Implementation and Verification of Asynchronous FIFO Under Boundary Condition

P Rajshekhar Rao
Dept. of Avionics
Inst. of Science & Technology, JNTUK
Kakinada, India

Manju Nanda
Aerospace Electronics & System division
CSIR – National Aerospace Laboratories
Bangalore, India

***Abstract*—First in First out(FIFOs) are frequently used to securely pass information starting with one clock area then onto the next asynchronous clock area. Utilizing a first in first out (FIFO) to pass information starting with one clock area then onto the next clock area. The main objective of this paper is the module level testing of FIFO based on functionality, performance, safety and coverage. This document describes and shows the module level testing of FIFO/Buffer and verifies the completeness, compliance and correctness of the implemented functionality with reference to the hardware requirement. This can be achieved by developing test cases under boundary condition.**

***Keywords*— FIFO; Asynchronous FIFO; Gray Counter; Assertion**

I. INTRODUCTION

An asynchronous FIFO basically works on the principal of buffer. To understand about the asynchronous FIFO clearly is to synchronous the clock frequency between two control signals which decides the criteria of performance based testing as well as safety measures. In this document it describes about Async_FIFO how to instantiate while declaring the component instance with independent clock for read and writes operation [1]. This instance stores packed data information about the coming bit-stream. Instance asynchronous FIFO(inst_async_fifo) shall be instantiated inside the buffer. To decide the functionality of FIFO it mostly depends on the control signals like rd_en and wr_en.

II. THE DESIGN UNDER TEST -FIFO

First-In-First-Outis a sort of line used to incidentally store the information and recover it. As the name infers, the information that goes-in to begin with, is recovered first from the line, basically to implement the async_FIFO, we need to synchronous two different clock frequency they are wr_clk and rd_clk to decide the functionality of FIFO or buffer is shown in Fig. 1, as well as to verify according to the industry like aerospace sectors need to fulfill the requirement as per DO-254. For testing the different scenario as per DO-254 [2], it should meet the entire requirement like RESET scenario i.e. reset is ‘high’ and observes output rd_data_o, full_o then the following output ports will be assigned to corresponding ports Full_o=’1’, Empty_o=’1’and DataOut_o will be 32 bit zero[1].DUT local signals can’t be observed in the Test Bench and with assertion (only input and output signals of DUT can be observed in Test Bench). Local signal can be observed in the simulated waveform and it can’t be logged. The information flow diagram is shown in Fig. 2.

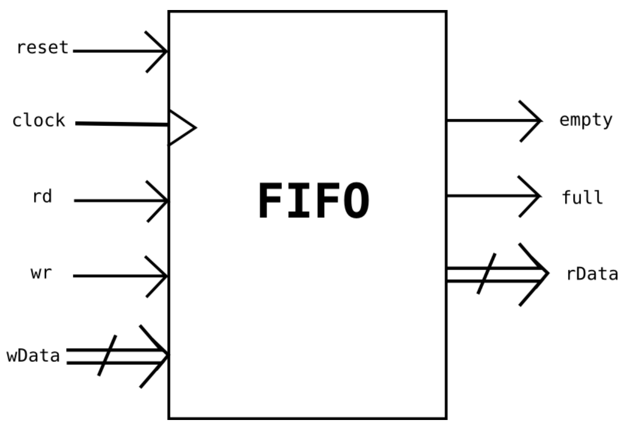


Fig. 1. RTL block diagram of FIFO [1]

From Fig. 2, as discussed in DO-254,any incapacity to confirm particular necessities by test on the device itself must be supported and elective methods for verification given. Affirmation experts support confirmation by test for formal check as a result of the basic reality that hardware flies, not simulation models. Necessities discussing FPGA pin level behavior must be confirmed by test.

