

Coverage Driven Verification of Synchronous FIFO using UVM

C. Anusha Bhanu^{1,*}, Rajesh Odela¹, G.R. Padmini²

Abstract

FIFO (first in first out) is a memory array in which the data written first into the memory will be read first with the help of the control signals read and write where the control signal write is used to write the data into the memory and the control signal read is used to read the data from the memory. To avoid the memory overflow and to know that whether there is a space in memory to write, a few status signals are asserted which are empty and full. The empty signal is high which means that all the memory location in the FIFO is empty i.e., no data written is into it and if the full signal is high then all the memory location in the FIFO is occupied with some data. The FIFO has been designed with a depth of 64 locations and the size of each location is 16-bit and verify it with the help of System Verilog-based UVM (universal verification methodology). The coverage driven verification is being used to verify the design functional accuracy. Constrained randomization has been used to attain the maximum amount of coverage (i.e., both code coverage and functional coverage). Code coverage of 99.43% and functional coverage of 100% have been achieved.

Keywords: Code coverage, design and verification, functional coverage, synchronous FIFO

INTRODUCTION

It is often required in electronic circuits to exchange data safely between memory modules. Synchronous FIFO is a memory queue used as a buffer between two synchronous clock domains. The FIFO receives information from one module, stores it in memory, and upon reading request, the data is read in a first in first out manner [1]. That means in a FIFO, it is not possible to write or read randomly. The full status pin is asserted when all the locations of the FIFO memory are fully written. Similarly, the empty status pin is asserted when all the memory locations are read out. Empty and full status pins function with the help of read pointer and write pointer. To ensure the proper functioning of the FIFO, it is required to test the design by giving inputs. This paper describes the verification of FIFO using System Verilog which is a hardware verification language. Inputs are given to the design using constrained randomization and tested for functionality. The environment for testing is implemented

such that FIFO with a similar structure can be implemented. Verification takes place much before the fabrication process to avoid the respin of the chip [2].

DESIGNING AND WORKING OF FIFO

Synchronous FIFO has been considered for designing. The building blocks of synchronous FIFO are memory array, write control logic, and read control logic [3]. The I/O signals are given in Table 1.

The memory array used in this module is 64 bits RAM with each location 16 bits wide. The read or write operation in the FIFO occurs according to we

*Author for Correspondence

C. Anusha Bhanu

¹Assistant Professor, Department of Electronics and Communication Engineering, Vasavi College of Engineering, Hyderabad, Telangana, India

²Associate Professor, Department of Electronics and Communication Engineering, Vasavi College of Engineering, Hyderabad, Telangana, India

Received Date: March 12, 2021

Accepted Date: April 10, 2021

Published Date: April 29, 2021

Citation: C. Anusha Bhanu, Rajesh Odela, G.R. Padmini. Coverage Driven Verification of Synchronous FIFO using UVM. Recent Trends in Electronics & Communication Systems. 2021; 8(1): 1–7p.

signal [2]. The memory array is designed such that when $we=0$ read operation is implemented when $we=1$ write operation is implemented.

Table 1. FIFO pin description.

S.N.	Signal	Description
1.	Clock	All signals are operated at the rising edge of the clock.
2.	Reset	When active, sets all output signals to zero.
3.	Data in	Data input
4.	we	When $we = 1$, data is written, when $we = 0$, data is read out from FIFO.
5.	Full	Indicates the availability of the memory location for data to be written.
6.	Empty	Indicates the availability of memory location for data to be read out.
7.	Data out	Data output

