Design And Verification of Auto-configurable SPI Controller

[1]Aayushi Dey, [2]Bhavini Kumawat

[1]Research Scholar, [2]Assistant Professor
Department of Electrical and Electronics Engineering
[1,2]Oriental University, Indore, Madhya Pradesh, India

***Abstract*: In today's world, more and more functionalities in the form of reusable verification environment cores are integrated into a single chip or SOC. The complexity of system level verification of large SOCs is much higher. The solution to this problem is to provide a reusable and pre-defined verification environment cores. The verification environments are independent, scalable, and have reusable verification components. The System Verilog language is based on Object Oriented Programming tool and is the widely used language to develop a complete verification environment with functional coverage, constrained random testing and assertions. The UVM, written in System Verilog, is a base class library of reusable verification components. It is an open-source System Verilog library that aims to form the verification process flexible by creating reusable verification components and assembling powerful test environments using constrained random stimulus generation and functional coverage methodologies. This paper discusses a UVM based verification environment for testing configurable SPI. A multi-layer testbench was developed which consists of a SPI master slave model, driver, scoreboard, coverage analysis, and assertions developed using various properties of System Verilog and the UVM library. Later, constrained random testing using vectors driven into the DUT for higher functional coverage is discussed. The result shows the effectiveness and feasibility of the proposed verification environment.**

***Keywords*: UVM (Universal Verification Methodology); SPI (Serial Peripheral Interface); DUT (Design Under Test)**

I. INTRODUCTION

There are interface protocols in communication. Interface can be defined as a boundary shared across two separate component of computer system to exchange information between software, computer hardware or peripheral devices. Protocols are defined as all objects that are not related for communicating with each other. Exchanging information between the objects in order to communicate and understand with each other is actually the meaning of interfacing protocol. From the definitions, there are many types of protocols in communication and it is very clear that these protocols are very important and each of protocols is different and has its own methods.

There are many different types of protocols. Most commonly used and popular protocol is Serial Peripheral interface (SPI). SPI was introduced in the year 1979 and was defined as external microcontroller bus which connected the four wires with microcontroller peripherals. The SPI are interfaced with microcontroller and other devices like peripheral EEPROMs, DACs and ADCs. There are different data rate communication protocols in the world of serial data communication. This protocol provides specific purpose for different type of communication. In a nutshell, SPI can be said that it can communicate between medium or small data transfer rate for integrated circuit (IC) with peripheral devices.

SOC design nowadays uses many reusable environment cores and requires a lot of effort for the functional verification of the system. Since the systems are becoming bigger, the requirement for speed on the verification environment is increasing. Not only speed but also the effectiveness of the environment should increase in order to cover most functionality in least time possible. Verilog is one of the ways to describe the hardware, but Verilog is designed for hardware modeling and lacks the features that are required for verification of complex SoC designs. To fix those issues System Verilog has been introduced with its specialization like UVM. This methodology is currently used in verification. Other than methodologies in the verification there are reusable environment cores.

One of the advantages of System Verilog is the functional coverage model [7]. System Verilog allows doing functional coverage-driven verification. In order to do that we need to code the technical documentation into a set of cover points and creates the coverage model. The coverage model will determine which features have been tested by the environment and which have not. These help in understanding the verification holes and, cover them to assure in a completely verified product. Other than the Coverage model System Verilog has the object-oriented programming (OOP) concepts integrated inside. That allows separating the verification environment into smaller parts. Those smaller parts are the base of the verification methodology. In general, the components are generators, for generating certain packets, drivers to send those packets, receivers to receive, monitors to monitor the interfaces, and a scoreboard to check the correctness of the operation. There are other components, but these are the widely used ones. This paper concentrates on developing a verification environment for configurable SPI interfaces. In upcoming chapters, the SPI interface overview is given to have a general idea of the interface. Also, the test environment architectures are described, and everything is concluded with the functional coverage results.

