International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6, August, 2019

# Design and Verification of Generic FIFO using Layered Test bench and Assertion Technique

**Mohini Akhare, Nitin Narkhede**

***Abstract*: Verification is must to ensure that the design is an exact representation of the specifications of the design without any bugs. Verification helps to avoid surprisess at later time so that product can enter the market on time with good quality and less cost. In the present research work, synchronous generic FIFO is designed using Verilog. Here the pointers will indicate the status of the FIFO, the flag information's like Full, Empty, Last, Second Last First and the FIFO will have a synchronous Reset ability and this FIFO is used as a DUT under verification environment. The verification is carried out using SystemVerilog layered testbench approach. As the designing of modules get complex, it is becoming more difficult to check that design, as it takes longer time to check all the combinations of design inputs. This problem can be solved by randomization and adding cover group and assertions. The verification plan involves test bench, verification properties, assertions, coverage sequences, application of test cases and verification procedures for the FIFO design. The functionality of the DUT is verified through layered testbench approach and coverage analysis. The response of DUT under random constrained inputs is compared with the predicted response in the scoreboard unit of the layered testbench. The research work achieved 80% code coverage and around 90% of functional coverage.***

***Index Terms: Verification, Synchronous FIFO, Generic FIFO, Code Coverage, Functional coverage, Covergroup, Assertions.***

## I. INTRODUCTION

The FIFO element will represent a layout written in System Verilog with System Verilog assertions. The FIFO usually interfaces to a controller for the synchronous pushing and popping of facts. This plan consists of characteristic extraction and test approach, check verification method [1].Nowadays, testing as a word has been substituted with check. Confirmation specialists need to guarantee what goes to the plant for assembling is an exact representation of the specification of configuration [2]. The modem, systematic and automated approaches has been required because of the continuous growth and complexity for creating test benches [3], given that up to 70% of the design period is spsent in the authentication process [4].The FIFO (First in First Out) is a genus of memory that is ordinarily used to holds the information, has to utilize consistently between various systems at distinct deferrals. The FIFO model permits the transmitter to send information and its collector is used to take out data. The data or information is filled the FIFO memory until the recipient starts emptying it.

Revised Manuscript Received on October 30, 2019.

* Correspondence Author

**Mohini Akhare*,** M. Tech. in VLSI Design from Shri Ramdeobaba College of Engineering and Management (RCOEM), Nagpur.

**Nitin Narkhede**, Associate Professor and M. Tech. (VLSI Design) Coordinator in Department of Electronics Engineering of Shri Ramdeobaba College of Engineering and Management (RCOEM), Nagpur



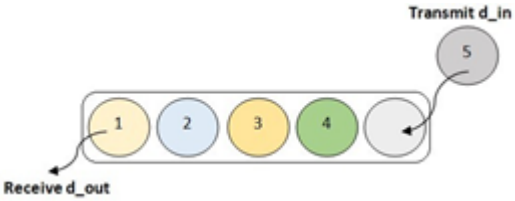



**Fig. 1: Functional Diagram of FIFO**

An overflow occurs as soon as the transmitter fills up the FIFO model and attempts to store more data before the receiver has read the data out. An underflow occurs when the receiver attempts to read data from the FIFO structure, but the transmitter did not fed any data into it. The Full and empty signals are used by the logic to strangle the transmitter and receiver respectively, in order to avoid these critical conditions. The Fig. 1 shows the functionality of the FIFO. We can represent the FIFO as a pipe that means we can imagine a FIFO as a pipe where the first element entering into the pipe is the first element that output from the pipe [5]. This paper describes the implementation of constrained random test stimuli, functional coverage, use of assertions technique also describes an approach for creating test cases that allow the use of both constrained random tests within a single environment. The environment built should also have the capability to be easily modified where a Device under Test (DUT) of similar structure can be verified. With this in mind the main goal is to develop a new and more effective intuitive way of designing test benches [6]. The more capable verification tools must be used, as circuits become more complex. This verification should takes place much earlier than the fabrication process. In this paper a verification environment is comprehend and implemented which may detect the maximum errors for proper functioning of the synchronous FIFO model.

## II. DESIGNING AND WORKING OF FIFO

Here we have considered the synchronous FIFO for designing. The synchronous FIFO consists of dual port RAM, write logic, read logic. Dual-port RAM or DPRAM is a type of RAM that allow multiple reads and writes in chorus at unlike addresses. In this design the write operation is accessed through port 1 where as read operation is accessed through port 2. The basic block diagram which demonstrates the architecture of FIFO mode has been shown in Fig 2.This design consists of the Dual Port RAM that is the memory block and Controller consists of Read control logic and

Write control logic blocks [2], [7], [8]. Here the memory array with the help of flip flopscan also be dual port RAM has been chosen. Instantaneous access is provided to read and write ports by this dual port RAM i.e. Read and Write operations can be performed simultaneously. The limitation with this design is that Read and Write operations cannot take place from the same memory location. So the key role of dual port RAM element (Memory block) has been to write and read the data simultaneously but not at the same memory location. This particular type of RAM has two unidirectional data ports, an input port for writing data and an output port for the reading data where each port is assigned to have their own data and address buses. The write port has signal named as "write" to allow writing of the data. The read port has signal known as "read" to enable the data output. The dual port RAM which is examined in this paper is synchronous and has a single clock for both ports, as depicted in Fig 3. At the rising or positive edge of clock both the reading and writing of data has been occurred. The synchronous FIFO has an exclusive clock port for both the Read and Write operations. The data which is given on the data input port is written on the next empty location. This happens only on the positive edge of the clock, when "write"signal of the write control logic block is high.

