Design and Functional Verification of I2C Master Core using OVM

B. Santosh Kumar, L. Ravi Chandra, A. L. G. N. Aditya, Fazal Noor Basha, T. Praveen Blessington

Abstract :- This paper contrasts physical implementation aspects of the protocol through a number of recent Xilinx's FPGA families, showing up the protocol features are responsible of substantial area overhead and power overhead. These help designers to make careful and tightly tailored architecture decisions. These RTL coding is carried out for the I2C protocol using the HDL code. The verification methodology carries a important role in design of the VLSI, As the functional verification of the I2C is covered using Open Verification Methodology (OVM) which does not interfere with DUT. This verification method provides the I2C with fault free and useable for modern day applications. The OVM is carried using Questasim10.0b.

Index Terms:- I2C, FPGA, OVM, Functional verification, HDL.

I. INTRODUCTION

The I2C bus was developed in the early 1980's by Philips Semiconductors. Its original purpose was to provide an easy way to connect a CPU to peripherals in a chipset. Peripheral devices in embedded systems are often connected to the MCU as memory-mapped I/O devices, using the microcontroller's parallel address and data bus. This result in lots of wiring on the PCB's to route the address and data lines, not to mention a number of address decoders and glue logic to connect everything. In mass production electronic equipment, this is not acceptable. Furthermore, lots of control lines implies that the systems is more susceptible to disturbances by Electromagnetic Interference (EMI) and Electrostatic Discharge (ESD). The bus is generally accepted in the industry as a de-facto standard.

The Open Verification Methodology (OVM) is a documented methodology with a supporting building-block library for the verification of semiconductor chip designs. The initial version, OVM 1.0, was released in January, 2008, and regular updates have expanded its functionality. The latest version is OVM 2.1.2, released in January, 2011. The current release and all previous releases are available, under the Apache License, on the OVM World site. Verifying a design consists of two major parts: stimulus generation and an analysis of the designs response. Stimulus generation sets up the device and puts it in a particular state, then the analysis part actually performs the verification. The analysis portion of a test bench is made up of components that observe behavior and make a judgment whether or not the device conforms to its specification. Examples of specified behavior include functional behavior, performance, and power utilization. The process by which the analysis section makes its judgment starts with observing response activity in the device under test (DUT). This is done by one or more monitors that observe the signal-level activity on the DUT through a virtual interface(s). The monitor converts signal-level activity into TLM transactions, and broadcasts the transactions to interested analysis components using analysis ports which are connected to subscribers. These subscribers capture the transactions and perform their analysis.

II. I2C BUS PROTOCOL

The I2C bus is a multi-master bus. This means that more than one IC capable of initiating a data transfer can be connected to it. The I2C protocol specification states that the IC that initiates a data transfer on the bus is considered the Bus Master. Consequently, at that time, all the other ICs are regarded to be Bus Slaves. As bus masters are generally microcontrollers, let's take a look at a general 'inter-IC chat' on the bus. Let's consider the following setup and assume the MCU wants to send data to one of its slaves.Proposed Delay Element.

III. SLAVES CONNECTED TO MASTER

First, the MCU will issue a START condition. This acts as an 'Attention' signal to all of the connected devices. All ICs on the bus will listen to the bus for incoming data.

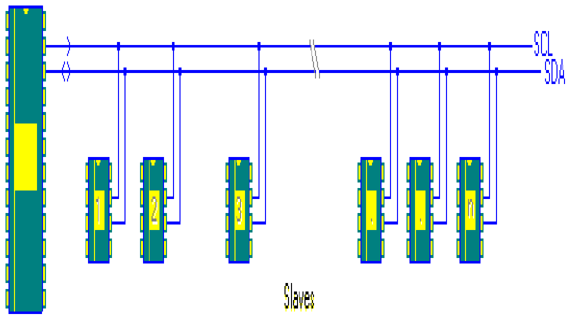


Figure 1: I2C bus Architecture

Manuscript received on April 26, 2012.

