Serial Peripheral Interface-Master Universal Verification Component using UVM

P. Rajashekar Reddy, Assistant Professor
CVR College of Engineering College, ECE, Hyderabad, India
Email: raju.sheker@gmail.com
P.Sreekanth,Assistant Professor
CVR College of Engineering College, ECE, Hyderabad, India
Email: sreekanth.isoft@gmail.com
K.Arun Kumar,Assistant Professor
CVR College of Engineering College, ECE, Hyderabad, India
Email: arun.katkoori@gmail.com

Abstract— System level verification with scalable and reusable components provides a solution for current complex SOC verification. UVM class library provides the building blocks needed to quickly develop reusable and well constructed verification components. In this work, a verification environment using UVM is developed for SPI-Master. SPI protocol is commonly used for communication in Integrated Circuits.

Index Terms—Universal Verification Methodology (UVM), Serial Peripheral Interface(SPI).

I. Introduction

Based on the way the data is transmitted between Integrated Circuits and On-board Peripherals, interfaces are divided into two types, namely serial interface and paralled interface. Serial communication is a common method of transmitting data between a computer and a peripheral device. Serial communication transmits data one bit at a time, sequentially, over a single communication line to a receiver. One of the well known Serial interfaces is the Serial Peripheral Interface (SPI). SPI was first developed by Motorola Semiconductor. SPI interfaces are usually full duplex in nature and operate as master-slave. SPI bus consists of four signals master out slave in (MOSI), master in slave out (MISO), serial clock (SCK), and active-low chip select (CS).

The paper is based on the development of verification environment using UVM. The UVM itself is a library of base classes which facilitate the creation of structured testbenches using code which is open source and can be run on any SystemVerilog IEEE 1800 simulator. In UVM, 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.

The Importance of verification is:

- 70% of design effort goes to verification.
- Verification is on the critical path.
- Verification time can be reduced through abstraction.
- Using abstraction reduces control over low level details.
- Verification time can be reduced through automation.

- Randomization can be used as an automation tool.

II. SERIAL PERIPHERAL INTERFACE

The SPI module allows a full duplex, synchronous, serial communication between the MCU and peripheral devices. Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.

The serial data transfer involves 4 serial signals – Serial clock (SCLK), Master in Slave out (MISO), Master out Slave in (MOSI) and Slave Select (SS). Serial clock generates clock based on the master clock. The SPI operates in 4 different modes, based on the data transmitting and receiving on rising or falling edge of the serial clock. The 4 different modes are controlled the two registers, namely – Clock Phase register (CPHA) and Clock Polarity register (CPOL). The CPHA register decides the clock edge at which the data to be transmitted. The CPOL register decides the clock edge at which the data to be sampled.

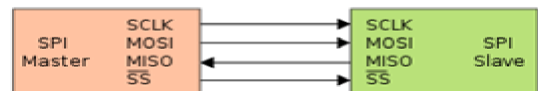

Figure 1: SPI Schematic diagram

- Master Out Slave In (MOSI) - The MOSI line is configured as output in a master device and as an input in a slave device. It is one of the two lines that transfer serial data in one direction, with the most significant bit sent first.
- Serial Clock (SCK) - The serial clock is used to synchronize data movement both in and out of the device through its MOSI and MISO lines. The Master and Slave devices are capable of exchanging a byte of information during a sequence of eight clock cycles. Since SCK is generated by the master device, this line becomes an input on a slave device.
- Slave Select (SS_bar) - The slave select input line is used to select a slave device. It has to be low prior to data transactions and must stay low for the duration of the transaction.

- Master In Slave Out (MISO) - The MISO line is configured as an input in a master device and as an output in a slave device. It is one of the two lines that transfer serial data in one direction, along with the most significant bit sent first. The MISO line of a slave device is placed in the high-impedance state if the slave is not selected.

III. SPI DATA TRANSMISSION

The SPI has four modes of operation. The clock polarity is specified by the CPOL control bit, which selects an active high or active low clock. The clock phase (CPHA) control bit selects one of the two fundamentally different transfer formats.

- CPOL-SPI Clock Polarity Bit

To transmit data between SPI modules, the SPI modules must have identical CPOL values. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state.
1 = Active-low clocks selected. In idle state SCK is high.
0 = Active-high clocks selected. In idle state SCK is low.