Then the written memory data is read out from the Read control logic block. This program steps forward when the "read" signal of the block is high. Whenever there is unavailability for writing data in the FIFO, Full signal goes high and indicates that the memory is full and there is no space for writing further data. The Empty signal indicates that there is no single data filled in the memory and vacant locations are available in the memory. Whenever any single data will get into the memory, Empty signal goes low. Along with Full and Empty flags, three more flags are present here and that are First, Second Last and Last. Here we have considered depth of memory is 8. So whenever the first data entered into the memory "First" signal goes high. Similarly whenever seventh location filled into the memory "SLast" signal indicates high value and for the last location i.e. the eight location signal names as "Last" goes high. All these flags are controlled by controller. The functionality of the dual port RAM can be articulated with three conditions, the first condition transpire when the reset signal is high and all the output signals i.e. output data (dataout), write address (wr_addr) and read address (rd_addr) reset to zero value. The next condition occur when write enable (write) signal is high and the incoming input data (datain) is written in the memory block generated by the write address. But this will take place only on the next rising edge of the clock. The write enable signal is only generated when the FIFO is not full so as to avoid corruption of data. The third stage show up when read enable (read) signal is high, then the data that is written in the memory is read out form the memory. The signal "rd_addr" will generate the read address through which the data can be read, this stage also can only exists at the next rising edge of the clock. The Read enable signal is only generated when the FIFO is not empty so that any corrupted data could not be read out from the memory [9], [10], [11]. In the presented Fig. 2, the block on the left implemented as a substitute of dual port RAM but to keep the design simple

side is the Write control logic block. This block is used to control the write operation of the implemented FIFO design. The block basically generates a binary coded write pointer (wr_ptr) and this pointer gets incremented by one location every time, the input data is written into the design. Also this block generates a Full signal to avoid overwriting a data in the memory block. The block on the right side is the Read control logic block, this block is used to control the read operation of the implemented FIFO design. The subunit generates a binary coded read pointer (rd_ptr) which gets incremented by one location every time, the written data is read from the design. Also, this block generates an Empty signal such that no invalid data can be read from memory. The verification phase is most significant step for any successful design.

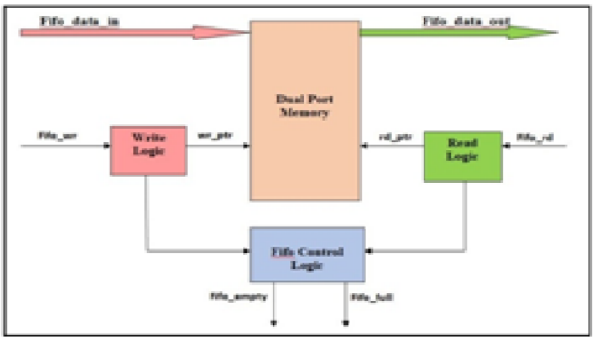



**Fig. 2 Block diagram of FIFO**

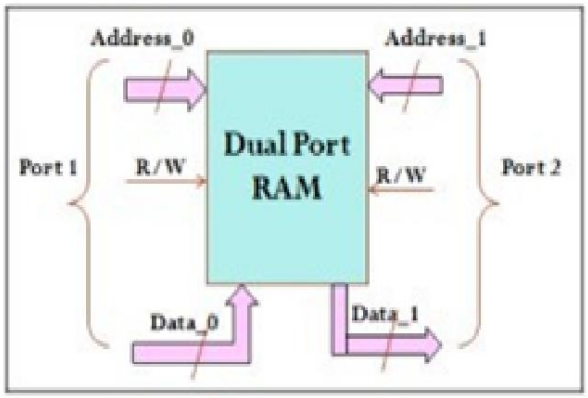



**Fig. 3 Block diagram of Dual port RAM**

## III. VERIFICATION PROCESS

### A. Verification through Layered Testbench

The verification plan must consist of the entire verification process [12] and formation of a good plan will save a lot of tedious and unprofitable time later. The whole plan should include the time for the completion of process as well as the authentication of the coverage result [13]. The verification planning are growing in a speedy manner hence it becomes more and more competently requirements to create a good plan before the verification has been finally started [14]. The theory about verification planning has been reported in a well organized and systematic manner [15].

Block diagram of the verification plan has been shown in Fig. 4 and all its components are explained in this section.

The very first component comes under verification environment is Transactor. Transaction class is used to declare fields required generating the stimulus, so first step is to declare the 'Fields' in the transaction class.

