UVM VERIFICATION OF AN SPI MASTER CORE

Mr.MVD Pavan Kumar[1], Mrs. D.N.V.S. VIJAYA LAKSHMI[2]
[1]Assistant Professor, International School of Technology and Sciences for Women, Rajanagaram, Andhra Pradesh-533294.
[2]Associate Professor, International School of Technology and Sciences for Women, Rajanagaram, Andhra Pradesh-533294.

ABSTRACT:

In today's world, more and more functionalities in the form of IP cores are integrated into a single chip or SOC. System-level verification of such large SOCs has become complex. The modern trend is to provide pre-designed IP cores with companion Verification IP. These Verification IPs are independent, scalable, and reusable verification components. The SystemVerilog language is based on object-oriented principles and is the most promising language to develop a complete verification environment with functional coverage, constrained random testing and assertions. The Universal Verification Methodology, written in SystemVerilog, is a base class library of reusable verification components. This paper discusses a Universal Verification Methodology based environment for testing a Wishbone compliant SPI master controller core. A multi-layer testbench was developed which consists of a Wishbone bus functional model, SPI slave model, driver, scoreboard, coverage analysis, and assertions developed using various properties of SystemVerilog an the UVM library. Later, constrained random testing using vectors driven into the DUT for higher functional coverage is discussed. The verification results shows the effectiveness and feasibility of the proposed verification environment.

Keywords: SOC, IC, SV, UVM, DUT.

INTRODUCTION

The rapid development of modern integrated circuits not only increased the complexity of integrated circuit (IC) design, but also made the IC verification equally challenging. Around 70% to 80% of the entire design cycle time is allotted to verification, and traditional verification methodologies are no longer able to support current verification requirements [1]. In 2002, the Accellera Systems Initiative released SystemVerilog (SV) a a unified hardware design and verification language. SystemVerilog language was an amalgamation of constructs from different languages such as Vera, Super Log, C, Verilog and VHDL languages. Moreover, in 2005 IEEE standardized

(1800-2005) SystemVerilog. SystemVerilog supports behavioral, register transfer level, and gate level descriptions. SystemVerilog also supports testbench development by the inclusion of object-oriented constructs, cover groups, assertions, constrained random constructs, application specific interface to other languages [2]. Universal Verification Methodology (UVM) is a standardized verification methodology for testbench creation an is derived form the Open Verification Methodology (OVM), and also inherits some features from Verification Methodology Manual (VMM). Use of the UVM standard enables an increase in verification productivity by creating a reusable verification platform and verification components. The verification results of this work show the effectiveness and feasibility of the proposed verification environment [3] System on Chip (SoC) is used for intelligent control feature with all the integrated components connected to each other in a single chip. To complete a full system, every SoC must be linked to other system components in an efficient way that allows a faster error-free communication. Data communication between core controller modules and other external devices like external EEPROMs, DACs, ADCs. is critical. Different forms of communication protocols exist such as high throughput protocols like Ethernet, USB, SATA, PCI-Express which are used for data exchanges between whole systems. The Serial Peripheral Interface (SPI) is often considered as light weight communication protocol. The primary purpose of the protocol is that it is suited for communication between integrated circuits for low and medium data transfer rates with onboard peripherals and the serial bus provides a significant cost advantage.

MAIN AIM:

The major contributions if this work include:

1. Research the SPI sub-system architecture, the Universal Verification Methodology, and SystemVerilog.
2. Development of a WISBONE bus function model acting as an interface between the test bench and the SPI master device under test (DUT) and SPI slave model in order to make the verification closed loop testing.
3. Build hierarchical testbench components using UVM libraries and SystemVerilog constructs, constrained random stimulus, coverage and assertions.
4. Verify transmission of data with different character width and data formats.

