Performance Analysis of Verilog Directed Testbench vs Constrained Random SystemVerilog Testbench

Deepika Ahlawat
ITM University, Gurgaon,
(Haryana), India

Neeraj Kr. Shukla
ITM University,Gurgaon,
(Haryana), India

ABSTRACT
SystemVerilog is the emerging language of choice for modern day VLSI design and verification. SystemVerilog (SV) brings a advanced level of abstraction to the system being modeled. The advanced constructs it utilizes its OOP capability make it stand apart from other verification languages. In this paper we will be analyzing the performance of SV testbench over Verilog testbench, using well defined comparison parameters tested against an actual IP design block, along with other features of the SV language.

Keywords
Assertions, Coverage, Environment, Mailbox, Randomization, SystemVerilog, Threads, Transactions, Testbench

1. INTRODUCTION
The advantages of SV are quite clear over Verilog for verification. SV has advance concepts, data types, and functionalities over Verilog. It also incorporates the OOP concepts [1]. We will be comparing the performance of both a Verilog and SV testbench in terms of test bench Compilation, Elaboration, Configuration, Running, and Wrap-up. The following sections outline the advantages of SV over the Verilog testbench.

1.1 Verification Overview
The complexity of verification environments has grown exponentially over the years. With the introduction of large multifunctional ASICs, verification needs to keep up with the increasing design complexity of SOCs with the integration of innumerable number of IP blocks with the processing element. Verification begins with a detailed analysis of the design specifications with a well-defined methodology and plan to achieve maximum coverage of verification. Coverage data can be ascertained using various ways: Code coverage, Functional coverage, and Assertion based coverage. The ultimate goal of verification is to boost verification coverage in a short period of time while minimizing verification costs [2].

Our analysis of the performance of SystemVerilog testbenches over traditional Verilog testbenches is organized in the following sections:

Section II highlights the merits of SV over Verilog for in terms of language advancements.

Section III describes the SPI Core Design under Test (DUT) to be used for the analysis.

Section IV describes the verification plan adopted for SV verification for the analysis.

Section V describes the System Verilog testbench details for SPI core.

Section VI describes the Verilog testbench details for SPI core

Section VII describes the comparison parameters to be used for the analysis

Section VIII describes the results of the analysis

Section IX describes our conclusions from the analysis made on both the testbenches.

2. ADVANTAGES OF SV
SV is world's first Hardware Verification Language (HVL). It has features for RTL design, assertions and verification. SV 2009 replaces Verilog for verification. SystemVerilog enhances extended and new constructs to Verilog-2001, some of them talk about below [3]:

1. Extensions to data types for improved encapsulation and compactness of code and for tighter specification
2. User defined types: enum, struct, union and typedef
3. Loops like for, foreach, while, do while, repeat, forever
4. Enhanced process control like fork, suspend, kill, wait and disable
5. Dynamic arrays – resizable and associative arrays.
6. Enhanced tasks and functions
7. Classes: Object-Oriented mechanism that offers abstraction, encapsulation, and safe pointer capabilities
8. Random constraints generation [4]
9. Interprocess communication synchronization – mailbox, semaphore
10. Cycle-Based Functionality: Clocking blocks and cycle-based attributes that allow for easy maintainability, and promote reusability
11. Assertion mechanism for verifying design objective and functional coverage objective [5].
12. Interfaces to encapsulate communication
13. Functional coverage, assertion based coverage, code coverage
14. Direct Programming Interface (DPI) for clear and efficient interoperation with other languages [3]

3. DUT- AS THE SPI CORE
The serial interface entails of slave select lines, serial clock lines along with input and output data lines. All the transfers are full duplex transfers comprising of a programmable number of bits per transfer. In respect to the falling or rising edge of the serial clock, it can drive the data to output data line. On the rising or falling edge of a serial clock line, it can

drive data on an input data line. It can also receive (transmit) the MSB first or the LSB first [6].