Different transactions depending on the test case configurations field selection has been generated in the are randomized so that to drive Random stimulus to DUT by changing the characteristics of data.

After Transactor, Generator class comes which is responsible for, Generating the stimulus by randomizing the transaction class & sending the randomized class to driver. Using the above environment, the transaction generator is defined within the Driver block i.e. the coding to the read, write and reset transactions are done in the Driver block itself. Each implemented transaction will generate a test case that is a random data which will initiate by the driver and given to the DUT to perform the specific operation utilizing the functioning of the incoming signals. The driver is the block that deciphers the transactions in to random inputs i.e. test cases. These are given to the DUT and the DUT performs specific operation depending upon the input given. The transaction generator generates a high level transaction like read, write or reset. The driver basically converts these transactions into actual inputs. A driver gets the information from the generator and drives it to the DUT by inspecting and driving the DUT signals [16]. It contains the hidden validation to drive the pins of the DUT as indicated by situation gave to it from the sequencer.

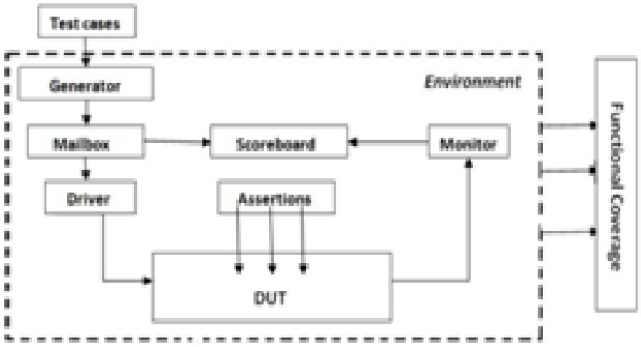



**Fig. 4 Verification Plan**

The scoreboard is used to record the operations of the driver and then displays these operations systematically. A screen, a distant commodity, is just to monitor the examples of the DUT signals [17] but cannot be used to drive them. A monitor collects information, extracts occasions, performs checking, scope and optionally prints follow data. It utilized the screen to sign sent to the pins of the DUT from the driver. As the name suggests, it basically monitors the operations performed by the driver and then it passes its data to the scoreboard to display the information. The scoreboard can also be named as tracker as it tracks all the operations. The dynamic data types and dynamic memory allocation makes the task much easier to implement scoreboard in the system Verilog. Normally, a scoreboard confirms whether there has been fitting operation of the configuration at a practical level. The scoreboard basically stores the data and address when the write operation is done and displays the results. transaction class. For the FIFO design which has to be implemented, the transactions are RESET, WRITE and READ operations and the fields which has been declared here are D_in, Write,
Read, Full, Empty, First, Last, SLast, D_out. Also the input signals (D_in, Write, Read)
Furthermore, it also records the data and addresses that had been previously read. It matches the similarity of the data and display the outcome [2].

Further the Environment class comes which is a container class contains Mailbox, Generator and Driver. It creates the mailbox, generator and driver, shares the mailbox handle across the Generator and Driver. Then Test class is present which is responsible for creating the environment. Also it configures the testbench i.e. setting the type and number of transactions to be generated as well as initiate the stimulus

driving. The Interface block is one of the essential modules throughout the verification plan. In this particular block all the commonly used signals in both the design and the verification environment. The interconnect block bridge as an interfaced to the design under-test and the check environment. The interface assimilates all the pin-level associations that are made to the DUT [18]. Basically an interface is a bundle of nets or variables. The coverage collector mainly covers the coverage related issues of the block. This block has covers the groups and cover points that are used to estimate the functional coverage of the design.

### B. Verification using Assertion Technique

Functional coverage verifies the functionality of the design based on its specification and Assertion-based verification (ABV) is a technique that aims to speed one of the most rapidly expanding parts of the design flow. We can measure how often these assertions are triggered during a test by using assertion coverage. A cover property observes sequences of signals, whereas a cover group samples data values and transactions during the simulation. These two constructs overlap in that a cover group can trigger when a sequence completes. Additionally, a sequence can collect information that can be used by a cover group. SystemVerilog has features to specify assertions of a system. An assertion specifies behaviour of the system. Basically SVA or System Verilog Assertions is based on PSL assertion that was developed earlier [3]. An assertion is simply a check against the specification of your design that you want to make sure never violates. If the specifications are violated, you want to see a failure. [19]

An assertion is basically a *"statement of fact"* or *"claim of truth"* made about a design by a design or verification engineer. An engineer will assert or "claim" that certain conditions are always true or never true about a design. If that claim can ever be proven false, then the assertion fails (the "*claim*" was false). SystemVerilog has two types of assertions: Immediate assertions and Concurrent assertions.

