# DUT Verification Through an Efficient and Reusable Environment with Optimum Assertion and Functional Coverage in SystemVerilog

Deepika Ahlawat
VLSI Group
Department of Electrical, Electronics & Communication Engineering, ITM University,
Gurgaon, (Haryana), India

Neeraj Kr. Shukla
VLSI Group
Department of Electrical, Electronics & Communication Engineering, ITM University,
Gurgaon, (Haryana), India

*Abstract*—**Verification is the most integral part of chip manufacturing and testing and is as important as the designing. Verification provides with the actual implementation and functionality of a Design under Test (DUT) and checks if it meets the specifications or not. In this paper, a communication protocol has been verified as per the design specifications. The environment so created completely wraps the design under verification and observes an optimum functional and assertion based coverage. The coverage so obtained is 100% assertion based coverage and 83.3% functional coverage using SV (SystemVerilog). The total coverage so obtained is 91.66%.**

***Keywords—Assertions; Coverage; Environment; Mailbox; Randomization;SystemVerilog; Threads; Transactions***

## I. Introduction

With increasing complexity of the input constraints and the need for better control of the statistical distribution, imperative test benches are being replaced by more declarative specification methods using languages such SystemVerilog [1].

### A. *Need of Verification*

Exponentially increasing complexity of chips particularly SOCs made verification more challenging. Major portion of development time (~70%) of a complex SOC is spent on verification. Reducing verification effort or time spent on verification has a strong impact on Time-to-Market (TTM). In order to satisfy such growing complex verification needs powerful verification languages and verification methodologies are employed [2].

In general IP Verification requires in depth verification with coverage based and constraint random simulation technique, which needs an advanced test bench equipped with various components such as coverage monitors and scoreboards. But if an IP was fully verified before and has a minor design change, it is not necessary to verify all features in detail. A few directed cases and simple checkers might be sufficient [3].

Except for simple cases, the behavioral specification of hardware designs is mostly incomplete, leaving the design's response to many input stimuli undefined. During verification, unspecified inputs must be excluded from examination to avoid undetermined or spurious erroneous behavior. In a simulation-based verification setting, the concept of a "test bench" is applied to specify valid input sequences as well as the expected design responses for them [4].

### B. *Need of System Verilog*

SV is built on top of Verilog 2001. SV improves the productivity, readability, and reusability of Verilog based code. It brings a higher level of abstraction to design and verification. The language enhancements in SV provide more concise hardware descriptions, while still providing an easy route with existing tools into current hardware implementation flows[5].

SV provides a complete verification environment, employing Directed and Constraint Random Generation, Assertion Based Verification and Coverage Driven Verification. These methods improve the verification process dramatically. It also provides enhanced hardware-modeling features, which improve the RTL (Register Transfer Level) design productivity and simplify the design process.

*Advantages of Using SV*

1) *SV was adopted as a standard by the Accellera organization, and is approval by IEEE. These ensure a wide embracing and support by multiple vendors of EDA (Electronics Design & Automation) tools and verification IP's, as well as interoperability between different tools and vendors [5].*

2) *Since SV is an extension of the popular Verilog language, the adoption process of SV by engineers is extremely easy and straightforward. SV enables engineers to adopt a modular approach for integrating new modules into any existing code. As a result, the risks and costs of adopting a new verification language are reduced.*

3) *Being an integral part of the simulation engine, eliminates the need for external verification tools and interfaces, and thus ensures optimal performance (running at least x2 faster than with any other verification languages) [5].*

4) *SV brings a higher level of abstraction to the Verilog designer. Constructs and commands like Interfaces, new Data types (logic, int), Enumerated types, Arrays, Hardware-specific always (always_ff, always_comb) and others allow modeling of RTL designs easily, and with less coding.*

5) *SV extends the modeling aspects of Verilog by adding a Direct Programming Interface which allows C, C++, SystemC and Verilog code to work together without the overhead of the Verilog PLI (Programmable Logic Interface).*

A declarative description of input constraints is significantly easier to develop in terms of avoiding over constraining or under constraining the inputs as well as controlling the desired distribution. It is expressed as a predicate on the design's input variables such that an input stimulus is valid if and only if the predicate evaluates to true. Advanced test benches must handle cases in which the validity of an input stimulus may differ from design state to design state, which makes the constraints dependent on state variables [4].