Data Transmission

The SPI Master core comprises of three parts as shown in the following figure:

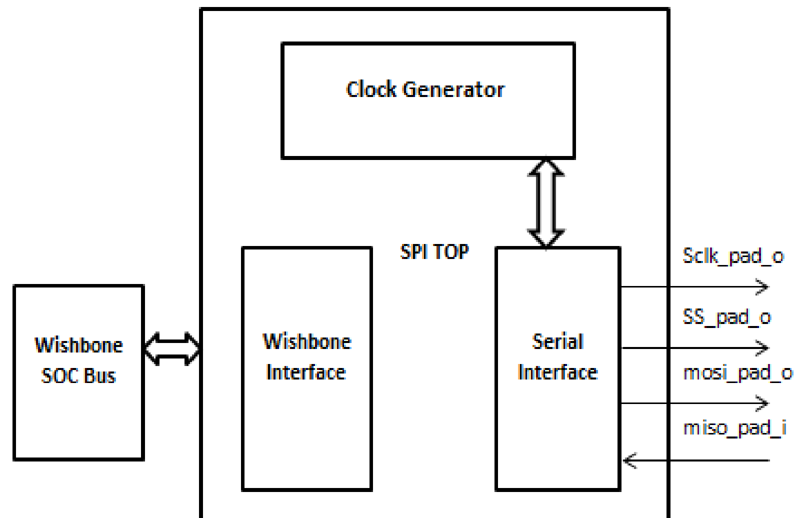

Fig 1: SPI Architecture [7]

A full duplex data transmission occurs during each SPI clock cycle [7]:

a. the master drives a bit on the MOSI line and the slave declaims it from that same line

b. the slave drives a bit on the MISO line and the master declaims it from that same line

Normally two shift registers of a given word size, such as eight bits, one in the slave and one in the master are involved in transmission; they are connected in a ring fashion. Data is shifted out with the most significant bit (MSB) first, while shifting a fresh least significant bit (LSB) into the same register. The master and slave have exchanged the values after that register has been shifted out. The device connected takes that value and performs something with it, for example writing it to memory. The

shift registers are loaded with a new data if there is more data to exchange and the process reprises [7].

Any number of clock cycles may be involved in transmission. The master stops toggling its clock when there is no more data left to be transmitted. Normally, then the slave is deselected. A master can start multiple such transmissions if it needs to; transmissions often consist of 8-bit words.

Every slave on the bus that hasn't been triggered using its chip select line must ignore the input clock and MOSI signals, and must not drive data on MISO. The master must select a single slave at a time [7].

4. SYSTEMVERILOG VERIFICATION METHODOLOGY

The SystemVerilog verification methodology relies on 3 building blocks [13]:

- Providing stimuli to the design using automatically generated random scenarios or constrained-random (CR) test generation.
- Check the conduct of the design through assertions and the output data through the checker or scoreboard to verify the correctness of operation.
- Measure the functional coverage to analyze progress of verification and provide feedback to the generation.

SV introduces Universal Verification Methodology. UVM is a methodology for functional verification using SystemVerilog, with a supporting library of SystemVerilog code.

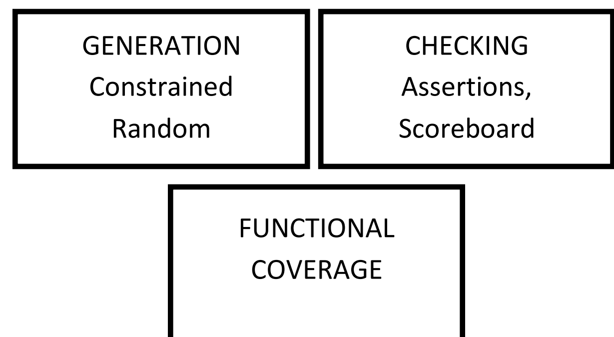

Fig.2.Verification Blocks [7]

Verification efficiency can be improved by reusing verification components, and this is a key objective of UVM. Verification reuse is facilitated by having a modular verification environment where each component has visibly defined responsibilities, by permitting flexibility in the way in which components are configured and used.

UVM facilitates the construction of verification environments and tests, both by providing reusable mechanism through usage of a library of SystemVerilog classes, also by providing a set of guidelines for finest practice when using SystemVerilog for verification. Thus the architecture of UVM has been designed to boost modular and layered verification environments, where verification components can be reused in diverse environments.

5. SYSTEMVERILOG TESTBENCH

Following are the methods which are defined in the environment class of the SV testbench [5]. The verification components made can be followed through fig 3.

a. build (): In this method, all the objects viz. 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 till all the required operations in other components are completed.

e. report (): This method is used for printing the results of the simulation, based on the error count.

f. run (): This method calls for the above declared methods in a sequenced manner.

