UNIVERSITY OF TEXAS AT DALLAS

800 W Campbell Rd, Richardson, TX 75080, USA

VLSI DESIGN (EECT 6325)

Project Done On:

FIRST IN FIRST OUT IMPLEMENTATION

Team Members:

GOKUL SAI RAGHUNATH GXR200021
ABHISHEK MAHESH KUMAR AXM200255
SOMA RAJ KUMAR RXS190135

Professor:

PROF. CARL SECHEN

ACKNOWLEDGEMENT

We are grateful to Professor CARL SECHEN for providing us an opportunity to explore and conduct projects based on VLSI Design.

We also take this opportunity to express our gratitude to MR VAIBHAV KUMARSWAMY SALIMATH for his guidance in conducting the project.

INTRODUCTION

FIFO is an acronym for First In First Out, which describes how data is managed relative to time or priority. In this case, the first data that arrives will also be the first data to leave from a group of data. A FIFO Buffer is a read/write memory array that automatically keep track of the order in which data enters the module and reads the data out in the same order. In hardware FIFO buffer is used for synchronization purposes. It is often implemented as a circular queue, and has two pointers:

I. Read Pointer/Read Address Register

II. Write Pointer/Write Address Register

Read and write addresses are initially both at the first memory location and the FIFO queue is Empty. When the difference between the read address and write address of the FIFO buffer is equal to the size of the memory array then the FIFO queue is Full.

BLOCK DIAGRAM

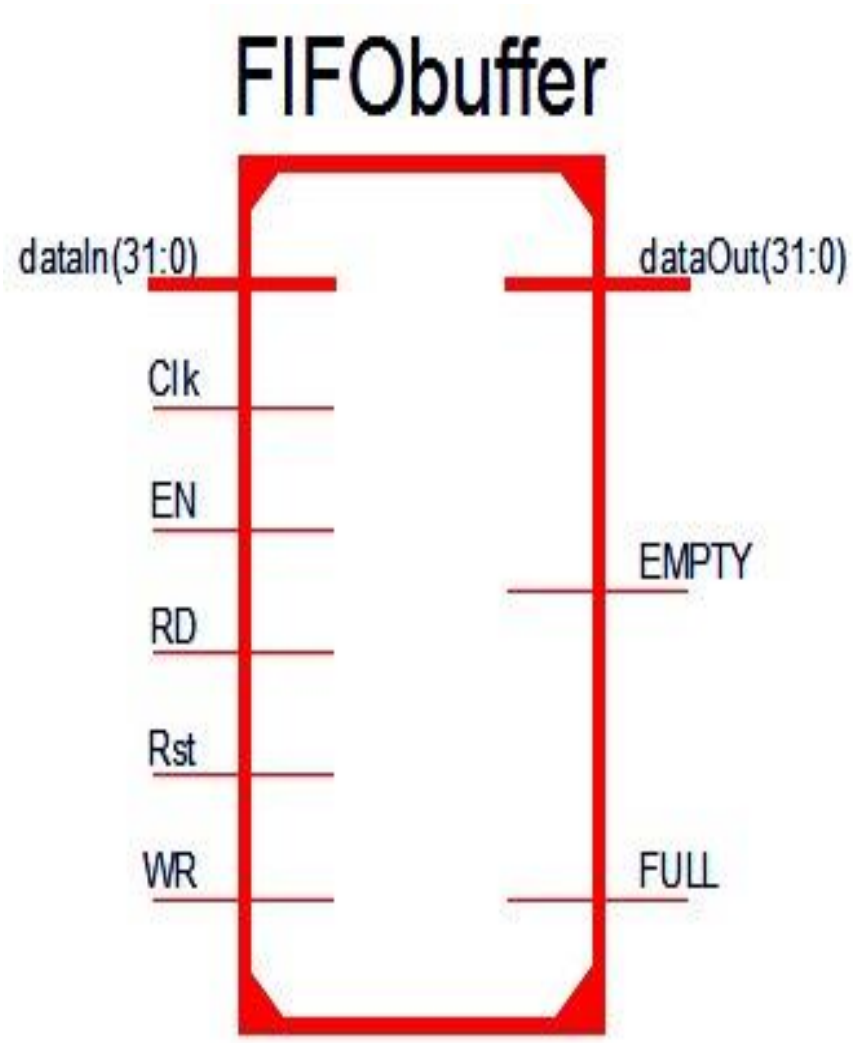


Figure – FIFO BUFFER

This FIFO Buffer can store eight 32-bit values. The FIFO Buffer module consists of a 32-bit data input line, dataIn and a 32-bit data output line, dataOut. The module is clocked using the 1-bit input clock line Clk. The module also has a 1-bit enable line, EN and a 1-bit active high reset line, Rst.

