UVM methodology based functional Verification of SPI Protocol

UVM methodology based functional Verification of SPI Protocol

Aman Kulkarni[1], **S M Sakthivel**[2]

[1,2] School of Electronics Engineering, Vellore Institute of Technology, Chennai, India

Email:[1]Aman.kulkarni55@gmail.com, [2]sakthivel.sm@vit.ac.in

Abstract. The scalability and complexity nature of the integrated circuit design makes the verification process more complicated and time-consuming. Therefore, in the present modern-day SOC's there is a strong need for verification architectures with increased reusability and easy accessibility. The UVM methodology-based verification architecture with reusable components is one of the widely accepted test bench architectures for carrying out such functional verification. This paper presents a UVM methodology based functional verification of the SPI protocol core with a dedicated architecture. First the SPI core is modeled using Verilog RTL. Then using the reusable components in UVM + System Verilog environment, the SPI core is verified under two modes such as i) SPI communication with wishbone interface and ii) SPI Master-Slave communication.

1. Introduction

Most of the SOC consists of many peripherals such as analog to digital converters, registers, memories, digital to analog converters, etc. Therefore, there is a need for transmission and reception of data among the several connected peripherals inside the SOC. Serial peripheral interface (SPI) is one of the most commonly used serial protocols for both inter-chip and intrachip for low/medium speed data-stream transfers. It is used to communicate between a processor and other devices like external EEPROMs, DACs, ADCs, etc. [1]. In the world of communication protocols, SPI is often considered as "little" communication protocol. It is important not to forget the purpose of each protocol. Ethernet, USB, and SATA are meant for "outside the box communications" and data exchanges between whole systems while SPI is aptly suited for communication between integrated circuits for low/medium data transfer speed with on-board peripherals [2], [3]. In SPI the data exchange takes place between the master and the slave device [4]. During the case of a device transmitting a data, the incoming data must be read before an attempt to transmit again. An exchange of data always takes place between the devices. In SPI protocol, a device cannot be just a transmitter only device or a receiver only device [3]. The master device controls the clock line SCK and the data exchange between the devices are controlled by SCK clock line as shown in Figure 1.

In this research work, the entire architecture is modeled using the Universal Verification Methodology (UVM) Class Library which provides the building blocks needed to develop reusable verification components and test environments [5,6]. Further, the rest of the paper is organized as follows: section 2, presents a discussion on SPI protocol overview and its working. Section 3 discusses the Proposed Universal Verification Methodology followed in generating the testbench

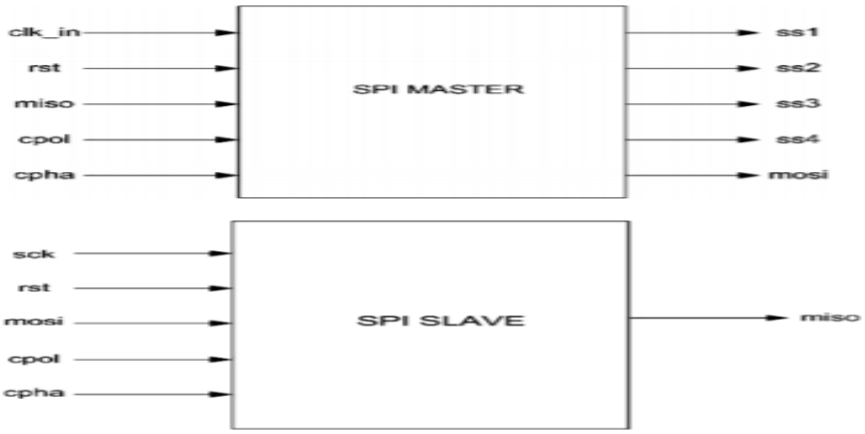

Figure 1. Block diagram of the SPI Master and Slave

architecture for functional verification, following this the section-IV presents the registers used in the SPI protocol. Finally, in section 5, the results and discussion of the implemented SPI protocol verification methodology is presented.