B. Santosh Kumar, M.tech VLSI, KL University, Vijayawada, India, Mobile No 9705965999, (e-mail:88santu@gamil.com).
L. Ravi Chandra, ECE, KL University Vijayawada, India, Mobile No 9849224324.
A. L. G. N. Aditya, M.tech VLSI, KL University, Vijayawada, India, Mobile No 9014936978, (e-mail:adityasind@gamil.com).
Fazal Noor Basha, ECE, KL University Vijayawada, India,(e-mail: fazalnoorbasha@kluniversity.in)
T. Praveen Blessington , ECE, KL University Vijayawada, India,(e-mail: praveentblessington@kluniversity.in)

Then the MCU sends the ADDRESS of the device it wants to access, along with an indication whether the access is a Read or Write operation. Having received the address, all IC's will compare it with their own address. If it doesn't match, they simply wait until the bus is released by the stop condition. If the address matches, however, the chip will produce a response called the ACKNOWLEDGE signal. Once the MCU receives the acknowledgment, it can start transmitting or receiving DATA. We have had several unique condition states on the bus in our example: START, ADDRESS, ACKNOWLEDGE, DATA , STOP.

IV. I2C BUS EVENTS: TRANSMITTING A BYTE TO A SLAVE

Once the start condition has been sent, a byte can be transmitted by the MASTER to the SLAVE. This first byte after a start condition will identify the slave on the bus (address) and will select the mode of operation.

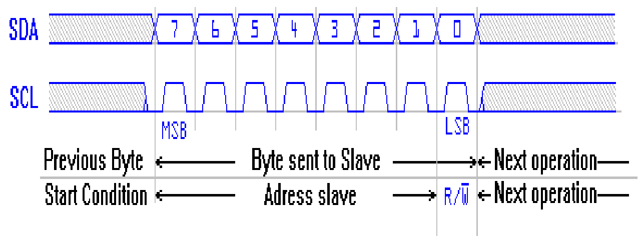


Figure 2: Timing analysis of the SDA and SCL

A number of addresses have been reserved for special purposes. One of these addresses is reserved for the "Extended Addressing Mode". Besides considering the limitation of I2c, that the number of available addresses was too small. Therefore, one of the reserved addresses has been allocated to a new task to switch to 10-bit addressing mode. If a standard slave doesn't initiate for the reception of reserved address.

I2C Bus Events: Receiving a byte from a slave

Once the slave has been addressed and the slave has acknowledged this, a byte can be received from the slave if the R/W bit in the address was set to READ (set to '1'). The protocol syntax is the same as in transmitting a byte to a slave, except that now the master is not allowed to touch the SDA line. Prior to sending the 8 clock pulses needed to clock in a byte on the SCL line, the master releases the SDA line. The slave will now take control of this line. The line will then go high if it wants to transmit a '1' or, if the slave wants to send a '0', remain low.

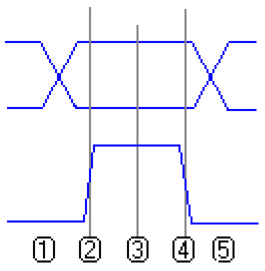


Figure 3: Clock diagram

All the master has to do is generate a rising edge on the SCL line (2), read the level on SDA (3) and generate a falling edge on the SCL line (4). The slave will not change the data during the time that SCL is high. (Otherwise a Start or Stop condition might inadvertently be generated) During (1) and (5), the slave may change the state of the SDA line.In total, this sequence has to be performed 8 times to complete the data byte. Bytes are always transmitted MSB first.