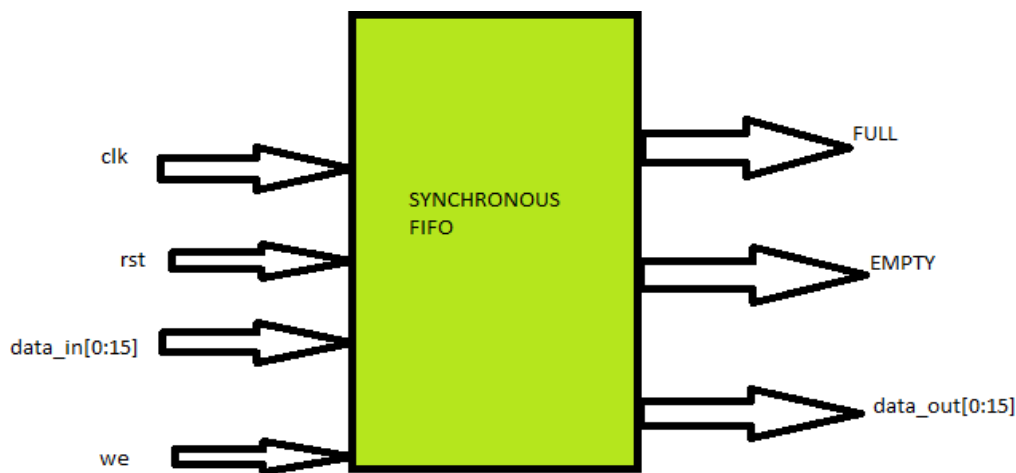


Figure 1. FIFO design.

At a rising edge of the clock, either the read operation or the write operation occurs. The memory array functions with the help of two pointers, a read pointer(rd_ptr) and a write pointer(wr_ptr). Upon every write request, the write pointer is incremented to the next empty location. Similarly, upon read request, the read pointer is incremented to the next available location. To avoid overflow of data, a full signal is asserted whenever the write pointer is at the last available empty location and the write pointer stops incrementing. When all the locations are read, an empty signal is asserted to denote FIFO is empty. The write and read operations take place only at the rising edge of the clock. The block diagram of synchronous FIFO is given in Figure 1. The verification is an important phase for any successful design [4].

VERIFICATION PROCESS

Verification aims to provide that the design meets the functional requirements. Randomized stimulus under the control of user-specified constraints generated by the environment is applied as input. This environment verifies each test applied automatically. Among the presently available technologies, UVM is best known for its features. UVM lays out a set of guidelines to be followed for the creation of test benches [5]. This will ensure testbench uniformity between different teams, flexibility, and ease of maintaining test benches. The UVM verification plan is shown in Figure 2 and each component is explained here.

There are two types of components in UVM i.e., static and dynamic. Static components are there since the beginning and exist even after the simulation. Dynamic components when generated, perform their task and their lifespan is completed at the end of the simulation cycle. Dynamic components are also called objects. The basic component of the verification environment is the sequencer. It consists of data packets required for generating stimulus. Sequence items are

randomized in sequences to generate stimulus. The driver receives the generated stimulus, converts them into test cases and they are given to DUT through the interface.

