Development of Verification Envioronment for SPI Master Interface Using SystemVerilog

Zhili Zhou, Zheng Xie, Xin'an Wang* and Teng Wang
Key Lab of Integrated Micro-Systems, Peking University Shenzhen Graduate School, Shenzhen, China
*E-mail : anxinwang@pku.edu.cn

Abstract—**System-level verification with scalable and reusable components provides a solution for current complex SOC verification and SystemVerilog with OOP is one of the most promising language to develop a complete verification environment with constrained random testing, functional coverage and assertions. In this paper, a uniform verification environment for SPI master interface is developed using SystemVerilog after a comprehensive analysis of the verification plan. The proposed multi-layer testbench is comprised of APB driver, SPI slave, scoreboard, checker, coverage analysis and assertions, which are implemented with different properties of SystemVerilog. Furthermore, constrained random testing vectors are generated automatically and driven into the DUT for higher functional coverage. The verification result shows the effectiveness of the proposed verification environment, which is of great feasibility for further extension and reuse.**

Keywords-SystemVerilog; object-oriented programming; SPI interface; functional coverage; assertion

I. INTRODUCTION

With the increasing complexity of digital systems driven by ever increasing demand for faster devices with more features, standard and compatible design and verification methodologies are in urgent need. The implementation of complex system-on-a-chip (SOC) design requires hardware platforms comprised of multiple processors, accelerators, peripherals and programmable logic arrays (PLA) with detailed cycle accurate description code, namely Verilog, VHDL hardware description language (HDL). However, the same language is not efficient enough for verification, especially for hardware-software co-emulation. With the traditional function simulation of hardware followed by software development and emulation, verification of a digital system is so time-consuming that some say more than 70 percents of the design circle is taken by verification with the risk of rework from the very beginning.

System-level verification with scalable and reusable components has been paid much attention these days. Major EDA tool vendors have proposed varieties of verification methodologies and languages, such as Specman E, SystemVerilog, VMM (Verification Methodology Manual), AVM (Advanced Verification Manual) and the emerging OVM (Open Verification Manual) [1]. Among all these methods, SystemVerilog with object-oriented programming (OOP) is considered as one of the most promising techniques for high-level function verification for current complex SOC designs [2].

In this paper, the development of the verification environment of a serial peripheral interface (SPI) using SystemVerilog with OOP technique is conducted. Functional coverage, score-boarding, assertions and constrained random vectors generation are implemented with the proposed integrated verification environment.

The remaining part of the paper is organized into the following sections. In section II, a brief introduction of SystemVerilog and verification plan of the SPI master interface is described, while section III addresses the proposed verification environment in details. The verification results are presented and discussed in section IV and conclusions are drawn in section V.

II. VERIFICATION PLAN OF SPI MASTER INTERFACE

A. SystemVerilog

As a set of extensions to the Verilog HDL, SystemVerilog gains the advantage of OOP, which can play a role of hardware description language as well as hardware verification language [3]. The main features of SystemVerilog contain constrained random techniques, communications between threads, assertions, functional coverage, and so on. It can conveniently describe design functions and scenarios, which is considerably suitable and effective for verification engineer. SystemVerilog and the related methodology offer an ideal way to build a verification environment and the main features of the verification environment with SystemVerilog can be concluded as follows [4, 5, 6]:

Object-Oriented Programming. The testbench is coded by OOP with every single verification component packaged as a class, which not only makes the testbench suitable for all test cases, but also makes the same component reusable for different designs in different environments. For example, the APB_master class proposed in this paper can also be used in

978-1-4673-2197-6/12/$31.00 ©2012 IEEE

the verification of other peripheral devices with APB interface, which is a way to decrease workloads.

Multi-layer Abstraction. The testbench is constructed with multiple abstraction layers for different functions and the relationships between layers are so clear that if any modification is in demand, the fixing of the corresponding layer is enough without affecting the function of other layers, which is convenient for the maintenance of the testbench and also makes the reuse of the testbench possible.

Assertions. SystemVerilog provides temporal constructs to describe properties of the design under test (DUT) with descriptive language. Verification can benefit from assertions in terms of convenient timing checkup, accurate bug-locating procedure and increased functional coverage. In this paper, assertions are applied in the testbench to detect interrupts from the SPI master.