The 1-bit RD line is used to signal a data read operation on the FIFO Buffer and the 1-bit WR line is used to signal a data write operation on the FIFO Buffer. Both the RD and WR lines are active high. The module also has two output lines FULL and EMPTY which are each 1-bit wide. The FULL line becomes high when the FIFO Buffer is or becomes full (internal counter becomes eight). The EMPTY line becomes high when the FIFO Buffer is or becomes empty (internal counter becomes zero).

FIFO BEHAVIORAL CODE

```
module FIFObuffer
#(parameter data_size=32,
parameter data_alloc=32,
parameter rdwd_c_bit=($clog2(data_alloc)+1))

(Clk, dataIn, RD, WR, EN, dataOut, Rst, EMPTY, FULL);

input  Clk, RD, WR, EN, Rst;
output  EMPTY, FULL;

input   [data_size-1:0] dataIn;
output reg [data_size-1:0] dataOut; // internal registers

reg [rdwd_c_bit:0]  Count = 0;
reg [data_size:0] FIFO [0:data_alloc];
reg [rdwd_c_bit:0]  readCounter = 0,writeCounter = 0;

assign EMPTY = (Count==0)? 1'b1:1'b0;
assign FULL = (Count==data_alloc)? 1'b1:1'b0;

always @ (posedge Clk)
begin
 if (EN==0);

 else
  begin
  if (Rst) begin
   readCounter = 0;
   writeCounter = 0;
  end

  else if (RD ==1'b1 && Count!=0)
  begin
   dataOut  = FIFO[readCounter];
   readCounter = readCounter+1;
  end
```

```
 else if (WR==1'b1 && Count<data_alloc)
 begin
  FIFO[writeCounter]  = dataIn;
  writeCounter  = writeCounter+1;
 end

 else;
 end

if (writeCounter==data_alloc)
 writeCounter=0;
else if (readCounter==data_alloc)
 readCounter=0;

else;

if (readCounter > writeCounter)
begin
 Count=readCounter-writeCounter;
end

else if (writeCounter > readCounter)
 Count=writeCounter-readCounter;

else;
end
endmodule
```