Immediate assertions execute once and are placed in line with the code. Immediate assertions are not exceptionally useful except in a few places. Concurrent assertions are the most valuable and most widely used type of assertion. Concurrent assertions activate properties (rules) that typically sample design signals or sequences of design signals just before each new active clock edge to determine if the design is behaving as it was claimed that it should behave. The most valuable assertion style that can be used in design and verification environments is the concurrent assertion. Concurrent assertions are little monitors that sit down inside of a block of code to periodically sample and test signals and generate error messages if the assertion ever fails. Concurrent assertions are typically sampled once per clock period at the end of the clock cycle, just before the next active clock edge. Concurrent assertions require the assertion of a property, where a property is basically a design rule that should always be true [20].

## IV.DESIGN AND SIMULATION ANALYSIS

Initially the outcomes embraced of the Reset, Read and Write condition results. Fig. 5 illustrates the scenario in which all the output signals are reset to zero which is shown with the help of waveform generated in the Questa Sim tool. As reported in the waveform the initial signal is the clock signal. This is a free running signal and all other signals will have to change their values only on the rising edge of this signal. The next signal is the reset signal; this is the signal that dictates the reset condition for all other signals. This status has forced the value 1 such that all the output signals would be reset to zero. The output signals D_out, rd_wire, wr_wire are set to zero.

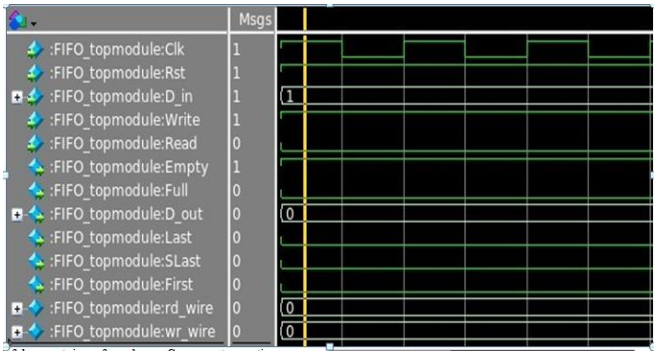



**Fig. 5 Reset condition waveform**

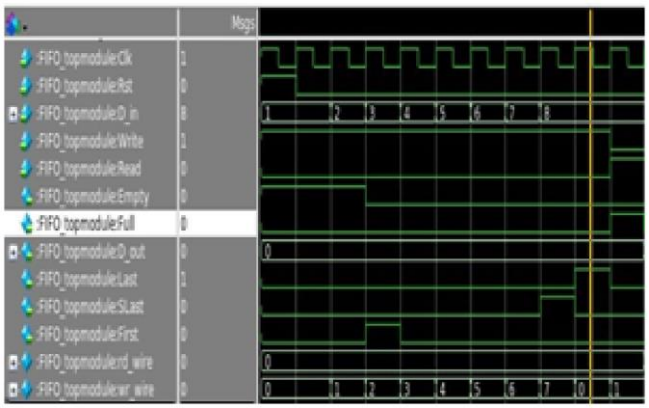


**Fig. 6 Write condition waveform**

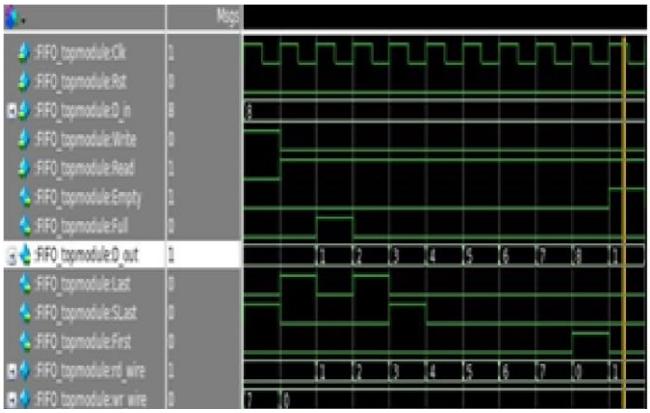


**Fig. 7 Read condition waveform**

The Fig. 6 shows the scenario in which all data is written in the FIFO through waveform generated by the QuestaSim tool. exhibited in the figure the first signal is the clock signal that is a free running signal. All other signals will change the values on the rising edge of the clock. The reset signal in this case has forced to zero. The depth of the memory is 8 i.e. up to eight locations FIFO can write the data in the memory. The 16 bit data is given on the D_in signal (0000000000001000). This data is the input data that is given in the FIFO memory. The Write signal is also set to 1, to activate the write control logic of the design. This block is responsible for writing the data in the memory. The FIFO "First" signal goes on high as the first data entered into the memory. Similarly, "SLast" and "Last" signal becomes 1 when the memory fills seventh and eight locations respectively. The FIFO full signal becomes 1 as the FIFO memory is filled with all the locations of memory and no space has been available to write more data. The wr_wire which is write pointer signal has been binary coded, gets incremented as it goes on filling the memory locations. As many times the rising edge of the clock is coming, the wr_wire gets incremented and the incoming data gets written into the FIFO memory on that write pointer location.