2. SPI Protocol Overview

In this section, the SPI protocol architecture and its operation are explained. Then the working principle of SPI in different modes of configuration and wishbone interface communication structure are also elaborated.

2.1. Serial Peripheral Interface.

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard named by Motorola that operates in full-duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines. Sometimes SPI is called a "four-wire" serial bus, contrasting with three-, two-, and one-wire serial buses. It is a synchronous serial data link that operates in full-duplex (signals carrying data go in both directions simultaneously).

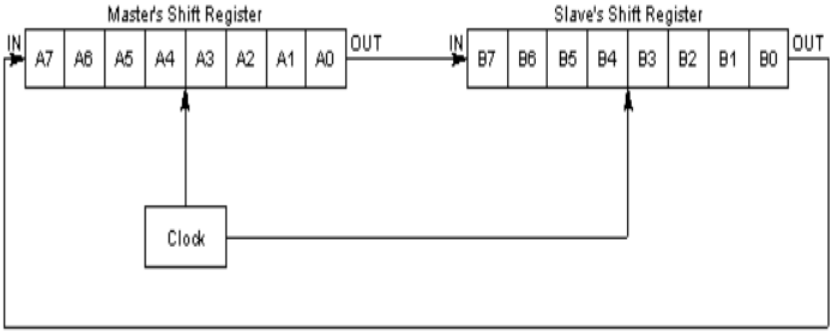

Figure 2. Data Shifting between SPI Master and Slave

In SPI, if the communication is initialized, then the master configures the system clock to different clock frequencies to connect with different slaves. According to the chip select bit and clock signal values, the communication is established in synchronized mode between the components. For establishing the synchronized communication, the shift register mode based data exchange action, as shown in Figure 2, is carried out to establish the communication. In this configuration, the SPI shift register configuration acts as a ring oscillator to configure the different operating frequencies. Thus by

this clock toggling mechanism and slave select option, the master communicates and transfers the data with different slaves.

During the master-slave communication, the SPI specifies four signals: clock (SCK1); master data output, slave data input (SI1); master data input, slave data output (SO1); and chip select (CS) for establishing the data connection between different slaves as shown in Figure 3. Here the SCK1 is generated by the master and input to all slaves. SI1 carries data from master to slave. SO1 carries data from slave back to the master. The SPI bus can operate with a single master device and with one or more slave devices. If a single slave device is used, the SS pin may be fixed to logic low if the slave permits it.

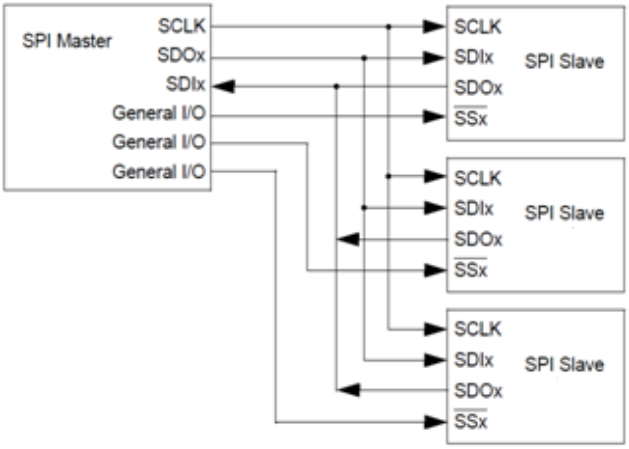

Figure 3. SPI single master communicating with multiple slaves

The SPI bus can operate with a single master device and with one or more slave devices. If a single slave device is used, the SS pin may be fixed to logic low if the slave permits it. From the above diagram, we can justify that data can be passed from one master to multiple slaves depends on the activation of the slave selection signal until that full-duplex communication is in existence. The master generates slave select signals using general-purpose discrete input/output pins or other logic. A pair of parameters called clock polarity (CPOL) and clock phase (CPHA) determines the edges of the clock signal on which the data are driven and sampled. Each of the two parameters has two possible states, which allows for four possible combinations, all of which are incompatible with one another.

