Original Research Paper

Volume - 7 | Issue - 5 | May - 2017 | ISSN - 2249-555X | IF : 4.894 | IC Value : 79.96

Engineering

An Overview of Unified Verification Methodology

P. N. Jayanthi	Assistant Professor, Dept. of Electronics and Communication Engineering, RVCE, Bengaluru, India
Pooja Ganesh	Student, Dept. of Electronics and Communication Engineering, RVCE, Bengaluru, India
Nikhil Guptha	Student, Dept. of Electronics and Communication Engineering, RVCE, Bengaluru, India

ABSTRACT

Abstract-In today's markets, industries have to focus on quality of their products to retain their competitive edge. It is a well-known fact that about one third of the total cost in the development of a new chip is devoted to hardware debugging and testing. Design verification is an important step to achieve reliability. Even a small defect in any part or block of a SoC can lead to improper functionality of the chip. Most of the existing verification methodologies involve providing test vectors as input and this may not cover all test cases in an exhaustive manner. Hence, a novel approach that involves randomization of test vectors proves to be a solution in this case. This paper provides the unified verification process and its importance. And the different aspects of verification are discussed for a given chip.

KEYWORDS :

I. INTRODUCTION

The design of a SoC begins with the definition of its specifications. This stage involves the definition of the features and functionalities of the chip and is hence is very important. The design at the macro and micro architectural level are derived from specifications mentioned in the previous step. The specification normally includes the required range of values for speed, size and power consumption of the chip in question. Once verified that the architecture meets the required specifications, the micro architecture can be drafted which helps in the transformation of the architecture concept to a design implementation, which is coded into a synthesizable RTL.

Next comes the process of verification. The goal of any hardware design is to create a device which performs a particular task based on a design specification. The verification process ensures that the design is an accurate representation of the specification. Tests which simulate real world situations are made to run on the high level representations of the design and in the case the design does not perform as expected, bugs appear which must then be rectified. The verification engineer is not responsible for verifying the behaviour of the device when not used for its original purpose but it is a good practice to determine where those boundaries lie.

The process of verification goes hand in hand with the design creation process. A designer must read and interpret the specification of the required block and replicate his interpretation of the logic into a machine-readable form, usually using a high level design code (RTL). To achieve this task, the designer must understand the different input signals and their respective formats, the transformation functions, and similarly the different output signals and their respective formats. A verification engineer must also read the hardware specification and create a corresponding verification plan. This is followed by test bench creation after which test cases are built and run on the RTL to show that it correctly implements the specified features. Redundancy makes sure that the interpretation is correct and can be achieved by having many people perform the same interpretation.

The verification process can be simplified by detecting and fixing bugs created by running specific tests on DUTs at the block level, which are generally created and owned by a single person. The next place to look for discrepancies is on the interfaces. In order to simulate a certain block, tests need to be created which generate stimuli from all the surrounding blocks which would be present in the physical implementation of the chip. The only benefit is that these simulations run very fast. However, bugs may be found both in the design and testbench.

The purpose of a testbench is to help carry out the verification plan. It is a virtual representation the real time environment of the design. The testbench performs the following functions.

- It generates the necessary stimulus required by the DUT
- It applies the stimulus to the DUT
- It captures the responses from the
- It checks the correctness of the responses
- The verification progress is then measured with respect to the verification plan

As design blocks start to integrate, they start to stimulate one other, thus reducing the required amount of work. Even though they may run a slower pace, this method of verification tends to produce more bugs. Verification at the highest level of the DUT ensures that the entire system is tested but the process speed takes a hit and simulation performance is greatly reduced. At this level sophisti cated tests can be run that have the ability to make the DUT execute multiple operations concurrently. Once verified that the DUT performs the required functions correctly, the engineer needs check operation of the same DUT when there are errors. Error injection and handling are the most challenging part of the verification process and as the design abstraction gets higher, so does the verification challenge. It can never be proven there are no bugs left, and so new verification tactics must be come up with.

This paper provides an overview of the different steps involved in the verification testbench. RTL verification is first introduced following which an insight is provided about gate level simulations. Finally, the process of logic equivalence checking is looked at.

I. UNIVERSAL VERIFICATION METHODOLOGY

A test bench based verification environment automatically randomi zed values for the chip inputs under control of certain constraints and the checks the results suitably. All simulation based verification suffers from the issue of insufficient test vectors to exhaustively test the whole design. One way to address this issue is using constrained random stimulus. The use of random stimulus provides significant benefits. Random stimulus is great for uncovering unexpected bugs. Many verification methodologies have been created to develop

constraint based-random value generating test benches in a uniform fashion and permit limited reuse of test bench components. The best known of these is the Universal Verification Methodology shown in figure 1.

