RTL SIMULATION AND SYNTHESIS WITH PLDS(R22D6809)

LABORATORY MANUAL

M. Tech – I Year – I Sem. (VLSI & Embedded Systems)

2022-2023

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGG

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

(Autonomous Institution –UGC, Govt. of India)

(Approved by AICTE- Accredited by NBA & NAAC- 'A' Grade-ISO 9001:2008 Certified)

Maisammaguda, Dhulapally, Secundrabad-500 100.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

(Affiliated to JNTU, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Kompally), Secunderabad – 500100, Telangana State, India.

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Department Vision & Mission

Vision

To strengthen the department into a centre of academic excellence with focus on advanced technologies & relevant research by delivering the best quality technical education to the students, so as to meet the current and future challenges along with emphasis on moral and ethical values.

Mission

To create and enrich academic environment with essential resources, so as to train and mould students in active learning & critical thinking with innovative ideas, so as to solve real-world problems in the field of Electrical & Electronics Engineering.

To motivate and strengthen the faculty to practice effective teaching & learning process and involve in advanced research & development work.

To enhance industry interaction and initiate best consultancy services.

Quality Policy

To develop, maintain and update global standards of excellence in all our areas of academic & research activities and facilities so as to impart a state of the art & value based technical education to the students commensurate with the rapidly changing industry needs.

To continuously adopt and implement concurrent & commensurate faculty development programmes towards achieving the institution's goals and objectives.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

(Affiliated to JNTU, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Kompally), Secunderabad – 500100, Telangana State, India.

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

M.TECH – VLSI & EMBEDDED SYSTEMS

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: TECHNICAL ACCOMPLISHMENTS

To pursue career in VLSI and Embedded Systems domain through state of the art learning and self directed approach on cutting edge technologies converging to substantial research work.

PEO 2: **PROFESSIONAL DEVELOPMENT**

To develop managerial skill and apply creative approaches in the domains of VLSI and Embedded Systems by incorporating automation, power consumption, miniaturization, sustainability leading to become a successful professional or an Entrepreneur.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1.

To acquire competency in areas of VLSI and Embedded Systems, Design, Testing, Verification IC Fabrication and prototype development with focus on applications.

PSO 2.

To integrate multiple sub-systems to develop System on Chip, optimize its performance and excel in industry sectors related to VLSI/ Embedded domain and to develop a start-up system.

RTL SIMULATION AND SYNTHESIS WITH PLDS

List of Experiments:

1) Verilog implementation of 8:1 Mux/Demux, Full Adder, 8-bit Magnitude comparator, Encoder/decoder, Priority encoder, D-FF, 4-bit Shift registers (SISO, SIPO, PISO, bidirectional), 3-bit Synchronous Counters, Binary to Gray converter, Parity generator.
2) Sequence generator/detectors, Synchronous FSM – Mealy and Moore machines.
3) Vending machines - Traffic Light controller, ATM, elevator control.
4) PCI Bus & arbiter and downloading on FPGA.
5) UART/ USART implementation in Verilog.
6) Realization of single port SRAM in Verilog.
7) Verilog implementation of Arithmetic circuits like serial adder/ subtractor, parallel adder/subtractor, serial/parallel multiplier.
8) Discrete Fourier transform/Fast Fourier Transform algorithm in Verilog.

Course Outcomes:

At the end of the laboratory work, students will be able to:

- Identify, formulate, solve and implement problems in signal processing, communication systems etc using RTL design tools.
  Use EDA tools like, Mentor Graphics and Xilinx.

CONTENTS

Exp.1- Verilog implementation of 8:1 Mux/Demux, Full Adder, 8-bit Magnitude comparator, Encoder/decoder, Priority encoder, D-FF, 4-bit Shift registers (SISO, SIPO, PISO, bidirectional), 3-bit Synchronous Counters, Binary to Gray converter, Parity generator.

AIM:

To develop the source code for multiplexer and demultiplexer by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement in FPGA.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3

ALGORITM:

Step1: Define the specifications and initialize the design.

Step2: Write the source code in VERILOG.

Step3: Check the syntax and debug the errors if found, obtain the synthesis report.