II. Overview of SPI

Motorola developed the Serial Peripheral Interface (SPI) module in the mid-1980s, which permits synchronous, serial, and full duplex communication between a microcontroller and peripheral devices. The structural link between the master and slave cores is depicted in Figure 3. The SPI bus is typically used to send and receive data between microcontrollers and other tiny peripherals such as shift registers, sensors, SD cards, and other similar devices. When compared to other protocols, the SPI protocol has the advantages of a relatively fast transmission speed, ease of usage, and a limited number of signal pins. For transmitting and receiving data, the protocol typically separates devices into master and slave. A master device generates distinct clock and data signal lines, as well as a chip-select line that selects the slave device for which communication is required. If there are several slave devices, the master device will need multiple chip select interfaces to control them.

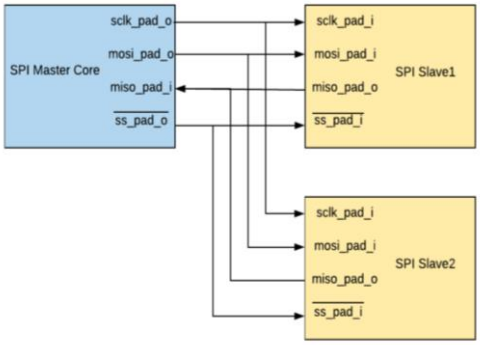


Figure 1 Master and Slave Connection

- Data Transmission - The SPI bus interface consists of four logic signals lines namely MOSI, MISO, Serial Clock (SCLK) and Slave Select (SS).

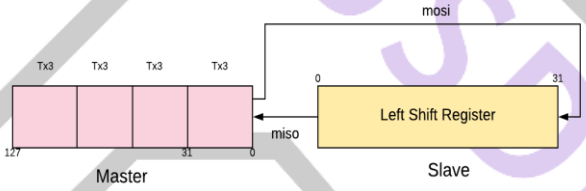


Figure 2 SPI Shift Register

- MOSI - The MOSI is a transversal signal line that can be used as both an output and an input signal line in a master and slave device. It is in charge of data transfer from master to slave in one direction.
- MISO – The MISO is a unidirectional signal line that can be used as an input in a master device and as an output in a slave device. It is in charge of data transmission from slave to master in one direction. The MISO line will be in a high impedance state if a specific slave is not selected.
- Slave Select (SS) - The slave select signal is used as a chip-select line to select the slave device. It is an active low signal and must stay low for the duration of the transaction.
- Serial Clock (SCLK) - The serial clock line is used to synchronize data transfer between both output MOSI and input MISO signal lines. Based on the number of bytes of transactions between the Master and Slave devices, required number of bit clock cycles are generated by the master device and received as input on a slave device.

2.1 Hardware Architecture

The designed SPI is compatible with the SPI protocol and bus principle. At the host side, the design is equivalent to the slave devices of AMBA bus specification. The overall structure of the AMBA complaint SPI Master core device can be divided into three functional units: Clock generator, Serial Interface and AMBA Interface.

2.2 Design of Clock Generation module (spi_clk_gen)

The clk gen is in charge of generating the clock signal from the external system clock wb_clk_i, according to the clock register's varied frequency factors, and producing the output signal s_clk_o. Because there is no response mechanism for the Serial Peripheral Interface, the clk_gen module can produce reliable serial clock transmission with odd or even frequency division in the register to assure timing reliability. By dividing the wb_clk_i, the core creates the s_clk_o; altering the value of the divider allows for arbitrary clock output frequency. fsclk = fwbclk/(DIVIDER + 1) x 2 is the expression for s_clk_o and wb_clk_i.