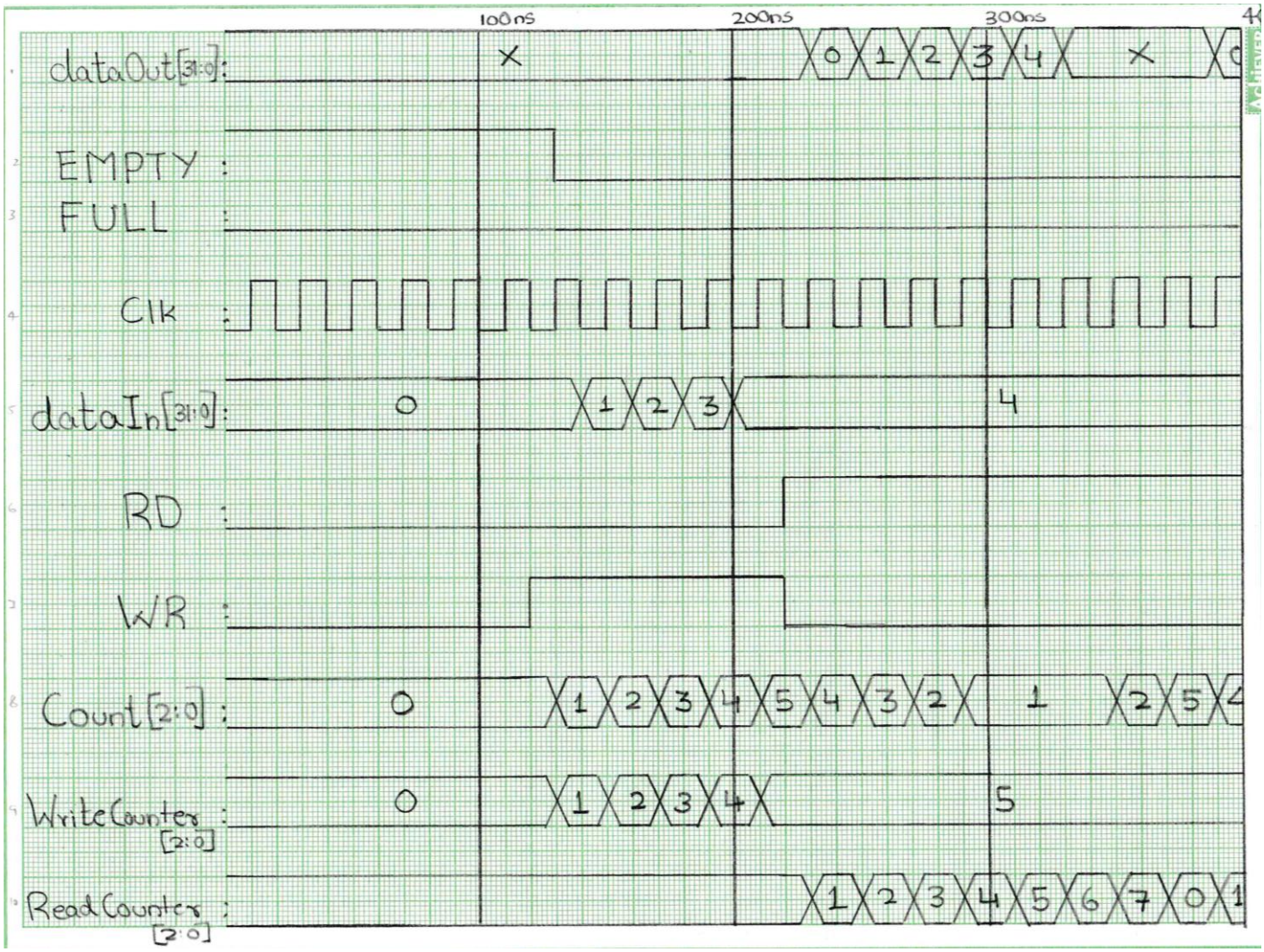
FIFO TESTBENCH CODE

```
module FIFObuffer_tb;
 // Inputs
 reg Clk;
 reg [31:0] dataIn;
 reg RD;
 reg WR;
 reg EN;
 reg Rst;

 // Outputs
 wire [31:0] dataOut;
 wire EMPTY;
 wire FULL;

 // Instantiate the Unit Under Test (UUT)
 FIFObuffer uut (
                .Clk(Clk),
                .dataIn(dataIn),
                .RD(RD),
                .WR(WR),
                .EN(EN),
                .dataOut(dataOut),
                .Rst(Rst),
                .EMPTY(EMPTY),
                 FULL(FULL)
                );

 initial begin
  // Initialize Inputs
  Clk  = 1'b0;
  dataIn  = 32'h0;
  RD  = 1'b0;
  WR  = 1'b0;
  EN  = 1'b0;
  Rst  = 1'b1;
```