The paper is organized as follows, after an overview of verification and advantages of using SV, section II describes the DUT taken and gives a brief introspection on its working. Section III discusses the test bench architecture and all the components it comprises of. Section IV describes how the SV environment works and the various phases of the test bench. Section V consists of the simulation results in the form of waveforms and the coverage report based on assertion coverage and functional coverage.

## II. DUT – THE SPI CORE

The serial interface consists of slave select lines, serial clock lines, as well as input and output data lines. All transfers are full duplex transfers of a programmable number of bits per transfer (up to 64 bits). It can drive data to the output data line in respect to the falling (SPI/Microwire compliant) or rising edge of the serial clock, and it can latch data on an input data line on the rising (SPI/Microwire compliant) or falling edge of a serial clock line [6].

Data Transmission

The bus master configures the clock first, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 10kHz–100 MHz [6].

During each SPI clock cycle, a full duplex data transmission occurs [7]:

*a) the master sends a bit on the MOSI line; the slave reads it from that same line*

*b) the slave sends a bit on the MISO line; the master reads it from that same line*

Transmissions may involve any number of clock cycles. When there is no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.

Transmissions often consist of 8-bit words, and a master can initiate multiple such transmissions if it wishes/needs. The master must select only one slave at a time [6].

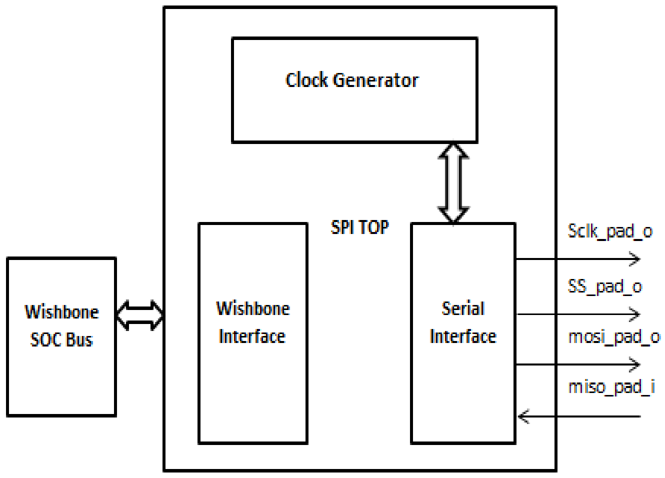


Fig. 1. SPI Architecture [7]

WISHBONE BUS

The Wishbone Busis an open source hardware computer bus, intended to allow parallel communication between the parts of an integrated circuit. This System-on-Chip interconnection architecture is used in order to create a common interface between different IP cores. The Wishbone interconnect is intended as a general purpose interface. As such, it defines a master / slave standard for data exchange between IP core modules, in terms of signals, clock cycles, and high & low levels.

## III. SYSTEMVERILOG TESTBENCH ARCHITECTURE

The testbench architecture has various modules as discussed below. The interconnection between these modules can be seen in figure 2.

### A. *Test Generation*

A Test case is a program block which provides an entry point for the test. The test case generator will provide all the valid test cases to the driver. The test cases are generated by randomizing certain inputs and registers while keeping some fixed.

To perform this type of randomization i.e. constraint randomization a function called random is created [8].

### B. *Driver*

The driver will reset and configure the DUT.

It will call the tasks from test generator and will form a packet in the packet generator module and will unpack the packet in the driver module and implement it on the DUT. The interfaces of the Driver are: clk_i, rst_i, add_i, data_i, sel_i, we_i, stb_i, cyc_i, sclk, miso.

