Verification IP for Routing Switch based on Network Layer Protocol, using SystemVerilog

Dipti Girdhar
School of Engineering and Technology,
ITM University, Gurgaon, India

Neeraj Sharma
School of Engineering and Technology,
ITM University, Gurgaon, India

Shankar
School of Engineering and Technology,
ITM University, Gurgaon, India

ABSTRACT
Today, in the world of ASICs and system-on-chip (SoC) designs which consists of millions of transistors and gates, verification is the process which consumes most of design efforts and time [4]. One of the major stresses for the verification engineer is to verify the given design in best possible manner [5]. For this he needs to cover almost all the hidden corners cases by applying various real time test cases. This paper will assist the verification engineers to understand the flow of verification environment for packet switch IP. We will also learn about the functional coverage. The language used for verification is SystemVerilog.

Keywords
SystemVerilog, Verification IP, Packet switch

1. INTRODUCTION
SystemVerilog is one of most preferred hardware verification language used worldwide [6]. To create any Verification IP (VIP), one need to understand various modules, which are the part of the verification environment. A design which is to be verified is commonly known as design under test (DUT). DUT is like a black box, and verification engineer is least bother about the internal schematic of the DUT. A set of specification is provided to verification engineer and he need to develop the complete VIP after reading the given specifications only.

2. COMMENCEMENT WITH DESIGN SPECIFICATION
To start with, readers must recall the basic thumb rule of verification as mentioned above, that the verification engineer is provided only with the specification booklet [7]. Based on the given specifications, the verification team develops a VIP. So it is paramount to understand the given specifications. So here we will start with understanding the specifications given for DUT.

We need to prepare a VIP for a simple switch which is used to drive an incoming packet to different output ports of the router. In a network, the switch acts as a router which has one input port and various output ports. In this case we have switch with one input ports and four output ports. Switch works on network layer of Open System Interconnection (OSI) model. The basic block diagram is shown in Figure 1.

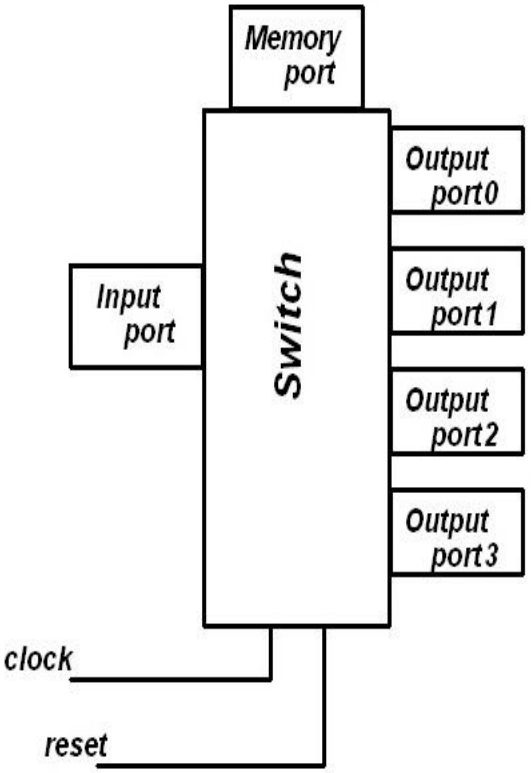

Figure 1: Block diagram of switch with various interfaces

2.1 Input condition for switch
Input port is responsible for collecting all the incoming packets. It consists of two signals; named as data_status and data as shown in Figure 2. Both the signals are active high.

Ideally, data_status signal is low, when there is no incoming packet on input port. As soon as any packet needs to enter to the switch through the input port, switch pulls up the data_status signal to high value at the rising edge of the clock. The data signal then carries the packet byte by byte. Switch releases the data_status signal to low value after receiving all the data bytes.

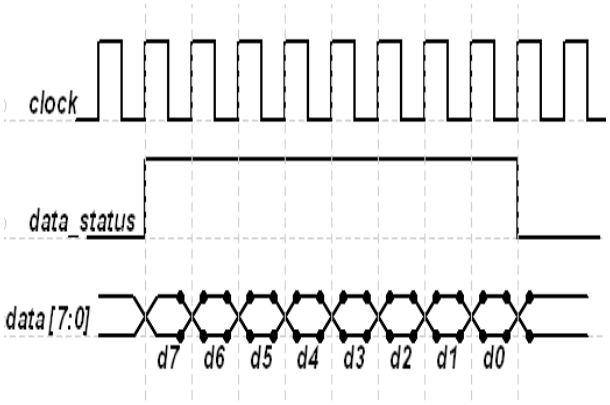

Figure 2: Shows the waveform for receiving condition

2.2 Morphology of packet

Packet header consists of three different fields as shown in Figure 3.