SURVEY OF RESEARCH

SPI protocol is one of the widely used serial protocols used in a SoC compared to other protocols such UART and I2C simply because SPI can operate in higher bandwidth and throughput [4]. SPI Protocol typically provides communication between the hosts side microcontroller and slave devices. It is widely used owing to fewer control signals to operate with [5]. At the host side, the specific SPI core studied in this work acts like a WISHBONE compliant slave device. The SPI master core controller consists of three main parts, Serial shift interface, clock generator and WISHBONE interface. The SPI core controller has five 32-bit registers which can be configured through the WISHBONE interface. The serial interface consists of slave select lines, serial clock lines, as well as input and output data lines. The data transfers are full duplex in nature and number of bits per transfer is programmable [6]. It is possible to have high speed SPI Master/Slave Implementation of range 900 – 1000 MHz. The core can be designed with greater ways to control SPI-bus such as the flexibility of handling two slaves at a time. One important feature is configured by programming the control register of the core through which the SPI module can be made to either operate in master or slave mode. During operation, the SPI status register gives information such as the current position of the data transfer operation, whether the data transfer has completed or not, etc. [7]. Another key feature is the flexibility of designing the SPI Interface IPs for multiple devices using parameterization method. Advanced design techniques, such as Time Sharing Multiplex (TSM), is used to automatically identify the master/slave devices and achieve multi-master devices. Using TSM the disadvantage of communication among multiple devices are overcome [8]. Owing to the increasing complexity of the modern SoC, the verification has become more challenging. In fact 70% of the product development time is spent on complex SoC verification. Reducing the verification effort is the key for time to market challenge. In order to cater to such growing complexity advanced verification methodologies are employed. IP verification requires in depth functional coverage with constraint random simulation technique. Various components such as coverage monitors and scoreboards are used for this purpose [9]. For a communication protocol like the SPI communication protocol, it has to be verified as per the design specifications. Applying

constrained random technique for higher functional coverage provides effective verification result [10]. For many years, EDA vendors have been proposing newer verification methodologies and languages. For any system level verification methodology and language to be successful, the key lies in the scalability and reusability of the verification components developed. SystemVerilog with object-oriented programming is considered as one of the most promising techniques for high level function verification for current complex SOC designs. SystemVerilog provide complete verification environment, with direct and constrained random generation, assertion based verification and coverage driven metrics [11].

SYSTEM DESIGN

Hardware description languages are tools used by engineers to specify abstract models of digital circuits to translate them into real hardware, as the design progresses towards completion, hardware verification is performed using Hardware verification languages like SystemVerilog. The purpose of verification is to demonstrate the functional correctness of a design. Verification is achieved by means of a testbench, which is an abstract system that provides stimulus to the inputs of design under test (DUT). Functional verification shows that design implementation is in correspondence to the specification. Typically, the testbench implements a reference model of the functionality that needs to be verified and compare the results from that model with the results of the design under test. The role of functional verification is to verify if the design meets the specification but not to prove it [16]. The traditional approach to functional verification relies on directed tests. Verification engineers conceive and apply a series of critical stimulus directly to the device under test, and check if the result is the expected one. This approach produces quick initial results because little effort is required for setting up the verification infrastructure. But as design complexity grows, it becomes a tedious and time-consuming task to write all the tests needed to cover 100% of the design. Random stimuli help to cover the unlikely cases and expose the bugs. However, in order to use random stimuli, the test environment requires automating process to generate random stimulus, there is a need of a block that predicts, keeps track of result and analyses them: a scoreboard. Additionally, functional coverage is a process used, to check what cases of the random stimulus were covered and what states of the design

have been reached. This kind of testbench may require a longer time to develop, however, random based testing can actually promote the verification of the design by covering cases not achieved with directed tests.

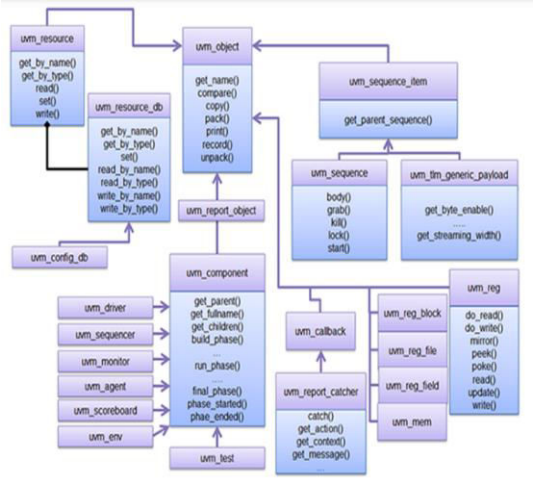

Fig.1. UVM hierarchy.