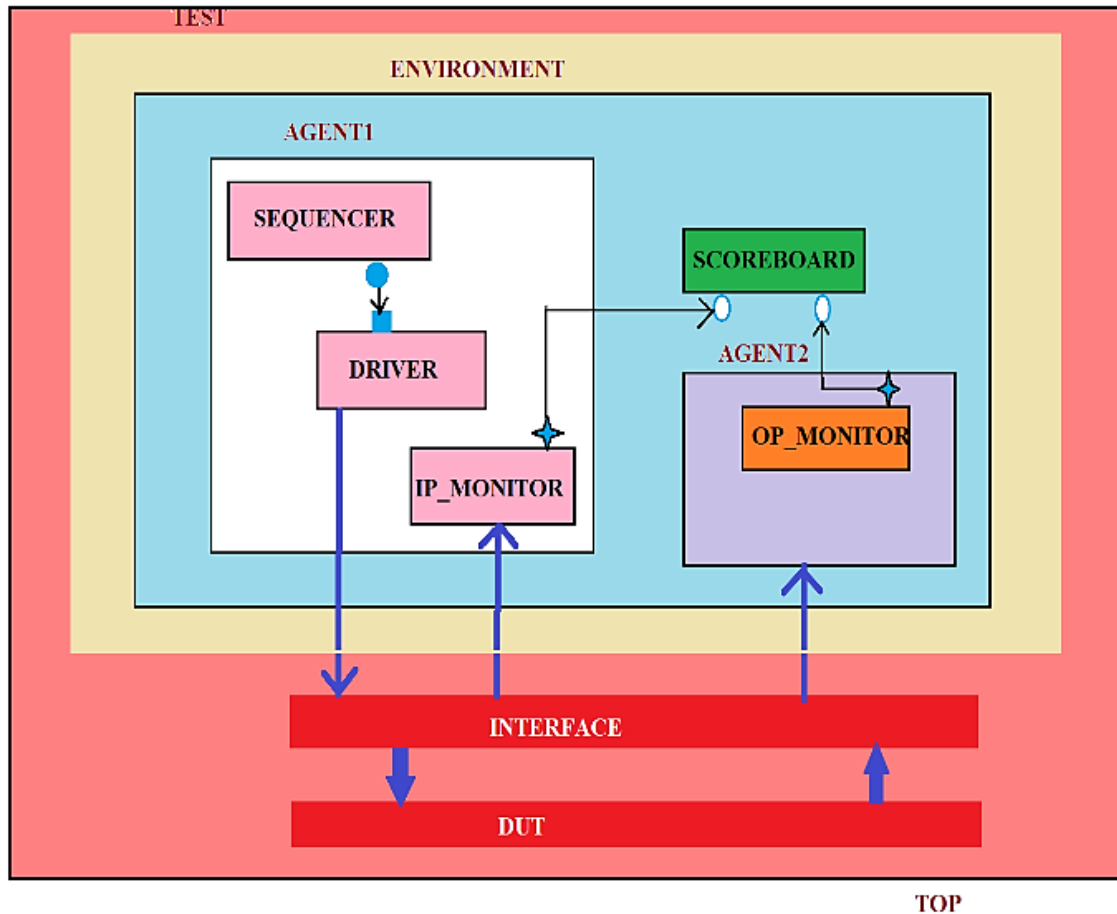


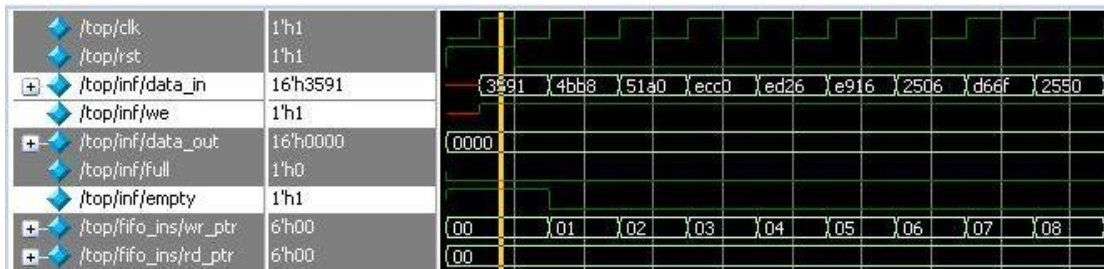
Figure 2. UVM hierarchy.

The sequencer and driver are connected via analysis ports. Analysis ports are required to establish connections between components for broadcasting transactions. Accordingly, import ports and export ports are defined. The monitor captures the signal activity from DUT and converts it into transaction-level objects to communicate with other objects. The input monitor is used to capture data sent to DUT and an output monitor to store data received from the DUT. An agent includes driver, sequencer, and monitor connected through TLM interfaces. Two agents are used here. The active agent which includes a sequencer, driver, and input monitor, enables data to be driven to DUT via driver. The passive agent which is used for checking and coverage includes an output monitor. The scoreboard is a component that verifies the functionality of the FIFO by comparing the input and output data of DUT received at packet level through input and output monitor via ports. It is possible to instantiate the scoreboard and agents directly but it is not recommended because tests are not reusable and any changes in the design will require updating of multiple files which is time-consuming. UVM environment encapsulates agents and scoreboards, and defines their default configuration required by FIFO. It can be instantiated within multiple tests. UVM test contains the test cases where sequences are created and started in the test [6]. An interface is a group of signals in a block, which connects DUT and testbench. UVM top module is the static container that instantiates test, interface, and DUT.

