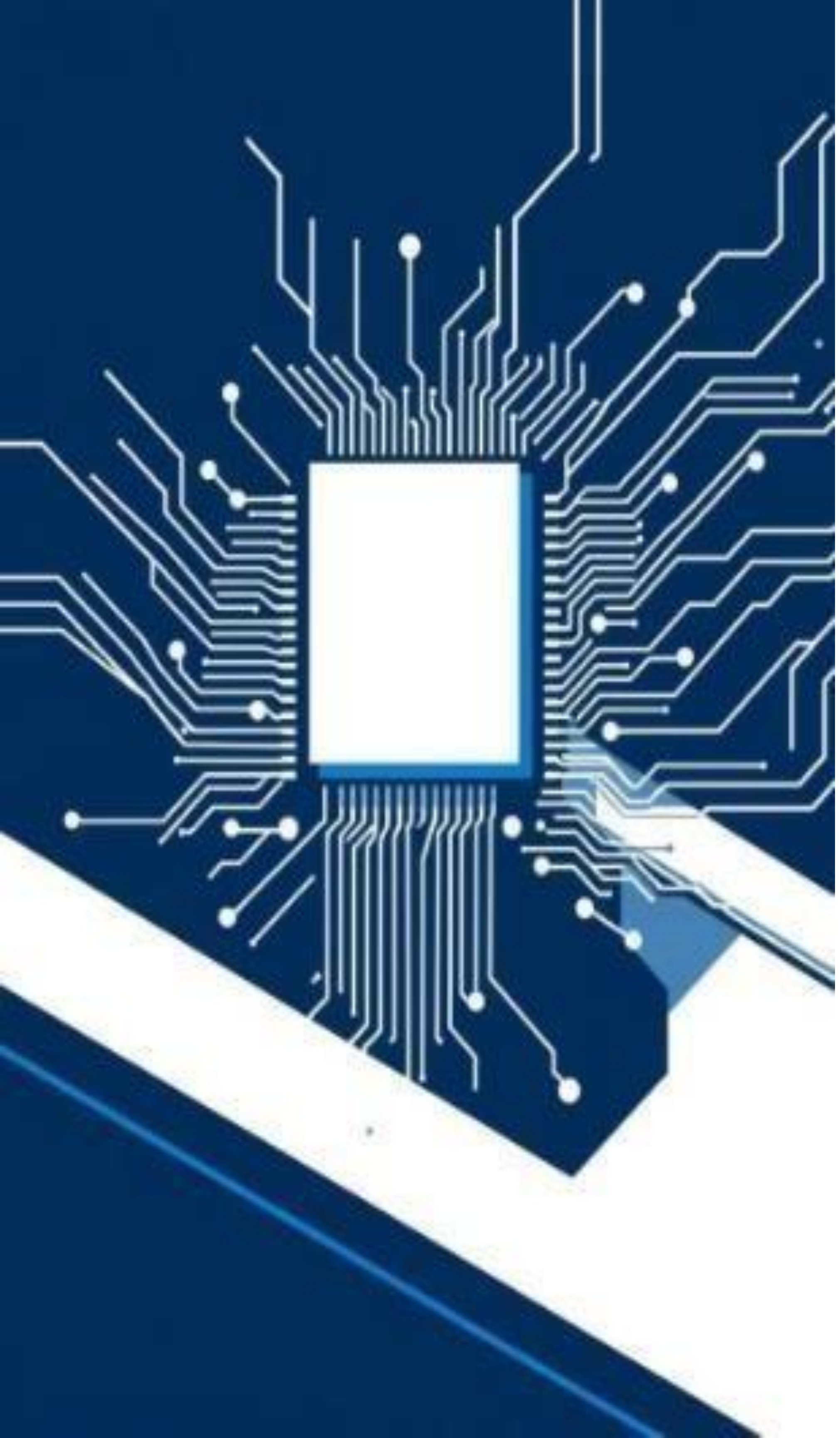


Design & Verification of Synchronous FIFO using Verilog

Presented By: - Aritra Banerjee

Table of Contents

Introduction to FIFO

FIFO (First-In-First-Out) is a data buffering mechanism used in digital systems, ensuring the first data element in is the first out.

Types:- There are two types of FIFO

1.Synchronous FIFO

Uses a single clock for both write and read operations, simplifying control logic.

2.Asynchronous FIFO

Employs different clocks for write and read operations, accommodating data transfer between systems with different clock domains.