The Fig. 7 configured the scenario in which the incoming data is read out from the FIFO. As depicted through the waveform, the first signal is the clock signal as convoluted in the write status explanation. The data that was written in the memory (0000000000001000) with the help of D_in signal is now read out form the FIFO memory. The Read signal is set to 1 so to activate the read control logic of the design. This block is fundamentally responsible for reading data from the memory. The FIFO empty signal is 1 and FIFO full signal becomes 0 in this circumstances as the FIFO is not full in this scenario. In this case, now the rd_wire that is the read pointer signal gets incremented by 1 value. Now the read pointer will traverse through all those locations on which the data was written by the write pointer. When all the data written would be read out then the FIFO will again be completely empty. The written data is read out from the FIFO with the help of D_out signal.

## V. VERIFICATION RESULTS AND ANALYSIS

### A. Verification Results through Layered Testbench

The Code Coverage of the testbench and design before adding covergroup is shown in Fig. 8 which demonstrates the code coverage of top module i.e. testbench is 80% whereas the code coverage of DUT is 98.9%. Here we have analyzed which Branches and Statements have been covered in the design. Also the missed Statements and Conditions have been detected.

The coverage report is shows in the Fig. 9 which is the foremost result in the verification analysis. In the coding, a

covergroup is created that is generated which is clearly visible through the coverage report. The coverpoints are created so as to cover the each important aspect of the design [21].

The practical coverage is the determination of the

covergroup cg@(posedge Clk); cover_point_D_in :
coverpoint D_in
{ bins a[] = {[1:4]};
bins b[] = {[5:8]};
bins c[] = {[9:12]};
bins d[] = {[13:16]}; }
cover_point_Write : coverpoint Write; cover_point_Read : coverpoint Read; endgroup

cg cg_inst = new();

initial // or task or function or always block
foreach(values[i])
begin
D_in = values[i]; Write = values[i]; Read = values[i];
cg_inst.sample(); end

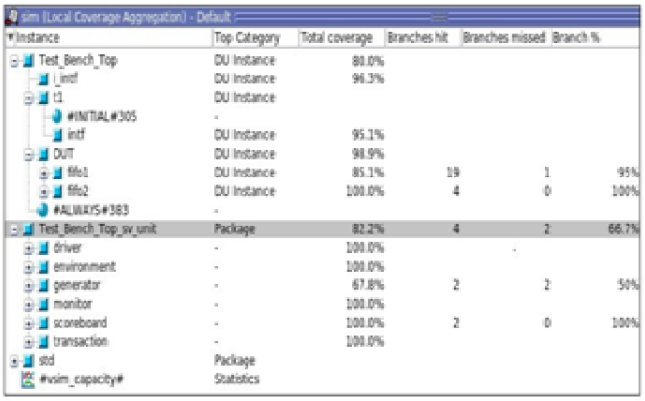



**Fig. 8 Code Coverage Report**

A covergroup in the coding is created that is generated which is clearly visible through the coverage report. The coverpoints are created so as to cover the each important aspect of the design [21]. Bins in cover group which have been generated automatically for "Read" and "Write" demonstrated in Fig. 10. This report essentially demonstrates the coverpoints that has been made to cover the different conditions of the design continuously. The cover_point_Read has covered the situations in which the Read control logic is selected in one situation and not selected in the other. It covers these conditions as this signal is given 0 and 1 value randomly so as to cover both the scenarios. This is the reason that the coverage for this signal is coming 100%. The cover_point_Write covers the scenarios in which the Write control logic is selected in one and not selected in the other phase. Both the amount of the design usefulness having induced by the verification environment [22]. Initially the code includes, the type of cover groups to screen the stimuli being put on the DUT. The responses and reaction to the stimuli are additionally checked to figure out what usefulness has been worked out. The cover groups ought to be indicated in the plan of verification. Inside a scenario of test, their handiness is learned by dissecting the RTL code and comprehension of the data. The cover points turn out to be all the more capable inside the recreation when they are crossed together to inside recognize more noteworthy levels of reflection of an outline. The cover groups give an effective and valuable system in distinguishing zones of useful scope inside of a
configuration [22]. The coding
portion of the coverage report has been given below:
situations have been covered by the signal which is given 0 and 1 value. Due to this same reason, the coverage for this signal is also showing to be 100 %.The most essential coverpoint of these is the first one i.e. cover_point_D_in. This coverpoint basically cover the scenarios in which random test data is generated automatically and is given to the FIFO design. Here explicit
bins are created for D_in to check whether it covers the data

given to the design or not. The separate bins have been created for each value in the given range of variable. The random data that is being written and read out form the design can be seen in the scoreboard report that is given in the next section.

The scoreboard report [22] is the next portion explained in this section. With the help of Scoreboard block of the verification plan, the scoreboard report has been generated. The scoreboard report generated in this case records the transactions that are taking place in the verification environment. For the above design it records the Reset, Write or Read scenario are being executed by the verification environment. It records the random test case data that is generated by the Write control logic to be written in the FIFO memory. It also records the test case data that is read out form the memory with the help of Read Control Logic.