6. VERILOG TESTBENCH

Modules are created for Wishbone and SPI master model. The testbench is a directed one. When wishbone is the master SPI acts as a slave as clock and data input is provided by the wishbone master. The functions included to execute the master side are:

Wishbone write cycle

Wait for acknowledge from slave

Wishbone read cycle

Wait for acknowledge from slave

Wishbone compare cycle (read data from location and compare with expected data)

When SPI is the master the slave or responder in this case sends the data miso when slave select and clock (sclk) is given by the master. Whether the data is latched on the posedge or negedge of the clock depends on the value of Rx_negedge.

In the top module SPI master model and Wishbone master model are instantiated. The values are initialized here and reset is provided to the design. The design core is configured here by setting and verifying the register values.

7. PERFORMANCE ANALYSIS PARAMETERS

To perform the performance analysis, various timing constructs were considerd. These constructs along with their brief description are given below. Comparisions have been drawn based on these performance constructs.

The various performance parameters on which the analysis has been performed are:

1. Elaboration Time: The elaboration process creates a design hierarchy based on the instantiation and configuration information in the design, establishes signal connectivity. Memory storage is allocated for the required signals. The elaboration process constructs a hierarchy of module instances that ends with primitive gates and statements [8].

   Simstats command :reports performance-related statistics about active simulations.The
   statistics measure the simulation kernal process (vsimk) for a single invocation of vsim [8].
   Syntax
   simstats [memory | working | time | cpu | context | faults]
   vsim -c -do "run -all; simstats" top-level-module"
   Arguments
   • Memory- Returns the amount of virtual memory that the OS has allocated for vsimk.
   • Working- Returns the portion of allocated virtual memory that is currently being used byvsimk. If this number exceeds the actual memory size, you will encounter performancedegradation.
   • Time- Returns the cumulative "wall clock time" of all run commands.
   • CPU- Returns the cumulative processor time of all run commands. Processor timediffers from wall clock time in that processor time is only counted when the CPU is actually running vsimk. If vsimk is swapped out for another process, CPU time does not increase[8].

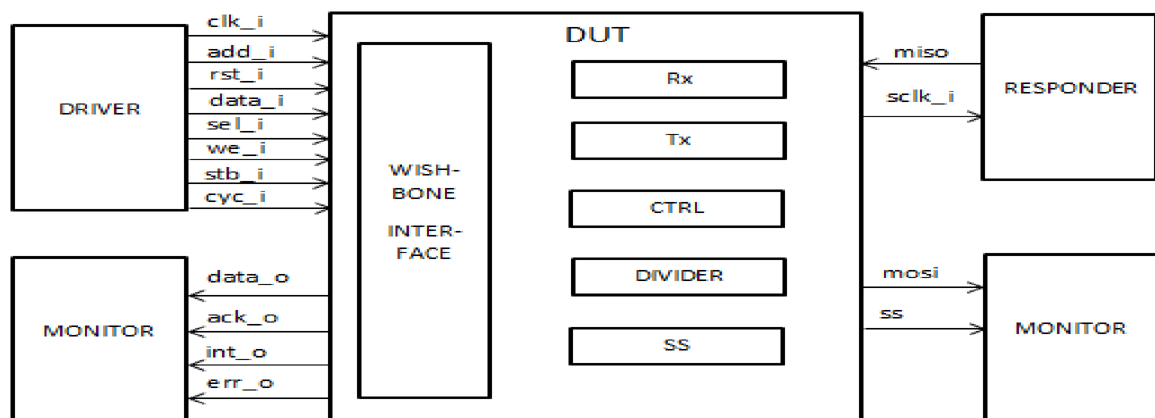

Fig 3: Architectural overview of the verification modules

The use of these arguments is optional and when executed without arguments, the command returns a list of pairs similar to the following [12]:

{{elab memory} 0} {{elab working set} 7245839} {{elab time} 0.942836}
{{elabcpu time} 0.1901574} {{elab context} 0} {{elab page faults} 1556}

{memory 0} {{working set} 0} {time 0} {{cpu time} 0} {context 0}
{{page faults} 0}

2. CPU Time: CPU time for completion of the test.

   CPU Time is the time the CPU actually spent executing your program. This need not be a continuous measurement of time, and since only 1 process may be executed on 1 CPU at any given time, the program is not executed continuously, but rather in chunks doled out by the kernel's CPU scheduler [9].