Block Diagram:-

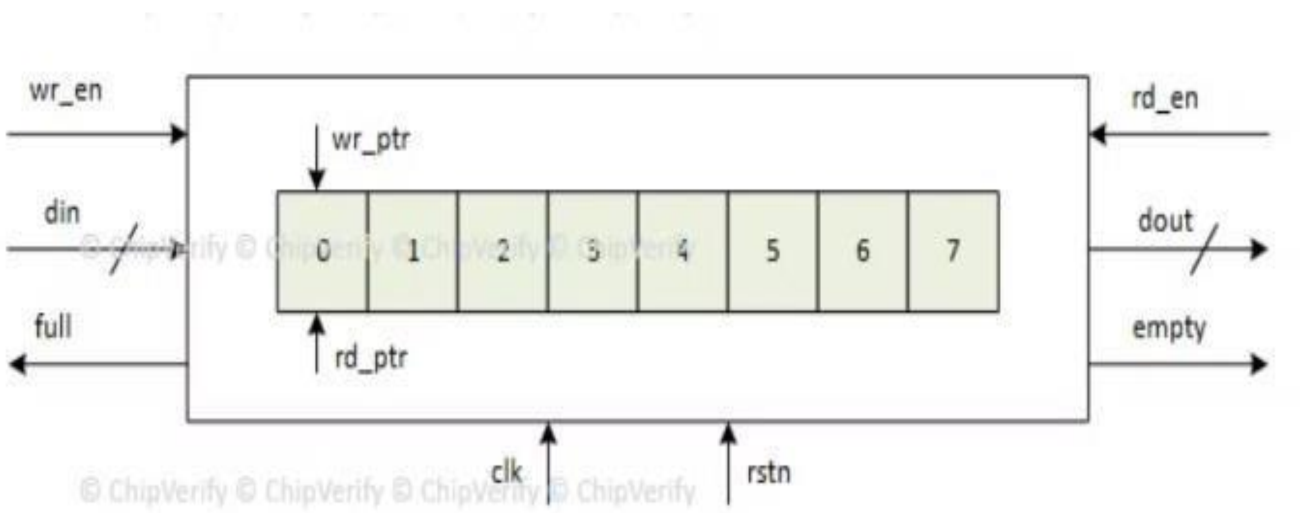


Fig1. Synchronous FIFO (Image source: chipverify.com)

Main IO Ports-
Data Ports- Write port & Read port
Memory Array- Stores the data element
Pointers- Read pointer & Write Pointer
Status Flags- Full & Empty
Clock Synchronization- Ensures proper timing of read/write operations

Depth :-

FIFO depth refers to the number of storage locations available in a FIFO buffer.

It determines how much data the FIFO can temporarily hold before it is read.

Design Details:-

Defines the inputs, outputs, and parameters of the FIFO module, ensuring a clear interface for interaction with other components.

Parameterize FIFO depth and data width for flexibility.

Implements the logic for full/empty flag control and pointer updates, ensuring correct FIFO behavior.

Synchronize read and write operations for data integrity.

Implement robust overflow and underflow handling.

Verilog Code :-

```
`timescale 1ns/1ps module sync_fifo #(parameter
Depth=8, Width=16)
(
   input clk, reset, w_enb, r_enb,   input [Width-1:0] din, output reg [Width-1:0] dout, output empty, full
);
```

```
reg   [$clog2(Depth)-1:0]   wptr;
reg    [$clog2(Depth)-1:0]    rptr;
reg  [Width-1:0]  fifo[0:Depth-1];
reg [$clog2(Depth):0] count;

always @ (posedge clk or posedge reset) begin
if  (reset)  begin       dout  <=  0;       wptr  <=  0;
rptr <= 0;      count <= 0;    end else begin
//          Write
operation
if  (w_enb  &&
!full)      begin
fifo[wptr]    <=
din;        wptr
<=  (wptr  +  1)
% Depth;
     end
     // Read
operation
```