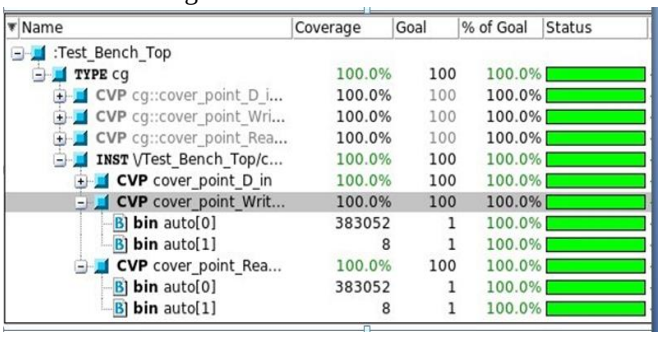



**Fig. 9 Coverage Report of Covergroup**

In addition, it compares both the data being written matches with the data read out. The Driver requests are given to the scoreboard through the monitor block. The generated scoreboard report for the design has been given as follows:

# Monitor : transaction no : 0 # Result is as Expected
# -------------------------
# - [Scoreboard ]
# -------------------------
# - D_in = 1, Write = 1, Read = 0
# - D_out = 0, - Full = 0, - Empty = 1, - Last = 0, - SLast = 0,
- First = 0
# -------------------------
# Monitor : transaction no : 1 # Result is as Expected
# -------------------------
# - [Scoreboard ]
# -------------------------

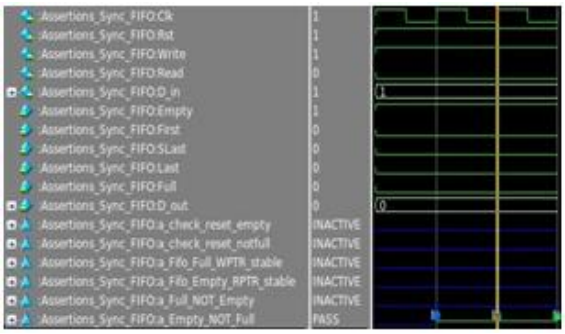
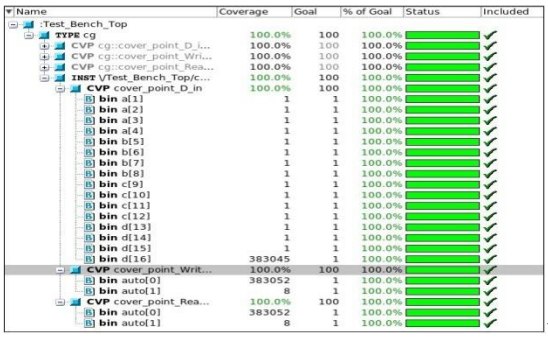


**Fig. 10: Bins Generation of covergroup**

**B. Verification Results using Assertions**



to

**Fig. 11: "Empty_NOT_Full" Condition**

Conditions check the code assertions are as follows:

#1 Check_reset_empty : This property checks to see that empty pointer should be high (i.e. Empty) when you reset the FIFO.

#2 Check_reset_notfull : This property checks to see that the FIFO should not be full when you reset the FIFO.

#3 Fifo_Full_WPTR_stable : This property checks to see that if the FIFO is full then the wr_ptr does not change.

#4 Fifo_Empty_RPTR_stable : This property checks to see that if the FIFO is empty then the rd_ptr does not change. "Empty" means that the rd_ptr remains the same as its value at the last clk—thus guaranteeing that the rd_ptr have not changed.

#5 Full_NOT_Empty : This property checks to see that if the FIFO is full (i.e. signal Full =1) at the same time signal "Empty" should not be 1.

#6 Empty_NOT_Full : This property checks to see that if FIFO is empty (i.e. signal Empty =1) at the same time signal "Full" should not be 1. This means that both full and empty signal should not assert at the same time.

Firstly the sixth property "Empty_NOT_Full" have been activated (i.e. Pass) is shown in Fig. 11. As at the starting, FIFO is empty and hence "Full" cannot be high at the same time. Then the first, second and fourth property "Check_reset_empty", "Check_reset_notfull" and "Fifo_Empty_RPTR_stable" respectively got activated and pass as depicted in Fig. 12. As the first two properties are based on

# - D_in = 1, Write = 1, Read = 0
# - D_out = 0, - Full = 0, - Empty = 0, - Last = 0, - SLast = 0,
- First = 1
# Monitor : transaction no : 2 # Result is as Expected
# -------------------------
# - [Scoreboard ]
# -------------------------
# - D_in = 2, Write = 1, Read = 1
# - D_out = 1, - Full = 0, - Empty = 0, - Last = 0, - SLast = 0,
- First = 1
#-------------------------

check on reset condition, when the reset is high Empty should be high and FIFO is not full and hence these properties got activated. At the same read pointer should be stable as FIFO is yet empty and it could not read any data from it. Hence the forth property got activated or pass. Fig. 13 shows the third and fifth property i.e. "Fifo_Full_WPTR_stable" and "Full_NOT_Empty" got activated or pass. Here when FIFO is full, write pointer should be stable and when full is high, empty could not be high at the same time. Cover Directives Window is shows in Fig. 14 which depicted about whether the given scenario ever happened in simulation or not.