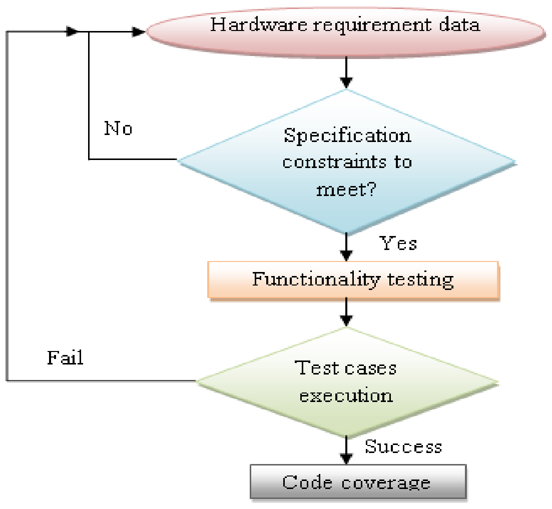


Fig. 2. Proposed flow chart for FIFO

The issue is that trying the field programmable gate array (FPGA) device at the board level gives low FPGA input control, accordingly making it hard to infuse certain signs for typical normal tests and robustness tests. The input ports are shown on left side pointing inside the block, while output ports appear on right side coming out from the block. The meanings of the each port have been clearly explained in Table I.

TABLE I. MEANING OF INPUT - OUTPUT PORTS [1]

Port	Direction	width (bits)	Significance
reset	in	1	all internal registers are reset to zero (0).
clock	in	1	the synchronizing signal for FIFO operation.
rd	in	1	a 1-byte of data is read from FIFO by asserting thissignal.
wr	in	1	a 1-byte of data is written to FIFO by asserting this signal.
wData	in	32	32-bit data to be written into FIFO.
empty	out	1	Indicates that FIFO contains no data to read from.
full	out	1	Indicates that FIFO has run out of storage.
rData	out	32	32-bit data read from the FIFO.

III. OPERATION OF FIFO

Endless supply of reset flag, every one of the registers including pointers and counter, are reset to 0. At that point, at each rising edge of clock flag, following conditions are checked [3]. There are total 4 cases they are clearing explained as:

A. *If 'rd = 1' and 'wr = 0'*

- The data pointed by bottom register is placed on rData outputport.
- Bottom register is incremented by 1.
- Counter register is decremented by 1.

B. *If 'rd = 0' and 'wr = 1'*

- The data present on wData input port is written to register pointed by top register.
- Top register is incremented by 1.
- Counter register is incremented by 1.

C. *If 'rd = 1' and 'wr = 1'*

- The data pointed by bottom register is placed on rData outputport.
- The data present on wData input port is written to register pointed by top register.
- Bottom register is incremented by 1.
- Top register is incremented by 1.
- Counter register is unaffected.

The *full* and *empty* flag outputs are asserted for following conditions:

1) *full = '1' if*

Counter register = 111_bindicating that all registers are written,andinput wr = 1.

2) *empty = '1' if*

Counter register = 000_bindicating that no register are written,andinput rd =1.

D. *If FIFO works simultaneously i.e. both read and write*

In this case, read and write operation will perform simultaneously by enabling both the control signals rd_en_i signal and wr_en_i. The FIFO first writes the incoming data into it and after some clock pulses it starts to read.

IV. BOUNDARY CONDITION TEST CASES

The test cases are developed for the Boundary condition verification. The test cases are written for all the possible combinations of reset signal (ie, reset_i) and input signals. Boundary condition testing is a process of testing the module by slightly varying the value of the input parameters above or below of its boundary value [8]. In this section the FIFO is made to operate at various scenarios like variation in the frequency of the clock, applying boundary conditions for the time duration for enabling the write and read operations. There are two possible cases. They are explained as:

A. *When RdClk_i = 80 MHZ and WrClk_i = 122.61 MHZ*

In this case, the frequency of read clock RdClk_i is increased to 80 MHZ which operates at 79.68 MHZ keeping write clock WrClk_i frequency constant at 122.61MHz. The reading operation takes 0.05 ns less time. This affects the performance of the module. There will be possible to form slack (Required Time- Arrival Time).

