UART IP CORE VERIFICATION BY USING UVM

[1]RENDUCHINTHALA H H S S PRASAD, [2]CH. SANTHI RANI

[1]M.Tech Student in DMS SVH College of Engineering, Machilipatnam, Krishna District, A.P., India
[2]Professor and HOD, ECE Department, DMS SVH College of Engineering, Machilipatnam, Krishna District, A.P., India
E-mail: [1]rhhssprasad@gmail.com, [2]santhirani.ece@gmail.com

Abstract— The objective of this paper is to verify the Universal Asynchronous Receiver/Transmitter (UART) protocol using Universal Verification Methodology (UVM). The UART allows serial communication between two systems running in different operating-frequencies, by converting parallel data into serial form and transmitting serially in frames. The frames are collected in the receiver by receiving bit-by-bit of a frame. Once the frame is collected, it converts the serial data into parallel data. This UART IP core is designed compatible with the industry standard National Semiconductors 16550A device. The key features of this paper are using an 8-bit WISHBONE interface, 16 bit FIFO in UART. RTL logic is written using Verilog HDL. It is verified using UVM test bench methodology. The main aim of this paper is to get 100% functional coverage by doing regression test cases. The UART also generates interrupts, which indicate errors, during transmission of data. The errors may arise due to mismatches in framing of transmitted data, parity-detection, etc. The operation of UART is simulated using Riviera Pro software. The result obtained in this paper is 100% functional coverage.

Keywords— UART,UVM,Wishbone interface,FIFO,IP Core.

I. INTRODUCTION

UART is a computer hardware device that is used to convert user's data to parallel and serial forms and vice versa. The word "Universal" is used to tell us that the format of the transmitted data and speed of the transmission are configurable. An UART is usually an Integrated Circuit(IC) which is used for serial communication over a computer or a peripheral device. UART's are now commonly used in microcontrollers.The universal asynchronous receiver/transmitter (UART) takes the input as bytes of data and transmits the individual bits in a sequential way.At the destination side, a second UART re-assembles the received bits into complete bytes. Each UART contains a shift register, which is basically used to convert serial to parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is less costly when compared with parallel transmission through multiple wires.

The Communication may be done in three different ways.

(A) Half-Duplex: - Both the UARTs can communicate with each other, but not simultaneously. The communication is only one-direction at a time.
(B) Full-Duplex: - Both the UARTS can communicate with each other simultaneously.
(C) Loop-Back mode: - This mode is for checking the accuracy of the UART. One UART will transmit some data, and the same UART will receive the transmitted data.

II. DATA FORMAT

A. Format:

The data consists of Start bit, DATA, Parity bit and stop bit. The start bit is an Active Low signal and stop bit is an Active High signal.

B. Transmitter side: The UART transmits in the following format

1 START bit + data bits (5, 6, 7, 8) + 1 PARITY bit (optional) + STOP bit (1, 1.5, 2).

It transmits 1 START bit; 5, 6, 7, or 8 data bits, depending on the data width selection;

1 PARITY bit, if parity is selected; and 1, 1.5, or 2 STOP bits, depending on the STOP bit selection.

C. Receiver Side: TheUARTreceives in the following format

1 START bit + data bits (5, 6, 7, 8) +
1 PARIT bit (optional) + 1 STOP bit

It receives 1 START bit; 5, 6, 7, or 8 data bits, depending on the data width selection; 1 PARITY bit, if parity is selected; and 1 STOP bit.