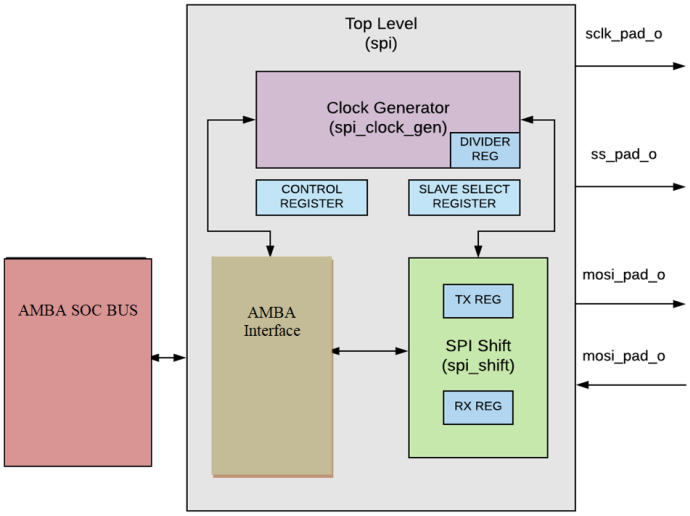


Figure 3 Clock Generation Module

2.3 Serial data transfer module design (spi_shift)

The data transfer core module is made up of serial data transfer modules. It's in charge of converting parallel input data into serial output data for MOSI transmission and MISO serial data into parallel out. The flip flops in the Receive and Transmit registers are the same. If no write access to the transmit register was done between the transfers, the data received from the input data line in one data transfer will be communicated on the output line in the next data transfer. The benefit is that it consumes less electricity because it uses less hardware resources. SPI on the host side receives input data and broadcasts output data in real time as the master device.

2.4 Top-level module (spi)

The top-level module's job is to ensure that the basic framework of high-speed reusable SPI bus sub-components functions properly. As a result, the SPI module's top-level controls the clock generator and serial data transmission modules' operational phase.

III. Functional Coverage and Simulation Results

3.1 Testbench Results

The functional verification of the SPI core controller was carried out successfully with the following results.

3.2 Data Transactions

The results published are for below configuration for a regression run of 10 M tests.