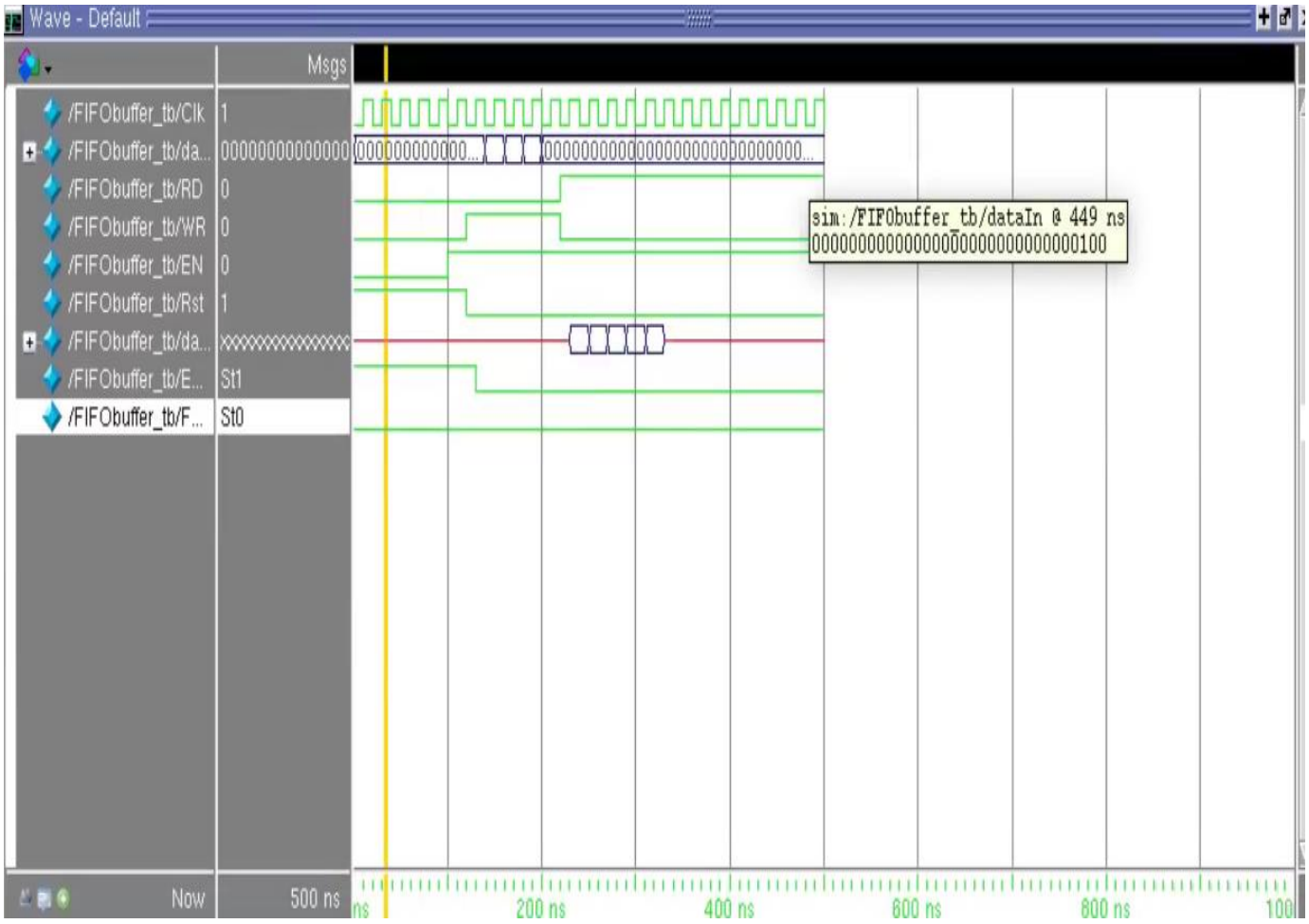
```
// Wait 100 ns for global reset to finish
 #100;

 // Add stimulus here
 EN  = 1'b1;
 Rst  = 1'b1;
 #20;
 Rst  = 1'b0;
 WR  = 1'b1;
 dataIn  = 32'h0;
 #20;
 dataIn  = 32'h1;
 #20;
 dataIn  = 32'h2;
 #20;
 dataIn  = 32'h3;
 #20;
 dataIn  = 32'h4;
 #20;
 WR = 1'b0;
 RD = 1'b1;
end

  always #10 Clk = ~Clk;

endmodule
```

At first the read flag is initialized to zero as the FIFO is now empty. Next, the write flag is high then the data in the registers are loaded with 0, 1, 2, 3, 4 and 5. While the enable is zero the FIFO does not function, and when the read enable is high, the values loaded to ‘data in’ register. The clockpulse applied has a time period of 20 nanoseconds and the EMPTY and FULL flags are used to detect if the FIFO is filled or empty.