3. Time: (optional) Returns the cumulative "wall clock time" of all run commands.

4. Time Elapsed: Elapsed time is the duration your program is running. This is measured

5. continuously from when a process is born until it dies.

   To obtain the elapsed time, you can use the TCL clock command:

   e.g.

   setstarttime [clock seconds]

   # 1087889342

   run 1000 ns

   setendtime [clock seconds]

   # 1087889359

   settotaltime [expr $endtime - $starttime]

   # 17
   echo "Simulation took $totaltime seconds"

6. SIMTIME Simulation time at completion of the test [12].

   (In Questa SIM, the $Now TCL variable contains the current simulation time in a string of the form: simtime_units.)

7. Simulation Time: Simulation is defined as the process of constructing a model of a system in order to identify and recognize those factors which control the system and/or to forecast the future behavior of the system [9].

   Here both the testbenches were simulated for a period of 3400ns.

   TIMEUNIT Units for simulation time: "fs", "ps", "ns", "us", "ms", "sec", "min", "hr".

8. PERFORMANCE ANALYSIS AND SIMULATION RESULT

All the analysis has been made through QuestaSim 10.0b tool from Mentor Graphics. Today Questa is the leading high performance SystemVerilog and Mixed simulator. Both the Verilog and SystemVerilog testbenches have been compiled and simulated on QuestaSim only.

Fig 4 shows the verification output of SPI core as all the signals are getting generated properly.

The elaboration statistics are measured one time at the end of elaboration. The simulation memory statistics are measured at the time simstats is invoked. The simulation time statistics are updated at the end of each run command. Units for time values are in seconds.