B. *When RdClk_i = 79.6875 MHZ and WrClk_i = 120 MHZ*

In this case, the frequency of write clock WrClk_i is reduced to 120 MHz which operates at 120 MHz keeping read clock RdClk_i frequency constant at 79.68MHz. The writing operation delayed by 0.1774 ns. This affects the performance of the module. There will be possible to form slack (Required Time- Arrival Time).

V. ASYNCHRONOUS FIFO POINTERS

With a specific to implement FIFO design, one should need to see how the async_FIFO pointers function works. There are essentially two pointers.

1. Address Pointer
2. Read Pointer

When Areset_i is high then all the signals are reset to its initial value. When the Areset_i is low and Wr_en_i is high then the 32 bit data from the different module is written into the Buffer with respect to the WrClk_i pulses. When the Areset_i is low and Rd_en_i is high then the 32 bit data which is stored in the data buffer will be fetched by the module with respect to the RdClk_i pulses. If the Buffer is full then the Full_0 signal goes high which indicating that the data is available to read [5].

So also, the Write Enable control signal to write the Data into the FIFO the design should have to satisfy this condition 1=Write; 0=Don't Write. Likewise the Read Enable control signal to read the FIFO the hardware design requirement should satisfy the following condition 1=read; 0=Don't Read. This async_fifo write pointer will write the data and store the date up to maximum of 8K RAM location it means the FIFO will read and stack out that much of location-stored data only. The test cases are written for all the possible combinations of reset signal (i.e., reset_i) &control signals (i.e.,wr_en_i and rd_en_i). The 32 bit data are passes from input port to the output port. The Boundary test cases are divided into 3 cases, first two cases are carried out in two different cycles and each cycle is equal to the capacity of the buffer [4]. The operation of

the buffer is divided into three cases with three different cycles as mentioned in Table II. To verify the boundary level of testing, some need to change in the design level like to change in the clock domain which gives the clock skew factors with slack effect., and to change in the input bit stream, suppose the stream consists of 4bit, to check the boundary condition change the 4bit MSB or LSB positions [14].

TABLE II. DETAILS ABOUT THE BOUNDARY TEST CASES

Sl.No.	Cases	Operation	Cycles	Range of count_wr and count_rd
1	Case1	Write only	First cycle	0 to 8976
2	Case2	Read only	Second cycle	8976 to 17952
3	Case3	Both Write and Read operation	Third cycle	17952 to 35904

ReadPort:

Process reads data from memory and sends to output ports

```
 ReadPort : process (RD_Clk_i)
 begin
  if (RD_Clk_i'event and RD_Clk_i = '1') then
   DataOut_l <=
ram_mem_l(To_integer(unsigned(read_addr_next_l)));
  end if;
 end process ReadPort;--end of ReadPort process
```

WritePort:

Process writes the input ports data to memory

```
 WritePort : process (WrClk_i)
 begin
  if (WrClk_i'event and WrClk_i = '1') then
   if (write_allow_l = '1') then
    ram_mem_l(To_integer(unsigned(write_addr_l))) <=
DataIn_i;
   end if;
  end if;
 end process WritePort;--end of WritePort process
```

VI. HANDLING FULL AND EMPTY CONDITIONS

Condition for designing the async_FIFO for FULL, it is based on the counter_wr. After writing the data in to the last location of the buffer and full_o become ‘1’ (count>8976) else full_o will be set to ‘0’. The typical scenarios, ‘when boundary condition for the time duration for enabling the read and write operation which handles the factor of full and empty conditions [9]. The boundary condition is given to the time duration of wr_en_i and the rd_en_i enabling. Whenever the write enable is ‘1’ the data buffer starts to write and whenever the read enable is ‘1’ the read operation starts in the data buffer.Allows flags determine whether the FIFO control logic can operate. If read_enable is driven high, and the FIFO is not empty, then reads are allowed. Similarly if the write_enable is driven high, and the FIFO is not full, then writes are allowed [13]. Empty flag is set on initial (reset) or when gray code counter are equal, or when there is one word in the FIFO and a read operation about to be performed.

```
read_allow_l  <= (read_enable_l and not empty_l);
write_allow_l <= (write_enable_l and not full_l);
```