FIFO EXPECTED WAVEFORM

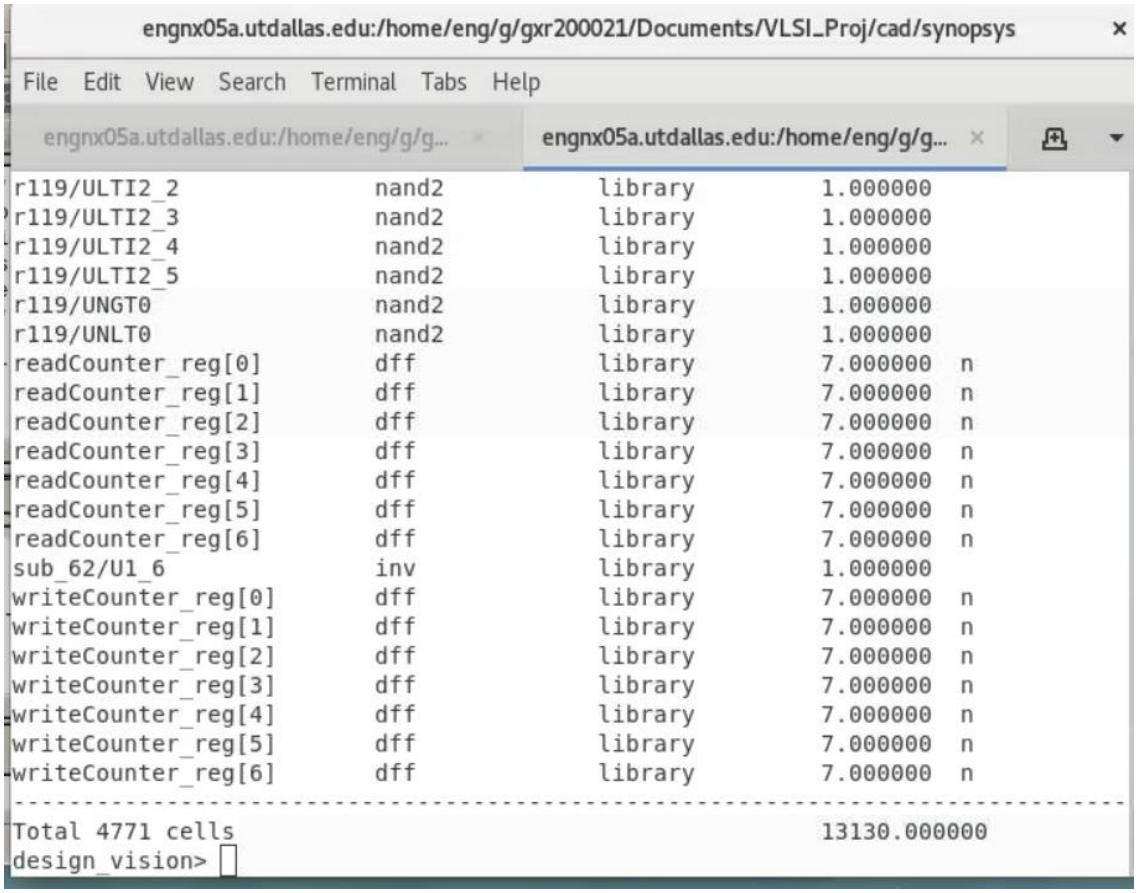


BEHAVIORAL CODE SIMULATION



From the simulation the fifo is functioning as expected and the expected waveform and the waveform simulation is compared.

FIFO CELL COUNT REPORT FROM DESIGN VISION



The total number of cells is 4771 as shown above.
Components being:
2 input NAND gate
3 input NAND gate
4 input NAND gate
2 input NOR gate
3 input NOR gate
4 input NOR gate
NOT gate (Inverter)
2 input XOR gate
2 input AND-OR-INVERT gate
3 input AND-OR-INVERT gate
2 input OR-AND-INVERT gate
3 input OR-AND-INVERT gate

DESIGN VISION CODE

Open | fifo_syno.v | ~/Documents/VLSI_Proj/cad/synopsys | Save

```
////////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : L-2016.03-SP3
// Date      : Mon Sep 13 09:08:59 2021
////////////////////////////////////////////////////////////////
module inv(in, out);
input in;
output out;
assign out = ~in;
endmodule

module nand2(a, b, out);
input a, b;
output out;
assign out = ~(a & b);
endmodule

module nand3(a, b, c, out);
input a, b, c;
output out;
assign out = ~(a & b & c);
endmodule

module nand4(a, b, c, d, out);
input a, b, c, d;
output out;
assign out = ~(a & b & c & d);
endmodule

module nor2(a, b, out);
input a, b;
output out;
assign out = ~(a | b);
endmodule

module nor3(a, b, c, out);
input a, b, c;
```