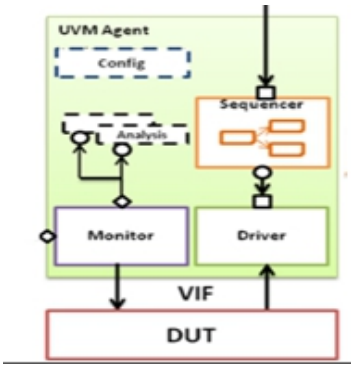

Fig 1. Block diagram of UVM methodology

The UVM methodology uses Transaction-Level Modelling (TLM) to communicate between the Device Under Test (DUT) and the testbench. A UVM class library brings automation to the System Verilog language [7]. The UVM library defines a set of base classes and utilities that facilitate the design of modular, scalable, reusable verification environments.

The major components of a UVM test bench include the uvm_agent, uvm_monitor, uvm_driver, uvm_scoreboard and the uvm_sequen cer. The uvm_driver is inherited from uvm_component and is responsible for driving the DUT[5]. It has transactions with the sequencer to achieve randomized testing vectors.A monitor is an entity that samples the DUT signals through virtual interface and converts the signal activity to transaction level. It samples DUT signals but does not drive it. Monitors usually consist of analysis ports. An agent is a combination of driver, monitor and sequencer and may be active or passive in nature. Active agents generate stimulus and drive the DUT and hence consist of driver, monitor and sequencer whereas a passive one doesn't drive the DUT.

The virtual sequencer along with the agent constitutes the reusable test bench. Figure 2 helps to represent this testbench.

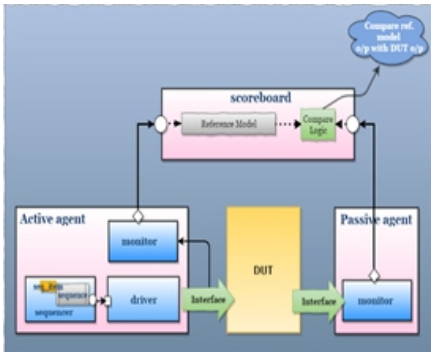

Fig 2. UVM Environment

UVM phases are required for carrying out events in a synchronized and sequential order. The major UVM phases are build, connect ,end of elaboration, start of simulation, run, extract, check and report phase. They are all derived from uvm_component [6].

The build, connect and end-of-elaboration come under the category of build phases. Start of simulation and run come under run phases and the rest fall under the category of clean up phases. The different UVM phases are as follows:

- Build: The build phase follows a top down approach and is used to construct the testbench components.
- Connect: The connect phase follows a bottom up approach and is used to connect TLM ports of components.
- End_of_elaboration: The end of elaboration phase is used to make any final adjustments to the structure, configuration or connectivity of the testbench before simulation starts.
- Start_of_simulation: The start of simulation phase is used for printing testbench topology or configuration information.
- Run: The run phase is used for stimulus generation, driving, monitoring and checking.
- Extract: The extract phase is used to retrieve and process information from scoreboards and functional coverage monitors.
- Check: The check phase is used to check that the DUT behaved correctly and to identify any errors that may have occurred during execution
- Report: The report phase is used to display the results of the simulation or to write the results to file
- Final: The final phase is used to complete any other outstanding actions that the testbench has not already completed

I. GATE LEVEL SIMULATIONS

An important part of the verification cycle is the running of simulations. Simulations can be performed at varying levels of abstraction, i.e., the transistor level, the gate level and the register transfer level. Most companies tend to sign off the design cycle after running simulations only at the RTL level. Lately, however, there has been an increasing trend in the industry, for reasons that will be discussed, to run gate level simulations (GLS) before going into the next stage of the VLSI design flow. GLS is a major step in the verification cycle [2]. GLS provide a variety of errors, the most common and most challenging of which is the "x" propagation debug. This error can come about for a variety of reasons such as timing violations, uninitialized memory and non-resettable flops.

A. Necessity of gate level simulations

GLS is carried out in order to:

- Verify the critical timing paths in asynchronous designs that is usually not performed in STA.
- Validate the constraints used in the process of STA and EC.
- Verify black boxes which may be present in EC.
- Verify the reset operation of the given design.
- Verify the power up operation of the given design.
- Verify that the design does not unintentionally depend on the initial conditions under which it is supposed to operate.
- Verify low power structures which are cannot be added in RTL and added only during synthesis phase.
- Verify that the digital and analog netlist have been successfully integrated.
- Verify DFT structures which are not present in RTL and which are added during synthesis.

B. Gate Level Simulation Flow

The gate level simulation flow is represented in figure 3. Given both a standard cell library and the required design constraints, the high level design (RTL) can be converted to its equivalent gate level representation, i.e., its gate level netlist. This process is known as logical synthesis. A standard cell library generally contains basic logic gates like AND, OR, NOT etc. and macro cells like adders and flip flops.