Similar to the SPI master-slave communication, the SPI establishes the data transfer between processor using the wishbone interface called as ITNERCON. In this data communication the connection and data transfer ins established using the i/o ports of the wishbone and with both master & slave. During this communication, the Wishbone-SPI uses handshake protocol, arbitration strategies to avoid contention-free transfer between the bus and the linked processor.

During the data transmission, the SPI bus interface involves our logic signals lines, namely Master Out Slave In (MOSI), Master in Slave Out (MISO), Serial Clock (SCLK) and Slave Select (SS). Master Out Slave In (MOSI) for establishing a synchronized transfer between the master and slave. It is also responsible for the transmission of data in uni/bi-direction from master to slave. The four signals are as follows:

- MISO - Master in Slave out.
- MOSI – Master out Slave in
- SS- Slave Select
- SCLK- Serial Clock

The operation of the respective ports is given as follows:

- Master Out Slave In (MOSI) - It is responsible for the transmission of data in one direction from master to slave.
- Master in Slave Out (MISO) - The MOSI is a unidirectional signal line and configured as input signal line in a master device and as an output signal line in a slave device.
- 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. Further during the data transmission, the SPI master and slave are configured accordingly then with respect to the data configuration and control signals, the transfer is initialized in 8, 16 and 32 bits wide.

3. Proposed Verification Environment of SPI Protocol

In the proposed verification environment of the SPI protocol, first, the SPI core is modelled using the Verilog RTL code. Then the respective test bench reusable components are generated using the UVM codes based on the verification test scenarios [7-9]. The constructed verification environment of the SPI protocol is shown in Figure 4. The proposed UVM methodology-based verification environment consists of the following components as UVM_TEST, UVM_Environment, UVM_Agent, UVM_Generator, UVM_Driver, UVM_Monitor, UVM_Scoreboard, UVM_Checker, and Interface.

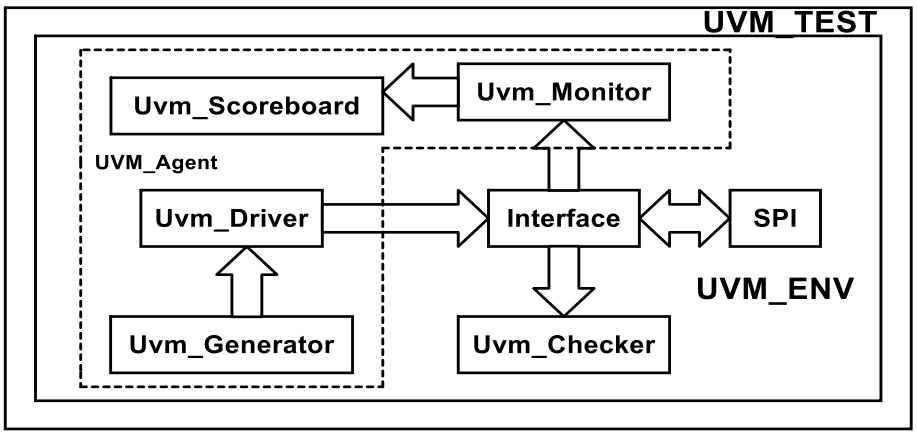

Figure 4. Proposed UVM methodology-based Verification Architecture of SPI protocol

A brief description of all the testbench components is given below:

- UVM TEST is the topmost class. It is responsible for configuring the testbench. Secondly, to initiate the test bench components construction process by building the next level down in the hierarchy. Third is to launch the stimulus by starting the sequence.
- UVM ENV Or Environment is a container component for grouping higher-level components like agents and scoreboard.
- UVM Sequence defines the sequence in which the data items need to be generated and sent or received to or from the driver.
- UVM Driver is responsible for driving the packet level data inside sequence_item into pin level i.e., to the DUT.
- UVM Sequence defines the sequence in which the data items need to be generated and sent or received to or from the driver.
- UVM Generator is responsible for generating the UVM sequence data packets or the sequence_item to the driver or vice versa.
- UVM Monitor observes pin level activity on interface signals and converts into packet level, which is sent to components such as scoreboards.