WISHBONE Interface

The WISHBONE System-on-Chip Interconnection Architecture shown in Figure 4.1 for portable and flexible IP Cores enables a design methodology for use with semiconductor IP cores. The WISHBONE interface alleviates System-on-Chip integration problems and results in faster design reuse by allowing different IP cores are connected to form a System-on-Chip. As defined, the WISHBONE bus uses both MASTER and SLAVE interfaces as part of the architecture. IP cores with MASTER interfaces initiate bus cycle transactions, and the participating IP cores with SLAVE interfaces can receive the designated bus cycles transactions. MASTER and SLAVE IP cores communicate through an interconnection interface called the INTERCON. The INTERCON is best thought of as a cloud that contains circuits and allows the communication with SLAVEs. INTERCON includes Point-to-point interconnection, Data flow interconnection, Shared bus interconnection and Crossbar switch interconnection [6]. WISHBONE Bus protocols include the implementation of an arbitration mechanism in centralized or distributed bus arbiters. The bus contention issue during the configuration of WISHBONE bus protocol is settled with the help of a Handshaking protocol and through the deployment of various arbitration schemes such as TDMA, Round Robin, CDMA, Token Passing, Static Priority etc. These strategies are applied based on the specific application in WISHBONE Bus.

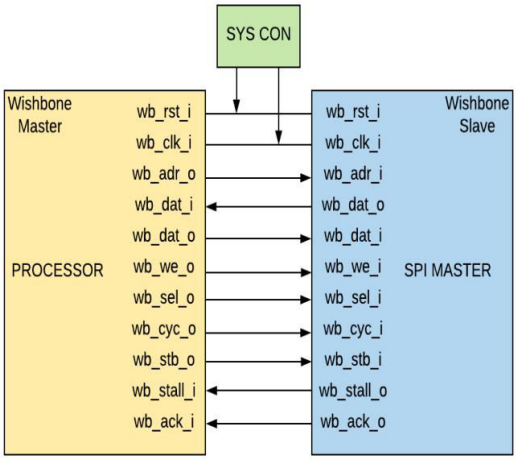

Fig.2. Wishbone Interface.

• wb_clk_i: All internal WISHBONE logic are sampled at the rising edge of the wb_clk_i clock input.

wb_rst_i: wb_rst_i is active low asynchronous reset input and forces the core to restart. All internal registers are preset, to a default value and all state-machines are set to an initial state.

• wb_int_o: The interrupt request output is asserted back to the host system when the core needs its service.

• wb_cyc_i: When the cycle input wb_cyc_i is asserted, it indicates that a valid bus cycle is in progress. It needs to become true on (or before) the first wb_stb_i clock and stays true until the last wb_ack_o. The logical AND function of wb_cyc_i and wb_stb_i indicates a valid transfer cycle to/from the core. This logic is usually taken care of by the bus master.

• wb_stb_i: The strobe input wb_stb_i is true for any bus transaction request. While wb_stb_i is true, the other wishbone slave inputs wb_we_i, wb_addr_i, wb_data_i, and wb_sel_i are valid and reference the current transaction. The transaction is accepted by the slave core any time when wb_stb_i is true, and at the same time, wb_stall_o is false.

Port	Width	Direction	Description
wb_clk_i	1	Input	Master clock input
wb_rst_i	1	Input	Asynchronous active low reset
wb_int_o	1	Output	Interrupt signal request
wb_cyc_i	1	Input	Valid bus cycle
wb_stb_i	1	Input	Strobe/core select
wb_adr_i	32	Input	Address bit
wb_we_i	1	Input	Write enable
wb_dat_i	32	Input	Data input
wb_dat_o	32	Output	Data output
wb_ack_o	1	Output	Normal bus termination
wb_stall_o	1	Output	Stall communication

Serial Peripheral Interface

A Serial Peripheral Interface (SPI) module allows synchronous, serial and full duplex communication between a Microcontroller unit and peripheral devices and was developed by Motorola in the mid 1980s. Figure represents the structural connection between master and slave core. The SPI bus is usually used to send and receive data between microcontrollers and other small peripherals units such as shift registers, sensors, SD cards, etc.

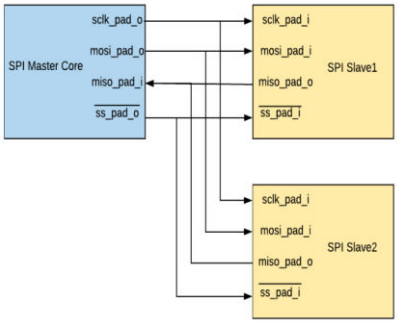

Fig.2. SPI Protocol.