Constrained Random Testing. Testbench by SystemVerilog can achieve automatic configuration of verification environment for different working condition of the DUT. Constrained random testing (CRT) can be conducted with a pseudo-random number generator (PRNG) so that a large set of test cases can be produced for the DUT. Along with the random configuration is the automatic comparison between outputs of DUT and the expected results, which saves a lot of labor and decrease manual mistakes. No matter how many times DUT being modified, the only thing to do is running the testbench as long as the requirements of DUT do not change.

Functional Coverage Analysis. As a means of measuring what has been verified in the DUT, functional coverage analysis along with code coverage is supported to check off items in the verification plan by collecting input types and internal states of the DUT, which can provide a guide for further verification to obtain a comprehensive verification of every feature of the DUT.

B. Functional Verification Requirement

SPI is a high speed, four-wire, full-duplex, synchronous serial communication protocol proposed by Motorola, which is widely used in kinds of SOCs for data transfer with on-board peripherals. SPI interface always consists four wires, namely SCK (serial clock output), MOSI (master output slave input), MISO (master input slave output), and SSN (slave select negedge). The proposed SPI master is designed to be applied in an APB system with an APB slave interface, the verification of which will not be discussed in this paper.

As shown in Fig.1, verification plan, which is comprised of functional verification requirements and verification environment requirements, plays an important role in the

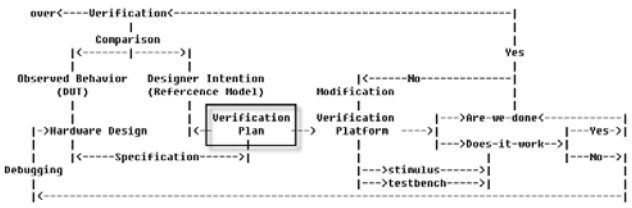

Figure 1. The importance of verification plan

verification flow and acts as a bridge between the designer's intention and the verification platform.

Following the specification of the SPI protocol, the functional verification requirements of the SPI master interface in this paper include [7]:

- Four kinds of transfer modes: transmit and receive; transmit only; receive only; EEPROM read.
- Four different clock mode combinations with the clock phase of 1'b0/1'b1 and clock polarity of 1'b0/1'b1.
- Thirteen choices of data frame size in the range of 4-16.
- Baud rate select with clock divider from 1/2 to 1/65536.
- Five kinds of internal interrupts, including: txe (transmit FIFO empty), txo (transmit FIFO overflow), rxf (receive FIFO full), rxo (receive FIFO overflow), rxu (receive FIFO underflow).
- Status registers for CPU access.

Meanwhile, there are also limits for the application of the SPI master in this paper according to the system requirement:

- No configuration to the SPI is accepted while the SPI master is enabled.

C. Verification Environment Requirement

On the base of the functional verification requirements mentioned above, verification environment requirements can be defined, which is essential to fulfill an expected design.

- Specify the configuration items which can be generated with constrained random testing and the dimensionalities of each item that can be constrained for the random number generator. Also, the stimulus sequences which cannot be generated by random method should be specified.
- Specify a scheme for results monitoring and checking along with a scoreboard to record and analyze the results.
- Specify the cover group or cover property as a functional coverage model to follow up the verification flow.
- Specify the exceptions of a normal working condition to trigger an interrupt.

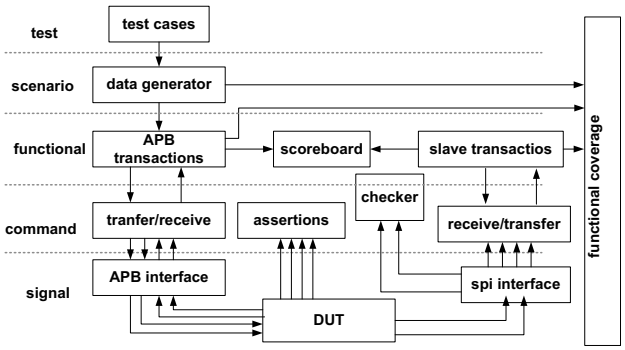

Figure 2. The proposed layered testbench for SPI verification

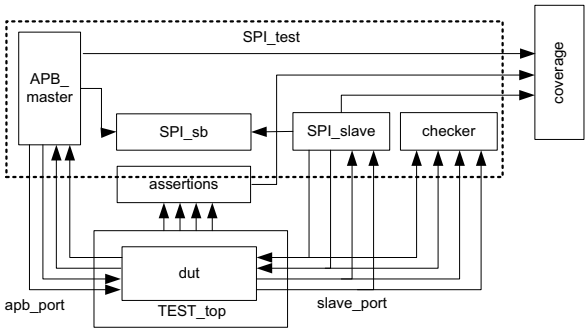

Figure 3. The implementation of the proposed testbench

• Specify the criterion with which the testing is sufficient enough to be terminated.