Open | fifo_syno.v | ~/Documents/VLSI_Proj/cad/synopsys | Save

```
module oai22(a, b, c, d, out);
input a, b, c, d;
output out;
assign out = ~((a | b) & (c | d));
endmodule

module dff( d, gclk, rnot, q);
input d, gclk, rnot;
output q;
reg q;
always @(posedge gclk or negedge rnot)
  if (rnot == 1'b0)
    q = 1'b0;
  else
    q = d;
endmodule

module FIFObuffer ( Clk, dataIn, RD, WR, EN, dataOut, Rst, EMPTY, FULL );
  input [31:0] dataIn;
  output [31:0] dataOut;
  input Clk, RD, WR, EN, Rst;
  output EMPTY, FULL;
  wire   N0, N1, N2, N3, N5, N6, N7, N8, N9, N10, N11, N12, N13, N14, N15, N16,
         N17, \FIFO[0][31] , \FIFO[0][30] , \FIFO[0][29] , \FIFO[0][28] ,
         \FIFO[0][27] , \FIFO[0][26] , \FIFO[0][25] , \FIFO[0][24] ,
         \FIFO[0][23] , \FIFO[0][22] , \FIFO[0][21] , \FIFO[0][20] ,
         \FIFO[0][19] , \FIFO[0][18] , \FIFO[0][17] , \FIFO[0][16] ,
         \FIFO[0][15] , \FIFO[0][14] , \FIFO[0][13] , \FIFO[0][12] ,
         \FIFO[0][11] , \FIFO[0][10] , \FIFO[0][9] , \FIFO[0][8] ,
         \FIFO[0][7] , \FIFO[0][6] , \FIFO[0][5] , \FIFO[0][4] , \FIFO[0][3] ,
         \FIFO[0][2] , \FIFO[0][1] , \FIFO[0][0] , \FIFO[1][31] ,
         \FIFO[1][30] , \FIFO[1][29] , \FIFO[1][28] , \FIFO[1][27] ,
         \FIFO[1][26] , \FIFO[1][25] , \FIFO[1][24] , \FIFO[1][23] ,
         \FIFO[1][22] , \FIFO[1][21] , \FIFO[1][20] , \FIFO[1][19] ,
         \FIFO[1][18] , \FIFO[1][17] , \FIFO[1][16] , \FIFO[1][15] ,
         \FIFO[1][14] , \FIFO[1][13] , \FIFO[1][12] , \FIFO[1][11] ,
         \FIFO[1][10] , \FIFO[1][9] , \FIFO[1][8] , \FIFO[1][7] , \FIFO[1][6] ,
```