### *C. Monitor*

Monitor will keep track of all the test cases provided to the driver. It will also look at all the signals coming from the DUT. Monitor thus will call the packet in the scoreboard module and compare it with the output from the DUT present in the checker. Hence the duty of monitor is to complete simulation when all cases have been read. It will also generate error message if there is any discrepancy in data out coming from DUT and the teat generator. The interfaces of the monitor are: data_out,int_o, ack_o, ss, err_o, mosi.

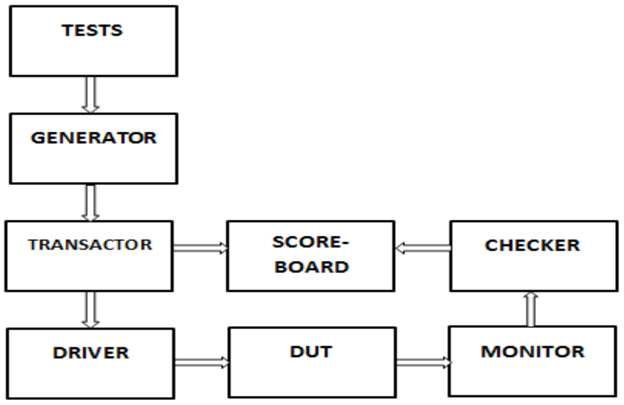


Fig. 2. Verification Flow [8]

### *D. Responder*

Responder is a block which acts as a slave and gives out miso_pad_i to the DUT which is processed or stored or read from the DUT. It is given sclk or slave clock from the DUT and sends miso to the core.

### *E. Scoreboard*

The output from monitor is checked with the expected output. Ihe output genetared by the DUT as observed by the monitor is passed to the scoreboard through mailbox. If the actual output does not match the expected output an error message is generated else if it matches a pass message is displayed.

### *F. Coverage*

Coverage will check the functional coverage of the DUT by the test cases tested by the driver and monitored by the monitor. It will also create an error counter which will show the TEST FAIL and TEST PASS status [8].

## IV. COMPILATION IN SYSTEMVERILOG

Following are the methods which defined in the environment class of the SV testbench[6].

*A. build (): In this method , all the objects like driver, output monitor and mailboxes are constructed.*

*B. reset (): in this method all the signals are put at a known state.*

*C. start (): in this method, all the methods which are declared in the other components like driver, output monitor and scoreboard are called.*

*D. wait_for_end (): this method is used to wait for the end of the simulation. Wait is done until all the required operations in other components are done.*

*E. report (): This method is used to print the results of the simulation, based on the error count.*

*F. run (): This method calls all the above declared methods in a sequence.*

The way the DUT interface with the driver, monitor and responder/slave can be seen in figure 3.

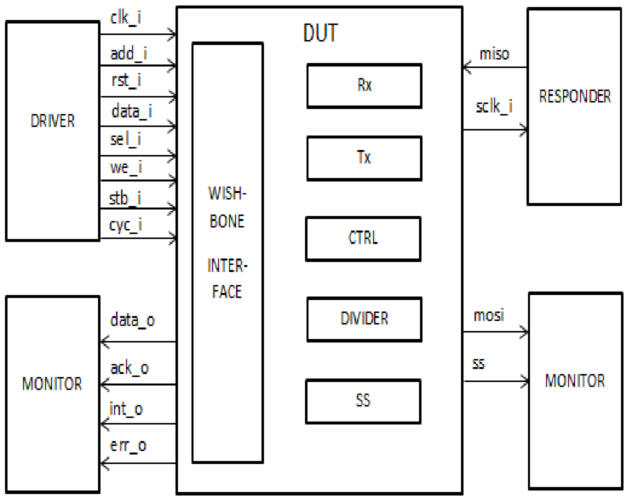
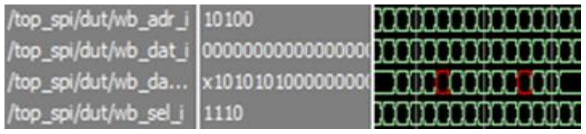


Fig. 3. Architectural overview of the verification modules as implemented in the proposed verification environment

## V. DESIGN SIMULATION

### *A. Randomization*

Random testing is more effective than a traditional approach of directed testing. One can easily create tests that can find hard-to-reach corner cases, by specifying constraints. SystemVerilog allows users to specify constraints in a more compact and declarative way. The constraints are then processed by a solver that generates random values that meet the constraints [5]. The stimuli randomizes are data input, slave select and address input as seen in figure 4.