DESIGN AND SIMULATION ANALYSIS

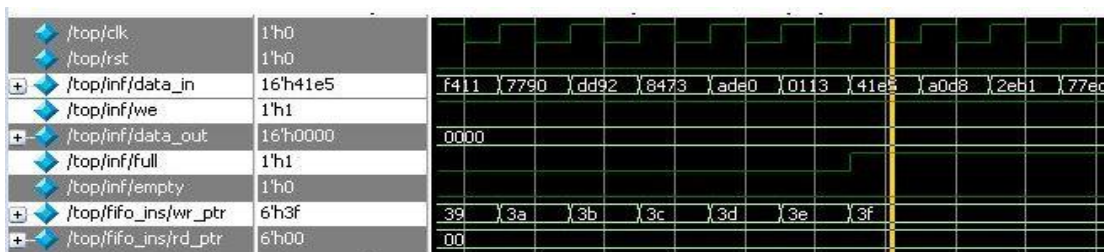
This project is designed and simulated using the Mentor Graphics Questa sim tool for simulation purposes. The first signal is the clock which is a free-running signal with a frequency of 5 Mhz. The

next signal is reset. When forced a value one, it sets all output signals to zero. The output signals data_out, full, empty, are set to zero. Figure 3 shows the scenario of data write in FIFO through the generated waveform. The reset signal has been forced to zero. Here, 64 depth memory, i.e. 64 locations each of 16-bit width, has been considered.



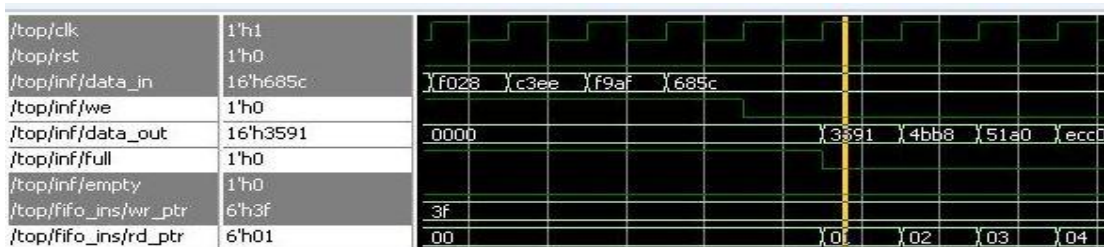
we=1 , i.e write operation & initially fifo is empty

Figure 3. FIFO write operation.



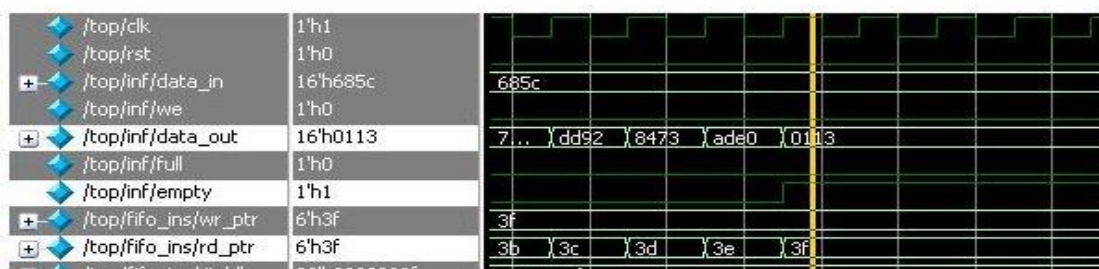
we=1, after 64 locations have been written then full=1 & write pointer(wr_ptr) stopped incrementing

Figure 4. Write operation completed.



we=0 i.e read operation , the data out is same as the data written previously, & rd_ptr is incremented every clock .

Figure 5. FIFO read operation.



once all the 64 loations have been read then the empty =1 & read pointer (rd_ptr) stopped incrementing

Figure 6. Read operation completed.

The data is given through the data_in signal of FIFO. Then we signal is set to 1 to enable the write operation. Initially, the full signal is zero and the empty signal is 1 as nothing is written in the FIFO. The write pointer starts incrementing at every rising edge of the clock and traverses through all the locations as long as the we=1 condition is satisfied. The empty signal becomes zero. Once the write pointer is at the 64th location, the full signal is asserted 1. This indicates that the memory is full and there is no more location to write data. This can be seen in Figure 4.