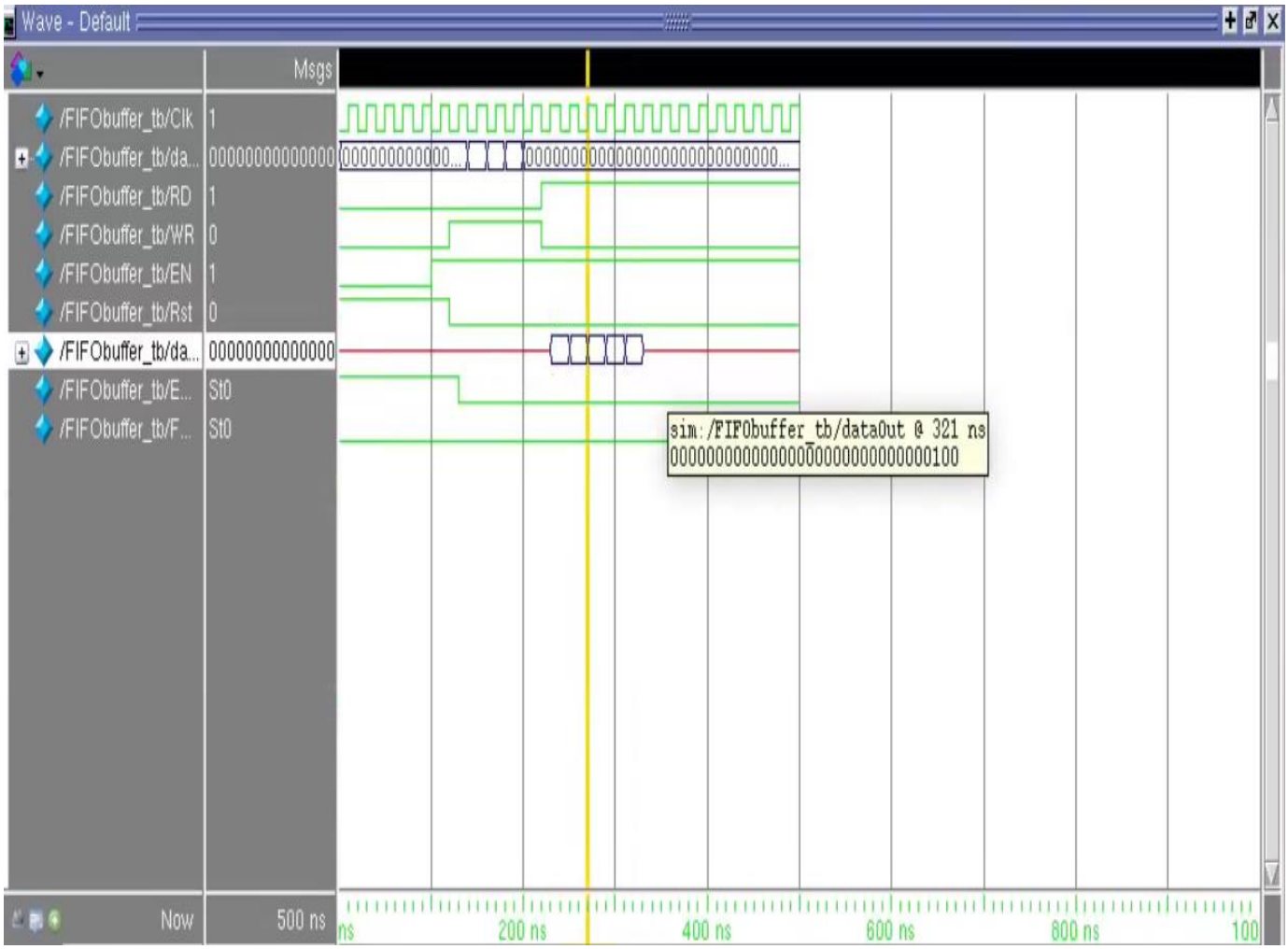
Verilog | Tab Width: 8 | Ln 102, Col 24 | INS