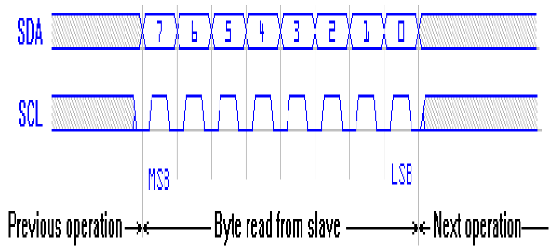


Figure 4: Analysis of SDA and SCL

I. I2C BUS EVENTS: THE START AND STOP CONDITIONS

Prior to any transaction on the bus, a START condition needs to be issued that enables the transmission/reception on the bus and control shifts towards the bus. After a message has been completed, a STOP condition is sent. This is the signal enables the idle state.

1. I2C Bus Events: Getting Acknowledge from a slave

When an address or data byte has been transmitted onto the bus then this must be acknowledged by the slave(s). In case of an address: If the address matches its own then that slave and only that slave will respond to the address with an ACK. In case of a byte transmitted to an already addressed slave then that slave will respond with an ACK as well. The slave that is going to give an ACK pulls the SDA line low immediately after reception of the 8th bit transmitted, or, in case of an address byte, immediately after evaluation of its address. In practical applications this will not be noticeable.

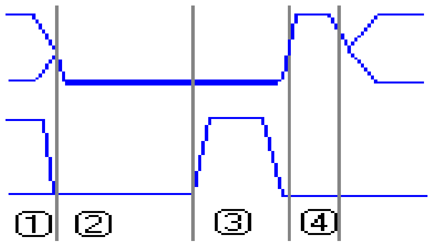


Figure 5: Timing diagram for SLAVES

V. I2C Bus Hardware

The bus physically consists of 2 active wires called SDA (data) and SCL (clock), and a ground connection. Both SDA and SCL are initially bi-directional. This means that in a particular device, these lines can be driven by the IC itself or from an external device. In order to achieve this functionality, these signals use open collector or open drain outputs (depending on the technology).

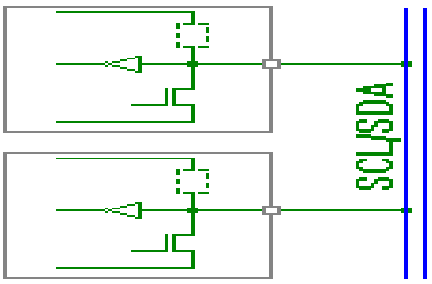


Figure 6: Basic diagram of I2C

The bus interface is built around an input buffer and an open drain or open collector transistor. When the bus is IDLE, the bus lines are in the logic HIGH state (note that external pull-up resistors are necessary for this which is easily forgotten). To put a signal on the bus, the chip drives its output transistor, thus pulling the bus to a LOW level. The "pull-up resistor" in the devices as seen in the figure is actually a small current source or even non-existent. If the bus is "occupied" by a chip that is sending a 0, then all other chips lose their right to access the bus this is referred as Built-in mastering technique. However, the open-collector technique has a drawback, too. If you have a long bus, this will have a serious effect on the speed you can obtain. Long lines present a capacitive load for the output drivers. Since the pull-up is passive, you are facing an RC constant which will reflect on the shapes of the signals. The higher this RC constant, the slower the bus. This is due to the effect that it influences the slew rate of the edges on the I2C bus, this undifferentiated the logic '0' and '1'.

At high speed the reflection is unavoidable can be so bad that "ghost signals" disturb your transmission and corrupt the data you transmit. The device developed by Philips consists of a twin charge pump. The moment the state changes, the dynamic resistor provide a large current (low dynamic resistance) to the bus, so it can charge the parasitic capacitor very quickly. Once the voltage has been increased the threshold level, the high current mode cuts out and the output current drops sharply. As long as the bus is kept low (transistor C is on), the charge pump is disabled because the gate of transistor B is kept low by transistor