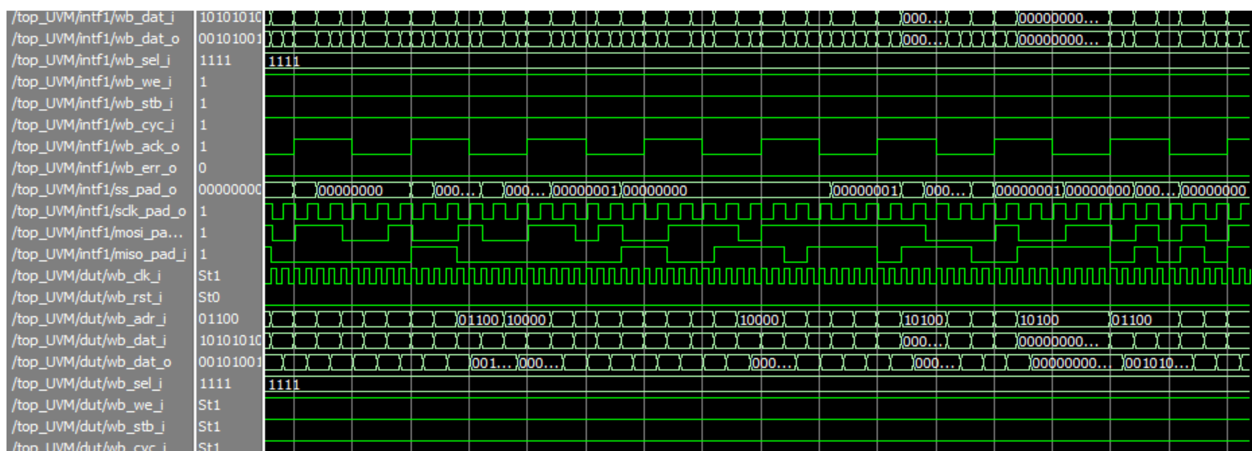

Fig 4: Simulation Waveform

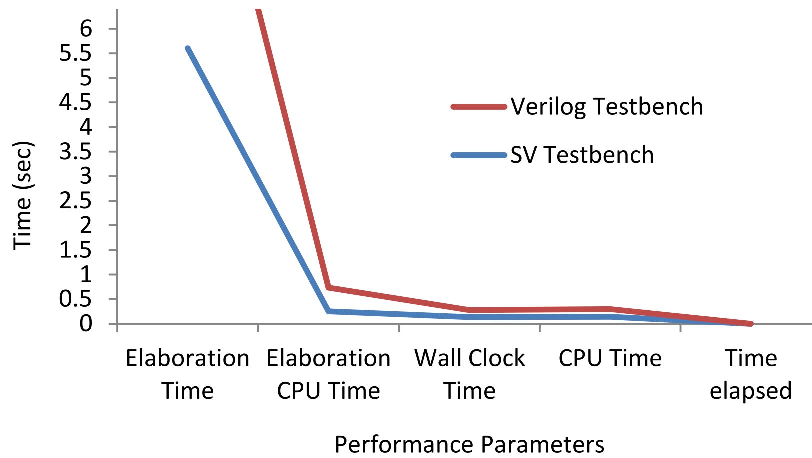

Fig. 5: Graph comparing the performance of SV & Verilog testbench for SPI as the DUT

The emerging System-on-a-Chip (SoC) business is enabling the rapid design of nearly complete systems on a single chip. Using a linear extrapolation, the growth rate indicates a trillion transistor chip within the next 10 years. As the size and complexity of SoC design grow, an efficient and structured verification environment is becoming more important than ever before. This capability is generating a flood of performance questions. Hence, , based on our extensive research, we have tried to come up with the performance analysis of a SoC using these two testbenches under the scanner.

We can see from Fig 5 and Fig 6 and Fig 7, the performance comparison based on these two testbenches is clear with good distinction in case of SoC as compared to the marginal differences seen in the performance comparison in case of our DUT i.e. SPI core.

Fig 8 demonstartes the testbench performance of Verilog and SystemVerilog testbench, based on the various phases involved like compilation phase, elabotation phase, run phase and wrap-up phase.