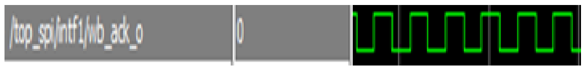
```
/top_spi/dut/wb_adr_i   10100
/top_spi/dut/wb_dat_i   0000000000000000(
/top_spi/dut/wb_da...   x101010100000000(
/top_spi/dut/wb_sel_i   1110
```

Fig. 4. Randomized value of signals

### *B. DUT Signals Generated*

*ack*

The acknowledge output [ack_o] indicates the normal termination of a valid bus cycle.
The ack signal obtained can be seen in the figure 5 below.



```
/top_spi/intf1/wb_ack_o   0
```

Fig. 5. Acknowledgment signal generated

*Sclk*

SCK [sck_o] (figure 6) is generated by the master device and synchronizes data movement in and out of the device through the MOSI [mosi_o] and MISO [miso_o] lines. The SPI clock is generated by dividing the WISHBONE clock [clk_i].

*Miso*

The Master In Slave Out line is a unidirectional serial data signal. It is an output from a slave device and an input to a master device (figure 6).

*Mosi*

The Master Out Slave In line is a unidirectional serial data signal. It is an output from a master device and an input to a slave device (figure 6).

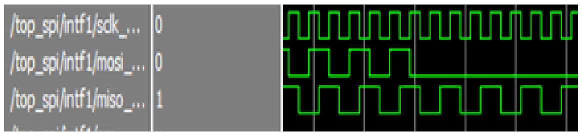

/top_spi/intf1/sclk_... 0
/top_spi/intf1/mosi_... 0
/top_spi/intf1/miso_... 1

Fig. 6. sclk, miso and mosi signals generated

## C. Output Waveform

The output waveform as shown in figure 7, displays all the signals being generated by the DUT. The internal registers are also seen to be crunching data and displaying corresponding outputs through SPI signals.

