Research Article

VERIFICATION OF I2C DUT USING SYSTEMVERILOG

[1] Purvi Mulani, [2]Jignesh Patoliya, [3]Hitesh Patel, [4]Dharmendra Chauhan

Address for Correspondence

[1,2,3] Dept.of Electronics and Communication Engg.

Charotar University of Science and Technology, Changa,Anand,Gujarat-388421(India)

[4]Electronics and Communication Dept S.P.B.Patel Engineering College, Linch

Mehsana - Ahmadabad Highway.

[1]purvi.mulani@gmail.com, [2]jigneshpatoliya@ecchanga.ac.in, [3]hiteshpatel.ec@ecchanga.ac.in,

[4]chauhan_ec@yahoo.co.in

ABSTRACT

Verification is the process used to demonstrate the functional correctness of a design prior to its fabrication. The lack of flexible verification environments that allow verification components reuse across ASIC design projects keep the verification cost very high. Design engineers have made design reuse central in bringing the design effort's complexity back to a manageable size and to reduce development time and effort. Considering the fact that verification consumes more resources than design does in a typical design project, it would be of great value to build verification components that are modular and reusable. This paper describes the verification of I2C DUT using System Verilog. The DUT has been verified for all four possible configurations, which are: Master TX, Master Rx, Slave TX, and Slave Rx. The verification environment is designed in System Verilog for verifying the DUT which acts as master if DUT is configured as slave and acts as slave if DUT is configured as master. The verification environment designed is reusable for any I2C DUT.

KEYWORDS: ASIC, DUT, I2C, SoC, System Verilog

I. INTRODUCTION

The tremendous progress of VLSI technology enables the integration of more than several million transistors in a single chip to make a SoC (System-on-Chip). This has made verification the most critical bottleneck in the chip design flow. Roughly 70 to 80 percent of the design cycle is spent in functional verification. [1]System Verilog is a special hardware verification language to be used in function verification. It provides the high-level data structures available in object-oriented languages, such as C++. These data structures enable a higher level of abstraction and modeling of complex data types. The System Verilog also provides constructs necessary for modeling hardware concepts such as cycles, tri-state values, wires, just like Verilog hardware languages. So System Verilog can be used to simulate the HDL design and verify them by high level test case. I2C is one module in this SoC and it has been verified for all possible configurations. During verifying the SoC, a great deal of visual simulation waveform inspection is required. The Simvision waveform viewer is used and the observed waveforms are also discussed in this paper.

II. VERIFICATION ENVIRONMENT ARCHITECTURE

The architecture of verification environment developed for I2C protocol is shown in the figure 1. The different modules of environment are explained.