Test Cases and Results:-

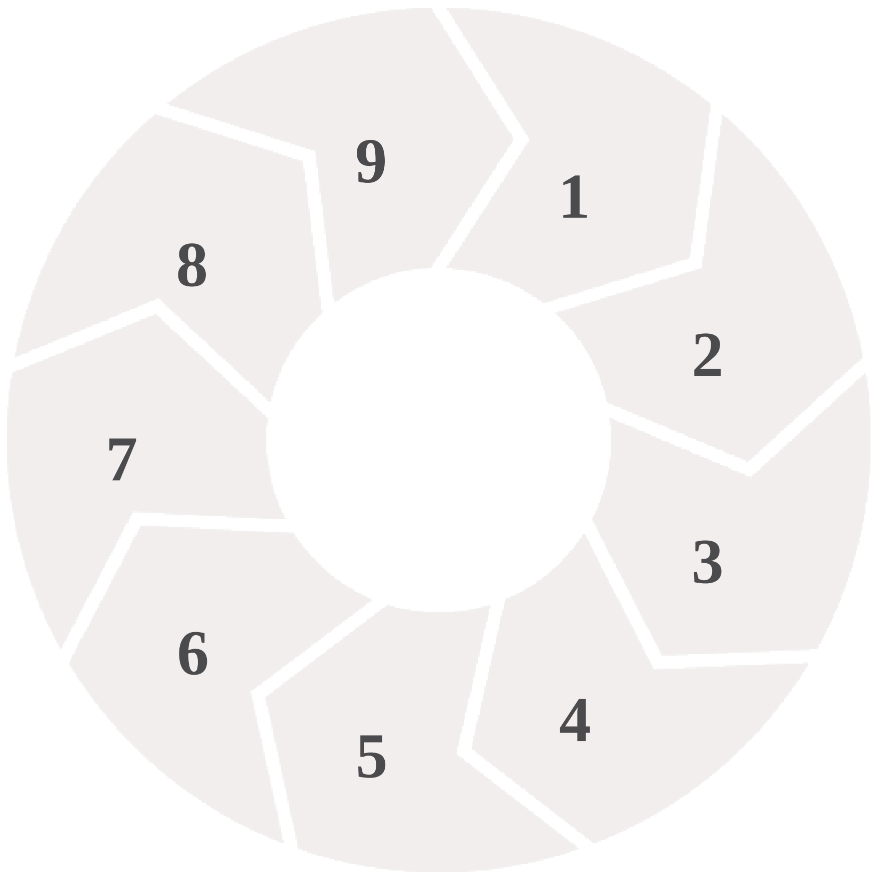
```
if (r_enb && !empty) begin
dout <= fifo[rptr];        rptr <=
(rptr + 1) % Depth;
     end
end   end

  // Count logic
  always @ (posedge clk or posedge reset) begin
if (reset) begin       count <= 0;     end else begin
     case ({w_enb && !full, r_enb && !empty})
       2'b10: count <= count + 1;                              // Write only
       2'b01: count <= count - 1;                              // Read only
       2'b11: count <= count;                                  // Simultaneous read and write
default: count <= count;                              // No change      endcase     end   end

  assign full = (count == Depth);    // Full when count reaches 8  assign
empty = (count == 0);        // Empty when count is 0

endmodule
```