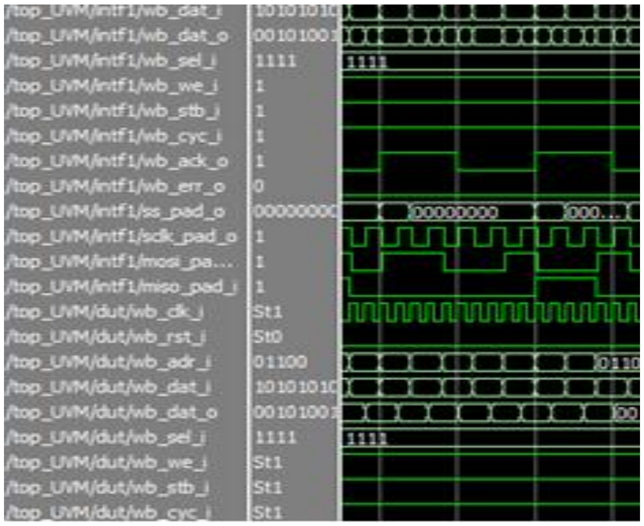

/top_UVM/intf1/wb_dat_i 1010101
/top_UVM/intf1/wb_dat_o 0010100
/top_UVM/intf1/wb_sel_i 1111
/top_UVM/intf1/wb_we_i 1
/top_UVM/intf1/wb_stb_i 1
/top_UVM/intf1/wb_cyc_i 1
/top_UVM/intf1/wb_ack_o 1
/top_UVM/intf1/wb_err_o 0
/top_UVM/intf1/ss_pad_o 00000000
/top_UVM/intf1/sclk_pad_o 1
/top_UVM/intf1/mosi_pa... 1
/top_UVM/intf1/miso_pad_i 1
/top_UVM/dut/wb_clk_i St1
/top_UVM/dut/wb_rst_i St0
/top_UVM/dut/wb_adr_i 01100
/top_UVM/dut/wb_dat_i 1010101
/top_UVM/dut/wb_dat_o 0010100
/top_UVM/dut/wb_sel_i 1111
/top_UVM/dut/wb_we_i St1
/top_UVM/dut/wb_stb_i St1
/top_UVM/dut/wb_cyc_i St1
1111
00000000
000...
0110
00
1111

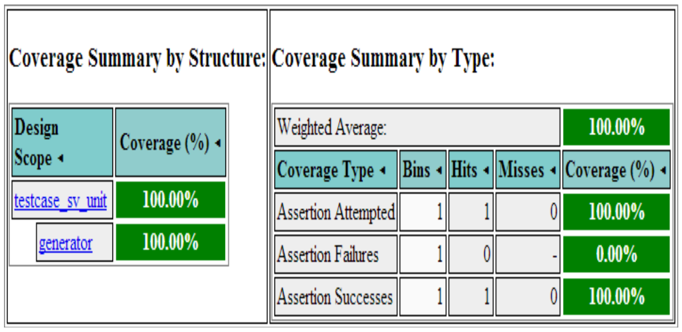
Fig. 7. Output Waveform of SPI Core

## D. Coverage

1) *Assertion Coverage*

Assertions are mechanism or tool used by HDL's (VHDL and Verilog) to detect a design's expected behavior. The assertion fails if a property that is being checked for in a simulation does not behave the way we expect it to or we can say that, the assertion fails if a property that is forbidden from happening in a design happens during simulation.

It helps capturing the designer's interpretation of the specification [5]. The assertion coverage based on randomization function assertion is shown in figure 8.



**Coverage Summary by Structure:**

| Design Scope | Coverage (%) |
|---|---|
| testcase_sv_unit | 100.00% |
| generator | 100.00% |

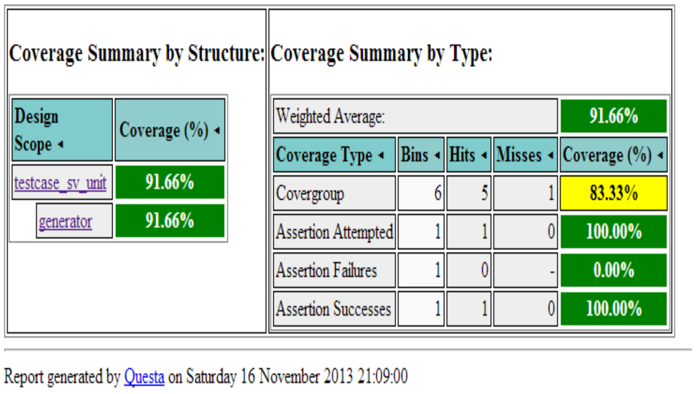
**Coverage Summary by Type:**

| Weighted Average: | | | | 100.00% |
|---|---|---|---|---|
| Coverage Type | Bins | Hits | Misses | Coverage (%) |
| Assertion Attempted | 1 | 1 | 0 | 100.00% |
| Assertion Failures | 1 | 0 | - | 0.00% |
| Assertion Successes | 1 | 1 | 0 | 100.00% |

Fig. 8. Coverage report based on assertion

2) *Total Coverage Percentage*

Total coverage here (figure 9) includes both the assertion based coverage and the functional coverage. The functional coverage is based on the coverpoints of the corresponding covergroup. Bins have been created and have been hit properly to generate functional coverage.