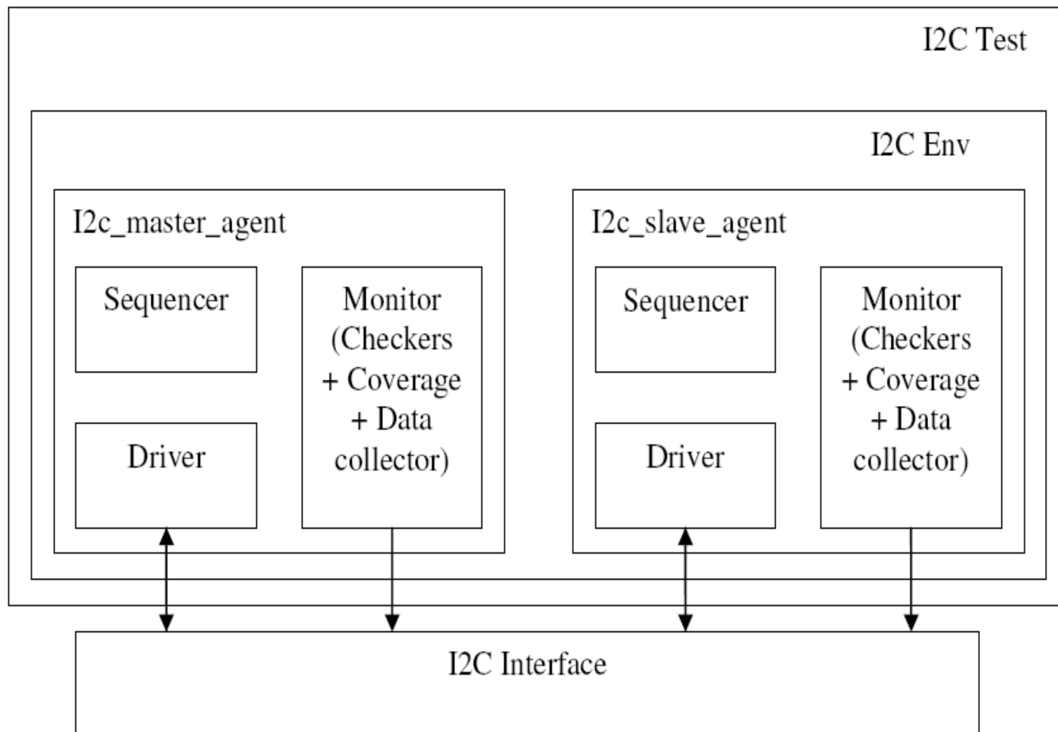

Figure 1. I2C verification environment architecture

A).Top module

This is test case which is class of system Verilog which contains instances of I2C Env, master agent and slave agent.

B). i2c_env

This is I2C component, containing Agent (master and slave). In addition, agent should be configurable for passive/active. All checkers and coverage are configurable to disable/enable.

C). i2c_env_config

This env config class contains the configurable parameters like number of masters, number of slaves present in the environment.

D). i2c_transfer

This is the basic transfer class, which will have all required parameters for I2C like address, read/write access, data size, etc.

E) .i2c_master_agent

Master agent is configurable either as a active or as a passive. Active contains Driver, Sequencer and monitor, while passive component contains only Monitor. Agent will also pass the interface of the DUT to each of the sub-sequent component.

F). i2c_master_agent_config

Master agent config has all the parameters like frequency of the master, timing parameters for the master like delay_to_drive_sda, delay_to_sample_sda, scl_high_width, scl_low_width, the duration for which glitch needs to be generated. These parameters can be configured at run time for the component. It contains two methods:

(1) i2c_ferq_update: to change the frequency according to user.

(2) i2c_scl_pulse_width: calculates scl pulse width according to frequency.

G). i2c_master_monitor

Master Monitor collects all the data from interface and makes a transaction. It also stores this data and emits an event for score-boarding. Based on

the collected transfer, it makes functional coverage and also does a data checking.

H). i2c_master_driver

Driver makes use of "i2c_transfer" as a basic item and does connect the sequences with this basic item. This will also contain the provision to make directed and random test selection.

I). i2c_master_seq

This will contain the sequences, which are going to be used for verification. By this, the test case can become shorten and easier.It contains instances of monitor, master_agent_config and master interface.

J). i2c_master_seq_library

This file contains the different sequences which are used in test cases to generate different scenarios. Each sequence is a class.

K). i2c_slave_agent

Slave agent is configurable as either active or passive. Active contains driver, sequence and monitor while passive component contains only Monitor. Agent will also pass the interface of the DUT to each of the sub-sequent component.

L). i2c_slave_agent_config

Slave agent config has all the parameters like address of the Slave, Glitch period to be detected on SDA and SCL line, busy bit to be set by slave. This parameters can be configured at run time for the component.

M). i2c_slave_monitor

Slave Monitor collects all the data from interface and makes a transaction. It also stores this data and emits an event for score-boarding. Based on the collected transfer, it makes functional coverage and also does data checking.

N). i2c_slave_seq

This will contain the sequences, which are going to be used for verification. By this, the test case can become shorten and easier.

It contains instances of monitor, slave_agent_config and slave interface.

O). i2c_slave_seq_library

This file contains the different sequences which are used in test cases to generate different scenarios. Each sequence is a class.

III. TESTCASES

A).DUT can work in following four modes.

Mater_TX:

DUT is master and is in transmit mode. So we have used slave part of our verification environment and we have to configure it in RX mode to receive data transmitted by DUT.