Reset Test



Write Test

Read Test

Full Test

Empty Test

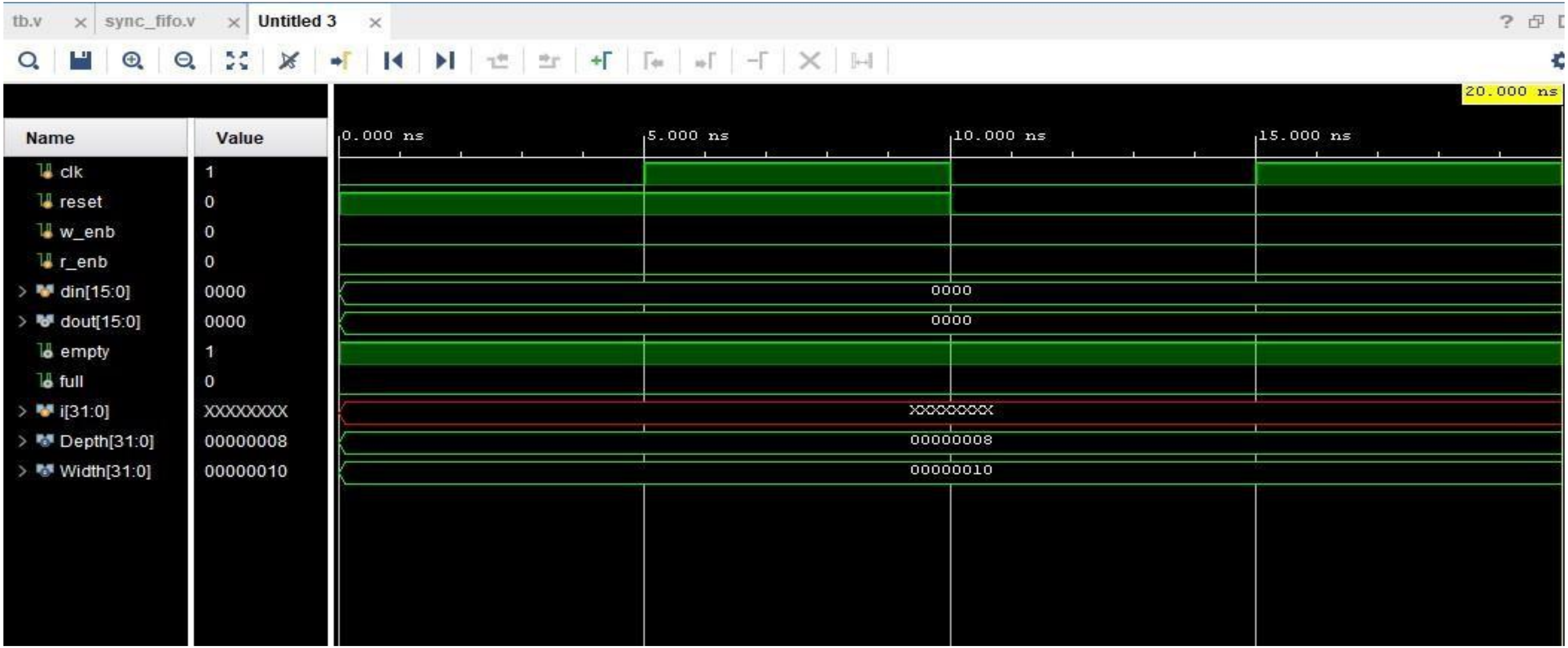
Simultaneous Read/Write

Alternate Read/Write

Overflow Test

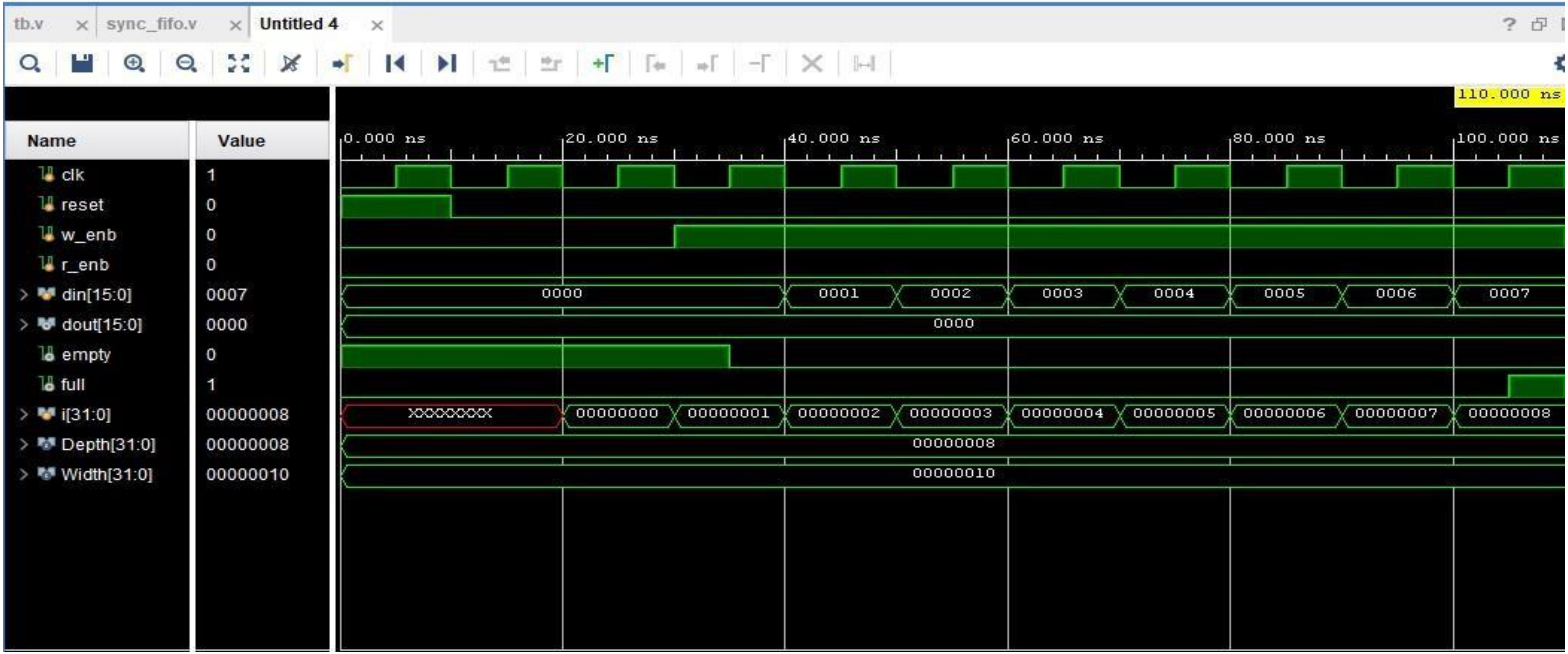
Underflow Test

Reset Test:-