III. DEVELOPMENT OF VERIFICATION ENVIORMENT

Janick Bergeron proposed a classic testbench, which is comprised of test layer, scenario layer, functional layer, command layer, and signal layer [8]. Accordingly, a layered testbench for SPI verification is proposed as shown in Fig.2. Based on the functional verification requirements and verification environment requirements along with the layered testbench in Fig.2, the scheme for partition of program modules and distribution of resources in the proposed testbench can be confirmed. Fig.3 presents the components which build up the proposed testbench and the relationships between them.

A. APB_master Class

The functions of APB_master class contain the configuration of SPI, and the commands of APB write/reads. Constrained random technique is applied in this class to generate transfer data and configuration parameters of environment requirements for the DUT, which is shown in Fig.4. The configuration parameters include transfer mode, clock phase and polarity, clock divider, and frame size.

A construction function and three tasks are built up in APB_master, which is constructed with the apb_port interface and functionalities for SPI configuration, APB read and write.

```
constraint c_cfg{
        tmod  inside {2'b00,2'b01,2'b10,2'b11};
        spha  inside {1'b0,1'b1};
       spol  inside {1'b0,1'b1};
       baurd inside {16'h0002,16'h0004,16'h0008};
      dfs   inside {[3:15]};}
```

Figure 4. The constrained random configuration parameters

B. SPI_slave Class

The SPI_slave class is built up with a constructor and two tasks. APB_slave connects with the slave_port interface and the two tasks are applied to receive or transmit data on the posedge or negedge of input clock according to the configured clock phase and polarity, while the transmit data is generated randomly.

C. SPI_sb Class

The SPI_sb (scoreboard) class realizes the comparisons between the data APB_master transmits/receives and the data SPI_slave receives/transmits. The two components in SPI_sb to buffer the comparison data are the transmit/receive mailboxes.

When APB_master transmits a data frame, it will be stored in the transmit mailbox. When SPI_slave has received a data, it will be compared with the one popped from the transmit mailbox. The receiver mailbox is used in the same way when SPI_slave transmits a data and APB_master receives the data.

D. Assertions

SystemVerilog assertion (SVA) is a declaration for the expected behavior. In this paper, assertions are applied to detect whether interrupts are triggered correctly, and they will induce the signals inside the design. For example, the txo interrupt can be expressed as presented in Fig.5.

```
……
@(posedge sck) disable iff (~rst)
    (u_apb_spi_master.tx_fifo_full                    &&
u_apb_spi_master.tx_fifo_wr |-> spi_txo_intr);

……
```

Figure 5. An example of assertion in the proposed testbench

As we can see from Fig.5, APB_master initiates a write transaction when the transmit FIFO is full, and the txo interrupt should be asserted, otherwise it means there is a design bug.

E. Functional Coverage

Modeling the functional verification requirements, functional coverage is an essential part for verification especially when constrained-random testing is applied.

```
covergroup apb_cvrg;
    tmod:coverpoint u_apb_model.tmod;
    spha:coverpoint u_apb_model.spha;
    spol:coverpoint u_apb_model.spol;
    buard:coverpoint u_apb_model.baurd{bins two = {2};
                          bins four = {4};
                          bins eight = {8};
                         }
    dfs:coverpoint u_apb_model.dfs{ bins length[] = {[3:15]};}
    spha_and_spol_c :cross spha,spol;
endgroup
```

(a)

```
covergroup slave_cvrg;
    tx_posedge:coverpoint u_slave_model.tx_posedge;
    rx_posedge:coverpoint u_slave_model.rx_posedge;
    dfs:coverpoint u_slave_model.data_size{bins length[] = {[3:15]};
    slave_mode:cross tx_posedge,rx_posedge{
        bins ptx_nrx = binsof(tx_posedge)intersect{1} && binsof(rx_posedge) intersect{0};
        bins ntx_prx = binsof(tx_posedge)intersect{1} && binsof(rx_posedge) intersect{0};
        ignore_bins bin0 = binsof(tx_posedge) intersect{0} && binsof(rx_posedge) intersect{0};
        ignore_bins bin1 = binsof(tx_posedge) intersect{1} && binsof(rx_posedge) intersect{1};  }
endgroup
```

(b)

Figure 6. The SystemVerilog description of functional coverage for (a) APB_master and (b) SPI_salve

Functional coverage model in this paper mainly gathers information from APB_master, SPI_slave, and assertions.