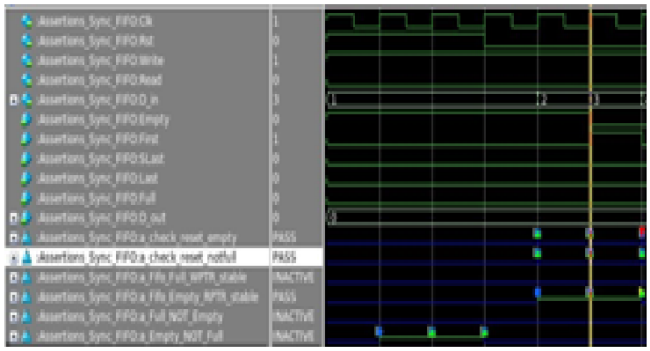


**Fig. 12: "Check_reset_empty", "Check_reset_notfull" and "Fifo_Empty_RPTR_stable" Condition**

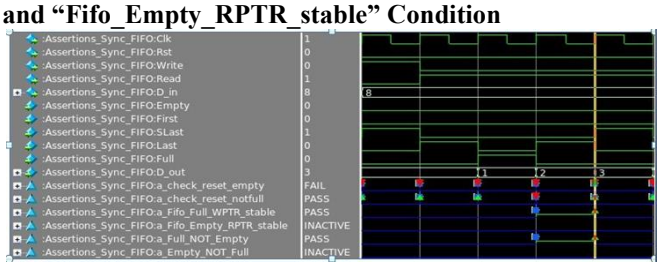


**Fig. 13: "Fifo_Full_WPTR_stable" and "Full_NOT_Empty" Condition**

## VI. CONCLUSION

The initial objective was to design and implement the FIFO using Verilog language. The said DUT was simulated and analysed for various input conditions, read and write logic and for its status flags like Full, Empty, Last, Second Last First. Further, the next step was to verify the DUT i.e. the FIFO module has been verified through layered test bench verification environment and using assertion techniques using System Verilog language which was the last and most crucial target of this work [23].

The layered testbench components were designed using systemverilog Hardware Verification Language (HVL). The DUT and layered testbench components were connected together using bind command to develop verification environment. The test cases which are the input data that need to be applied have been randomized and automated using the implemented verification model so as to reach to each and every corner cases to detect the bugs. The scoreboard report was analysed for various test cases against the predicted outputs and it is found that for all the test cases the response of the DUT was matching to the predicted results. The systemverilog functional coverage methodology verifies the functionality of the design in most effective way. 80% code coverage has been achieved for the constraints that were applied to the design. The applied random constrained inputs and the transactions that took place achieved 90% functional coverage for the DUT.

The functional coverage of the DUT has been substantially increased by adding assertions in the design [1], [24]. By using assertions all the possible conditions of FIFO have been checked which is required to check the functional behaviour of design. The benefit of using assertions is it improved the observability of the design and thereby reduced the overall debug effort. Also the system verilog coding knowledge got enumerated through this research. The UVM is another form of functional verification that we can use for

| Name | Language | Enabled | Log | Count | AtLeast | Limit | Weight | Cmplt % | Cmplt graph | Included |
|---|---|---|---|---|---|---|---|---|---|---|
| Assertions_Sync_FIFO:c_check_reset_empty | SVA | ✓ | Off | 3 | 1 | Unlimited | 1 | 100% | | ✓ |
| Assertions_Sync_FIFO:c_check_reset_notfull | SVA | ✓ | Off | 23 | 1 | Unlimited | 1 | 100% | | ✓ |
| Assertions_Sync_FIFO:c_Fifo_Full_WPTR_stable | SVA | ✓ | Off | 1 | 1 | Unlimited | 1 | 100% | | ✓ |
| Assertions_Sync_FIFO:c_Full_NOT_Empty | SVA | ✓ | Off | 1 | 1 | Unlimited | 1 | 100% | | ✓ |
| Assertions_Sync_FIFO:c_Empty_NOT_Full | SVA | ✓ | Off | 2 | 1 | Unlimited | 1 | 100% | | ✓ |

**Fig. 14: Cover Directives Window**

verifying designs as it has reusable verification components and assembling powerful test environment.

## ACKNOWLEDGMENT

The authors are thankful to Mr. Ankit Somani, MGR I ASIC Digital Design, Synopsys India Pvt. Ltd. Pune for his guidance and technical support for completing the said research work.