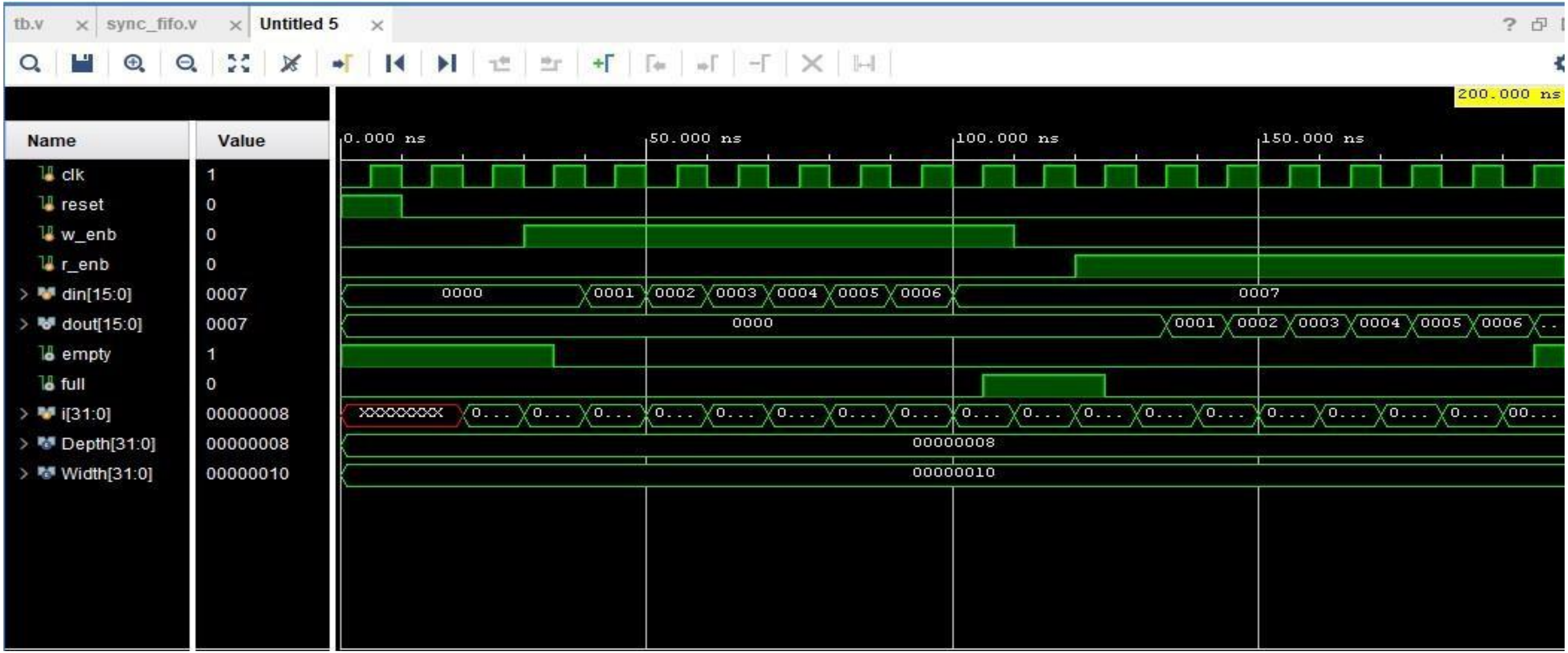


tb.v
sync_fifo.v
Untitled 3
20.000 ns
Name
Value
0.000 ns
5.000 ns
10.000 ns
15.000 ns
clk
1
reset
0
w_enb
0
r_enb
0
din[15:0]
0000
dout[15:0]
0000
empty
1
full
0
i[31:0]
XXXXXXXX
Depth[31:0]
00000008
Width[31:0]
00000010