III. WISHBONE INTERFACE

The WISHBONE System-On-Chip (SoC) Interconnect Architecture for Portable IP Cores is used as a portable interface in semiconductor IP cores. Its main purpose is to faster the design reuse to overcome system-on-chip integration problems. This is mainly used to create a common and logical interface between IP cores. It defines the standard data exchange between the IP Core modules. This improves the portability and reliability of the system, and results in faster time-to-market for the customer. WISHBONE interface itself is not an IP core. It is used as a specification for creating IP cores. OpenCores recommends the WISHBONE System-On-Chip Interconnect as the interface to all cores that require interfacing to other cores inside a chip (FPGA, ASIC, etc.).Wishbone is made to let designers combine several designs written in Verilog or VHDL or some other Hardware Description Languages (HDL) for Electronic Design

Automation(EDA). Wishbone provides a standard way for Design Engineers to combine these hardware logic designs. Wishbone interface can have 8, 16, 32, and 64-bit buses.

Table-1: Wishbone Interface Signals

Port	Width	Direction	Description
CLK	1	Input	Block's clock input
WB_RST_I	1	Input	Asynchronous Reset
WB_ADDR_I	5 or 3	Input	Used for register selection
WB_SEL_I	4	Input	Select Signal
WB_DAT_I	32 or 8	Input	Data Input
WB_DAT_O	32 or 8	Output	Data Output
WB_WE_I	1	Input	Write or Read Cycle Selection
WB_STB_I	1	Input	Specifies transfer Cycle
WB_CYC_I	1	Input	A bus cycle is in progress
WB_ACK_O	1	Output	Acknowledge of a transfer

IV. IP CORE

An IP (intellectual property) core is like a block of logic or data that is used as basic building blocks of Field Programmable Gate Array (FPGA) or Application-Specific Integrated Circuit (ASIC) for a product. IP cores are the basic elements of design-reuse. IP Cores are the growing Electronic Design Automation (EDA) industry trend towards the use of previously designed components repeatedly. Ideally, an IP core should be entirely portable - that is, able to easily be inserted into any vendor technology or design methodology. Universal Asynchronous Receiver/Transmitter (UART s),central processing units (CPU s), Ethernet controllers, and PCI express interfaces etc are all examples of IP cores.

V. BAUD GENERATOR

An external clock signal is given to the Processor's clock generator and it produces an UART input clock with a user defined frequency. The UART contains a programmable baud generator that takes an input clock from clock generator of the processor and divides it by a divisor latch value in the range between 1 and (2^{16} - 1) to produce a baud clock (BCLK). In this way the UART BCLK can be obtained from the input clock to the UART. The frequency of the BCLK is sixteen times (16×) the baud rate (each received or transmitted bit lasts 16 BCLK cycles) or thirteen times (13×) the baud rate (each received or transmitted bit lasts 13 BCLK cycles). When the UART is receiving, the bit is sampled in the 8th BCLK cycle for 16× over sampling mode and in the 6th BCLK cycle for 13× oversampling mode.

The formula to calculate the divisor is:

(A) If MDR.OSM_SEL=0

$$\text{Divisor} = \frac{\text{UART input clock frequency}}{16 * \text{Desired Baud rate}}$$

(B) If MDR.OSM_SEL=1

$$\text{Divisor} = \frac{\text{UART input clock frequency}}{13 * \text{Desired Baud rate}}$$

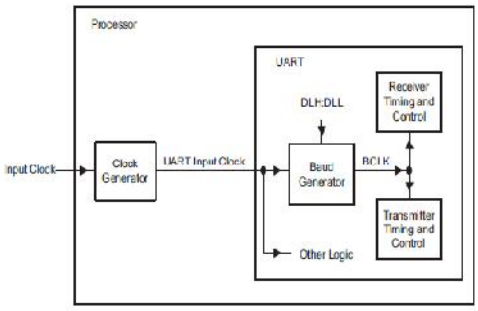


Figure-1: UART Clock Generation Diagram

VI. REGISTERS USED IN UART

A. INTERRUPT ENABLE REGISTER(IER)

The Interrupt Enable Register (IER) is used to enable or disable each type of interrupt request individually that can be generated by the UART. It is an 8-bit register. Each interrupt request that is enabled in the IER is forwarded to the CPU. It has both Read/Write access.

