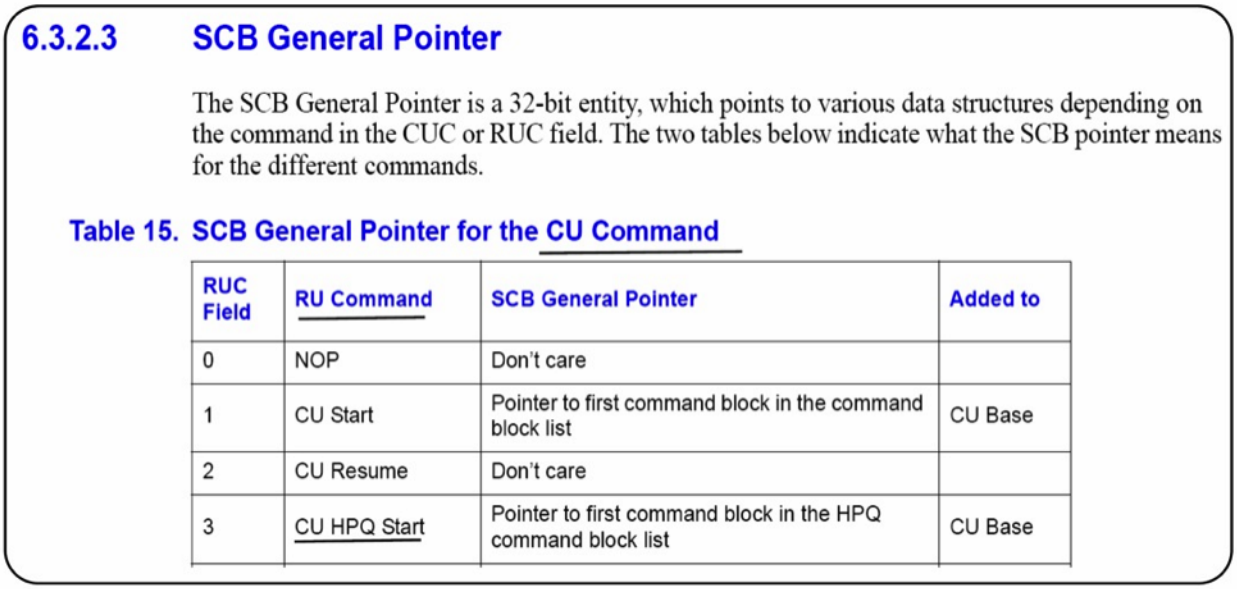
Applying random stimulus has two very significant advantages. The first one is that a random stimulus is great for spotting unexpected bugs, because it makes it possible to explore the entire state space of the design without any selective bias that can be imposed by a human test writer. Secondly, due to the random nature of the test vectors,

it is possible to run parallel computing and utilize compute farms to speed up the verification time.

Needless to say that purely random stimulus would defeat the purpose. It is up to the verification engineer to add relevant constraints and put the required amount of stimulus for each particular part of an elaborate design. Constrained-random stimulus is one the great supported features that SystemVerilog offers. In the process of constraining the stimulus, you may want to put more stress on particular input values and shape the stimulus that are randomly fed into the most sensitive part of the design in order to push the design into corner cases that need more verification. You can add more redundancy for particular input values to verify the reliability of the system over a large number of repetitions.

Formalization of HW/SW Interface Specifications

Most of the hardware/Software interface specifications are in English. Since English does not have formal semantics, these specifications can have ambiguities and inconsistencies. An example of such is presented in Figure 1 [1]. As can be seen, the CU HPQ Start command is not defined or even mentioned anywhere in the document. Another problem with this example is inconsistency of the content of the table and its title (underlined). In this document, RU stands for Receive Unit and CU stands for Command Unit.



6.3.2.3 SCB General Pointer

The SCB General Pointer is a 32-bit entity, which points to various data structures depending on the command in the CUC or RUC field. The two tables below indicate what the SCB pointer means for the different commands.