Data Transfer	Sent First	Transmit	Receive
32bit	MSB	posedge	negedge

Table 1 Data Transaction

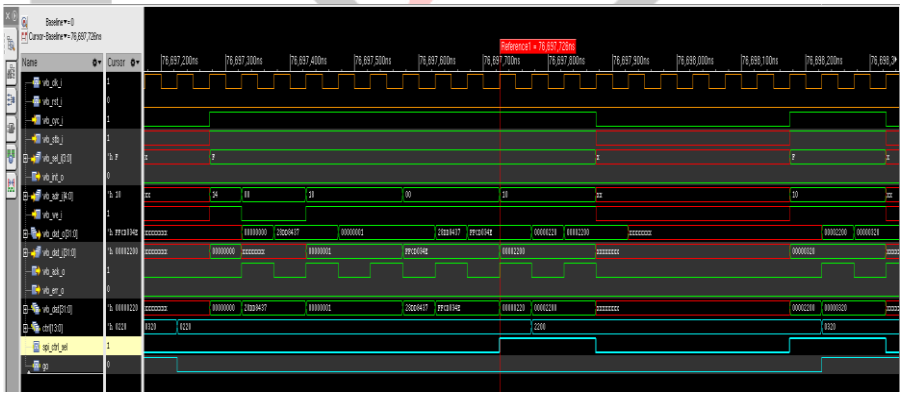


Figure 4 SPI Master Communication

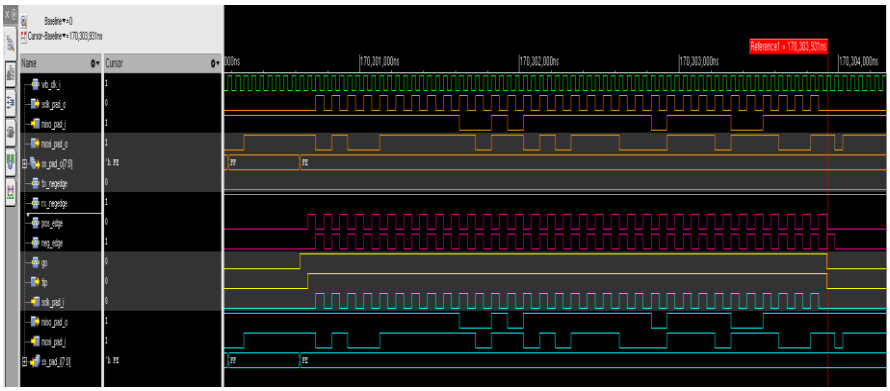


Figure 5 SPI Slave Communications

3.3 Code Coverage

Code coverage keeps track of how many lines of code, expressions, and blocks have been executed. Code coverage, on the other hand, isn't comprehensive and can't detect circumstances that aren't present in the code.

Because not all sections of the code are covered, block coverage is not 100 percent. Except for the AMBA interrupt acknowledgement portion, expression coverage is 100%. Finally, toggle coverage is low since zeros and ones are not covered for all input, output wires, and registers.

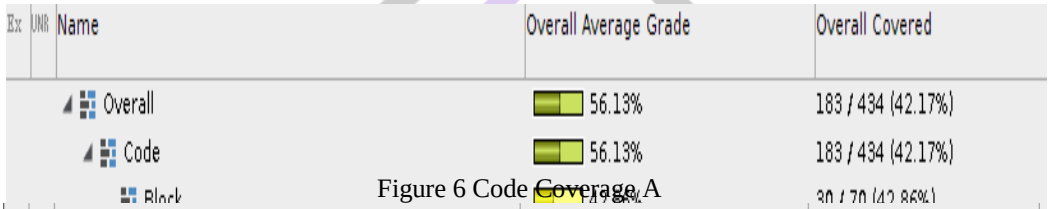


Figure 6 Code Coverage A

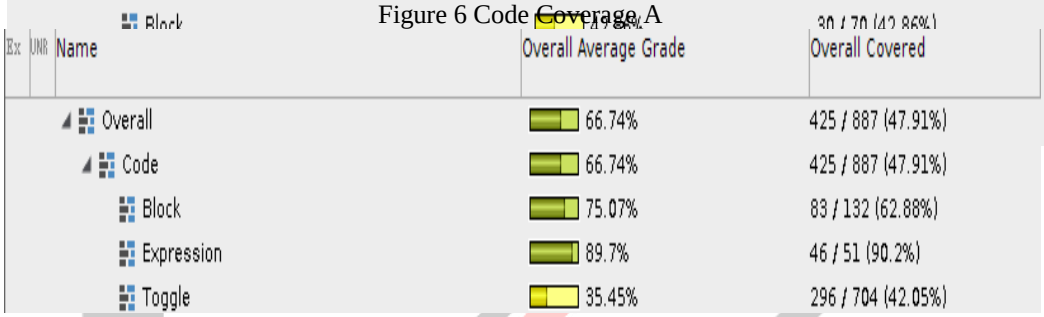
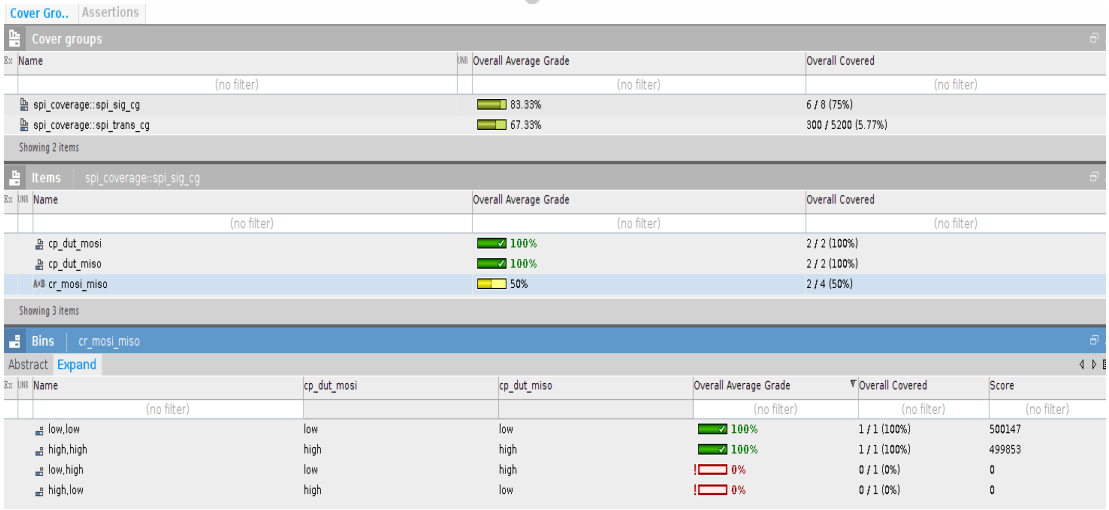