B. INTERRUPT IDENTIFICATION REGISTER(IIR)

The Interrupt Identification Register (IIR) is an 8-bit register which has read-only access. It shares the same address as of FIFO control register (FCR), which has write-only access. When an interrupt is generated and enabled in the Interrupt Enable Register (IER), the IIR shows that an interrupt is pending. The UART has an on-chip interrupt generation and prioritization capability that allows flexible communication with the CPU. The UART provides three priority levels of interrupts:

- Priority 1 - Receiver line status (highest priority)
- Priority 2 – Receiver data ready or receiver timeout
- Priority 3 – Transmitter holding register empty

C.FIFO CONTROL REGISTER (FCR): -

The FIFO Control Register (FCR) is a write-only 8-bit register at the same address as the interrupt identification register (IIR), which has read-only

access. Use the FCR to enable and clear the FIFOs and to select the receiver FIFO trigger level.

D. LINE CONTROL REGISTER (LCR): -

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. In addition, the programmer can retrieve, inspect, and modify the content of the LCR. This excludes the need for separate storage of the line characteristics in system memory.

E. LINE STATUS REGISTER (LSR): -

The Line Status Register (LSR) has the ability to provide information to the CPU regarding the status of data transfers. It has read-only access. Bits 1 through 4 can record the error conditions that produce a receiver line status interrupt.

F. DIVISOR LATCHES (DLL AND DLH):-

There are two 8-bit register fields (DLL and DLH), called divisor latch registers. They have the ability to store the 16-bit divisor for the generation of the baud clock in the baud generator. The latches are in the DLH and DLL. The DLH register holds the most-significant bits of the divisor, and the DLL register holds the least-significant bits of the divisor. These divisor latches should be loaded during the initialization process of the UART to ensure that the desired operation of the baud generator to get the BCLK.

VII. VERIFICATION PLAN

(A) **Agent:** The most important and basic element in UVM Architecture is the Universal Verification Component (UVC) or Agent. Because of Agent, the Test-bench of UVM is re-usable. Agent is an encapsulation of Driver, Monitor and sequencer. An UVM environment can consists of one or more agents. Agent can be configurable. There are two types of Agents-Active agent and Passive agent. If the agent is Active, then the agent will have all the driver, monitor and sequencer. But if the Agent is Passive, then the agent will have only Monitor.

(B) **Driver:** Driver is an active entity that emulates the logic that drives the DUT. It fetches data repeatedly from the sequencer. Driver has to drive the DUT according to the protocol using the virtual interface. Driver drives the data to DUV using an interface.

(C) **Monitor:** Monitor is a passive entity that can sample the DUT signals, but does not drives them. A monitor extracts signal information from the bus and translates the information into a transaction that can be made available to other components and to the test case writer.

(D) **Sequencer:** The sequencer in UVM just acts as a gateway between sequence and driver. This is the only "non-virtual" class in our UVM testbench architecture. The sequencer is also parameterized by transaction class. The sequencer takes the randomized data from sequence and passes it to the driver to which it is connected, thus it connects several sequences to driver.

(E) **Sequence:** The actual driven data is randomized in the sequence. From the sequence, this randomized data is given to driver via sequencer.

(F) **Test:** Test is a place where we start the sequences on sequencer. In base test we will set all the parameters of configuration database class according to our requirements. We will also get the interface set in top and again set it to the interface handles in local configuration database classes. The base test also creates the environment. All these things happen in build phase of base test. The further child tests will be made from this base test class. In child test we just create the handle of virtual sequence and start it on virtual sequencer, before starting the sequence an objection is raised and this objection is dropped again after starting the sequence. If we don't raise the objections, the simulator will think that there is no run phase to execute, so the simulator can jump directly to extract phase. The total number of raised objections should be equal to the number of dropped objections.

(G) **Environment:** The object of this class is created in test and is a most important component of UVM test bench. Environment creates agents, scoreboard, virtual sequencer etc. The environment makes connection between monitors and scoreboard. If it has virtual sequencer, here in environment we have to connect physical sequencers with the handles of physical sequencers in virtual sequencer.