Table 15. SCB General Pointer for the CU Command

RUC Field	RU Command	SCB General Pointer	Added to
0	NOP	Don't care	
1	CU Start	Pointer to first command block in the command block list	CU Base
2	CU Resume	Don't care	
3	CU HPQ Start	Pointer to first command block in the HPQ command block list	CU Base

Figure 1 An example of issues with none-formalize HW/SW interface specification sheet (From Intel Ethernet controller document) [1].

Studies have been done which suggest that we can formalize HW/SW interface specification schemes [1]. Having such formalized documents can reduce the implementation time and errors. Also if the verification engineer

has the required skills, it is a good practice to write scripts that optimally automate the process of exporting the formalized specs into an RTL model (automatic implementations of interfaces). Writing such scripts for an inexperienced engineer might not save significant time, however the end result would be company-wide consistent and error-free implemented interfaces which can save significant time in long run.

Where to find reliable resources?

One of the safest way to get familiarized with UVM and SystemVerilog language is to check the available UVM manuals and user guides[2]. Also we encourage the novice engineer to subscribe to available online courses that help them to adopt UVM standards into their design. One example of online courses are provided by Mentor Graphics [3]. Of course access to high quality course material would not be granted free of charge.

Engineers can learn from the Internet, but as is with any sufficiently complex system, there are several ways of creating verification test benches and doing verification using UVM. Many of the methods cause problems downstream, for example, some groups of users prefer to use Virtual Sequencers and Virtual Sequences, and others simply use virtual sequences and subsequences. Another example, some groups heavily depend on the SV Macros. Others discount it. Which methodology is better within the UVM framework can only be determined by a qualified instructor or experienced peers in the verification team.

Note that when seeking training on UVM, by using an experienced teacher with years in design and verification, the user is sure to get seasoned advice and comprehensive training on UVM techniques.

Having an experienced verification engineer to teach you the necessary skills and techniques to productively implement the test benches with UVM is like finding a gold mine; however, it is not in their best interest to spend most of their time teaching instead of doing their jobs. That is why it is quite hard to find expert peers in companies that are willing to dedicate a substantial portion of their time teaching the basics of UVM to new engineers.

Professional trainers, assuming they are qualified, draw upon an extensive array of experience so that they can for-warn the students about the potential pit-falls in utilizing certain methods. A good teacher will know how to teach, in other words, s/he will know what issues cause problems for beginners. This can save a lot of time in the long run.

Avoid Mistakes and Speed-Up the Process

As the complexity of the Design Under Test (DUT) increases, the possibility of making mistakes in generating the class-based testbench, which is complying with UVM standards, will rise. Note that if there is an error in your design, the verification code (test bench) should spot it; however, if there is a mistake in the verification code, it might hide behind some of the design flaws and never get uncovered.

Generally there is no automated verification test bench for the verification test bench itself, therefore, any mistakes in implementing the verification test bench are intolerable.

Reusing verification components is a principal objective of UVM. By having a modular verification environment where each configurable component has defined responsibilities. The configurable component can be easily reused by simply reconfiguring it for a different application. If the consistency between different applications (in the company) is preserved, such reusable components can reduce errors and save significant verification time.

The architecture of UVM has been designed to support modular and layered verification environments, where verification components at all layers can be reused in different situations. Low-level driver and monitor components can be reused across multiple designs-under-test. The whole verification environment can be reused by multiple tests and configured top-down by those tests. Finally, test scenarios can be reused from application to application. This degree of reuse is enabled by having UVM verification components configured in a very flexible way without modification to their source code. This flexibility is built into the UVM class library.

Some bugs are caused by errors during MANUAL implemention of the interfaces from non-formalized HW/SW specifications sheets. EDA engineers developed tools that can automatically create the test bench templates that conform to UVM standards. These generators are key to increase the productivity of verification engineers, reduce errors and increase code conformity.

Note that, it is very common to auto generate code from same input. These code generators work great if the generated code can be used as-is without any manual modification. A code generator will be successful, if it is not restrictive on the users. If the user needs to modify the generated file manually or if the code generator does not have the flexibility of changing the output from the source, then the code generator is virtually unusable.