Figure 7 Code Coverage B

3.4 Functional Coverage - Signal Level

Signal level functional coverage is usually applied in the monitor component of the UVM test bench. Signal level exercise the checking at the DUT output pin level. At SPI signal level below three cover points are incorporated:

Figure 8 Functional Coverage – Signal Level



3.5 Functional Coverage – Transaction Level

Transaction level functional coverage is usually applied in the scoreboard component of the UVM test bench. Signal level exercises the checking at the DUT transaction class outputs. At SPI signal level below six cover points are incorporated:

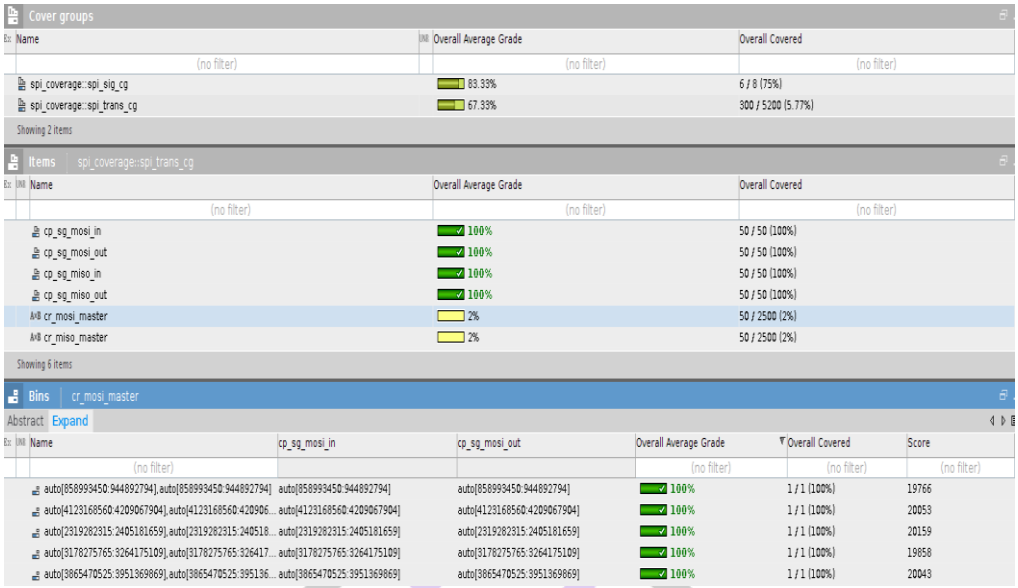


Figure 9 Functional Coverage – Transaction Level

CONCLUSION

A reusable SystemVerilog based UVM environment for a configurable SPI is created in this work. The verification environment was created around the AMBA System on Chip bus, making it even easier to combine both the core and verification environments. The testbench can be configured to meet unique attributes thanks to the configuration feature. The testbench allows us to track and validate the whole data transmission between both the master and slave for varying widths and metadata.