Transfer Read and Write Gray pointers to the other clock domain. Convert Gray pointers to binary equivalent. Delay Read and Write Binary counters in respective domain and use this value with the boundary crossed value from other domain to create Length values [11]. The delay of the Binary values is done to have binary value for the current active Gray value in each of the clock domains.

```
EmptyFlag : process (RD_Clk_i, Areset_i)
 begin
  if (Areset_i = '1') then
   empty_l <= '1';
  elsif (RD_Clk_i'event and RD_Clk_i = '1') then
   if (empty_allow_l = '1') then
    empty_l <= emptyg_l;
   end if;
  end if;
 end process EmptyFlag;--end of EmptyFlag process
```

A. Use of Grey Counter

Generally, grey counter is to improve theperformance ofFIFO functionality into two ways, Readaddress_counter and Writeaddress_counter. The primary one is binary (read_addr) and the grey code are generated via pipelining the binary-to-gray code result. The initial values are important, so they are in sequence. Grey code addresses are used so that the registered full and empty are always clean, and never in an unknown condition or state due to the asynchronous relationship of the write and read clocks. In the worst case scenario, full and empty would simply stay active one cycle longer; it would not generate an error or a false value [7].

```
ReadAddrCnt : process (RD_Clk_i, Areset_i)
 begin
  if (Areset_i = '1') then
   read_addr_l <= (others => '0');
  elsif (RD_Clk_i'event and RD_Clk_i = '1') then
   --Read the next address pointer value
         read_addr_l <= read_addr_next_l;
  end if;
 end process ReadAddrCnt;--end of ReadAddrCnt process
```

The two conditions decoded with special carry logic are Empty and Full (gated versions). These are used to determine the next state of the Full/Empty flags. Carry logic is used for optimal speed. (The previous implementation of AlmostEmpty and AlmostFull have been wrapped into the corresponding carry chains for faster performance).When write_addrgray_l is equal to read_addrgray_l, the FIFO is Empty, and emptyg_l (combinatorial) is asserted. Or, when write_addrgray_l is equal to read_nextgray_l (1 word in the FIFO) then the FIFO potentially could be going Empty, so emptyg_l is asserted, and the Empty flip-flop enable is gated with empty_allow, which is conditioned with a valid read.Similarly, when read_lastgray_l is equal to write_addrgray_l, the FIFO is full (511 addresses)[10]. Or, when read_lastgray_l is equal to write_nextgray_l, then the FIFO potentially could be going Full, so fullg_l is asserted, and the Full flip-flop enable is gated with full_allow, which is conditioned with a valid write.

Presently to look at Mealy and Moore machine one case is taken. Think about the instance of a circuit to distinguish a couple of 1's or 0's in the single piece input. On the off chance that two maybe a couple's zero's comes in a steady progression, yield ought to go high. Generally yield ought to be low [9].

Note: To have utilized the full address space (512) would have required extra logic to determine Full/Empty on equal addresses, and this would have slowed down the overall performance, which was the top priority.

```
if (reset_i = '1') then
write_nextgray_l(write_nextgray_l'high-1 downto 0) <= (others => '0');
write_nextgray_l(write_nextgray_l'high) <= '1';
elseif (WrClk_i'event and WrClk_i = '1') then
if (write_allow_l = '1') then
write_nextgray_l <= Bin2Gray(write_addr_l);
end if;
```

VII. TEST ENVIRONMENT

The test environment provides the details about the test bench used for the verification of functionality of FIFO. The software tools used and their setting is mentioned clearly in Fig. 3, which represents the detailed test environment.