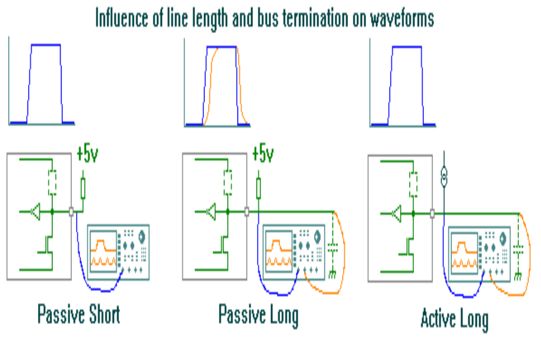


Figure 7: Influence of line length and bus termination on waveforms

As soon as the chip releases the bus (A and C turn off), the capacitor will start charging, drawing current trough all four of the resistors (1 - 4). The voltage drop over resistor 2 will cause the transistor B to turn on, shorting out resistor 3. Since resistor 3 is a relatively low value, the current will rise. At a certain point in time, the drop between transistor B's gate and source will not be big enough to keep it switched on. It will then switch off and the charge injection will stop. At that time, only the external pull-up resistor remains to overcome the charge leakage on the bus.

VI. I2C Bus Arbitration

The I2C bus is explained with one master in operation but in practical it was originally developed as a multi-master bus. When using only one master on the bus there is no real risk of corrupted data, except if a slave device is malfunctioning or if there is a fault condition involving the SDA / SCL bus lines. This situation changes with 2 MCU's:

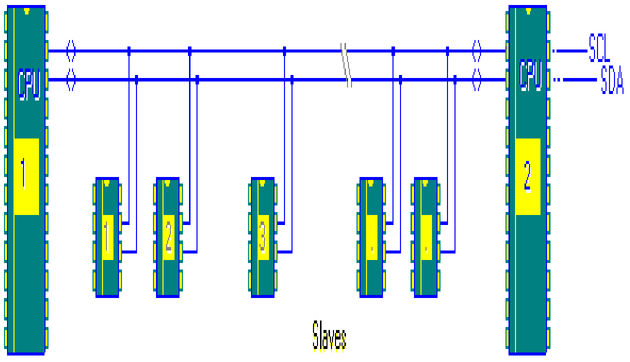


Fig 8 Master-Salve-Master with I2C bus

When MCU 1 issues a start condition and sends an address, all slaves will listen (including MCU 2 which at that time is considered a slave as well). If the address does not match the address of CPU 2, this device has to hold back any activity until the bus becomes idle again after a stop condition. As long as the two MCU's monitor what is going on the bus (start and stop) and as long as they are aware that a transaction is going on because the last issued command was not a STOP, there is no problem.

Let's assume one of the MCU's missed the START condition and still thinks the bus is idle, or it just came out of reset and wants to start talking on the bus which could very well happen in a real-life scenario. This could lead to problems. But the physical bus setup helps us out as the bus structure is a wired AND (if one device pulls a line low it stays low), you can test if the bus is idle or occupied. When a master changes the state of a line to HIGH, it MUST always check that the line really has gone to HIGH. If it stays low then this is an indication that the bus is occupied and some other device is pulling the line low. Therefore the general rule of thumb is: If a master can't get a certain line to go high, it lost arbitration and needs to back off and wait until a stop condition is seen before making another attempt to start transmitting.

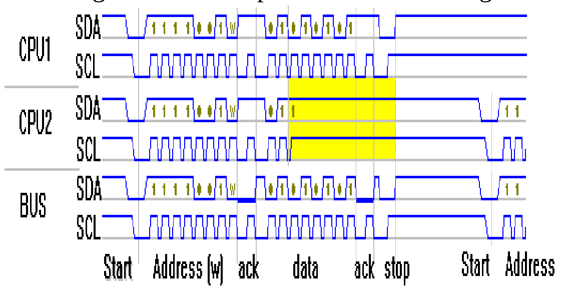


Figure 9: Analysis of SDA SCL

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

VII. CLOCK SYNCHRONIZATION

All masters generate their own clock on the SCL line to transfer messages on the I2C-bus. Data is only valid during the HIGH period of the clock. A defined clock is therefore needed for the bit-by-bit arbitration procedure to take place. Clock synchronization is performed using the wired-AND connection of I2C interfaces to the SCL line. This means that a HIGH to LOW transition on the SCL line will cause the devices concerned to start counting off their LOW period and, once a device clock has gone LOW, it will hold the SCL line in that state until the clock HIGH state is reached.