Mater_RX:

DUT is master and is in receive mode. So we have used slave part of our verification environment and we have to configure it in TX mode to transmit data to DUT.

Slave_TX:

DUT is slave and is in transmit mode. So we have used master part of our verification environment and we have to configure it in RX mode to receive data transmitted by DUT.

Slave_RX:

DUT is slave and is in receive mode. So we have used master part of our verification environment and we have to configure it in TX mode to transmit data to DUT.

Testcases are written for these four modes to verify the DUT.

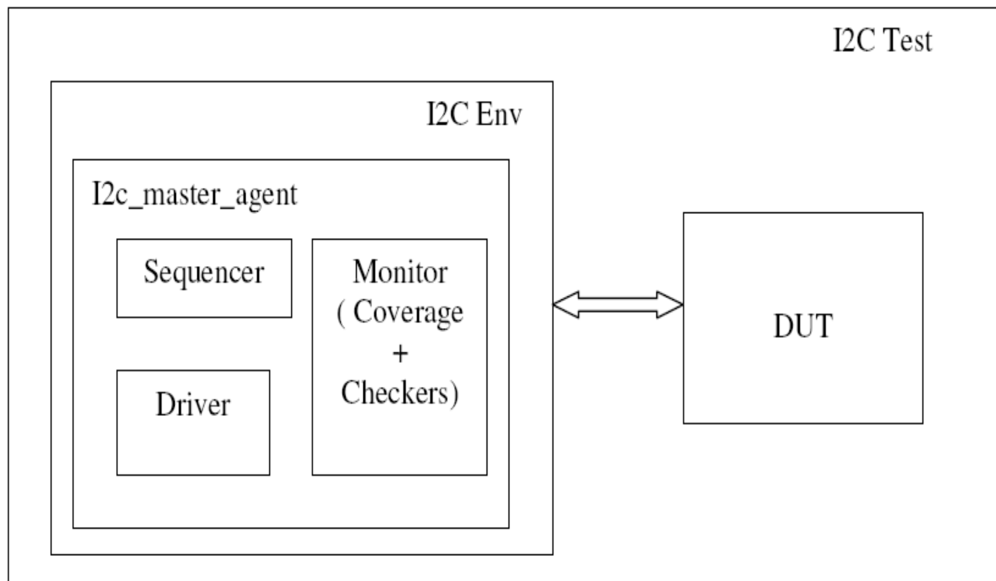

Figure 2. Environment with DUT configured as slave

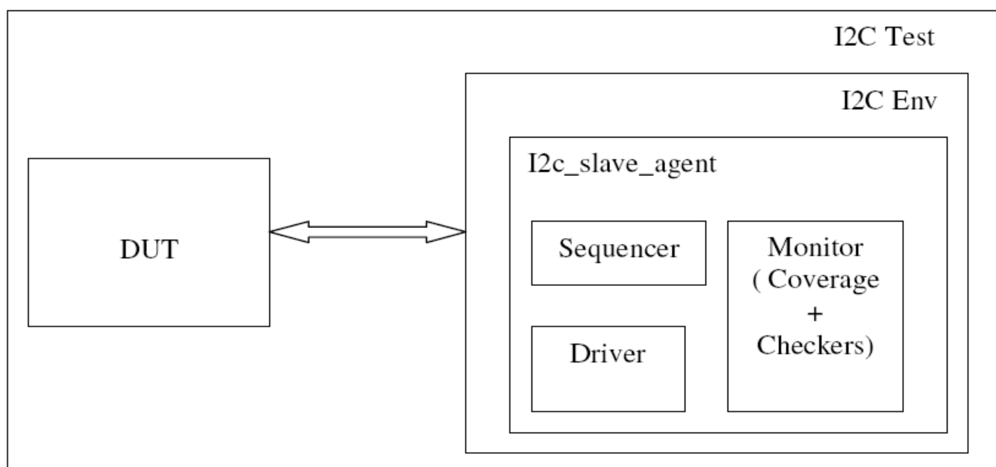

Figure 3. Environment with DUT configured as master

B). i2c_slave_rx_test.sv:

DUT has been configured as slave and it is configured as a receiver. To verify the DUT, the verification environment is developed which is master and it transmits the data which is received by DUT which is shown in figure 3. Verification environment asa master will generate clock and start condition. After that, it will send address of the slave to which it wants to communicate which is shown by signal sda_in_iic in figure. This address is received by DUT and then DUT will send acknowledgement to the verification environment (master). After receiving acknowledgement, Verification environment will send data to DUT on the pin sig_sda which will be received by slave DUT on sda_in_iic pin. Then after data transfer has been finished, verification environment as a master will generate stop condition which indicates completion of transfer.

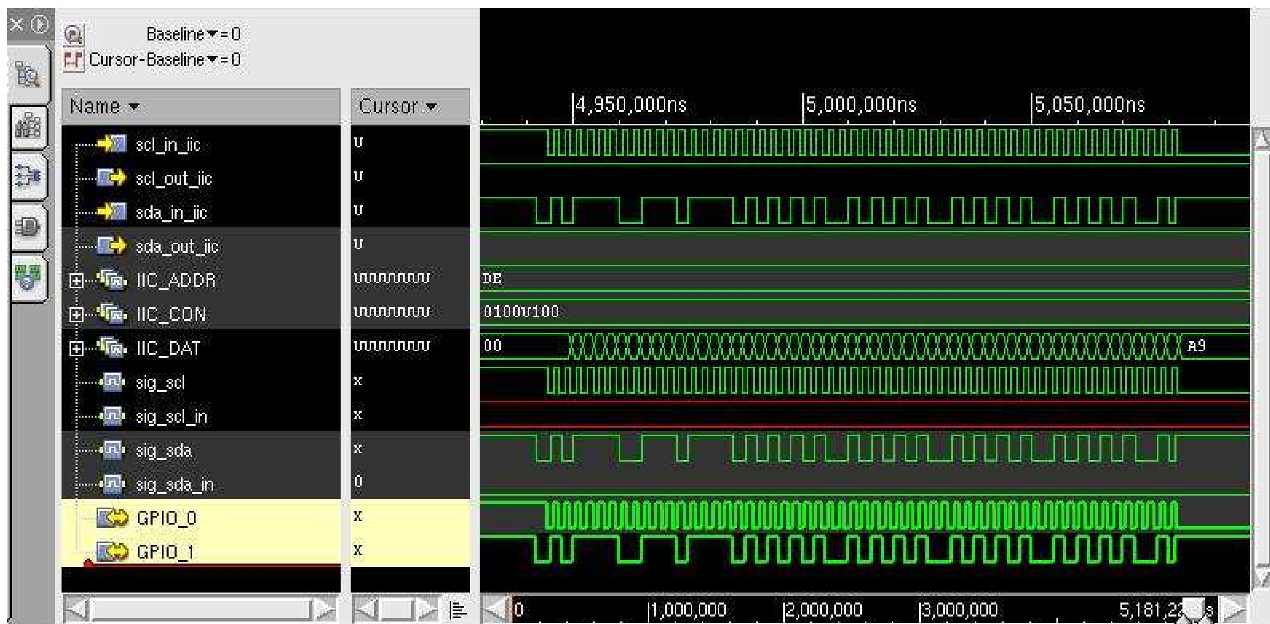

Figure 4. i2c_slave_rx_test

IV. CONCLUSION

In this paper, we have used System Verilog to put up a verification intellectual property which is reusable to verify any I2C DUT. By using this verification environment the DUT has been verified for its functionality.

REFERENCES

1. Han Ke, Deng Zhongliang, Shu Qiong "Verification of AMBA Bus Model Using SystemVerilog" in The Eighth International Conference on Electronic Measurement and Instruments
2. SystemVerilog 3.1a Language Reference Manual Accellera's Extensions to Verilog
3. UM10204 I2C-bus specification and user manual Rev. 03 — 19 June 2007
4. Micro computer control small area network specialists
5. I2C bus Inter Integrated Circuits bus by Philips Semiconductors TomášMatoušek tmd.havit.cz