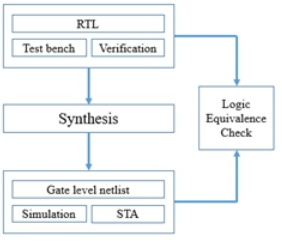

Fig 3. Gate Level Simulation Flow

A. Static Timing Analysis

Static Timing Analysis is a method used to determine as to whether a circuit has met the required timing constraints without having to

perform simulations. It is much faster than timing-driven, gate-level simulations. Proper circuit functionality is not checked and vector generation is not required. Static Timing Analysis involves the following steps:

- The first step is to break the circuit down into a set of timing paths.
- The next step is to calculate the delay of each timing path. This delay is calculated by summing both the net delays and cell delays along the given path.
- Finally, the path delays are checked to see if the required timing constraints have been met.

1) Setup Time and Hold Time

Data which must be sampled by a flop on a given clock edge must be stable for a minimum period of time both before and after the respective clock edge in order to be latched correctly. The minimum amount of time before the edge for which the data must be stable is more commonly call the setup time. Similarly, the minimum amount of time after the clock edge for which the data must be stable is more commonly known as the hold time. Any violation with respect to these timing values leads to incorrect data to be captured by the flop and is known as a timing violation. The respective violations are called the setup and hold time violations. The equations for calculating the setup and hold times is discussed below. The equations for setup time and hold time can be derived from figure 4 [3]. The figure shows two flops talking to one another, where the first flop acts as the origin of the data which must be captured by the second flop. The equations for setup and hold time which will be defined will be true for any flop in the design and is not restricted only to the scenario shown in figure 4.

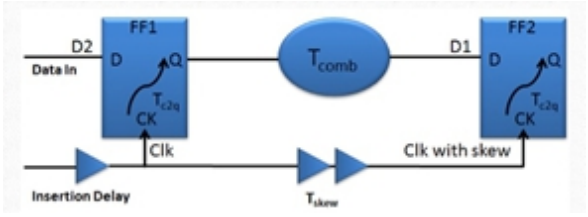

Fig 4. Two talking flops scenario

At time zero, the first flip flop (FF1) processes the data at its input port, D2. The time taken for D2 to propagate to the second flip flop (Ff2), taking the positive clock edge at FF1 as reference, can be represented as the sum of the delay for the corresponding output to appear at the output of FF1, T_{c2q}, and the delay of the combinational logic, Tcomb. For FF2 to successfully latch this data, D2 has to be maintained at D of FF2 for T_{setup} time before the clock tree sends the next positive edge of the clock to FF2. Hence to meet the setup time, eqn. 3.1 must be satisfied.

$$T_{c2q} + T_{comb} + T_{setup} \leq T_{clk} + T_{skew} \quad (3.1)$$

Now, in order to make sure the hold time is met at the first flop, the data D1 must remain stable for some time (T_{hold}) after the positive clock edge.

Equation 3.2 must be satisfied to ensure that the hold time is met.

$$T_{c2q} + T_{comb} \geq T_{hold} + T_{skew} \quad (3.2)$$

The timing diagram in figure 5 [3] helps to provide a figurative representation of the equations discussed above.

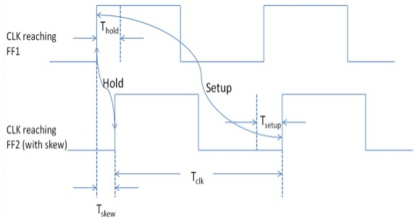

Fig 5. Setup and Hold timing diagram

As seen above, setup time violations can be corrected with changes in the clock period, or in other words, clock frequency. However hold time violations cannot be corrected in this manner. Hold time violations indicate design issues and must be corrected from the design perspective before being verified again.

1) Standard Delay Format File

SDF file is used to represent circuit delays. SDF or Standard Delay Format is an IEEE specification [4]. The STA tool is used to generate SDF files which will contain both interconnect and cell delay [4]. The SDF file is created for different variations in process, voltage and temperature (PVT). Process variations account for deviations in the semiconductor fabrication process. These variations are caused due to uniformity not being maintained during the process of during the diffusion of the impurities. This leads to change in sheet resistance and transistor parameters such as threshold voltage. ICs are generally produced in lots of 50 to 100 wafers with approximately 100 dice per wafer. Each lot can differ in terms of their electrical properties. Slight differences can even exist among wafers in a single lot, even in a single manufactured chip. As a result transistors end up having different transistor lengths in a chip. This in turn affects the propagation delay to vary in a chip, as a smaller transistor is faster and vice versa.