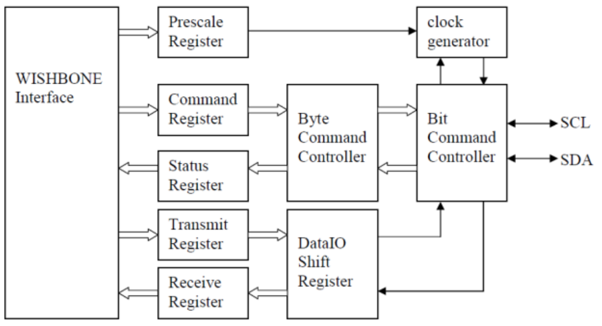


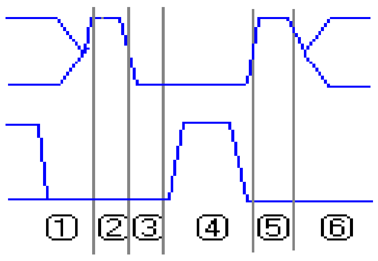
Figure 10: Block diagram of generic system using I2C

However, the LOW to HIGH transition of this clock may not change the state of the SCL line if another clock is still within its LOW period. The SCL line will therefore be held LOW by the device with the longest LOW period.

VIII. I2C BUS EVENTS

A. Giving Acknowledge to a slave

Upon reception of a byte from a slave, the master must acknowledge this to the slave device. The master is in full control of the SDA and the SCL line.



After transmission of the last bit to the master (1) the slave will release the SDA line. The SDA line should then go high (2). The Master will now pull the SDA line low (3) Next, the master will put a clock pulse on the SCL line (4). After completion of this clock pulse, the master will again release the SDA line (5). The slave will now regain control of the SDA line (6). Note: The above waveform is slightly exaggerated. You will not notice SDA going high in (2) and (5). A small spike might barely be visible.

If the master wants to stop receiving data from the slave, it must be able to send a stop condition. Since the slave regains control of the SDA line after the ACK cycle issued by the master, this could lead to problems. Let's assume the next bit ready to be sent to the master is a 0. The SDA line would be pulled low by the slave immediately after the master takes the SCL line low. The master now attempts to generate a Stop condition on the bus. It releases the SCL line first and then tries to release the SDA line - which is held low by the slave. Conclusion: No Stop condition has been generated on the bus. This condition is called a NACK : Not Acknowledge . Do not confuse this with No Acknowledge

B. No Acknowledge (from slave to master)

This is not exactly a condition. It is merely a state in the data flow between master and slave. If, after transmission of the 8th bit from the master to the slave the slave does not pull the SDA line low, then this is considered a No ACK condition.

C. Enhanced I2C (FAST mode)

In the FAST mode, the physical bus parameters are not altered. The protocol, bus levels, capacitive load etc. remain unchanged. However, the data rate has been increased to 400 Kbit/s and a constraint has been set on the level of noise that can be present in the system. The input of the FAST mode devices all include Schmitt triggers to suppress noise. The output buffers include slope control for the falling edges of the SDA and SCL signals. If the power supply of a FAST mode device is switched off, the bus pins must be floating so that they do not obstruct the bus.

IX. OPEN VERIFICATION METHODOLOGY

OVM provides the best framework to achieve coverage-driven verification (CDV). CDV combines automatic test generation, self-checking test benches, and coverage metrics to significantly reduce the time spent verifying a design. The purpose of CDV is to:

■ Eliminate the effort and time spent creating hundreds of tests.

■ Ensure thorough verification using up-front goal setting.