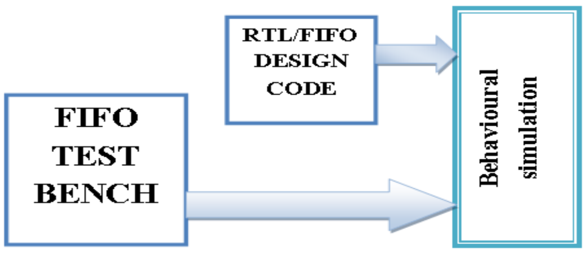


Fig. 3. Tool Environment

VIII. RESULTS

There are two possible cases to study. They are when FIFO is full and second one is when FIFO is empty. It can be observed that in the two cases the results reveals the control signal status.

A. *when full is equal to '1'under boundary condition*

Scenario: when FIFO write only (WR), then full = 1 and read (RD) =1 and empty=1. As observed from Fig. 4, FIFO is full or FIFO_FULL='1', which indicates the incoming data is full, so no data will be stored in the buffer until the control signal like wr_en depends on wr_clk will high.

B. *when empty is equal to '1'under boundary condition*

Scenario: when read (RD) only, so write (WR) =0, empty=1 and full=0. As observed fromFig. 5, FIFO is empty or FIFO_EMPTY='1', which indicates the incoming data is full, need to empty to allow the next data to store. So data will be stored in the buffer until the control signal like rd_en depends on rd_clk will high.

IX. CONCLUSION

. The performance and safety analysis is carried out by varying the frequency in inputs clocks (wr_clk_i and rd_clk_i) that is carried in boundary condition.The performance metrics of the buffer/FIFO are latency and throughput and data loss. The Latency is the time taken by the buffer to receive the data from instantiation and the time taken by the buffer to send the data to the output port. Latency may occur due to propagation delay of data or transmission delay from input port to output port of the buffer. The throughput is the amount of data successfully transmitted in a given time period. This is computed based on the time period of write clock (Wclk_i) and the read clock (Rclk_i). The variation in these clocks like clock skew, clock drift and the clock distortion have an impact on wr_en_i and rd_en_i duration and variation in time period in storing and retrieving data in the buffer.

Since writing and reading of data in the buffer/FIFO is depended on clocks, delayed clocks will leads to delay in the entire process of the module that impact on other modules connected to its output port. Data loss may occur with distorted clock pulses.

X. FUTURE WORK

The implementation of asynchronous FIFO and verification of FIFO under boundary is an crucial role for an industry whenever they need to instantiation the ASYNC_FIFO as to store the frame or any sort of data, need to check/ verify all scenario like one of method/ test case i.e. boundary presently. In future verification of FIFO as per DO-254 designing, it carry the test case to be in robustness method using one of transcript techniques like SELF-CHECKING test bench to be added into stimulus and further one more techniques to verify the FIFO for better instance which is constrained random verification, which improves the performance and safety analysis of the FIFO which is more secure in the sense, no data loss, encryption and decryption or read/write to be done fastly including time factor.

XI. REFERENCES

[1] Dadhania Prashant C., “Designing Asynchronous FIFO”, International journal of information, knowledge and research in electronics and communication engineering, Volume 02, issue 02, pp-561-563.

[2] Stuart Sutherland, “Verilog HDL quick reference Guide”, Based on the Verilog – 2001 standard, IEEE standard1364-2001, www.sutherland-hdl.com

[3] “Design Compiler User Guide” version X-2005.09, Synopsys,Sep 2005.

[4] SamirPalnitkar,“Verilog HDL: a guide to digital design and synthesis”, second edition, Prentice Hall PTR publications, Feb 21, 2003.

[5] L.K.Hu and Q.H wang,”UART Based reliable communication and performance analysis”, Computer Engineering Vol32 No.10,may2006,pp5-21

[6] Charles H.Roth “Digital system design using VHDL “2nd edition, Thomson publication.