**Coverage Summary by Structure:**

| Design Scope | Coverage (%) |
|---|---|
| testcase_sv_unit | 91.66% |
| generator | 91.66% |

**Coverage Summary by Type:**

| Weighted Average: | | | | 91.66% |
|---|---|---|---|---|
| Coverage Type | Bins | Hits | Misses | Coverage (%) |
| Covergroup | 6 | 5 | 1 | 83.33% |
| Assertion Attempted | 1 | 1 | 0 | 100.00% |
| Assertion Failures | 1 | 0 | - | 0.00% |
| Assertion Successes | 1 | 1 | 0 | 100.00% |

Report generated by Questa on Saturday 16 November 2013 21:09:00

Fig. 9. Coverage report including functional and assertion based coverage

## VI. CONCLUSIONS

The code for environment has been simulated. The outputs from DUT have been observed. Environment contains the instances or the objects of the driver, monitor, scoreboard and the DUT. The task performed by the monitor, driver and scoreboard is called along with the mailboxes which contain the received and sent information in the form of randomized packets. The mailbox implemented to carry the packets shows results after every transaction. The environment so created completely wraps the design under verification and observes an optimum functional and assertion based coverage. Bins have been created based on the constraints and 85-100% functional coverage has been obtained on them. The coverage so obtained is 100% assertion based coverage and 83.3% functional coverage using System Verilog. The total coverage so obtained is 91.66%.

## ACKNOWLEDGMENT

The authors are grateful to their respective organization for help and support.

## REFERENCES

[1] Sutherland S, Davidmann S, Flake P, "SystemVerilog for Design: A Guide to Using SystemVerilog for Hardware Design and Modeling," Norwell, MA: Kluwer Academic Publishers, 2003.

[2] SudhishNaveen, BR Raghavendra, YagainHarish, "An Efficient Method for Using Transaction Level Assertions in a Class Based Verification Environment," International Symposium on Electronic System Design,pp.72-76, 2011

[3] Yun Young-Nam, Kim Jae-Beom, Kim Nam-Do, Min Byeong, "Beyond UVM for practical SoC verification," SoC Design Conference (ISOCC), pp. 158 – 162, Nov 2011

[4] Welp Tobias, Kitchen Nathan, and Kuehlmann Andreas, "Hardware Acceleration for Constraint Solving for Random Simulation,"IEEE Transactions On Computer-Aided Design of Integrated Circuits And Systems, vol-31, No. 5, May 2012

[5] [Online]Available: http://www.systemverilog.in/systemverilog_introduction.php

[6] K.Aditya,M.Sivakumar,FazalNoorbasha, T.PraveenBlessington, "Design and Functional Verification of A SPI Master Slave Core Using System Verilog," International Journal of Soft Computing and Engineering (IJSCE), vol-2, Issue-2, May 2012

[7] SrotSimon, "SPI Master Core Specification,"Rev. 0.6, March 15, 2004

[8] RaoAbhiram. *What is SystemVerilog?*[Online] Available:http://electrosofts.com/systemverilog/introduction.html

## About The Authors

DeepikaAhlawat,completed her B.Tech in Electronics and Communication Engineering from Gurgaon College of Engineering for Women, Gurgaon in 2012. She is now pursuing her Master of Technology (M.Tech) in VLSI Design at ITM University, Gurgaon. Her interest includes Digital Design, ASIC Design, VLSI Testing and FPGA prototyping.

Dr. Neeraj Kr. Shukla (IEEE, IACSIT, IAENG, IETE, IE, CSI, ISTE, VSI-India), an Asst. Professor in the Department of Electrical, Electronics & Communication Engineering, ITM University, Gurgaon, (Haryana) India. He has received his M.Tech. Degree in Electronics Engineering and B.Tech. Degree in Electronics & Telecommunication Engineering from the J.K. Institute of Applied Physics & Technology, University of Allahabad, Allahabad (Uttar Pradesh) India in the year of 1998 and 2000, respectively. His main research interests are in Low-Power Digital VLSI Design and its Multimedia Applications, Digital Hardware Design, Open Source EDA, Scripting and their role in VLSI Design, and RTL Design.