The design's supply voltage can vary from the designed ideal value during daily operation. The delay of a cell depends on the saturation current of the cell which in turn depends on the supply voltage. The power supply varies throughout the core and hence so does the propagation delay. The voltage drop across the chip occurs due to internal resistance as well as the self-inductance in supply wires. A higher voltage leads to higher operating frequencies which in turn makes the cell faster.

Temperature variations are unavoidable in the everyday operation of an electronic device. While a chip is operating, the temperature can vary throughout the chip. This is caused due to power dissipation from MOS-transistor operation which can be attributed to switching, short-circuit and leakage power consumption. Temperature has a direct effect on electron and hole mobility. For temperatures above -50C, the mobility in Si decreases with increased temperature. Decrease in mobility leads to an increase in the propagation delay and hence an increase in temperature leads to higher propagation delay. The different combinations of PVT variations represent "corners". The design must provide the required performance for all PVT variations, i.e., the design must pass on these corners. There are many combinations of different PVT values possible. However, specific corners are chosen which would cover all such combinations.

D. Annotation

The process of adding delays defined in a given source to the various cells in a netlist during netlist simulation is called Back Annotation. This source of delays is the SDF file described above. This is represented in figure 6.

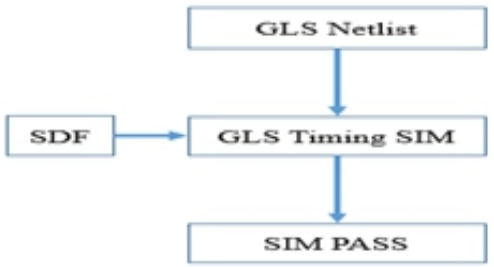

Fig 6. Back Annotation

The delay values for each cell are initially determined from simulations. But the delays determined are not the actual delays of the cells. Rather they are the delays the cells would display when they are used in different environments. These environments can have varying locations, varying loads and varying fan-in. Two similar cells placed at different locations on the core can have significant differences based on the factorsjust mentioned above. In order to simulate the different cells at these delay values, SDFs must be written by an EDA tool. This tool defines the delays of each cell instance in the netlist depending on these factors. During simulations each cell in the netlist has its corresponding delay read from the SDF file. Before annotation is carried out with SDF files, a zero delay gate level simulation is carried in order to check whether the simulation is working correctly with the netlist or not. Ideally, without an SDF there should not be any issue, but 'X' propagation can still occur due to gate delay and sync cells.

III. LOGIC EQUIVALENCE CHECK

Once verified that the synthesized netlist meets the required timing requirements, it must be ensured that the netlist created still maintains the same functionality as our RTL code. A logic synthesis tool can help to ensure that the netlist is logically equivalent to the RTL source code. This process is called as logic equivalence checking. There are several inputs required for LEC but some are very crucial and sometimes overlooked. They include:

- Robust black box matching
- Correct scan constraints
- Port mismatch between timing models and behavioural code.

IV. CONCLUSION

Verification is the most important and the most time-consuming part of chip design. New chip designs coming out within months, verification needs to be done faster to achieve its goal. Test benches designed to be robust and flexible at the same time to address this issue. Constrained random testing was created to reduce the amount of time required and run directed tests. To improve portability and interoperability for RTL verification, a standard called UVM was created. The paper introduced the UVM standard and its flow. Once RTL verification is carried out, the timing requirements for the DUT under different conditions must be verified. This verification is carried with the help of GLS. Finally, once the timing requirements of the gate level netlists is verified, these netlists must be checked for functional equivalence with the RTL. This is carried out through the process of LEC.

REFERENCES

[1] Harry Foster, "Trends in functional verification: a 2014 industry study", DAC, 2015.
[2] Ankit Khandelwal, Abhinav Gaur and Deepak Mahajan. "Gate level simulations: verification flow and challenges", Internet: http://www.edn.com/design/integrated-circuit-design/4429282/Gate-level-simulations--verification-flow-and-challenges, Apr. 12, 2017.
[3] Deepak Kumar Behera and Karthik Rao C.G. "Equations and Impacts of Setup and Hold Time", Internet: http://www.edn.com/design/systems-design/ 4392195/ Equations-and-Impacts-of-Setup-and-Hold-Time, Apr. 21, 2017.
[4] Sini Mukundan. "Standard Delay Format", Internet: http://vlsi.pro/standard-delay-format-sdf-file/, Apr. 12, 2017.
[5] Ke Wang,Hong-Bo Zhu,Wei-Gou Lu,Zhen-An Liu,"A testbench research based on UVM for ABCStar",Published in Real Time Conference (RT), 2016 IEEE-NPSS, 6-10 June 2016.
[6] Wei Ni,Jichun Zhang, "Research of reusability based on UVM verification", ASIC (ASICON), 2015 IEEE 11th International Conference on 21st July 2016.
[7] Hao Wang, "The Preamp chip's verification and analysis based on UVM Methodology", 2014