Open | fifo_syno.v | ~/Documents/VLSI_Proj/cad/synopsys | Save

```
nand2 C2914 ( .a(n3293), .b(n3294), .out(N300) );
nand2 C2903 ( .a(n3292), .b(N390), .out(N478) );
inv I_100 ( .in(N478), .out(N479) );
nand2 C2901 ( .a(WR), .b(N479), .out(n2664) );
inv I_24 ( .in(readCounter[1]), .out(N17) );
inv I_23 ( .in(readCounter[0]), .out(N16) );
inv I_22 ( .in(readCounter[2]), .out(N15) );
inv I_21 ( .in(readCounter[3]), .out(N14) );
inv I_20 ( .in(readCounter[4]), .out(N13) );
inv I_19 ( .in(readCounter[5]), .out(N12) );
nand2 C2789_5 ( .a(N11), .b(N17), .out(n2665) );
nand2 C2789_4 ( .a(N10), .b(N16), .out(n3291) );
nand2 C2789_3 ( .a(N9), .b(N15), .out(n3290) );
nand2 C2789_2 ( .a(N8), .b(N14), .out(n3289) );
nand2 C2789_1 ( .a(N12), .b(N13), .out(n3288) );
nand2 C2751 ( .a(N430), .b(N443), .out(n1522) );
nand2 C2750 ( .a(N430), .b(N442), .out(n1531) );
nand2 C2749 ( .a(N430), .b(N441), .out(n1539) );
nand2 C2748 ( .a(N430), .b(N440), .out(n1547) );
nand2 C2747 ( .a(N430), .b(N439), .out(n1527) );
nand2 C2746 ( .a(N430), .b(N438), .out(n1535) );
nand2 C2745 ( .a(N430), .b(N437), .out(n1543) );
nand2 C2744 ( .a(N430), .b(N436), .out(n1551) );
nand2 C2743 ( .a(N429), .b(N443), .out(n1523) );
nand2 C2742 ( .a(N429), .b(N442), .out(n1532) );
nand2 C2741 ( .a(N429), .b(N441), .out(n1540) );
nand2 C2740 ( .a(N429), .b(N440), .out(n1548) );
nand2 C2739 ( .a(N429), .b(N439), .out(n1528) );
nand2 C2738 ( .a(N429), .b(N438), .out(n1536) );
nand2 C2737 ( .a(N429), .b(N437), .out(n1544) );
nand2 C2736 ( .a(N429), .b(N436), .out(n1552) );
nand2 C2735 ( .a(N428), .b(N443), .out(n1524) );
nand2 C2734 ( .a(N428), .b(N442), .out(n1533) );
nand2 C2733 ( .a(N428), .b(N441), .out(n1541) );
nand2 C2732 ( .a(N428), .b(N440), .out(n1549) );
nand2 C2731 ( .a(N428), .b(N439), .out(n1529) );
nand2 C2730 ( .a(N428), .b(N438), .out(n1537) );
```

```
dff \FIFO_reg[8][12]  ( .d(n2590), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][12] ) );
dff \FIFO_reg[8][11]  ( .d(n2589), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][11] ) );
dff \FIFO_reg[8][10]  ( .d(n2588), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][10] ) );
dff \FIFO_reg[8][9]  ( .d(n2587), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][9] )
      );
dff \FIFO_reg[8][8]  ( .d(n2586), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][8] )
      );
dff \FIFO_reg[8][7]  ( .d(n2585), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][7] )
      );
dff \FIFO_reg[8][6]  ( .d(n2584), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][6] )
      );
dff \FIFO_reg[8][5]  ( .d(n2583), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][5] )
      );
dff \FIFO_reg[8][4]  ( .d(n2582), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][4] )
      );
dff \FIFO_reg[8][3]  ( .d(n2581), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][3] )
      );
dff \FIFO_reg[8][2]  ( .d(n2580), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][2] )
      );
dff \FIFO_reg[8][1]  ( .d(n2579), .gclk(Clk), .rnot(1'b1), .q(\FIFO[8][1] )
      );
dff \FIFO_reg[16][0]  ( .d(n2578), .gclk(Clk), .rnot(1'b1), .q(\FIFO[16][0] ) );
dff \FIFO_reg[16][31]  ( .d(n2577), .gclk(Clk), .rnot(1'b1), .q(
      \FIFO[16][31] ) );
dff \FIFO_reg[16][30]  ( .d(n2576), .gclk(Clk), .rnot(1'b1), .q(
      \FIFO[16][30] ) );
dff \FIFO_reg[16][29]  ( .d(n2575), .gclk(Clk), .rnot(1'b1), .q(
      \FIFO[16][29] ) );
dff \FIFO_reg[16][28]  ( .d(n2574), .gclk(Clk), .rnot(1'b1), .q(
      \FIFO[16][28] ) );
dff \FIFO_reg[16][27]  ( .d(n2573), .gclk(Clk), .rnot(1'b1), .q(
      \FIFO[16][27] ) );
dff \FIFO_reg[16][26]  ( .d(n2572), .gclk(Clk), .rnot(1'b1), .q(
      \FIFO[16][26] ) );
dff \FIFO_reg[16][25]  ( .d(n2571), .gclk(Clk), .rnot(1'b1), .q(
      \FIFO[16][25] ) );
dff \FIFO_reg[16][24]  ( .d(n2570), .gclk(Clk), .rnot(1'b1), .q(
```

FIFO SIMULATION AFTER CONVERSION TO GATE LEVEL NETLIST USING DESIGN VISION



From both simulations it can be realised that the netlist generated by the Design Vision and the behavioural code gives the same output.

CONCLUSION

The waveform output from the simulation through ModelSim and the simulation of the netlist generated from Design Vision have the same result and the netlist code generated has 4771 cells.

Code and resource used for this project:
https://github.com/Abhishekmaheshkumar/VLSI_Design-EECT6325/tree/main/Project%202