For APB_master, a cover group named apb_cvgr is used to cover configuration information, and cover points such as transfer modes, combinations of clock phase and polarity, clock dividers, and data frame size are taken into account, in which cross coverage is applied for clock phase and polarity, as shown in Fig.6 (a). For SPI_slave, as presented in Fig.6 (b), the launch/capture edge of the serial clock as well as the data frame size is involved in the cover group of slave_cvgr, while cover property checking out whether all interrupts have been triggered correctly is tested out for assertion coverage.

F. SPI_test and TEST_top

SPI_test is a program block, which controls the overall verification flow. As shown in Fig.7, SPI_test starts from constrained random testing vector generation and ends at automatic comparison between the actual data and the expected data. Another important functionality of SPI_test is scheduling the corresponding tasks of APB_master and SPI_slave to imitate the real application environment.

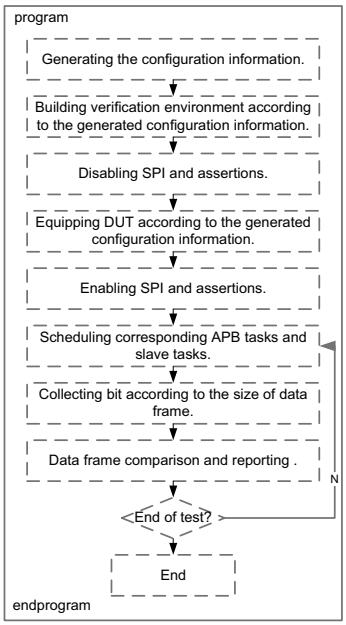

Figure 7. The SPI_test program

TEST_top module works as the entrance of the simulation, in which apb_port, slave_port and DUT are instantiated and connected with each other with signals. This is the bottom layer of the verification environment.

IV. SIMULATION RESULTS

The proposed verification environment applies constrained random technique to fulfill the configuration of verification environment and DUT, and functional coverage is employed to make sure that DUT has realized all the expected functional requirements. And the automaticity of the proposed verification environment improves efficiency of verification largely.

Fig.8 (a) shows part of the simulation result when SPI works in transmit-and-receive mode and the combination of phase and polarity of the clock is {0, 0}. The clock divider and data frame size are set as 4 and 16 respectively. At the time of 145ns, APB_master writes a data frame 0x6f89 to the transmit FIFO of the DUT, which is stored in the transmit mailbox as well, while the first transmission with a data of 0xe53c by SPI_slave is scheduled at the same time and the data is stored in the receive mailbox. The data in SPI_slave begins to be transmitted by slave once a negedge of clock is detected. At the time of 816ns, slave receives data 0x6f89, which is compared automatically with the data from transmit mailbox, showing the correctness of the transmit process. At the time of 855ns, APB_master receives a data of 0xe53c from salve, which is compared with the data from the receive mailbox and shows the correctness of the receive process.