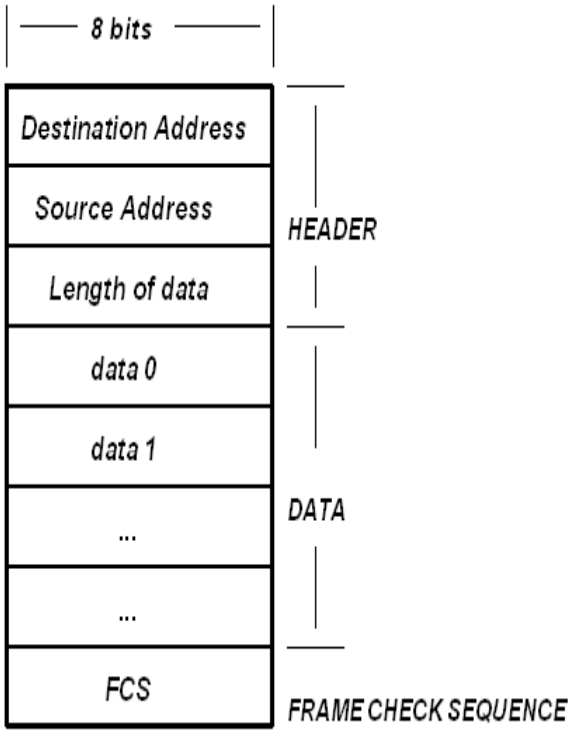

Figure 3: Shows the different fields of simple packet

All the incoming packets usually consist of 8 bits long destination address. This 8 bits long destination address is a unique port address for each output port. This destination address helps switch to drive the packets to the respective output port. However, the 8 bits long source address implies the origin of the packet. Length field is also 8 bits long which imply that we can have maximum 2^8 data bytes. If the length field of the packet contains numeric value 2, it means that particular packet contains 2 bytes of data. The last field is responsible frame check sequence (FCS). FCS is an extra checksum added for error detection.

2.3 Memory

As we know that switch comprises of four output ports and each output port have special port address which is 8 bits long. Memory interface is one of important interface which is responsible for configuring each output port to the unique address.

Memory interface consists of four signals mem_en, mem_rd_wr, mem_add and mem_data as shown in Fig. Ideally, mem_en signal is at "low" value. To configure the output port address, switch asserts the "high" value to mem_en signal. This enables the process of configuring the output port addresses. Mem_add signal carried the port no and mem_data carries 8 bits long address for the respective port. In this manner a table is generated for addressing each output port, as shown in Table 1.

Table 1: Port Adrresses

Port Number	Port Address
0	0000_0000
1	0000_0001
2	0000_0010
3	0000_0011

2.4 Output condition for switch

Output ports are used to send the packets to different devices connected to switch in a network. For this switch need to take care of ready, read and data signals as shown in Figure 4.

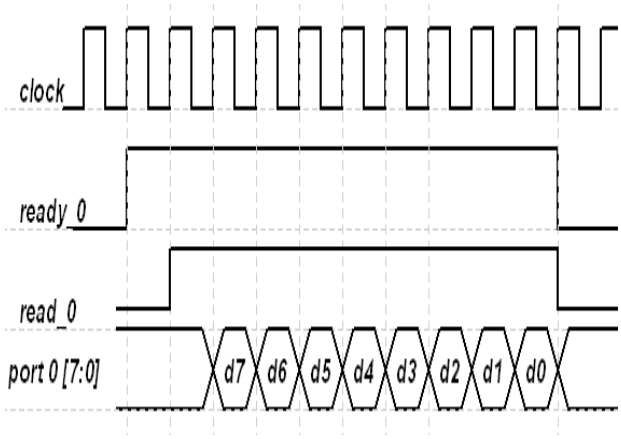

Figure 4: Shows the waveform for output port condition

When the packet is ready to hurl through the output port, switch pulls up the ready signal. Now when ready signal is high and read signal is assert to high value, data can be received from the data signal.