- CPHA- SPI Clock Phase Bit

This bit is used to select the SPI clock format. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state.
1 = Sampling of data occurs at even edges (2, 4, 6) of the SCK clock.
0 = Sampling of data occurs at odd edges (1, 3, 5) of the SCK clock.

The communication is initiated by the master all the time. The master first configures the clock, using a frequency, which is less than or equal to the maximum frequency that the slave device supports. The master then selects the desired slave for communication by pulling the chip select (SS) line of that particular slave-peripheral to low state. Data transfer is organized by using Shift register with some given word size such

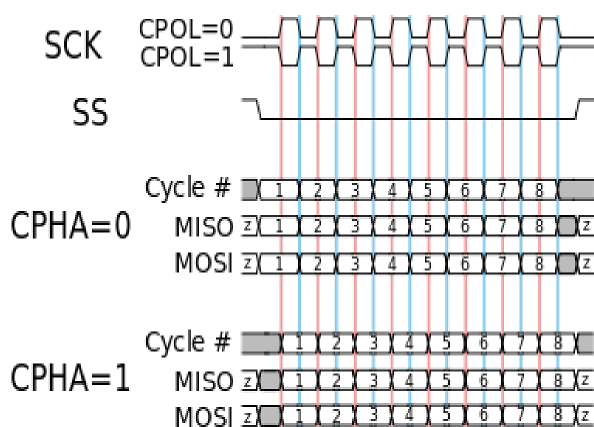

Figure 2: Timing diagram of different SPI modes

as 8- bits in both master and slave. While master shifts register value out through MOSI line, the slave shifts data in to its shift register and sends data to master from slave by MISO line

IV. UNIVERSAL VERIFICATION METHODOLOGY(UVM)

To verify the functionality of the design, a Verification environment is created in Universal Verification Methodology (UVM). The UVM environment is based on System Verilog. Based on the requirements for the project, the following points are considered while the verification architecture is built.

- Re-usability of the verification IP.
- Which building blocks the verification language can support.
- Controllability in generation of the stimulus.
- Next phase is building the Verification environment.
- Final phase would be to verify the DUT (RTL code) using the constructed verification environment.

A design and testbench are first compiled, and then the design and testbench are elaborated. Design and elaboration happen before the start of simulation at time-0. At time-0, the procedural blocks (initial and always blocks) in the top-level module and in the rest of the design start running. In the top-level module is an initial block that calls the run_test( ) task from uvm_top, which is the testcase we want to run. It is passed to simulator by passing the test name or by using “+UVM_TESTNAME=” switch. When run_test() is called at time-0, the UVM pre-run() global function phases (build(), connect(), end_of_elaboration(), start_of_simulation() all execute and complete. The run() phase is a taskbased phase that executes the entire simulation, consuming all of the simulation time. When the run() phase stops, the UVM post-run() global function phases (extract(), check(), report() all run in the last time slot before simulation ends. By default, when run_test() is done, $finish is called to terminate the simulation. Phases are a synchronizing mechanism for the environment. The UVM provides the following predefined phases for all uvm_components.

- **Build** - Depending on configuration and factory settings, create and configure additional component hierarchies.
- **Connect** - Connect ports, exports, and implementations.
- **End_of_elaboration** - Perform final configuration, topology, connection, and other integrity checks.
- **Start_of_simulation** - Do pre-run activities such as printing banners, pre-loading memories, etc.
- **Run** - Most verification is done in this time-consuming phase.
- **Extract** - Collect information from the run in preparation for checking.
- **Check** - Check simulation results against expected outcome.
- **Report** - Report simulation results.

A) UVM Verification Components

- **Design Under Test** - The design that is intended to be verified .This is generally RTL description in any of the HDL (Verilog, VHDL and System Verilog).This

completely describes the functionality of the design as well the features to be verified.