Figure 5 depicts the read operation scenario. The data written in the FIFO is now read out by forcing 0 on we signal. The read pointer which is at the first location initially starts incrementing for every rising clock edge [7]. The full signal now becomes zero. The read pointer starts incrementing for every rising edge of the clock and traverses through all the 64 locations. The data_out signal gives the data written in the memory. Once the read pointer reaches the last location, data from all the locations are read out, and now the FIFO becomes empty, the empty signal is asserted 1, which is depicted in Figure 6.

VERIFICATION RESULTS AND ANALYSIS

The scoreboard report is explained in this section as depicted in Figure 7. The scoreboard report has been generated with the help of the scoreboard component. Test case data are written and data read out from the FIFO are stored in the scoreboard. A data match occurs only when input data equals output data. The display statements come from the scoreboard.

```
UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
-----
Name                                     Type                                     Size  Value
-----
uvm_test_top                             fifo_test                             -      0471
  env                                     fifo_env                             -      0478
    agent1                                 fifo_agent1                           -      0485
      drv                                 fifo_drv                             -      0710
        rsp_port                         uvm_analysis_port                     -      0725
        seq_item_port                   uvm_seq_item_pull_port                -      0717
      mon1                                 fifo_mon1                             -      0733
        item_collected                 uvm_analysis_port                     -      0742
      seqr                                 fifo_seqr                             -      0601
        rsp_export                      uvm_analysis_export                  -      0608
        seq_item_export                 uvm_seq_item_pull_imp                 -      0702
        arbitration_queue               array                                0      -
        lock_queue                     array                                0      -
        num_last_reqs                   integral                             32     'd1
        num_last_rsps                   integral                             32     'd1
    agent2                                 fifo_agent2                           -      0492
      mon2                                 fifo_mon2                             -      0756
        item_collected_port1           uvm_analysis_port                     -      0763
      sb                                   fifo_sb                               -      0499
        ip_fifo                         uvm_tlm_analysis_fifo #(T)           -      0506
        analysis_export                 uvm_analysis_imp                     -      0545
        get_ap                          uvm_analysis_port                     -      0537
        get_peek_export                 uvm_get_peek_imp                     -      0521
        put_ap                          uvm_analysis_port                     -      0529
        put_export                      uvm_put_imp                           -      0513
        op_fifo                         uvm_tlm_analysis_fifo #(T)           -      0553
        analysis_export                 uvm_analysis_imp                     -      0592
        get_ap                          uvm_analysis_port                     -      0584
# -----
# 1025driver pkt::inf.data=26716...inf.we=0
# UVM_INFO fifo_driver.sv(30) @ 1025: uvm_test_top.env.agent1.drv [DRV] DRV TRANSACTION TO DUT
# 1025outmonitrrrrrrrrrr=13713
# UVM_INFO fifo_mon2.sv(35) @ 1025: uvm_test_top.env.agent2.mon2 [MON2] MON2 TRANSACTIONS
# UVM_INFO fifo_seql.sv(40) @ 1025: uvm_test_top.env.agent1.seqr@seq [SEQ] SEQUENCE TRANSACTIONS
# 1025queue2 values are13713
# UVM_INFO fifo_scoreb.sv(59) @ 1025: uvm_test_top.env.sb [SB1] DATA MATCHED
# 1025MATH***datain=13713,dataout=13713
# 1025mmeue126716
```

Figure 7. Scoreboard report generation.

Coverage Summary By Instance:

Scope ▾	TOTAL ▾	Statement ▾	Branch ▾	FEC Expression ▾	FEC Condition ▾	Toggle ▾
TOTAL	99.43	100.00	100.00	100.00	100.00	97.15
top	100.00	100.00	--	--	--	100.00
inf	98.64	--	--	--	--	98.64
fifo_ins	99.18	100.00	100.00	100.00	100.00	95.91

Figure 8. Code coverage report.