RESULTS EXPLANATION

The SPI master core is verified along with the SPI slave model. Initially, the SPI master and slave have configured appropriately (for example at the master end no. of bits-32, transmit-posedge, receive-negedge). The basic idea of the verification is to send data from both master and slave ends. And after the transfer is completed, verify the exchanged data at both the ends. The Figure shows the test bench module approach. Below each of the components is explained.

Test top The top-level module is responsible for integrating the test bench module with the device under test. This module instantiates two interfaces, one for the master and another for the slave. Then the master interface is wired with SPI master core and likewise slave interface with SPI slave model. The top module also generates the clock and registers the interface into the config database so that other subscribing blocks can retrieve. Finally, the module calls the run_test function which starts to run the uvm_root.

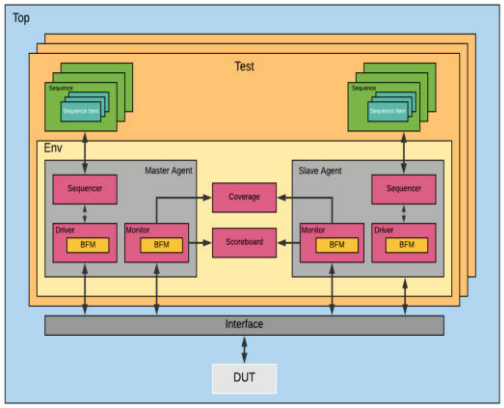

Fig.3. UVM Testbench model

spi_interface

The interface block declares all the WISHBONE slave logic signals. The communication with the master and slave core happens through WISHBONE bus function model. The block also samples the input and output signals using two different clocking blocks, one for driver and another for the monitor. Clocking block helps to synchronize all logic signals to a particular clock. It also helps to separate the timing details from the structural, functional and procedural elements of the test bench.

spi_package The package class typically includes all SystemVerilog testbench components and make the scope available to the entire build process.

spi_test

The test class is created by extending the uvm_test class. Then the class is registered to factory using

uvm_component_utils macro. In the build phase, the lower level SPI environment class is created and configured. Instead of the run phase, the test class contains two of the twelve scheduled phases. Reset phase typically resets the device under test. The main phase used to create the sequences and start running the sequencer for the required number of tests. Whenever there needs to be a blocking phase execution, phase raise objection is invoked and like to unblock phase drop objection is used.

spi_environment

SPI environment is a container component containing the agent and scoreboard. It is created using uvm_env virtual base class. In the build phase components within the environment are instantiated. And in the connect phase, the connections are made between components.

spi_agent

Currently, there is only one agent container component is used within the project. The SPI agent container is configured as an active component. SPI agent is created using uvm_agent virtual base class. In the build phase, the agent builds Sequencer, Driver and Monitor components. In the connect phase, the driver and sequencer are connected.

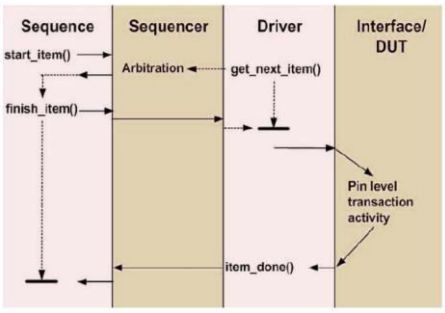

Fig.4. UVM Sequencer Driver Communication.

RESULTS

SPI Master Controller Synthesis Benchmarking The project aims to create a functional verification environment for SPI controller. For this purpose the IP core was reused from Opencores, but with some modification. The logic synthesis of the module was performed in the TSMC 180nm, 65nm and SAED 32nm technology. Area, Power and Timing of the final module were captured.

Type	Technology node	32 nm	65 nm	180 nm
Area	Sequential Area (μm^2)	2096.68	2520.35	18990.41
	Combinational Area (μm^2)	2527.97	2209.68	17071.08
	Buf/Inv Area (μm^2)	314.37	71.28	1862.78
	Total Area (μm^2)	5847.47	4730.03	36061.50
Power	Internal Power (μW)	32.59	47.34	335.80
	Switching Power (μW)	1.844	3.58	74.86
	Leakage Power (μW)	452.2	0.189	0.145
	Total Power (μW)	486.6	51.11	410.8
Timing	Slack (ns)	18.375	17.958	12.983
DFT Coverage		100%	100%	100%
Latency (Clock cycles)				