- **Interface** - Interface serves as the actual link between the design- under- verification and the verification environment. It is a SystemVerilog interface. The interface describes the pin - level description of the DUT. An interface is basically a bundle of nets or wires.
- **Virtual Interfaces** - It provide a mechanism for separating abstract models from the actual signals of the design. A virtual interface allows the same instance or the subprogram to operate on different parts of the design. It dynamically controls the set of signals associated with the subprogram, this allows passing the same data over all the components.
- **Transactions** - Interfaces represent the input to the DUT. The fields and attributes of transactions are derived from the transaction's specification. In a test, many data items are generated and those are sent to the DUT via driver. Generally data item fields are randomized using System Verilog constraints many number of tests can be created.
- **Agents** - Most DUTs have a number of different signal interfaces, each of which have their own protocol. The UVM agent collects together a group of uvm_components focused around a specific pin-level interface. The purpose of the agent is to provide a verification component which allows users to generate and monitor pin level transactions.
- **Sequence And Sequencer** - A sequence is the series of transaction and sequencer is used to control the flow of transaction generation. A sequence is extended from uvm_sequence class. uvm_sequencer does the generation of this sequence of transaction. Driver (extension of uvm_driver) takes the transactions from Sequencer and processes the packets of data or drives them to other component or to the DUT. It allows the addition of constraints to the data item generated in the sequence, thus bringing forth the corner cases.
- **Driver** - Driver is defined by extending uvm_driver. Driver takes the transactions from the sequencer by using seq_item_port. These transactions will be driven to DUT as per the interface signal specifications. Then it sends the transaction to scoreboard using uvm_analysis_port. Task for resetting DUT and configuring the DUT are also declared here. An instance of the driver class is created in the environment class and the sequencer is connected to it.
- **Monitor** - A monitor is a passive entity that samples DUT signals but doesn‘t drive them. A monitor collects transactions (data items), extracts events, performs checking and coverage, Optionally prints trace information, checking typically consists of protocol and data checkers to verify that the DUT Output meets the protocol specification. Coverage is collected in the monitor. It is implemented by extending the uvm_monitor class and an instance is created in the environment for hooking it up with DUT signals.
- **Scoreboard** - Scoreboard is implemented by extending uvm_scorboard. Scoreboard has 2 analysis imports. One is used to for getting the packets from the driver and other from the receiver. Then the packets are compared and if they don't match, then error is asserted. Compare function of transaction class is used for comparison.
- **Environment** - Environment class is used to implement verification environments in UVM. It is extension on uvm_env class. The testbench simulation needs some systematic flow like building the components, connection the components, starting the components etc. uvm_env base class has methods formalize the simulation steps. All the methods inside environment class are declared virtual. Virtual interface is created in the environment and all other virtual functions of environment class are extended. Our environment is the top level of the class based part of the testbench.
- **Testcases -** The uvm_test class defines the test scenario for the testbench for the DUT and is specified in the test. Testcase contains the instance of the environment class. This testcase creates an Environment object and defines the required test specific functionality. Verification environment contains the declarations of the virtual interfaces. These virtual interfaces are pointed to the physical interfaces which are declared in the top module. These virtual interfaces are made to point to physical interface in the testcase.
- **Top Module -** SystemVerilog interface instance is created in this module. DUT instance is created and hooked up with the interface instance. Clock generator is implemented here. run_test method is called from here. The test name can be implicitly passed or can be passed as a command line argument during simulation.

V. RESULTS

The below shown figures are the individual mode output of the SPI with that of different 8-bit data for each mode.