■ Receive early error notifications and deploy run-time checking and error analysis to

Simplify debugging. The CDV flow is different than the traditional directed-testing flow. With CDV, you start by setting verification goals using an organized planning process. You then create a smart test bench that generates legal stimuli and sends it to the DUT. Coverage monitors are added to the environment to measure progress and identify non-exercised functionality. Checkers are added to identify undesired DUT behavior. Simulations are launched after both the coverage model and testbench have been implemented. Verification then can be achieved. Using CDV, you can thoroughly verify your design by changing testbench parameters or changing the randomization seed. Test constraints can be added on top of the smart infrastructure to tune the simulation to meet verification goals sooner. Ranking technology allows you to identify the tests and seeds that contribute to the verification goals, and to remove redundant tests from a test-suite regression.

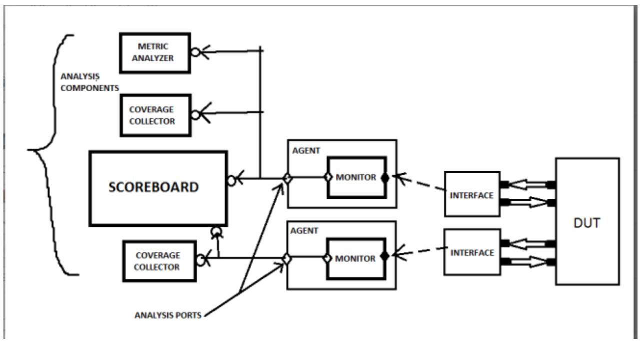


Figure 12: OVM test bench setup

CDV environments support both directed and constrained-random testing. However, the preferred approach is to let constrained-random testing do most of the work before devoting effort to writing time-consuming, deterministic tests to reach specific scenarios that are too difficult to reach randomly. Significant efficiency and visibility into the verification process can be achieved by proper planning. Creating an executable plan with concrete metrics enables you to accurately measure progress and thoroughness throughout the design and verification project. By using this method, sources of coverage can be planned, observed, ranked, and reported at the feature level. Using an abstracted, feature-based approach (and not relying on implementation details) enables you to have a more readable, scalable, and reusable verification plan.

OVC Overview

The following subsections describe the components of an OVC:

Data Item (Transaction)
Driver (BFM)
Sequencer
Monitor
Agent
Environment

Data Item:

Data items represent the input to the DUT. Examples include networking packets, bus transactions, and instructions. The fields and attributes of a data item are derived from the data item's specification. For example, the Ethernet protocol specification defines valid values and attributes for an Ethernet data packet. In a typical test, many data items are generated and sent to the DUT. By intelligently randomizing data item fields using System Verilog constraints, you can create a large number of meaningful tests and maximize coverage.

Driver (BFM):

A driver is an active entity that emulates logic that drives the DUT. A typical driver repeatedly receives a data item and drives it to the DUT by sampling and driving the DUT signals. (If you have created a verification environment in the past, you probably have implemented driver functionality.) For example, a driver controls the read/write signal, address bus, and data bus for a number of clocks cycles to perform a write transfer.

Sequencer:

A sequencer is an advanced stimulus generator that controls the items that are provided to the driver for execution. By default, a sequencer behaves similarly to a simple stimulus generator and returns a random data item upon request from the driver. This default behavior allows you to add constraints to the data item class in order to control the distribution of randomized values. Unlike generators that randomize arrays of transactions or one transaction at a time, a sequencer captures important randomization requirements out-of-the box. A partial list of the sequencer's built-in capabilities includes.

Monitor:

A monitor is a passive entity that samples DUT signals but does not drive them. Monitors collect coverage information and perform checking. Even though reusable drivers and sequencers drive bus traffic, they are not used for coverage and checking. Monitors are used instead. A monitor: Collects transactions (data items). A monitor extracts signal information from a bus and translates the information into a transaction that can be made available to other components and to the test writer.

■ Extracts events: The monitor detects the availability of information (such as a transaction), structures the data, and emits an event to notify other components of the availability of the transaction. A monitor also captures status information so it is available to other components and to the test writer.