To make the SPI master core verification as complete as possible, an SPI slave model was constructed. A link between the testbench components and the device under test was also successfully created using an AMBA BFM. Basic read and write capabilities are provided by the AMBA BFM. Functional coverage was effectively incorporated into the testing environment, allowing for coverage-based verification criteria to be achieved

REFERENCES

[1] W. Ni and J. Zhang, “Research of reusability based on UVM verification” in 2015 IEEE 11th International Conference on ASIC (ASICON), Nov 2015, pp. 1–4.

[2] K. Fathy and K. Salah, “An Efficient Scenario Based Testing Methodology Using UVM” in 2016 17th International Workshop on Microprocessor and SOC Test and Verification, Dec 2016, pp. 57–60.

[3] P Rajashekar Reddy, P. Sreekanth, and K. Arun Kumar, “Serial Peripheral Interface-Master Universal Verification Component using UVM” International Journal of Advanced Scientific Technologies in Engineering and Management Sciences, vol. 3, p. 27, 06 2017.

[4] R. Prasad and C. S. Rani, “UART IP Core Verification by using UVM” IRF International Conference, 15 2016.

[5] P. Roopesh D, P. Siddesha K, and B. M. Kavitha Narayan, “RTL Design And Verification of SPI master slave using UVM” International Journal of Advanced Research in Electronics and Communication Engineering, vol. 4, p. 4, 08 2015.

[6] K. Aditya, M. Sivakumar, F. Noorbasha, and P. B. Thummalakunta, “Design and Functional Verification of A SPI Master Slave Core Using SystemVerilog” International Journal Of Computational Engineering Research, 05 2018.

[7] Chris Spear, "System Verilog for Verification, A guide to Learning the Testbench Language Features", 3rd ed., pp 295-332, 2012.

[8] T. Liu and Y. Wang, “IP design of universal multiple devices SPI interface” in Anti-counterfeiting, Security and Identification (ASID), 2011 IEEE International Conference on. IEEE, 2011, pp. 169–172.

[9] D. Ahlawat and N. K. Shukla, “DUT Verification through an Efficient and Reusable Environment with Optimum Assertion and Functional Coverage in SystemVerilog” International Journal of Advanced Computer Science and Applications, vol. 5, no. 4, 2014.

[10] N. Gopal, “SPI Controller Core: Verification,” SSRG International Journal of VLSI & Signal Processing, vol. 2, 09 2015.

[11] Z. Zhou, Z. Xie, X. Wang, and T. Wang, “Development of verification environment for SPI master interface using SystemVerilog” in Signal Processing (ICSP), 2012 IEEE 11th International Conference on, vol. 3. IEEE, 2012, pp. 2188–2192.

[12] J. Francesconi, J. A. Rodriguez, and P. M. Julian, “UVM based testbench architecture for unit verification” in Nano electronics, Technology and Applications (EAMTA), 2014 Argentine Conference on. IEEE, 2014, pp. 89–94.

[13] A. K. Swain and K. Mahapatra, "Design and verification of WISHBONE bus interface for System-on-Chip integration" in India Conference, 2010 Annual IEEE 2010, pp. 1–4.
[14] A. K. Oudjida, M. L. Berrandjia, A. Liacha, R. Tiar, K. Tahraoui, and Y. N. Alhoumays, "Design and test of general-purpose SPI Master/Slave IPs on OPB bus" in Systems Signals and Devices (SSD), 2010 7th International Multi-Conference on. IEEE, 2010, pp. 1–6.
[15] Mahendra.B.M and Ramachandra.A.C, "Bus Functional Model Verification IP Development of AXI Protocol," International Conference on Engineering Technology and Science, vol. 3, 02 2014.