3. INTRODUCTION TO VERIFICATION ENVIRONMENT

Any verification methodology usually consists of layered Testbench [1]. This layered structure tends to make the verification task easier by dividing the complete code into smaller target modules. This makes it easy to develop and debug these smaller modules. Figure shows the various layers of the Testbench.

The Signal layer is bottom most layer of the testbench that contains DUT and the various signals that connects it to testbench [2].

The command layer is the next higher level to signal layer. Command layer consists of driver, assertion and monitor block. Driver is the class of verification environment which is responsible for driving the inputs for DUT. Monitor usually performs the task of collecting and grouping the signal transitions into commands. Command layer also holds assertions, which are responsible for monitoring the individual signals and changes across an entire command layer.

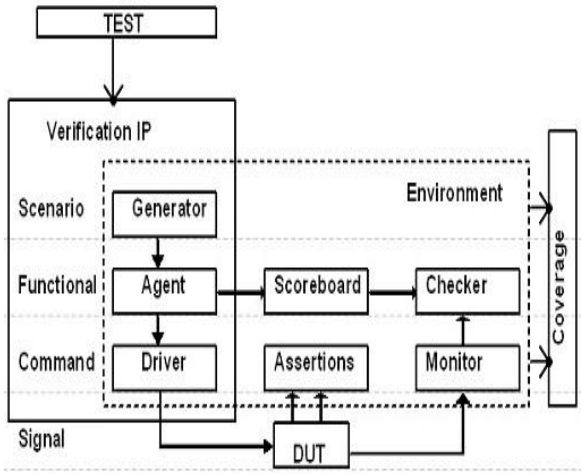

Figure 5: Shows the various layers of Testbench [3]

The functional layer contains agent block, scoreboard and checker block. Agent block is commonly known as transactor which receives higher level transactions and cleft them into individual transactions. The scoreboard is responsible to receive these individual transactions and predicts the results. The checker is responsible for inspecting the commands received from monitor and scoreboard. The scenario is used to drive the functional layer. It provides the protocol specific scenario generation.

The basic verification environment is shown in Figure 5. Here, generator is used to create constrained random test vectors which are fed into the driver and hence can stimulate the DUT. Monitor is also known as receiver which receives the output from DUT for the given stimulus generated by the generator and generates the verification report. This report is fed into the checker and is compared with the accepted report generated by scoreboard. Hence, adding to the coverage report. The object-oriented feature of SystemVerilog can be used to reinforce the reusability of these test bench components.

The verification environment also involves the special class known as interface. This class is used to inter connect the test program to the DUT. Top module includes instance of memory interface, input interface, output interface, testcase and DUT.

4. CONCLUSION

In today's world all the electronic design automation (EDA) tools are hastily in fabrication labs or for even programming design functionality into programmable IC's [8]. EDA companies make a huge turn over in terms of money. Field Programmable Gate Arrays (FPGAs) have become a critical part of every system design [9]. Developing and reusing IP for SOC verification is always a desired but challenging methodology [10].

This paper presents a method of creating a simple verification environment involving almost all the essential objects of the verification plan. The packet can have a bad fcs kind, if it has length greater than the data size and similarly it can have good fcs, if it has length equal to data size. Figure 6, shows simulation result for one of the packet which can be called as bad test packet

Figure 7, shows simulation result for one of the packet which can be called as good test packet

The field with the index value 0 and 1 shows the destination address and the source address respectively. The field with index value 2 shows length of the data. As we have $(ed)_{16}$ which is equivalent to $(237)_{10.}$ This is reason we have the data byte at 237 index value. The next address value (i.e. 238) hold the hex value which is correspondent to some fcs calculation. The simulation results also show that scoreboard has successfully received the packet from the receiver. This received packed is compared with packet earlier received from the driver. Results show that all the test packets generated have successfully passed the test. All this have been achieved by creating an efficient verification environment by using SystemVerilog language.