Coverage Summary by Structure:			Coverage Summary by Type:						
Design Scope ▾	Hits % ▾	Coverage % ▾	Total Coverage:		100.00%	100.00%			
			Coverage Type ▾	Bins ▾	Hits ▾	Misses ▾	Weight ▾	% Hit ▾	Coverage ▾
fifo_top_sv_unit	100.00%	100.00%	Covergroups	17	17	0	1	100.00%	100.00%
fifo_coverage	100.00%	100.00%							

Report generated by Questa (ver. 10.6c) on Monday 02 November 2020 12:28:20 with command line:

Figure 9. Functional coverage report.

Coverage can be defined as the ratio of the verification objectives met to the total number of verification objectives. Code coverage and functional coverage are two types of coverages analyzed. Code coverage accounts to the amount of code utilized. Toggle coverage, branch coverage, conditional coverage, and expression coverage are subparts of code coverage. The design specification that has been met according to predicted results is verified through functional coverage. A coverage report has been generated using constraint randomization. It shows that 99.43% code coverage has been achieved. The coverage report of the design has been analyzed. Toggle coverage assures how many times the signals have been toggled [4]. Statement coverage is the number of code lines (statements) executed according to the applied stimulus. Every statement has been executed so this is 100%. Branch coverage and conditional coverage will report the true or false of the branch-like if-else, case, and ternary operator [8]. Branch coverage of 100% has been achieved. Expression coverage is the ratio of the number of test cases checked to the total number of cases present. The coverage report is shown in Figure 8.

Constraint randomization allows us to define random values that satisfy the required condition. Using constraint randomization, 100% functional coverage has been achieved. This is depicted in Figure 9.

CONCLUSION

The main aim of this paper is to design, verify, and achieve maximum coverage for synchronous FIFO. The DUT was designed, simulated, and analyzed for input conditions like read and write controls, and for its status flags which are full and empty [5]. The FIFO module was verified using layered test bench verification and the constraint randomization technique which is the most important part of this paper. The layered test bench was designed using components. The verification environment and the DUT were connected through the interface [9]. The input stimuli were randomized and automated using the verification environment to cover every possible corner case to detect any bugs [10]. The scoreboard report was analyzed and we found that all the test cases were matching as desired. Using randomized input stimuli and constrained randomization, we have achieved a code coverage of 99.43% and functional coverage of 100% whereas the [4] has achieved code coverage of 80% and functional coverage of 90%.

REFERENCES

1. Akhare Mohini, Narkhede Nitin. Design and verification of generic FIFO using layered test bench and assertion technique. International Journal of Engineering and Advanced Technology (IJEAT). 2019; 8(6): 5254–5260.
2. Vinoth Nagarajan. The design and verification of a synchronous first-in-first-out (FIFO) module using system verilog based universal verification methodology (UVM). [Master's Project]. New York: Rochester Institute of Technology. 2018.
3. Morris Mano, Emeritus, Michael D. Digital Design: with an Introduction to the Verilog HDL. 5th ed. Pearson Education: Uttar Pradesh, India: January 2013.
4. Bergeron Janick. Writing Testbenches Using System Verilog. Berlin, Germany: Springer; 2006.
5. Maurya Shilpa. Design of RTL synthesizable 32-Bit FIFO memory. International Journal of Engineering Research & Technology (IJERT). 2016; 5(11): 591–593.
6. Bergeron Janick. Writing Testbenches: Functional Verification of HDL Models. Germany: Springer; 2003.
7. Samir Palnitkar. Verilog HDL: A Guide to Digital Design and Synthesis. 2nd ed. Hoboken, New Jersey: Prentice Hall PTR; 2003.
8. Navaid Zafar Rizvi, Rajat Arora. Implementation and verification of synchronous FIFO using system verilog verification methodology. Journal of Communications Technology, Electronics and Computer Science. 2015; 2: 18.
9. Amit Kumar, Shankar, Neeraj Sharma. Verification of asynchronous FIFO using system. International Journal of Computer Applications. 2014; 86(11): 0975–8887.
10. Mehta Ashok B. System Verilog Assertions and Functional Coverage. 2nd ed. New York: Springer Verlag; 2013.