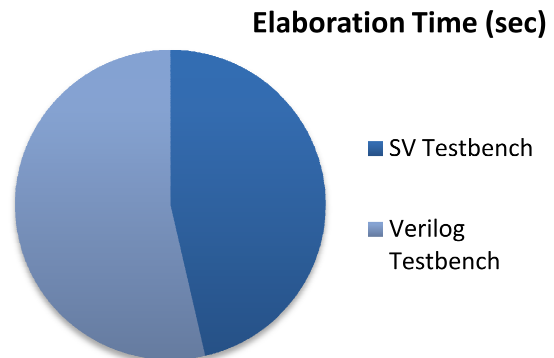

Fig. 6(i): Elaboration Time analysis for a large SoC

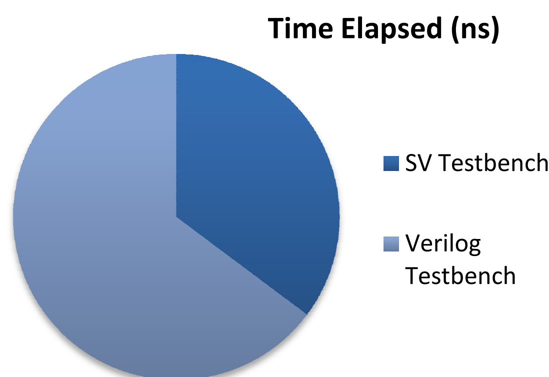

Fig. 6(ii): Time Elapsed analysis for a large SoC

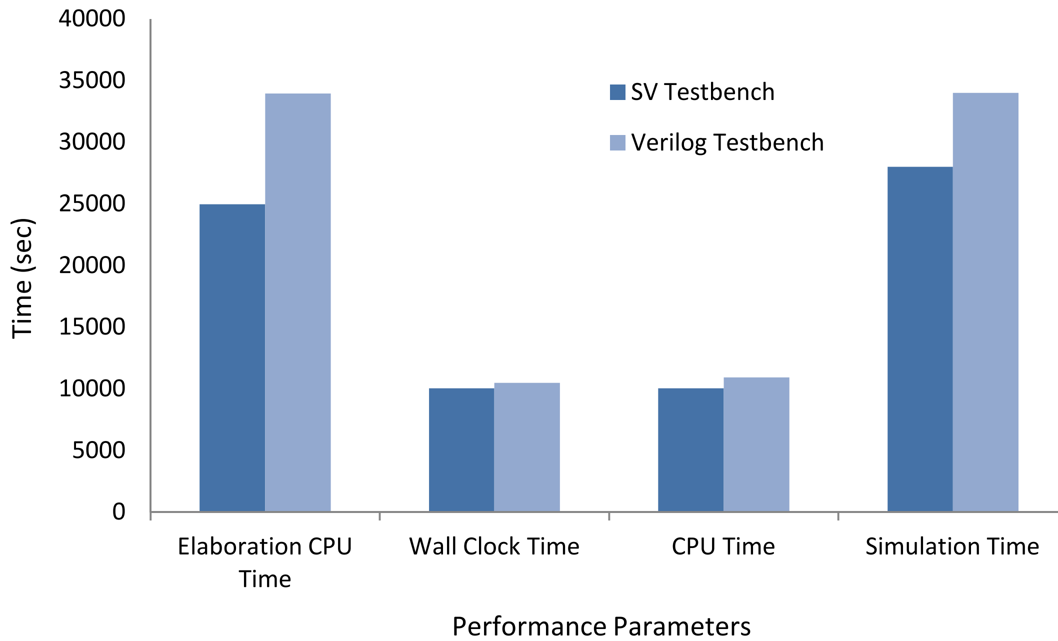

Fig. 7: Performance analysis for a large SoC

9. CONCLUSIONS

Through the performance analysis on both the testbenches it can be established that the performance and testbench parameters like elaboration time, time elapsed, CPU time, compilation time, run time and wrap-up time were less in the case of SV testbench. Both the testbenches perform the same functionality test operations; SV in addition performs the coverage analysis as well. Performance study has also been conducted for a larger and complex design. It can be concluded that when analysis is done using a much complex and larger DUT as in a SoC, the distinction in performance for various parameters is more visible. Hence it can be established that SV apart from being a superior verification language also gives better performance results as compared to a Verilog testbench.

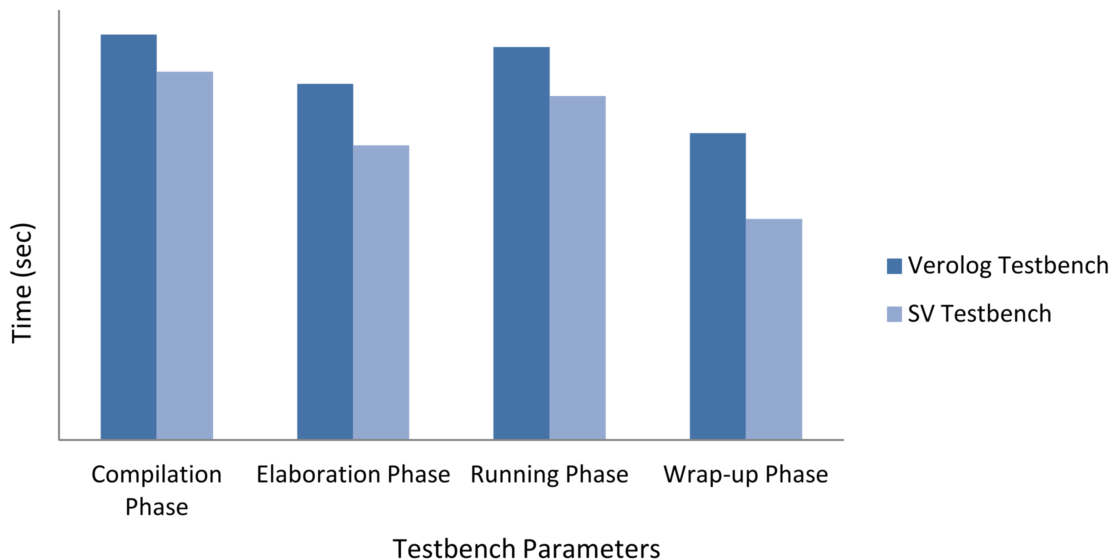

Fig. 8: Testbench performance comparison of SV and Verilog Testbench of a SoC

10. ACKNOWLEDGEMENT

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

11. REFERENCES

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

[2] Stuart Sutherland, “Don’t Forget the Little Things That Can Make Verification Easier,” Verification Horizons, Mentor Graphics

[3] SystemVerilog 3.1a, Language Reference Manual

[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] 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

[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, May 2012, Issue-2.

[7] Srot Simon, “ SPI Master Core Specification,”Rev. 0.6, March 15, 2004

[8] Questa® SIM User’s Manual, Software Version 10.0d

[9] ModelSim® Reference Manual, Software Version 6.5e