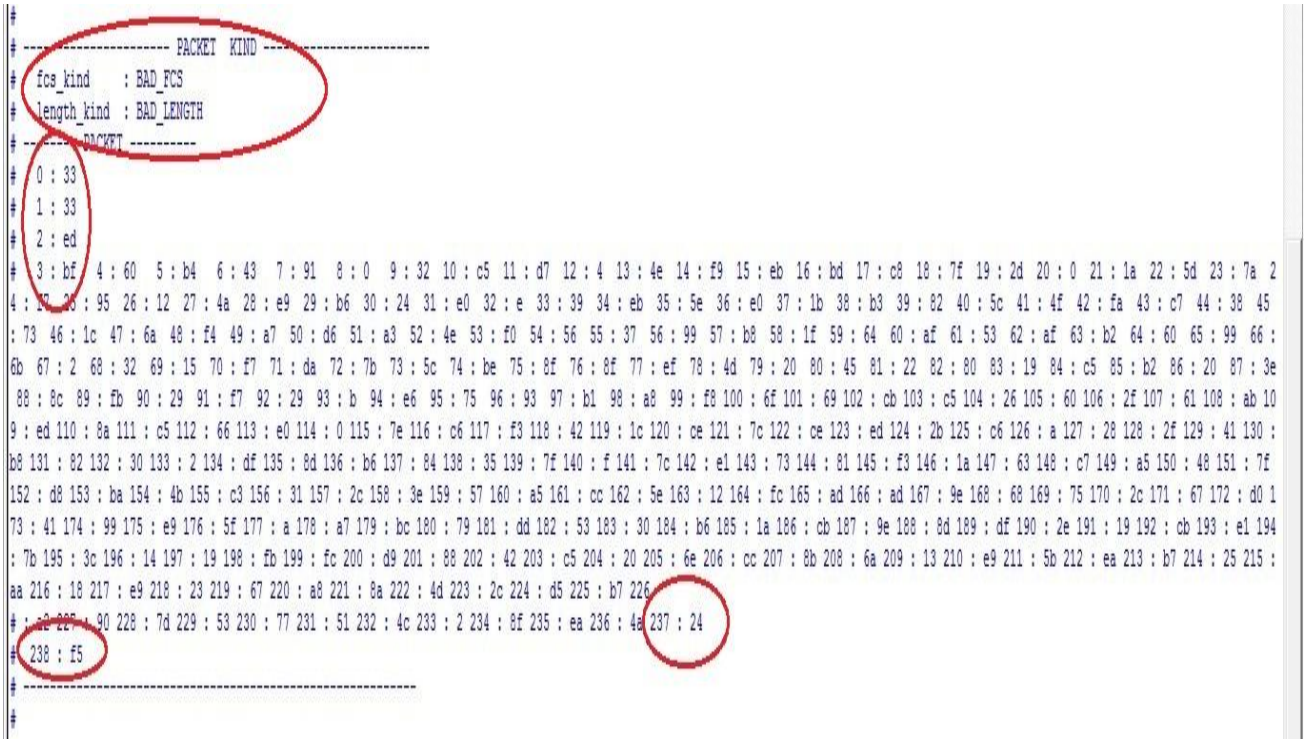

Figure 6: Shows the simulation result for packet with bad FCS kind

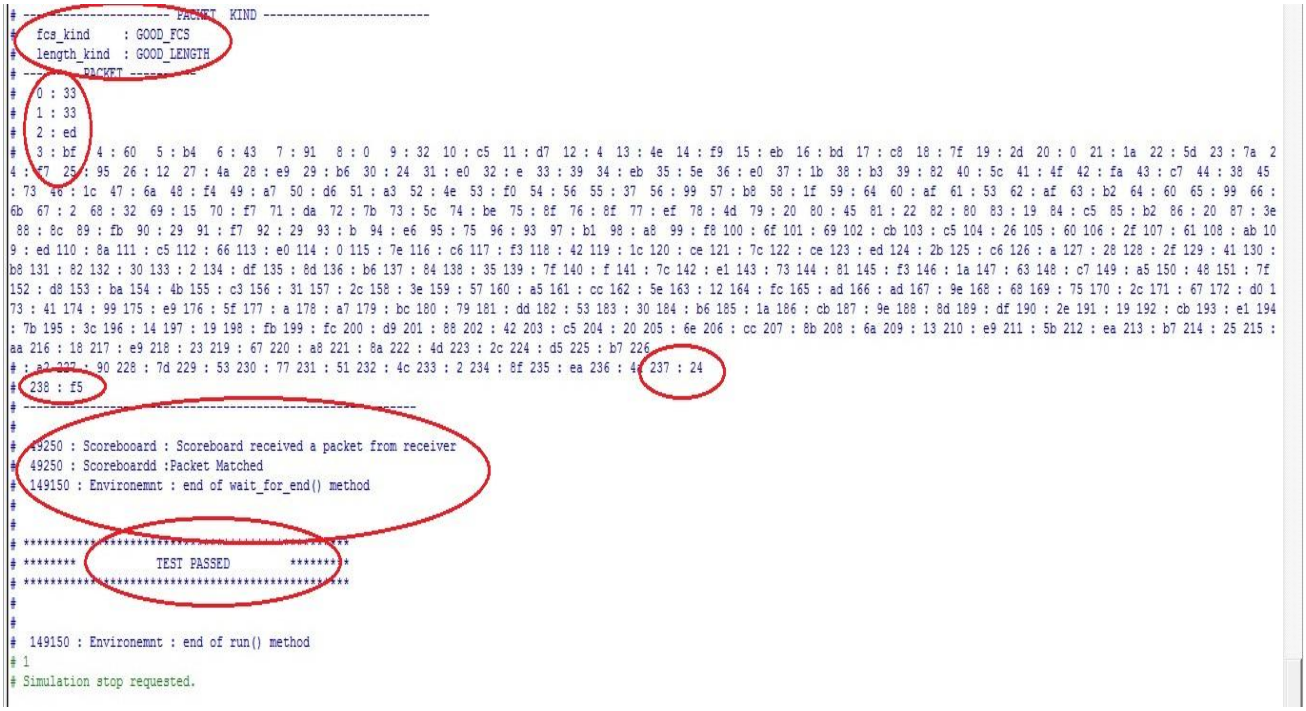

Figure 7: Shows the simulation result for packet with good FCS kind

5. REFERENCES

[1] Chris Spear, Greg Tumbush, Verification Guidelines, *SystemVerilog for Verification*, Third Edition (Ney York: Springer), pp. 58.

[2] Chris Spear, Greg Tumbush, Verification Guidelines, *SystemVerilog for Verification*, Third Edition (Ney York: Springer), pp. 61.

[3] Chris Spear, Greg Tumbush, Verification Guidelines, *SystemVerilog for Verification*, Third Edition (Ney York: Springer), pp. 62

[4] Purvi D. Mulani, "SoC Level Verification using SystemVerilog," *In the proceedings of Emerging Trends in Engineering and Technology (ICETET) Conference*, Nagpur, pp. 378-380, Dec. 2009

[5] Young-Jin Oh Gi-Yong Song, "Simple hardware verification platform using SystemVerilog," *In the proceedings of TENCON 2011 - 2011 IEEE Region 10 Conference*, Bali, pp. 1414 – 1417, Nov. 2011.