- UVM Scoreboard receives data items from monitors and compares with expected values. The expected values can be either original reference values or generated from reference model.
- UVM Agent groups the uvm_components specific to an interface or protocol.

4. Registers used in the SPI protocol during data transmission

In the SPI protocol, master_slave core uses several registers as mentioned below for data transmission with individual functionalities as stated below

- ASS: If this bit is set, ss_pad_o signals are generated automatically. This means that the slave select signal, which is selected in SS register, is asserted by the SPI controller when transfer is started by setting CTRL[GO_BSY] and is de-asserted after the transfer is finished.
- IE: If this bit is set, the interrupt output is set active after a transfer is finished. The Interrupt signal is de-asserted after a Read or Write to any register.
- LSB_Register: When LSB bit is set to 0x1, the least significant bit is sent first on the line (bit TxL[0]), and the first bit received from the line will be put in the least significant bit position in the Rx register (bit RxL[0]). When this bit is cleared, the MSB is transmitted /received first (CHAR_LEN field in the CTRL register selects which bit in TxX/RxX register).
- Tx_NEG_Register: The system programmer has the ability to control the format of the asynchronous data communication exchange by using the Line Control Register (LCR). This is an 8-bit register.
- Rx_NEG Register: When Rx_NEG bit is set, the miso_pad_i signal is received on the falling edge of a sclk_pad_o clock signal, or otherwise, the miso_pad_i signal is received on the rising edge of sclk_pad_o.
- GO_BSY Register: Writing 0x1 to this bit starts the transfer and remains set during the transfer, automatically cleared after the transfer is finished. Writing 0x0 to this bit has no effect.
- CHAR_LEN Register: This field specifies the number of bits to be transmitted in one transfer. Can send up to 64 bits in one transfer.

BITS	31:14	13	12	11	10	9	8	7	6:0
ACCESS	R	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
NAME	Reserved	ASS	IE	LSB	Tx_NEG	Rx_NEG	GO_BSY	Reserved	CHAR_LEN

Control and Status register

Figure 5. Control status register format used in the communication of SPI core for both master_slave mode and wishbone interface interaction

In the data transmission process, the SPI establishes the communication between the master & slave and also with the wishbone interface with the help of the control and status register bit values [10], as shown in Figure 5. Depending on the values in the status register, the corresponding communication mode is checked and verified for each of the transactions in the SPI core. Thus, with the help of the status and control register, the communication failure, bottlenecks and errors are corrected automatically in the SPI communication.

5. Proposed Verification Environment of SPI Protocol

In this section, the simulation results and discussion of the proposed verification architectures are discussed in detail. First, the SPI core and wishbone interface are modeled using the Verilog HDL. Then the reusable verification testbench architecture is constructed using the UVM base class library functions using system Verilog language. During the functional simulation, each UVM components are registered with the UVM factory settings and executed in the predefined UVM phases. The functional verification is carried out for the following two verification scenarios as: i) SPI

Master_Slave communication and ii) SPI_ Wishbone communication. For the respective test cases, the test environment is configured and the sequence is generated by the UVM_Generator. The generated sequence is driven by the UVM_Driver and the respective outputs are monitored and checked by UVM_Monitor and UVM_Checker. The two respective functional simulations is carried out using the opensource simulation environment called EDA playground. During the simulation in the open source environment, the SPI modeled codes and UVM_testbench architecture are executed under UVM 1.2 library support using the tools Mentor_Questa and EP wave. The verification of the functionality for the aforementioned respective test scenarios are carried out in such a way that the transmission is occurring in posedge and reception is performed in negedge of the clock signal. Also, the individual test case scenario is explained briefly as follows:

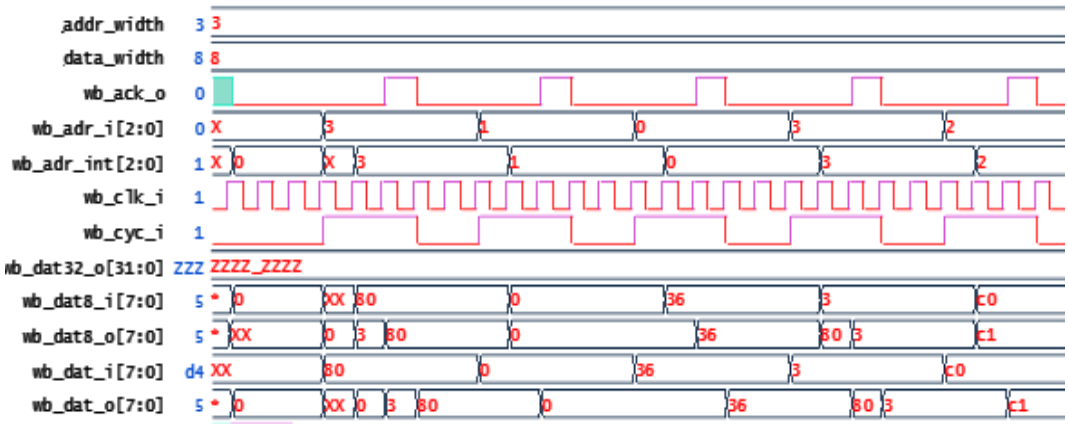

Figure 6. Functional simulation response of SPI Master_Slave communication

5.1. SPI Master-Slave communication

In this mode, the communication between the SPI master and slave is carried out in a synchronized manner using the clock signal values. During the initial phase of the communication, the Master_Slave control registered is configured accordingly, then with the value of the flag registers of transmission and reception, the signal values are tracked. As the read and write operation is carried out in the same buffer, the flag register for transmit and receive will have opposite data values for the corresponding operation. in this data transaction, the master configures the divider register and slave select register accordingly for the error-free communication. After checking the status register values, the control signal values are asserted, thereby initializing the data transactions between SPI master and slave. Thus, in the developed verification environment for every clock cycle, the generated data transactions are verified & checked by driver, monitor and checker components. The sample functional simulation response of the SPI Master_Slave communication is shown in the Figure 6.

5.2. Wishbone to SPI communication

In this mode of data transactions, the SPI initializes the read, write and reset operation on the bus interface with respect to the wishbone protocol. During the data communication, the write data cycle involves strobe, write enable and select signals to assert the WISHBONE to the corresponding address. That is another word, the master asserts the slave with appropriate address and data at the same time on the bus. Then the data transaction is identified by an acknowledgment from the corresponding slave. Upon receiving the acknowledge single, the master frees the bus by suspending the current signal operation in the clock cycle. Then in the next cycle, the terminated operation is initialized and executed, thereby establishing the communication between wishbones to SPI in this mode of operation.The sample simulated function output response of the WISHBONE to SPI communication is shown in Figure 7.