TIP

The best way to approach the verification process is to start with simple directed (non-random) tests to bring up the design, then move to random tests to explore the state space in a broad fashion and flush out as many bugs as possible with minimum human effort devoted to test writing. This will typically achieve less than 100% functional coverage, and the remainder of the verification process is spent defining a series of tests, each of which constrains and shapes the random stimulus in a different way to push the design into corner cases.

Such EDA tools are the result of many tests and has been approved by many customers, and therefore the generated test bench are fully optimized. Applying such tools not only will save the verification engineering significant time and headaches, they can also deploy UVM in a company-wide, consistent and repeatable manner. Applying EDA tools make it easier to understand complex test benches by analyzing them, automatically create the necessary build functions and finally produce the required documents in no time.

Concluding Remarks

If you are working in a relatively small company, the best way to start is to sign up for high quality online courses and materials that can boost your knowledge and educate you about the basics of the UVM framework. For larger companies, it is highly advised to schedule one of the company's course for improving the verification team's UVM knowledge.

AGNISYS
ACCELERATING DESIGN VERIFICATION

Purchase a tool that can automatically generate registers for the design and a second tool for DV that streamlines the use of UVM libraries. Tools like this, help you to decrease the time for UVM based verification. However, both help in a different ways!

The first, is a tool that addresses code generation in a proper way:

- The output from tools like IDesignSpec™ do not need to be modified manually. For example, it creates a register model class that can be used as-is.
- User can specify properties and code snippets that will change the output, for example, hdl_path, coverage and constraints properties.

Purchase a second tool that enables users to create correct by construction code. It guides the user and alerts him/her about the issues that will come up later. It saves time because no time is wasted in debugging code. SV/UVM has many guidelines, which, if not followed, cause problems downstream. For example, many bugs would not be discovered until the compiler stage, and if it catches the error, typically the error message is gibberish yielding a very tedious and time consuming debug session.

There are always some engineers are hesitant to adopt the features that are provided by automatic and assistive CAD tools. Some of the reasons for them to resist are:

1. They may be used to doing it the old way and resistant to change.
2. They may have questions about the stability of the solution.
3. They might not even know such tools exist.

Many companies are offering similar tools to improve the productivity and reduce or eliminate the mistakes during the implementation and verification phases. Some of these tools are too difficult to adopt. . To avoid such tools that are written by inexperienced engineers who don't have deep knowledge of verifications using UVM environments, there is one rule of thumb. Always look for companies that aside from design automation tools offer related training programs that demonstrate their depth of knowledge regarding these topics.

Agnisys Inc. is a pioneer in design automation tools for design and verification of IPs, SoCs, ASIC designs and FPGA based designed. This company offers a wide range of topics and materials in the format of on-site or at their locations. The instructors are all experienced design and verification engineers. The SystemVerilog courses are one of the categories that can considerably accelerate the verification engineering team in adopting and fully utilizing the features and reusable verification components that are available in the latest version of UVM. For more information, you can visit the website provided in [4].

This company also offers award-wining software called IDesignSpec that can automatically create the UVM implementation of register maps that are created from design specification sheets. That can save considerable

time by eliminating the error-prone task of implementing interfaces in System Verilog from register map specifications. Some of the benefits of this software are:

- Create synthesizable code for registers.
- Get a jump-start for Device Driver, Firmware and application software development.
- Automatically create documentation for customers and Tech-Pubs.
- Supports Architecture, Design, Verification, Diagnostics, Firmware, Application Software and Documentation groups.

Another useful tool from Agnisys Inc is DVinsight™. With this program, you can streamline SystemVerilog and UVM code generation. It is a smart editor that checks your code as you write it while providing useful feedback. DVinsight is an excellent tool in the toolbox of experts and novice Verification engineers. Code created in DVinsight requires very little debugging because it prevents bugs from germinating in the first place by highlighting them while the engineer is coding. This product accelerates design

verification and leads to higher quality code in less time.

References

[1] J . Li; F. Xie et al," Formalizing hardware/software interface specifications," 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp.143,152, 6-10 Nov. 2011.

[2] Available online at: http://www.accellera.org/downloads/standards/uvm

[3] Available online at: https://verificationacademy.com/cookbook

[4] List of training services available from Agnisys Inc. is available online at: http://agnify.com/our-services/training-services