[6] Ma Pei-Jun, Ma Wen-Bo, Li Kang, Shi Jiang-Yi, others, "The verification of network processor Fast Bus Interface using SystemVerilog," *In the proceedings of Electron Devices and Solid-State Circuits (EDSSC), 2011 International Conference*, Tianjin, pp. 1 – 2, Nov. 2011.

[7] Simmons, M., Geishauser, J., "FlexBench: reuse of verification IP to increase productivity," *In the proceedings of Design, Automation and Test in Europe Conference and Exhibition*, Paris, 2002.

[8] Chia-Chih Yen, Jing-Yang Jou, An-Che Cheng, "A formal method to improve SystemVerilog functional coverage," *In the proceedings of High Level Design Validation and Test Workshop (HLDVT), 2012 IEEE International Conference*, Huntington Beach, CA, pp. 56 – 63, 9 – 10 Nov. 2012,

[9] Young-Jin Oh, Gi-Yong Song, Jae-Jin Lee, "Design and verification of an application-specific PLD using VHDL and SystemVerilog**,"** *In the proceedings of ASIC (ASICON), 2011 IEEE 9th International Conference*, Xiamen, pp. 159 – 162, 25 – 28 Oct. 2011,

[10] Pierres, A. Hu Shiqing, Chen Fang and others, "Practical and efficient SOC verification flow by reusing IP testcase and testbench," *In the proceedings of SoC Design Conference (ISOCC), 2012 International Conference*, Jeji Island, pp. 175 – 178, 4 – 7 Nov. 2012