## REFERENCES

1. Naveen T V, M V Latte, Sathish Shet, Shashidhara H R, " Assertion based verification strategy for a generic first in first out (FIFO)", Mtech in VLSI Design and Embedded Systems, Dept of ECE, JSSATE Bengaluru, IRJET, Volume: 04 Issue: 05, June -2017.
2. Navaid Z. Rizvi, Rajat Arora and Niraj Agrawal," Implementation and Verification of Synchronous fifo using system verilog verification methodology ", School of ICT, Gautam Buddha University Greater Noida, India, Journal of Communications Technology, Electronics and Computer Science, Issue 2, 2015.
3. Chris Spear,"System Verilog for Verification", Springer, Vol 1 pp. 1-5, 2005.
4. Janick Bergeron," Writing Testbenches Using SystemVerilog", Springer, Vol I pp. 1-10, 2006.
5. Bergeron, Janick, Writing testbenches: functional verification of HDL models, Springer, Edition 2003.
6. Takimoto, Y., "Recent activities on millimeter wave indoor LAN system development in Japan," Dig. IEEE Microwave Theory and Techniques Society Int. Symp., 405-408, Jun. 1995.
7. Haytham Ashour, "Design, Simulation and realization of a parametrizable, configurable and Modular asynchronous FIFO", Mentor Graphics, Emulation Division, Science and Information Conference, London, UK, July 28-30, 2015, pp. 1392-13995.
8. Hemant Kaushali, Tushar Puri," Design of synthesizable asynchronous FIFO and implementation on FPGA", School of Electronics, M.TechvlsiCDAC–Noida, International Journal of Engineering Research and Development, Volume 13, Issue 7, July 2017, PP.06-12.
9. Shilpa Maurya," Design of RTL Synthesizable 32-Bit FIFO Memory", International Journal of Engineering Research & Technology (IJERT), Vol. 5 Issue 11, November-2016. pp. 591-593.
10. Gengting Liu, James Garside, Steve Furber, Luis A. Plana, Dirk Koch," Asynchronous Interface FIFO Design on FPGA for High-throughput NRZ Synchronisation", IEEE Xplore, Science and Information Conference (SAI), London, UK, 03 September, 2015.
11. V. Nagarajan," The Design and Verification of a synchronous First-In First-Out (FIFO) Module Using SystemVerilog Based Universal Verification Methodology(UVM)", Rochester Institute of Technology, Rochester, New York, December 2018.
12. Yakovlev, Alexandre V., Albert M. Koelmans, and Luciano Lavagno. "High-level modeling and design of asynchronous interface logic." IEEE Design & Test of Computers, Vol 12.1, pp 32-40, 1995.
13. K.K. Yi, "The Design of a Self-Timed Low Power FIFO Using a Word- Slice Structure", M.Phil Thesis, University of Manchester, September 1998.
14. Wang, Xin, Tapani Ahonen, and Jari Nurmi, "A synthesizable RTL design of asynchronous FIFO", System-on-Chip Proceedings, 2004 International Symposium on IEEE, pp. 123-128, 2004.
15. T., Nowick, S.M., "Low-latency asynchronous FIFO's using token rings", Advanced Research in Asynchronous Circuits and Systems, Proceedings. Sixth International Symposium, Vol 2-6, pp 210 – 220, April 2000.
16. Doulos Ltd, "SystemVerilog Golden Reference Guide", Vol II, pp. 1-11, 2003.
17. J. Bergeron, E. Cerny, A. Hunter, and A. Nightingale, "Verification Methodology Manual for System Verilog", Springer, 2006.
18. Andreas Meyer, "Principles of Functional Verification", Vol I, pp. 1-10, 2004.
19. Ashok B. Mehta, "SystemVerilog Assertions and Functional Coverage", (2nd ed.) [Online]. Available: http://extras.springer.com
20. Clifford Cummings," SystemVerilog Assertions - Bindfiles & Best known practices for simple SVA usage", Sunburst Design (SNUG), 2009.
21. Synopsys, "System Verilog Assertions Checker Library Quick Reference", April, 2006.
22. S. Vijayaraghavan and M. Ramanathan, "A Practical Guide for System Verilog Assertions". Springer, 2005.
23. SystemVerilog—Unified Hardware Design, Specification, and Verification Language, IEEE Std. 1800, 2018.
24. Clifford Cummings," SystemVerilog Assertions - Bindfiles & Best known practices for simple SVA usage", Sunburst Design (SNUG), 2016.

## AUTHORS PROFILE

**Mohini Akhare** received the B.E. degree in the Electronics and Telecommunication Engineering in the year 2017 and currently pursuing M. Tech. in VLSI Design from Shri Ramdeobaba College of Engineering and Management (RCOEM), Nagpur.

**Dr. Nitin Narkhede** received his Bachelor's degree from Shri Guru Gobind Singhji Institute of Engineering and Technology Nanded (M.S.) in 2000, Master's and Doctoral degree in the year 2006 and 2012 respectively from Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur. He is working as Associate Professor and M. Tech. (VLSI Design) Coordinator in Department of Electronics Engineering of Shri Ramdeobaba College of Engineering and Management (RCOEM), Nagpur. He has 16 years of teaching experience. He has published 25 research papers in reputed indexed Journals (SCOPUS, ESCI etc.) and conferences. His research interest includes Digital VLSI Design, Digital system Verification, Micro / Nano Sensors etc. He is a life member of ISTE and recipient of 'Rashtra Sewa Award' for his contribution in the field of education.