WISHBONE to SPI Master communication using BFM

The communication between the WISHBONE and SPI master is performed using WISHBONE bus

function model. The model mainly implements read, write and reset functionalities w.r.t WISHBONE B.3 protocol. In the below Figure shows the WISHBONE protocol. Initially when there is a write data is involved cycle, strobe and write enable signals along with select lines of WISHBONE are asserted to 0x1 by the bus master. The WISHBONE address and data at the same time is placed on the bus. The bus model waits until a receive acknowledgment from the slave is received. Then the bus master frees the bus by terminating the cycle signal to 0x0. For example, if the control register needs to be configured, then control register address 0x10 is sent along with the data value 0x2200, referred at reference 1 in the Figure.. Correspondingly, the SPI control select flag is selected, and in the next cycle, the value is written to the local control register of the device under test.

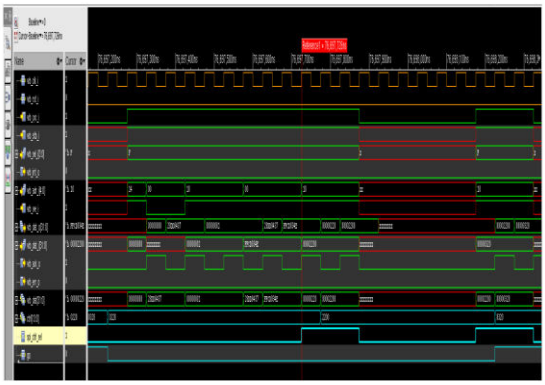

Fig.6. WISHBONE to SPI communication

These two flags should have opposite values to each other since the SPI read input and write output takes place at the same single buffer in a shift register fashion. The master also configures its divider register and slave select register. Once all SPI registers are initially set up, then go flag of the control signal is asserted, which starts the transfer. The testbench uses the flag transfer in progress to synchronize driver and monitor respective forever loop part. Finally as given in Figure after 32 clock cycles, the transfer in progress signal is de-asserted and thus informs the end of communication for the WISHBONE interface to collect the data.

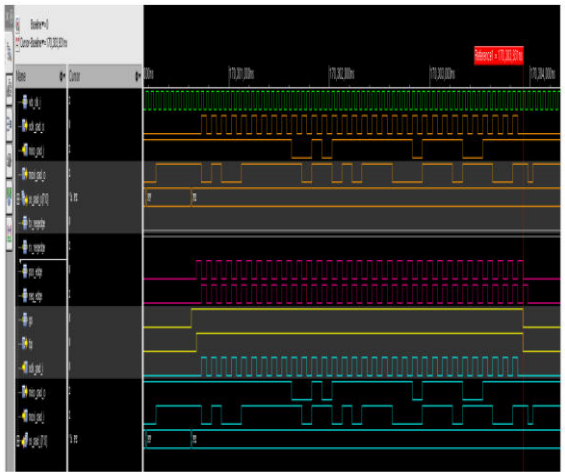

Fig.7. SPI Master - Slave communication

Functional coverage is essential to any verification plan, in the project it the coverage is retrieved using Cadence Integrated Metrics Centre tool. Functional coverage is a way to tell the

effectiveness of the test plan. Functional coverage infers results such if an end to end code checked if an important set of values corresponding to interface or design requirement and boundary conditions have been exercised or not. 100% Functional coverage combined with 100% Code coverage indicates the exhaustiveness of the verification plan coverage.

CONCLUSION

In this work, a reusable SystemVerilog based UVM environment is created for an SPI master core controller. The verification environment is built around WISHBONE System on Chip bus thus making both core IP, and verification IP easy to integrate. Configuration capability is provided to configure the testbench to suit different protocol characteristics. The testbench enables to verify and validate the full duplex data transfer between the master core and slave core for various character lengths and data formats respectively. An SPI slave model was created to enhance the SPI master core verification as end to end feasible. In addition, a WISHBONE BFM was successfully established to form the link between the testbench components and the device under test. The WISHBONE BFM provides basic read and write functionalities. Functional coverage was successfully integrated into the testing environment in order to achieve coverage driven verification metrics.

REFERANCES

[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 (MTV), 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] N. Anand, G. Joseph, S. S. Oommen, and R. Dhanabal, “Design and implementation of a high speed Serial Peripheral Interface,” in 2014 International Conference on Advances in Electrical Engineering (ICAEE), Jan 2014, pp. 1–3.

[8] T. Liu and Y. Wang, “IP design of universal multiple devices SPI interface,” in AntiCounterfeiting, 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.