[7] Zainalabedin Nawabi “VHDl Analysis and modelling of Digital system” 2nd edition, McGraw-Hill, Hardcover, published January 1998.

[8] X.D.Wu and B.Dai, “Design of interface Between high speed A/D and DSP based on FIFO”, journal of Beijing Institute of petrochemical Technology vol14 No.12, June 2006, pp 26-29.

[9] Clifford E. Cummings and Peter Alfke, “Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparison ,” SNUG 2002 (Synopsys User Group Conference, San Jose, CA,2002) User Papers, March 2002, Section TB2, 3rd Paper.

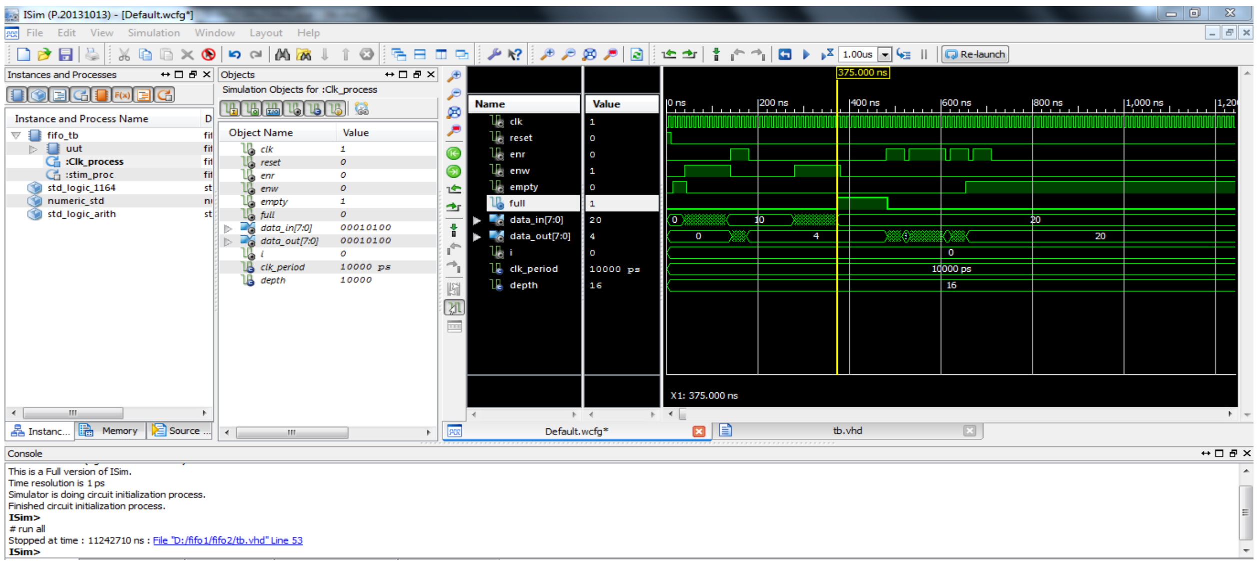
[10] A.V. Yakovlev A.M. Koelmans L. Lavagno "High-Level Modeling and Design of Asynchronous Interface Logic" IEEE Design and Test of Computers Spring 1995.

[11] Q. T. Ho J. B. Rigaud L. Fesquet M. Renaudin R. Rolland "Implementing Asynchronous Circuits on LUT Based FPGAs" Proceedings of 12th International Conference on Field-Programmable Logic and Applications September 2002.

[12] C. Law B. H. Gwee J. S. Chang "Modeling and Synthesis of Asynchronous Pipelines" IEEE Transactions on Very Large Scale Integration (VLSI) Systems vol. 19 no. 4 pp. 682-695 April 2011.

[13] H. A. Farouk M. T. El-Hadidi "Implementing Globally Asynchronous Locally Synchronous Processor Pipeline on Commercial Synchronous FPGAs" IEEE 17th International Conference on Telecommunications (ICT) pp. 989-994 April 2010.