■ Performs checking and coverage

Agent: Sequencers, drivers, and monitors can be reused independently, but this requires the environment integrator to learn the names, roles, configuration, and hookup of each of these entities. To reduce the amount of work and knowledge required by the test writer, OVM recommends that environment developers create a more abstract container called an agent. Agents can emulate and verify DUT devices. They encapsulate a driver, sequencer, and monitor.. Agents should be configurable so that they can be either active or passive. Active agents emulate devices and drive transactions according to test directives. Passive agents only monitor DUT activity.

Environment :

The environment (env) is the top-level component of the OVC. It contains one or more agents,as well as other components such as a bus monitor. The env contains configuration properties that enable you to customize the topology and behavior and make it reusable. For example, active agents can be changed into passive agents when the verification environment is reused in system verification.

X. RESULTS

The I2C is implemented using the VHDL with full duplex mode which allows the communication between the master and the salve through the handshaking protocol. When a slow slave is attached to the bus then problems may occur. This mechanism works on the SCL line only. The slave that wants the master to wait simply pulls the SCL low as long as needed. If the SCL gets stuck due to an electrical failure of a circuit, the master can go into deadlock and this can be handled by timeout counters. Another drawback is speed. The bus is locked at that moment.. Other masters cannot use the bus at that time either. his technique does not interfere with the previously introduced arbitration mechanism because the low SCL line will lead to back-off situations in other devices which possibly would want to "claim" the bus. So there is no real drawback to this technique except the loss of speed / bandwidth and some software overhead in the masters. You can use this mechanism between masters in a multi-master environment. As soon as you have three or more masters this is very handy rather than 2 masters. So the implementation is carried with three slaves and master.

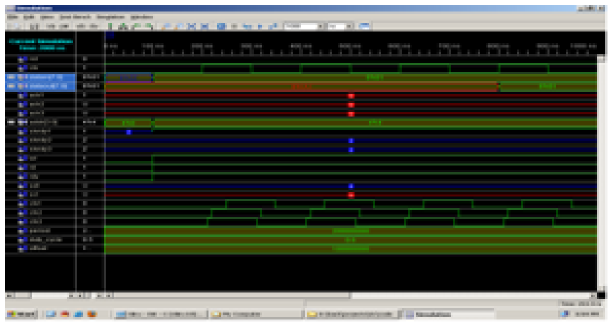


Fig 11 simulation of I2C bus

B. Functional Verifiaction of I2C using OVM

The functional verifiaction of the I2C is carried using the Questasim10.0b for the complete coverage of the RTL design of the I2C. The verification is carried for both the read and writre operation.