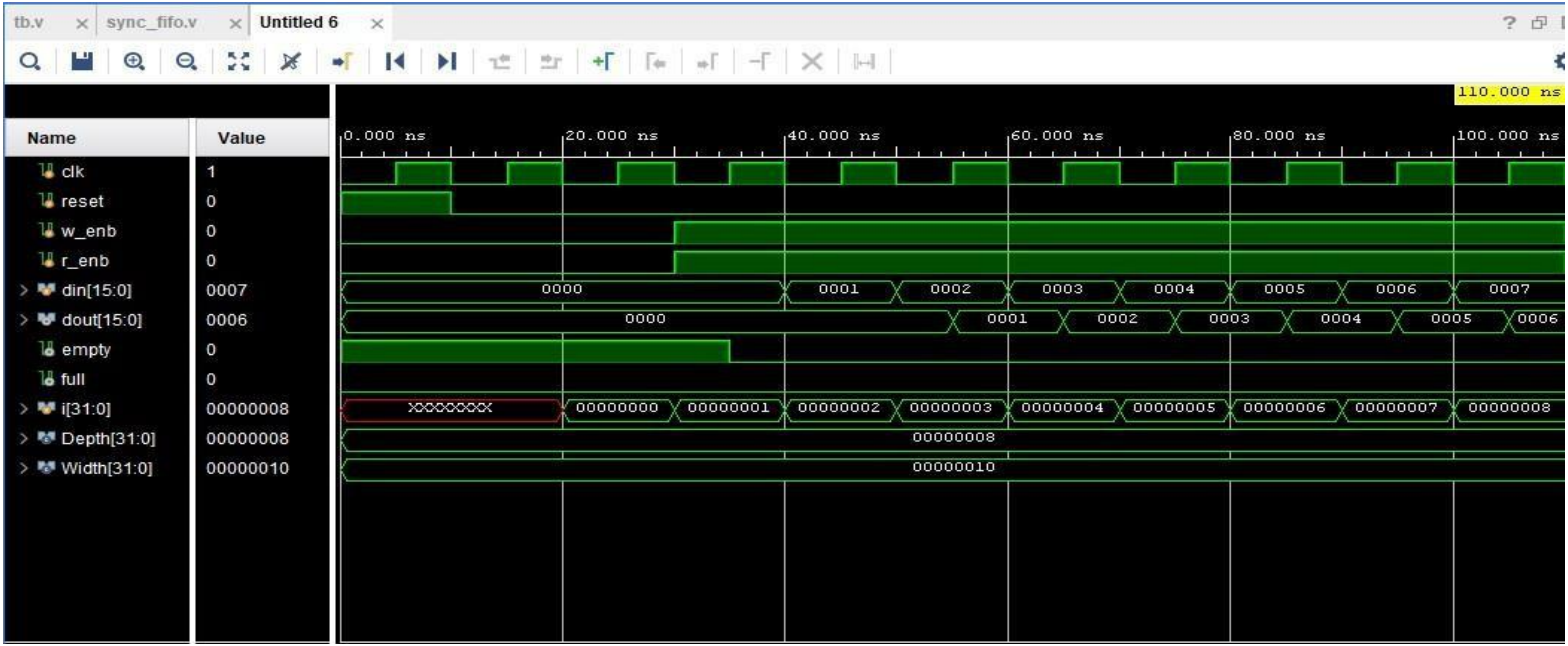
Write & Full Condition Test:-



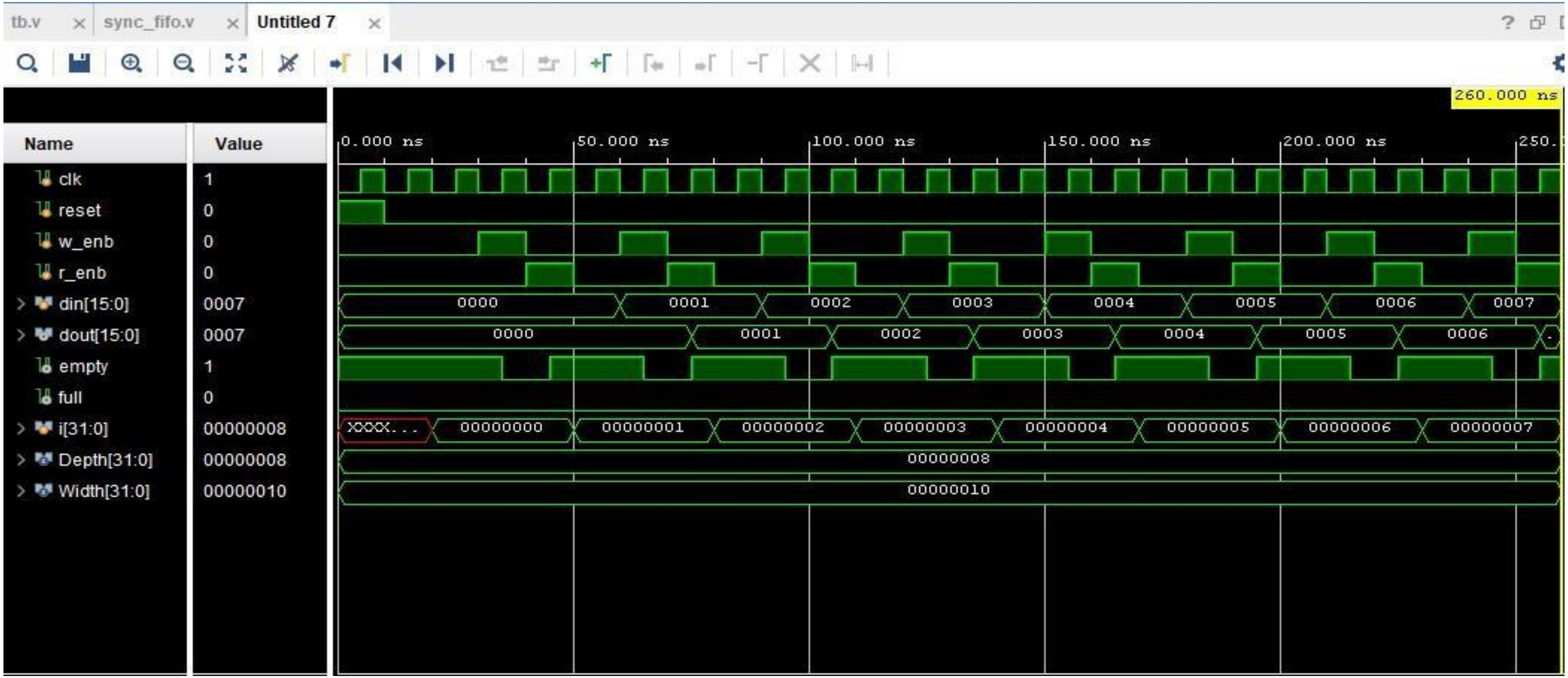
Read & Empty Condition Test:-



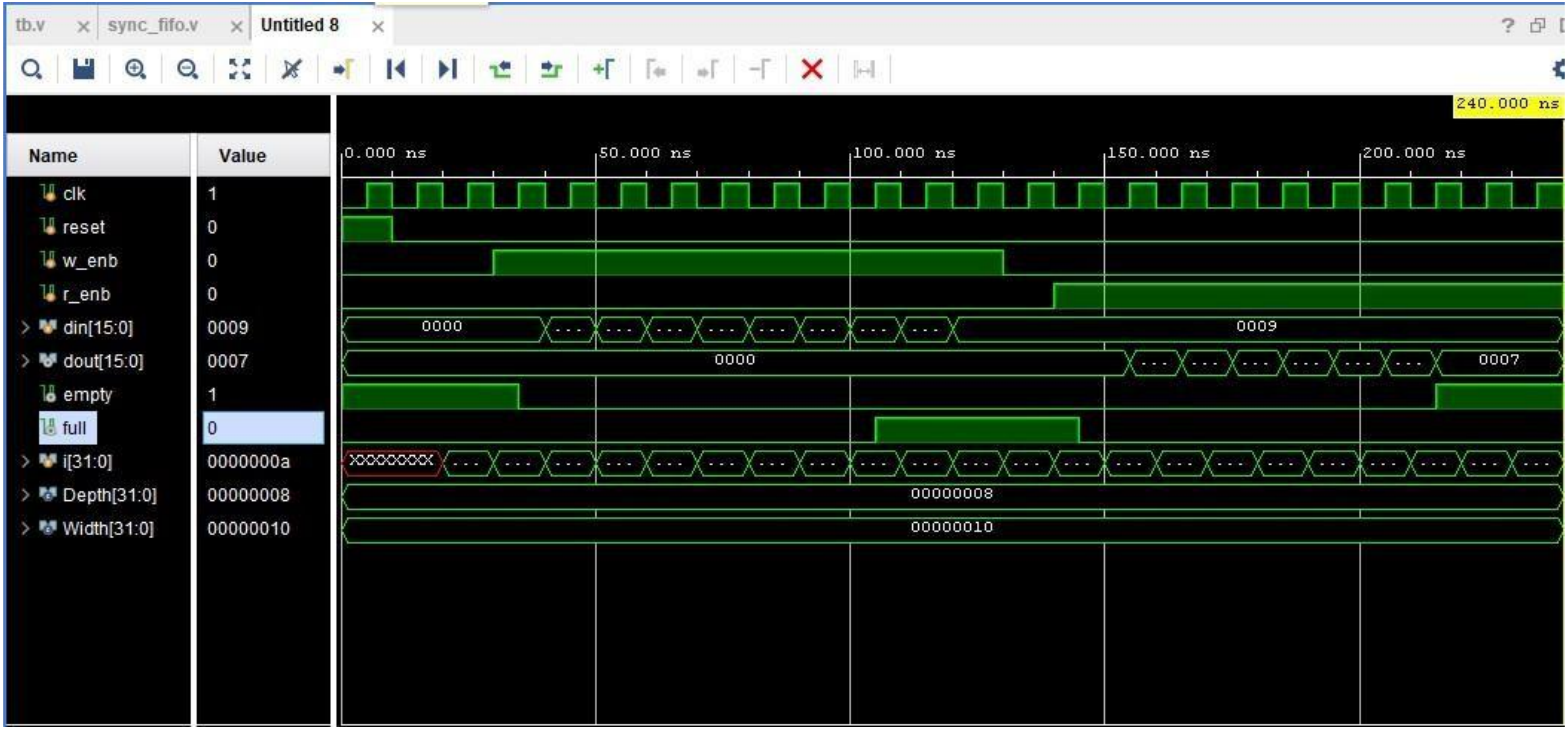
Simultaneous Read/Write :-



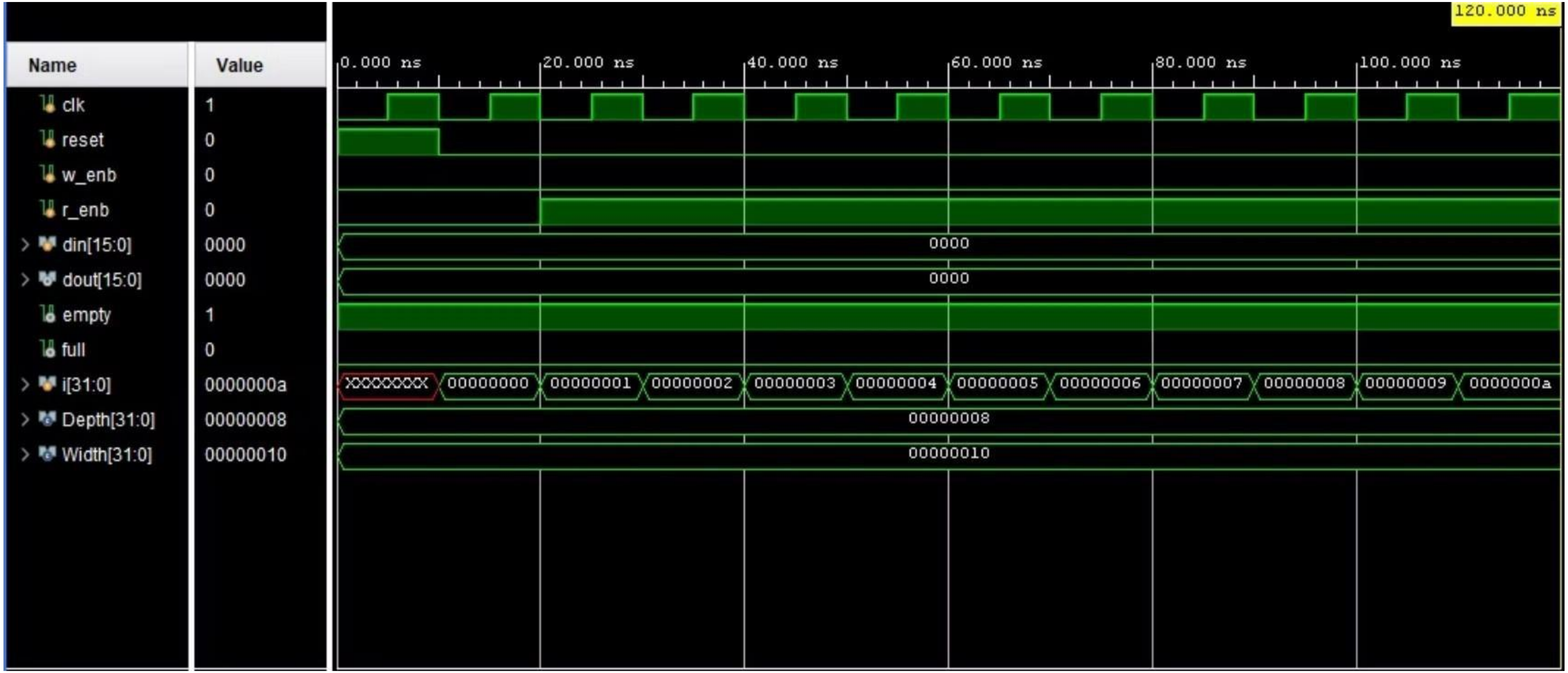
Alternating Read/Write :-



Overflow Test :-



Underflow Test :-



Tcl Console × Messages Log

```
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window. If you want to open a wave window go to
#   }
# }
# run 1000ns
Starting FIFO Testbench...
Reset Test: PASSED
Write Operation Test: PASSED
Read Operation Test: PASSED
Full Condition Test: PASSED
Empty Condition Test: PASSED
Simultaneous Read/Write Test: PASSED
Alternating Read/Write Test: PASSED
Overflow Test: PASSED
Underflow Test: PASSED
FIFO Testbench Completed.
$finish called at time : 850 ns : File "D:/Viavdo Programs/synchronus fifo/synchronus fifo.srcs/sim_1/new/tb.v" Line 124
xsim: Time (s): cpu = 00:00:08 ; elapsed = 00:00:08 . Memory (MB): peak = 1737.957 ; gain = 0.000
INFO: [USF-XSim-96] XSim completed. Design snapshot 'sync_fifo_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:10 ; elapsed = 00:00:25 . Memory (MB): peak = 1737.957 ; gain = 0.000
```

Conclusion :-

A well-structured synchronous FIFO can significantly enhance data throughput and buffering efficiency in digital systems, making it a fundamental component in memory interfaces, pipelines, and communication protocols.

Thank You