[14] A. Yousefzadeh L. A. Plana S. Temple T. Serrano-Gotarredona S. B. Furber B. Linares-Barranco "Fast Predictive Handshaking in Synchronous FPGAs for Fully Asynchronous Multisymbol Chip Links: Application to SpiNNaker 2-of-7 Links" IEEE Transactions on Circuits and Systems II: Express Briefs vol. 63 no. 8 pp. 763-767 2016.



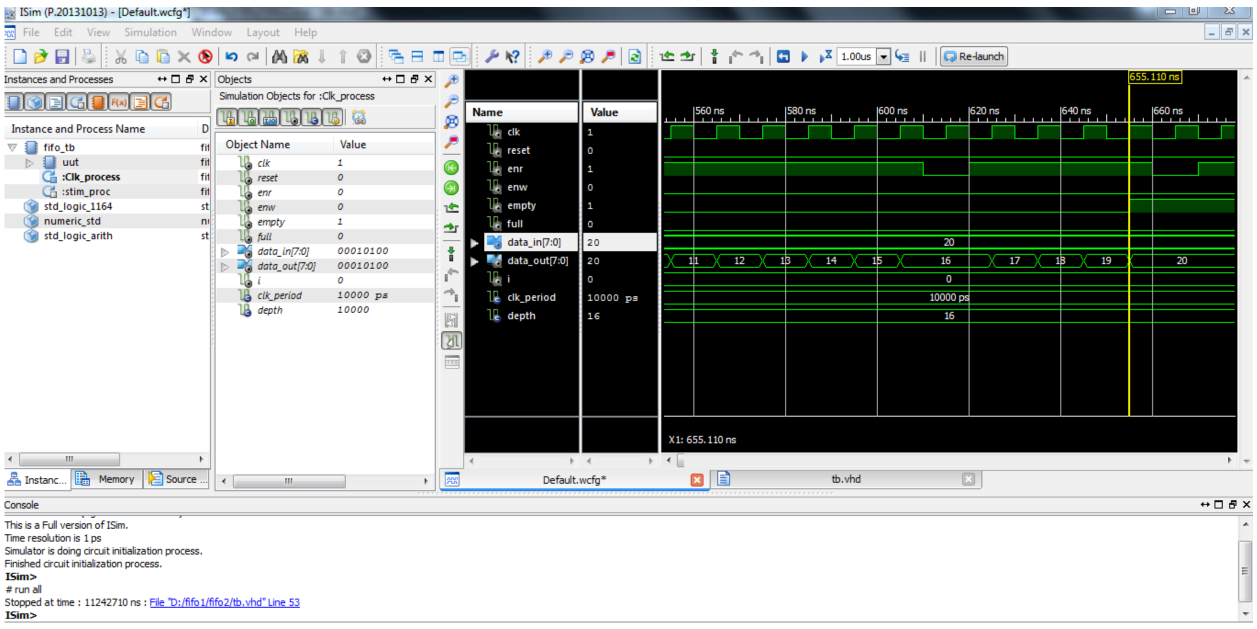
Instance and Process Name
fifo_tb
uut
:Clk_process
:stim_proc
std_logic_1164
numeric_std
std_logic_arith

Object Name	Value
clk	1
reset	0
enr	0
enw	0
empty	1
full	0
data_in[7:0]	00010100
data_out[7:0]	00010100
i	0
clk_period	10000 ps
depth	10000

Name	Value
clk	1
reset	0
enr	0
enw	1
empty	0
full	1
data_in[7:0]	20
data_out[7:0]	4
i	0
clk_period	10000 ps
depth	16

```
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
# run all
Stopped at time : 11242710 ns : File "D:/fifo1/fifo2/tb.vhd" Line 53
ISim>
```

Fig. 4. Simulation results when FIFO is full (Case 1)



```
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
# run all
Stopped at time : 11242710 ns : File "D:/fifo1/fifo2/tb.vhd" Line 53
ISim>
```

Fig. 5. Simulation results when FIFO is empty (Case 2)