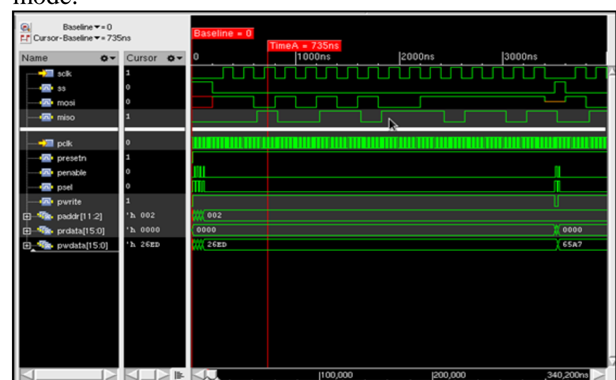

Figure 3: Output of SPI Mode 1

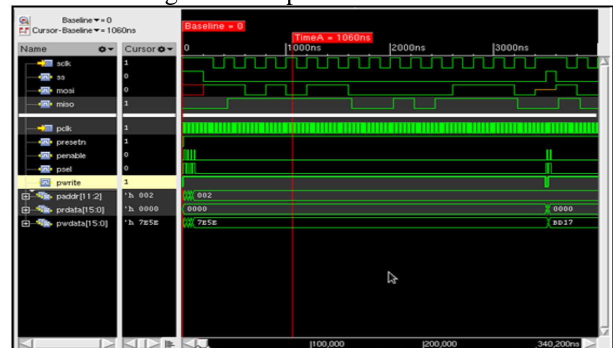

Figure 4: Output of SPI Mode 2

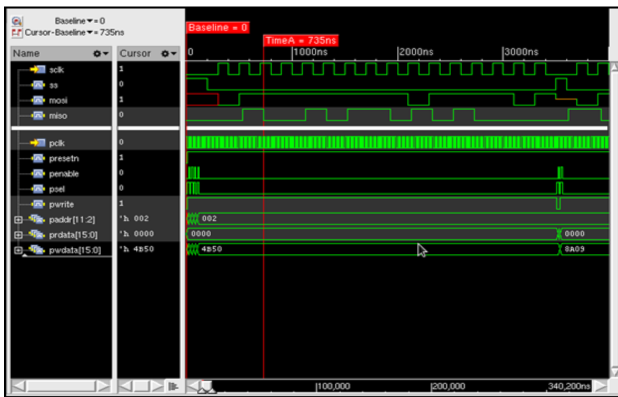

Figure 5: Output of SPI Mode 3

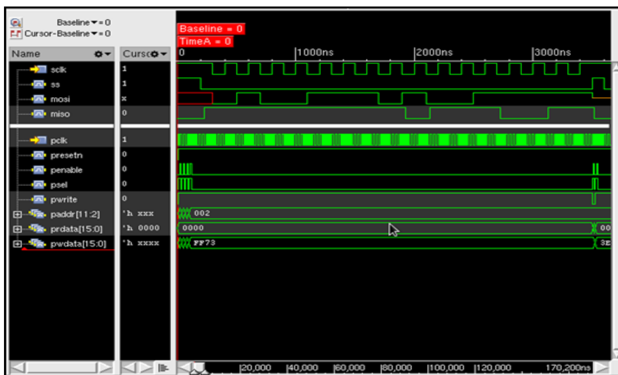

Figure 6: Output of SPI Mode 4

As per the SPI protocol the 16 bits data are transferred between the SPI master and slave. As seen, during simulation that the 16 bits data transfer takes 16 serial clock pulses. Also, the data transmission between the SPI master and slave is full duplex. The 16 bits data transfer occurs in MISO and MOSI serial signals. The proposed verification environment applies constrained random technique to fulfill the configuration of verification environment and DUT.

VI. CONCLUSION

In this paper, a uniform verification environment for SPI master interface is developed with UVM. The proposed multi-layer testbench is comprised of APB driver, SPI slave, scoreboard, which are implemented with OOPs concept. Furthermore, constrained random technique is applied. The verification result provides good evidence for the effectiveness of the proposed verification environment.

REFERENCES

[1] Zhili Zhou, Zheng Xie, Xin'an Wang and Teng Wang, "Development of verification Environment for SPI Master Interface Using SystemVerilog", 978-1-4673-2197-6/12/$31.00 ©2012 IEEE.

[2] Tianxiang Liu "IP Design of Universal Multiple Devices SPI Interface" IEEE.978-1-61284-632-3, 2011.

[3] M.Sandya1, K.Rajasekhar, "Design and Verification of Serial Peripheral Interface", International Journal of Engineering Trends and Technology- Volume 3 Issue 4- 2012.

[4] Juan Francesconi, J. Agustin Rodriguez, Pedro M. Julian, 2014. UVM Based Testbench Architecture for Unit Verification. ISBN: 978-987-1907-86-1 IEEE Catalog Number CFP1454E-CDR.

[5] Alexander W. Rath, Volkan Esen and Wolfgang Ecker, 2014.ATransaction-OrientedUVM-BasedLibraryfor Verification of Analog Behavior, IEEE- 978-1-4799- 2816-3, pp 806-811.

[6] K.Aditya, M. Sivakumar, Fazal Noorbasha and T. Praveen Blessington "Design and Functional Verification of A SPI Master Slave Core Using System Verilog" International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, May 2012.

[7] Motorola Inc., "SPI Block Guide V03.06," February 2003.

[8] Ccelleraorganization,"UniversalVerificationMethodology (UVM) 1.1 Class Reference", June 2011.