(H) **Virtual Sequence/Virtual Sequencer:** To reduce the dependency of test case writer and TB developer, these two virtual sequence and virtual sequencer were developed.

(I) **Score-Board:** It compares the data sent from one side to another i.e. from master to slave and vice-versa. This also includes the cover groups which help in knowing the functional coverage. It gets the data from both write monitor and read monitor and compares both of them. Also it samples the signal values which have been covered.

(J) **DUV:** The Device which is being verified is called as a Device/Design under Verification (DUV). The DUV is driven with a group of inputs and the outputs are compared with the reference model in the score-board. The functionality of the Design is checked using Functional Coverage.

(K) **Configuration database:** It is reusable and efficient mechanism for organizing the configuration of the test bench. If we want to build re-usable test bench, we have to design Test bench such that the components are easily configurable. To build a re-usable test bench, the main thing is UVM Configuration. We have to use configuration API uvm_config_db.

VIII. COVERAGE:

Coverage is a generic term used to measure the progress to complete verification for DUV. The coverage tools gather information during the simulation and post process it to produce a progress report. We can use this report to look for coverage holes and then modify existing test cases or create new test cases to fill the holes. This iterative process continues until we are satisfied with the coverage level. There are two types of Coverage. They are:

A) **Code Coverage**: The easiest way to measure the Verification progress is with Code Coverage. Here we are measuring how many lines of the RTL code have been executed (Line coverage),which paths through the code and expressions have been executed (Path Coverage), which single variables have had the values 0 or 1(Toggle Coverage). We don't need to write any extra HDL Code. The tool captures our design automatically by analysing the source code and tool will add the hidden code to gather statistics about the code coverage. Code Coverage is used to measure the efficiency of verification process. Code coverage provides a quantitative measurement of the testing space. It describes the degree to which the source code of a DUV has been tested. Code coverage is simulator dependent. Code Coverage is also called as "Structural Coverage".

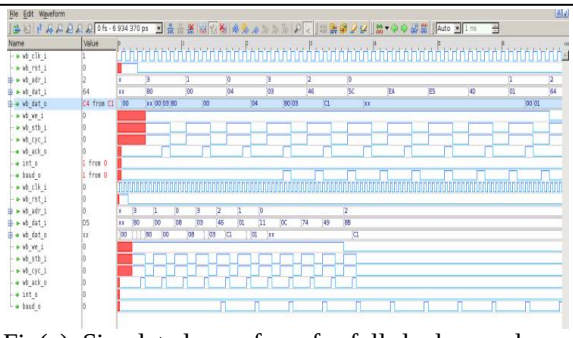
B) **Functional Coverage**: Code coverage does not know anything about what the design is supposed to do. There is no way to find what is missing in the code by the code coverage, but a functional coverage can catch the missing functionality in the Design code. The main goal of verification is to make sure that a design behaves correctly in its real environment. Functional Coverage is user specified to tie the verification environment to the design intent or functionality. Functional coverage tells us about whether all the functionalities given in specification are included in the RTL or not. Functional coverage is also called as "Specification Coverage".

IX. TEST CASES

Different test cases like half duplex mode, full duplex mode, loop back mode, Parity error, break interrupt, framing error, time out indication and over-run error are verified successfully.

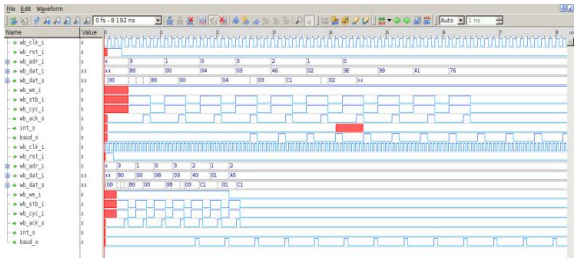
X. RESULTS

The Design is done using Verilog HDL and verified using UVM Test bench. The coverage is done using Riviera-Pro tool.