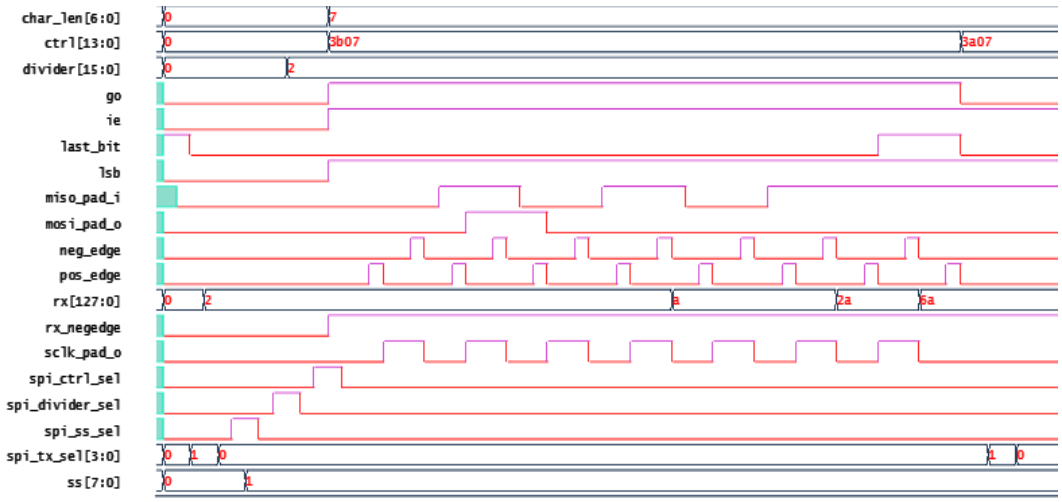

Figure 7. Functional simulation response of WISHBONE -SPI communication

6. Conclusion

This research work presents a UVM methodology based functional verification of SPI protocol using Mentor Questa and EP Wave in the EDA playground cloud. In the verification process, initially, the SPI core protocol was modeled using Verilog HDL, then the verification environment is constructed using the system Verilog based UVM 1.2 base class libraries. Then the functionality of the SPI is verified for two test scenarios, such as SPI Master_Slave and WISHBONE_SPI communication. For this verification, the reusable verification components are generated using UVM base class libraries and the sequence input are generator using random transactions. The generated testbench enables the verification and validation of the full-duplex data transfer between the master core and slave core for various character lengths and data formats, respectively. During the two modes of verification, the read & write functionalities are checked with respect to the posedge and negedge of the clock cycle for every transaction simultaneously. Further, the proposed verification environment can also be used as verification IP in SOC architecture for performing the functional verification.

References

[1] Ni W and Zhang J 2015 Research of reusability based on UVM verification *IEEE 11th Int. Conf. on ASIC* pp 1–4.
[2] Rajashekar Reddy P, Sreekanth P and Arun Kumar K 2017 Serial peripheral interface-master universal verification component using UVM *Int., Journal of Advanced Scientific Technologies in Engineering and Management Sciences* **3** p 27.
[3] Prasad R and Rani C S 2016 UART IP core verification using UVM *IRF Int. Conf.*
[4] Aditya K, Sivakumar M, Noorbasha F and Thummalakunta P B 2018 Design and functional verification of a SPI master slave core using system verilog *Int. Journal of Computational Engineering Research*.
[5] Roopesh P D, Siddesha P K and Kavitha Narayan B M 2015 RTL design and verification of spi master-slave using UVM *Int. Journal of Advanced Research in Electronics and Communication Engineering* **4** p 4.
[6] Mahesh G and Sakthivel S M 2015 Verification of memory transactions in axi protocol using system verilog approach *Proc. Of Int. conf. on Communication and Signal Processing* 0860-0864
[7] Swetha S and Sakthivel S M 2018 Design and verification of AMBA AXI3 protocol *Proc. of Int. conf. on VLSI Design: Circuits, Systems and Applications Lecture Notes in Electrical Engineering* 247-259
[8] Shyam S and Sakthivel S M 2019 Implementation and verification of rgb to grayscale

converter ip using system verilog *Int. Journal of Innovative Technology and Exploring Engineering* **8(7)** 645-652

[9] Murali.M, Umadevi.S, Sakthivel.S.M 2017 Verififcation IP for AMBA AXI Protocol using System Verilog, International Journal of Applied Engineering Research **12(17)** 6534-6541

[10] Liu T and Wang Y 2011 IP design of universal multiple devices SPI interface *Anti Counter feiting, Security and Identification Int. Conf. on. IEEE* pp. 169–172.