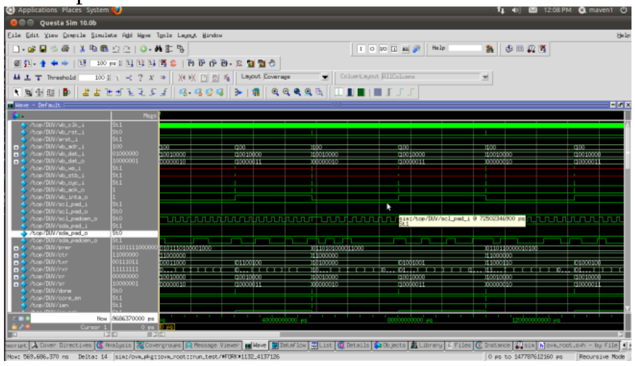


Fig 12. Simulation of the I2C Functional verification for Read operation

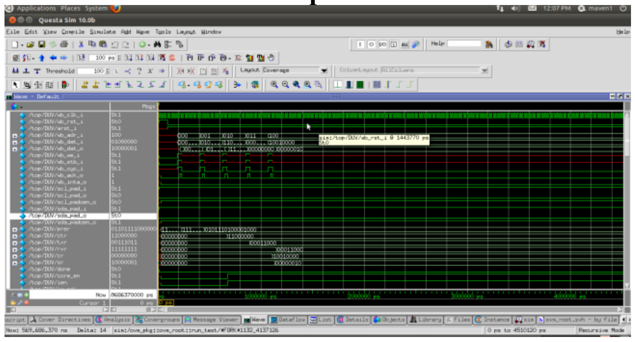


Fig 13. Simulation of the I2C functional verification for write operation

XI. CONCLUSION

The I2C IP core for intercommunication bus is designed using the HDL for master and salve and for the master to master communication in full duplex mode. This RTL design is carried out the functional verification using OVM in Questasim and Xilinx. This methodolgy provides the complete coverage of the RTL design so as to acquire the fault free Protocol design of I2C. So that can be implemented in real time systems.

REFERENCES

1. AN_108_Command_Processor_for_MPSSE_anMCU_Host_Bus_Emulation_Modes. Philips I2C datasheet.
2. D2XX Programmer"s Guide
3. Datasheet for FT2232H V202 I2C protocol
4. Datasheet for Microchip 24LC256 – 2K I2C Serial EEPROM.
5. "Introduction to I2C Bus": Available at *http://www.semiconductors.philips.com/i2c*.
6. P. Venkateswaran, "FPGA Based Efficient Interface Model for Scalefree Computer Network using I2C Bus Protocol"; Spl. Issue – Advances in Computer Sci. & Engg., ISSN 1870-4069, Pub. By
7. National Polytechnic Institute, Mexico, Vol.23, pp. 191- 198, Nov. 21-24, 2006.

AUTHORS PROFILE

B.santosh kumar was born in vijayawada, krishna (Dist.), AP, India. He received B.Tech. in Electronics & Communication Engineering from Bapatla engg college. ,Bapatla ,prakasam (Dist.,),AP, India ,M.Tech from KL University , Vijayawada, AP, India

L.Ravi Chandra Assistant professor, ECE department, K.L.UNIVERSITY, He received his M.TECH in the branch of VLSI, His interests are in digital front end design.

A.L.G.N Aditya was born in vizag, vizag(dist),AP, India. He received B.Tech. in Electronics & Communication Engineering from TPIST,AP. India ,M.Tech from KL University , Vijayawada, AP, India. He has undergone 8 international conferences and 1 publishment in IEEE

Dr. Fazal Noorbasha, Presently working as an Assistant Professor, Department of Electronics and Communication Engineering, KL University, Guntur, Andhra Pradesh, India, where he has been engaged in teaching and research, VLSI Research Group Head, Department Curriculum Committee (DCC) Member. His interest of research and development is Low-power VLSI, High-speed CMOS VLSI SoC, Memory Processors LSI's, Digital Image Processing, Embedded Systems and Nanotechnology. He has published and presented over 35 Science and Technical papers in various International and National reputed journals and conferences. He is a Scientific and Technical Committee & Editorial Review Board Member in Engineering and Applied Sciences of World Academy of Science Engineering and Technology (WASET), Advisory Board Member of International Journal of Advances Engineering & Technology (IJAET), Life Member of Indian Society for Technical Education (ISTE-India), Member of International Association of Engineers (IAENG-China) and Senior Member of International Association of Computer Science and Information Technology (IACSIT-Singapore).
E-Mail: fazalnoorbasha@kluniversity.in

T.Praveen Blessington, Presently working as an Associate Professor & research scholar in Department of ECE, KL University, Guntur, Andhra Pradesh, India, where he has been engaged in teaching and research in VLSI & embedded designs. He is a member of VLSI Research Group, Department Curriculum Committee (DCC) Member. His interest is research and development in SOC, NOC Architectures, Low-Power VLSI & Embedded Systems. He has published and presented various International and National reputed journals and conferences. He is a life member of IETE, ISTE and SCIEI. E-Mail: praveentblessington@kluniversity.in