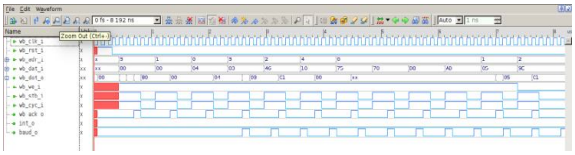
Fig(a). Simulated waveform for full-duplex mode

Here the address, data-in, data-out, enable signal, interrupt and baud-out of both UART's are shown. The input applied at UART-1 can be seen at output of UART-2 after interrupt goes low and input applied at UART-2 can be seen at output of UART-1 after interrupt goes low.



Fig(b). Simulated waveform for half-duplex mode

Here the address, data-in, data-out, enable signal, interrupt and baud-out of both UART's are shown. The input applied at UART-1 can be seen at output of UART-2 after interrupt goes low.



Fig(c). Simulated waveform for loop-back mode

Here the address, data-in, data-out, enable signal, interrupt and baud-out of UART-1 is shown. The input applied at UART-1 can be seen at output of the same UART-1 after interrupt goes low.

Covergroup	Hits	Goal / At Least	Status
TYPE /package uart_pkg\/uart_sb\/uart_cov_rd	100.000%	100.000%	Covered
INSTANCE <UNNAMED1>	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::RESET	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::WR_RD	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::STB	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::CYC	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::SEL	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::READ_ADDRESS	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::DATA_IN	100.000%	100.000%	Covered
TYPE /package uart_pkg\/uart_sb\/uart_cov_wr	100.000%	100.000%	Covered
INSTANCE <UNNAMED1>	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::RESET	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::WR_RD	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::STB	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::CYC	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::SEL	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::READ_ADDRESS	100.000%	100.000%	Covered
COVERPOINT <UNNAMED1>::DATA_IN	100.000%	100.000%	Covered

Fig(d). 100% Functional Coverage

Here, all the cover points are covered. The functional coverage obtained is 100%.

CONCLUSION

The functionality of UART IP Core has been verified successfully and different test cases are successfully implemented. The functional coverage obtained for this verification of UART IP Core is 100%.

REFERENCES

[1] Yi-yuan Fang, Xue-Jun-Chen, Design and simulation of UART serial communication module based on VHDL, In proceedings of the 3rd International Workshop on Intelligent Systems and Applications(ISA) ,May 2011

[2] Dipanjan Bhadra, Vikas S.Vij, Kenneth S.Stevens, A Low power UART Design based on asynchronous techniques, IEEE 56th International Midwest Symposium on Circuits and Systems ,pp 21-24,Aug. 2013

[3] He-Chun-Zhi, Xia Yin-Shui, Wang Lun-Yao, A Universal asynchronous receiver transmitter design, International conference on Electronics, Communication and Control, pp. 691-694,Sept. 2011

[4] R.Gallo,M.Delvai, W.Elmenreich, A. Steininger, Revision and verification of an enhanced UART, IEEE International workshop on Factory communication systems, pp. 315-318, Sept. 2004.

[5] Yongcheng Wang, Kefei Song, A New approach to realize UART, International conference on Electronic and Mechanical Engineering and Information Technology, vol.5, pp. 2749-2752, Aug. 2011.

[6] Garima Bandhawarkar Wakhle, Iti Aggarwal ; Shweta Gaba, Synthesis and Implement tation of UART using VHDL codes, International symposium on Computer, Consumer and Control,pp. 1-3, June. 2012

[7] Nithin Patel, Naresh Patel,. VHDL Implementation of UART with BIST Capability, Fourth International Conference on computing, Communications, and Networking Technologies,pp 1-5, July. 2013.

[8] Universal Verification Methodology (UVM) 1.1 User's Guide, May,2011

[9] UVM Cookbook, Mentor Graphics, Sept, 2013.

★★★