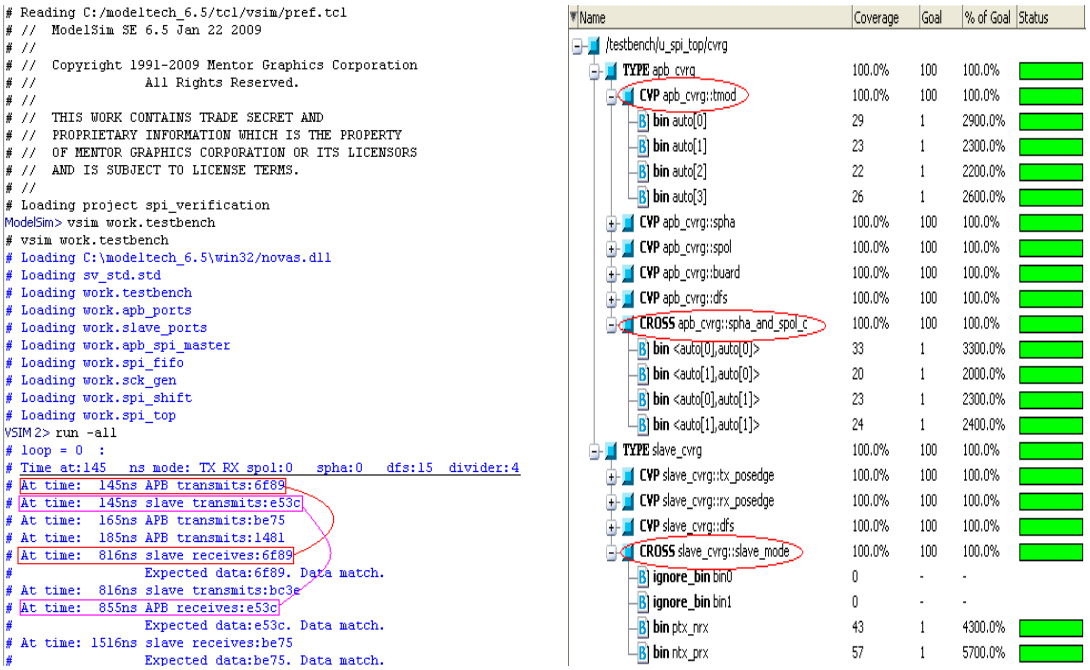

Figure 8. (a) Simulation results; (b) Functional coverage results.

Fig.8 (b) presents the results of functional coverage for the proposed verification environment. The results show that the environment parameters of the DUT are randomly configured 100 times. The apb_cvrg::tmod indicates that the DUT works in one of the four modes randomly. The apb_cvrg::spha _and_spol_c is the cross coverage of phase and polarity of the clock, which demonstrates that each kind of the combinations of phase and polarity has been hit more than 20 times. And the slave_cvrg::slave_mode represents the work state of SPI_slave. The bin "ptx_nrx" means that slave transmits data on the posedge of the clock and receives data on the negedge of the clock, and vice versa for bin "ntx_prx". Since it is impossible for slave to transmit and receive data at the same edge of the clock, there are two bins ignored.

V. CONCLUSION

In this paper, a uniform verification environment for SPI master interface is developed with SystemVerilog. The proposed multi-layer testbench is comprised of APB driver, SPI slave, scoreboard, checker, coverage analysis and assertions, which are implemented with OOP. Furthermore, constrained random technique is applied for higher functional coverage. The verification result provides good evidence for the effectiveness of the proposed verification environment.

ACKNOWLEDGMENT

This work is supported by National High Technology Research and Development Program of China ("863" Program, Grant No.2009AA01Z127). Z.L. Zhou thanks X. Chen in IMS lab of SZPKU for his discussions and advices on the application of SystemVerilog assertions.

REFERENCES

[1] Martin Keavency, Anthony McMahon, et al. "The development of advanced verification environments using systemverilog". Proceedings of ISSC2008, pp.325-330, 2008.

[2] Myoung-Keun You, Gi-Yong Song. "SystemVerilog-based Verification Environment Using SystemC Custom Hierarchical Channel", IEEE 8th conference on ASIC, pp.1-4, 2009.

[3] Han Ke, Deng Zhongliang, Shu Qiong. "Verification of AMBA Bus Model Using SystemVerilog. 8th International Conference on Electronic Measurement and Instruments", pp.776-780. 2007.

[4] Chris Spear, SystemVerilog for Verification (2nd Edition): A Guide to Learning the Testbench Language Features, Springer, 2008.

[5] Yong-Jim Oh, Gi-Yong Song. "Simple hardware verification platform using systemverilog". IEEE TENCON2011, pp.1414-1417, 2011.

[6] H. Foster, "Applied assertion-based verification: An industry perspective," Foundations and Trends in Electronic Design Automation, vol.3, no.1, pp.1–95, 2009.

[7] A.K. Oudjida, M.L. Berrandjia, et al. "Design and Test of General-Purpose SPI Master/Slave IPs on OPB Bus". 7th International Multi-Conference on Systems, Signals and Devices. 2010.

[8] Janick Bergeron, Eduard Cerny, et al. Verification Methodology Manual for SystemVerilog. Springer Publisher. 2005.