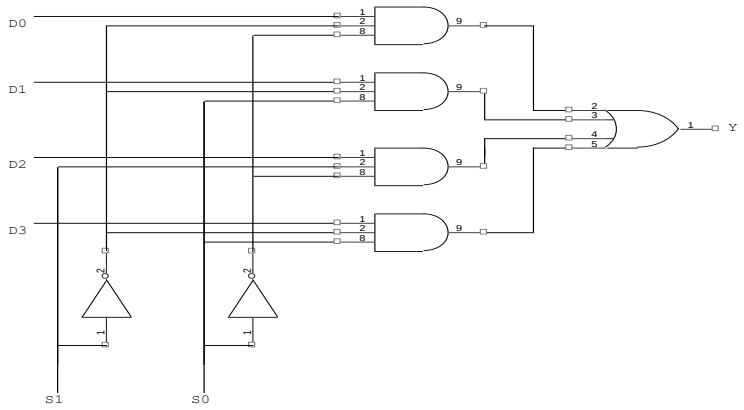
Step4: Verify the output by simulating the source code.

Step5: Write all possible combinations of input using the test bench.

Step6: Obtain the place and route report.

MULTIPLEXER:

LOGIC DIAGRAM:



TRUTH TABLE:

SELECT INPUT		OUTPUT
S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module muxdataflow(s, d, y);
  input [1:0] s;
  input [3:0] d;
  output y;
        wire p,q,r,s,sobar,s1bar;
        assign #sobar = ~s[1];
        assign #s1bar = ~s[0];
        assign p = (d[0]&sobar&s1bar);
        assign q = (d[0]&sobar&s1);
        assign r = (d[0]&so&s1bar);
        assign s = (d[0]&so&s1);
        assign y = (p|q|r|s);
endmodule
```

Behavioral Modeling:

```
module mux_behv(d, s0, s1, y);
  input [3:0] d;
  input s0;
  input s1;
  output y;
        reg y;
reg s0bar,s1bar;
reg p,q,r,s;
        always@(d or s0 or s1)
begin
        p=(d[0]&sobar&s1bar);
q=(d[1]&sobar&s1);

r=(d[2]&so&s1bar);

s=(d[3]&so&s1);
endmodule
```

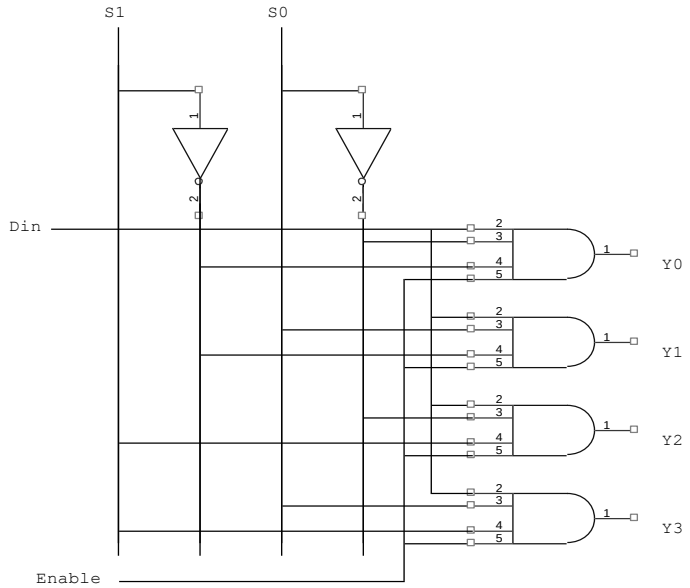
Structural Modeling:

```
module mux_struct(d, s0, s1, y);
  input [3:0] d;
  input s0;
  input s1;
  output y;
        wire p,q,r,s;
        and
```

```
        a1(p,d[0],~s0,~s1);
        a2(q,d[1],~s0, s1);
        a3(r,d[2], s0, ~s1);
        a4(s,d[3], s0, s1);
        or
        o1(yp,q,r,s);
endmodule
```

DEMULTIPLEXER:

LOGIC DIAGRAM:



TRUTH TABLE:

INPUT			OUTPUT			
D	S0	S1	Y0	Y1	Y2	Y3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module demuxdataflow(s0,s1,d,y,e);
  input s0,s1,d,e;
  output [3:0] y;
        wire s0bar,s1bar;
        assign #2s0bar=~s0;
        assign #2 s1bar=~s1;
        assign #3 y[0]=(d&s0bar&s1bar&e);
        assign #3 y[1]=(d&s0bar&s1&e);
        assign #3 y[2]=(d&s0&s1bar&e);
        assign #3 y[3]=(d&s0&s1&e);
```

```
endmodule
```

Behavioral Modeling:

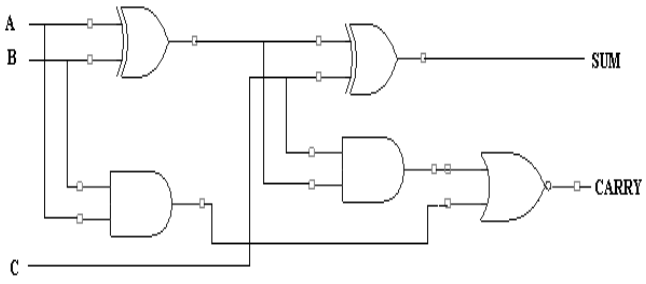
```
module demux_behv(s0, s1,d, y,e);
  input s0;
  input s1;
  input d,e;
  output [3:0] y;
        reg [3:0] y;
        reg s0bar,s1bar;
        always@(d or s0 or s1)
        begin
        s0bar=~s0;
        s1bar=~s1;
        y[0]=(d&s0bar&s1bar&e);
        y[1]=(d&s0bar&s1&e);
        y[2]=(d&s0&s1bar&e);
        y[3]=(d&s0&s1&e);
        end
        endmodule
```

Structural Modeling:

```
module demux_struct(s0, s1, d,e,y);
  input s0;
  input s1;
  input d,e;
  output [3:0] y;
        and
        a1(y[0],d,~s0,~s1,e);
        a2(y[1],d,~s0, s1,e);
        a3(y[2, s0, ~s1,e);
        a4(y[3], s0, s1,e);
endmodule
```

FULL ADDER:

LOGIC DIAGRAM:



TRUTH TABLE:

A	B	C	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module fulladddataflow(a, b, c, sum, carry);
   input a;
   input b;
   input c;
   output sum;
   output carry;
assign#2 p=a&b;
assign#2 q=b&c;
assign#2 r=c&a;
assign#4 sum=a^b^c;
assign#4carry =(p1 | p2) | p3;
 endmodule
```

Behavioral Modeling:

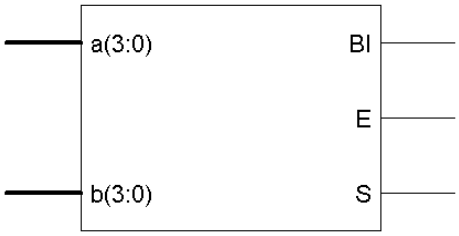
```
module fuladbehavioral(a, b, c, sum, carry);
   input a;
   input b;
   input c;
   output sum;
   output carry;
         reg sum,carry;
         reg p1,p2,p3;
         always @ (a or b or c) begin
         sum = (a^b)^c;
         p1=a & b;
         p2=b & c;
         p3=a & c;
         carry=(p1 | p2) | p3;
         end
endmodule
```

Structural Modeling:

```
module fa_struct(a, b, c, sum, carry);
   input a;
   input b;
   input c;
   output sum;
   output carry;
         wire t1,t2,t3,s1
         xor
         x1(t1a,b),
         x2(sum,s1,c);
         and
         a1(t1,a,b),
         a2(t2,b,c),
         a3(t3,a,c);
         or
         o1(carry,t1,t2,t3);
endmodule
```

4 BIT COMPARATOR:

LOGIC DIAGRAM:



VERILOG SOURCE CODE:

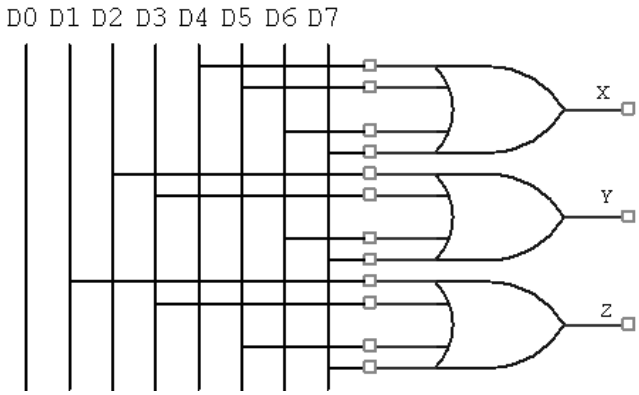
Behavioral Modeling:

```
module comparatorbehvioral(a, b, x, y, z);
  input [3:0] a;
  input [3:0] b;
  output x;
  output y;
  output z;
        reg x,y,z;
        always @ (a,b) begin
        if(a==b)
        begin
        x=1'b1;
        y=1'b0;
        z=1'b0;
        end
        else if (a<b)  begin
        x=1'b0;
        y=1'b1;
        z=1'b0;
  end
        else
        begin
        x=1'b0;
        y=1'b0;
        z=1'b1;
        end
        end
        endmodule
```

ENCODERS AND DECODERS

ENCODER:

LOGIC DIAGRAM:



TRUTH TABLE:

D0	D1	D2	D3	D4	D5	D6	D7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module encod_data(d, a.b.c);
   input [7:0] d;
   output a,b,c;
          assign#3 a=d[4]|d[5]|d[6]|d[7];
          assign#3 b=d[2]|d[3]|d[6]|d[7];
          assign#3 c=d[1]|d[3]|d[5]|d[7];
endmodule
```

Behavioral Modeling:

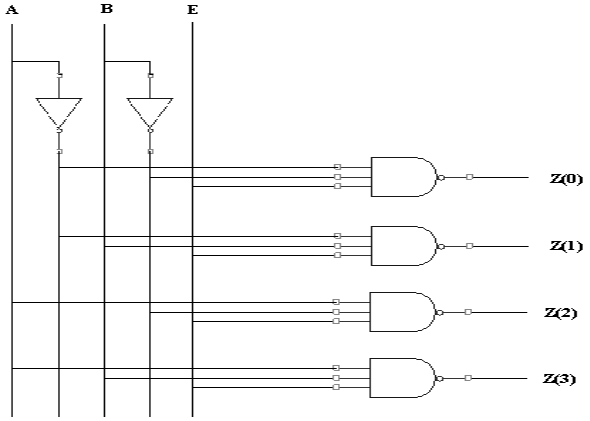
```
module encoderbehav(d, a,b,c);
   input [7:0] d;
   output x;
   output y;
   output z;
          reg a,b,c;
          always @ (d [7:0]) begin
          a= d[4] | d[5] | d[6] | d[7];
          b= d[2] | d[3] | d[6] | d[7];
          c= d[1] | d[3] | d[5] | d[7];
          end
   endmodule
```

Structural Modeling:

```
module encod_struct(d, a,b,c);

   input [7:0] d;

   output a,b,c;

          or

          o1(a,d[4],d[5],d[6],d[7]),

          o2(b,d[2],d[3],d[6],d[7]),

          o3(c,d[1],d[3],d[5],d[7]);

endmodule
```

DECODERS:

LOGIC DIAGRAM:



TRUTH TABLE:

A	B	C	Z(0)	Z(1)	Z(2)	Z(3)
0	0	1	0	1	1	1
0	1	1	1	0	1	1
1	0	1	1	1	0	1
1	1	1	1	1	1	0

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module decoderdataflow(a,b,en, z);
   input a,b,en;
   output [0:3] z;
          wire  abar,bbar;
          assign # 1 abar=~a;
          assign # 1 bbar =~b;
          assign # 2 z[0]=(abar & bbar & en);
          assign # 2 z[1]=(abar & b & en);
                    assign # 2 z[2]=(a & bbar & en);
                              assign # 2 z[0]=(a & b & en);
endmodule
```

Behavioral Modeling:

```
module decoderbehv(a, b, en, z);
   input a;
   input b;
   input en;
   output [3:0] z;
          reg [3:0] z;
reg abar,bbar;
          always @ (a,b,en) begin
```

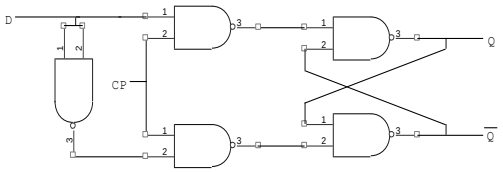
```
        z[0] = (abar&bbar&en);
        z[1] = (abar&b&en);
        z[2] = (a&bbar&en);
        z[3] = (a&b&en);
        end
endmodule
```

Structural Modeling:

```
module decod_struct(a, b, e, z);
  input a;
  input b;
  input e;
  output [3:0] z;
        wire abar,bbar;
        not
        n1(abar,a),
        n2(bbar,b);
        and
        a1(z[0],abar,bbar,e),
        a2(z[1],abar,b,e),
        a3(z[2],a,bbar,e),
        a4(z[3],a,b,e);
        endmodule
```

D FLIPFLOP:

LOGIC DIAGRAM:



TRUTH TABLE:

Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

VERILOG SOURCE CODE:

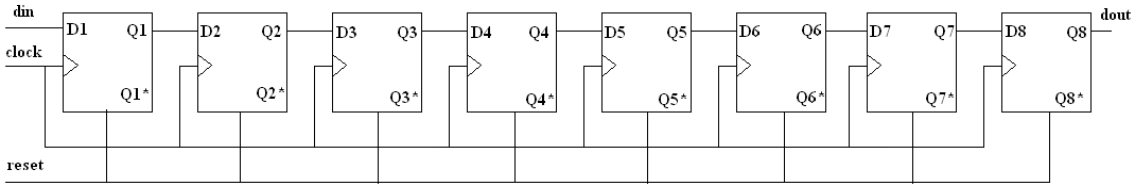
Behavioral Modeling:

```
module dff(d, clk, rst, q, qbar);
  input d;
  input clk;
  input rst;
  output q;
  output qbar;
        reg q;
        reg qbar;
        always @ (posedge(clk) or posedge(rst)) begin
        if (rst==1'b1)
        begin
        q=1'b0;
        qbar=1'b1;
        end
        else if (d==1'b0)
        begin
        q=1'b0;
        qbar=1'b1;
        end
        else
        begin
        q=1'b1;
        qbar=1'b0;
        end
        end
  endmodule
```

DESIGN OF SHIFTERS

SERIAL-IN SERIAL-OUT SHIFT REGISTER:

LOGIC DIAGRAM :



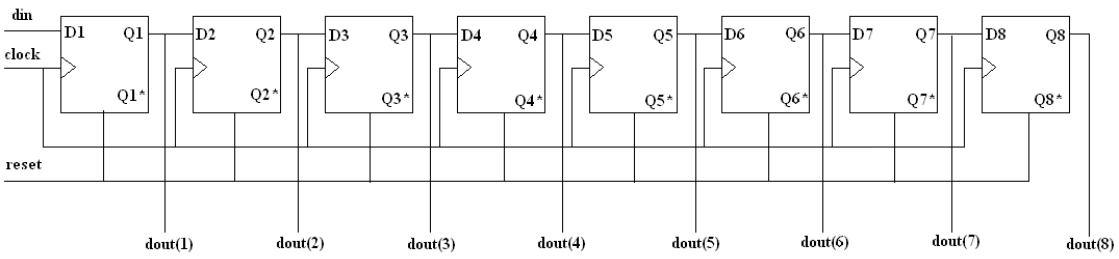
VERILOG SOURCE CODE:

Behavioral Modeling:

```
module siso(din, clk, rst, dout);
   input din;
   input clk;
   input rst;
   output dout;
          reg dout;
          reg [7:0]x;
          always @ (posedge(clk) or posedge(rst)) begin
          if (rst==1'b1)
          begin
          dout=8'hzz;
          end
          else
          begin
          x={x[6:0],din};
          dout=x[7];
          end
          end
          endmodule
```

SERIAL IN PARALLEL OUT SHIFT REGISTER:

LOGIC DIAGRAM :



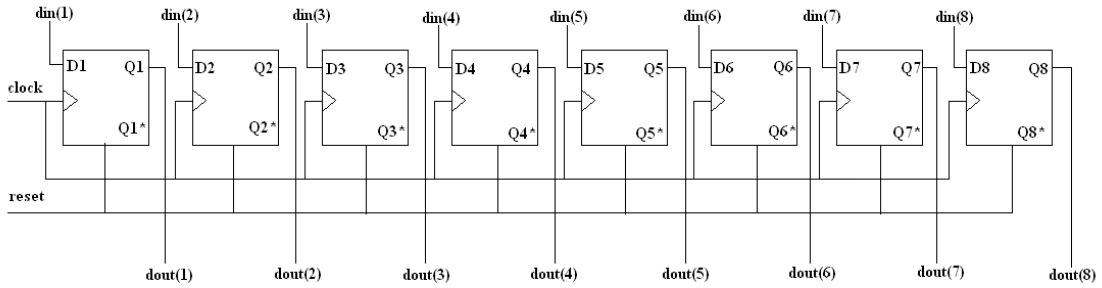
VERILOG SOURCE CODE:

Behavioral Modeling:

```
module sipo(din, clk, rst, dout);
   input din;
   input clk;
   input rst;
   output [7:0] dout;
          reg[7:0]dout;
          reg[7:0]x;
          always @ (posedge(clk) or posedge(rst))
          begin
          if(rst)
          dout=8'hzz;
          else
          begin
          x={x[6:0],din};
          dout=x;
          end
          end
endmodule
```

PARALLEL-IN PARELLEL-OUT SHIFT REGISTER:

LOGIC DIAGRAM :



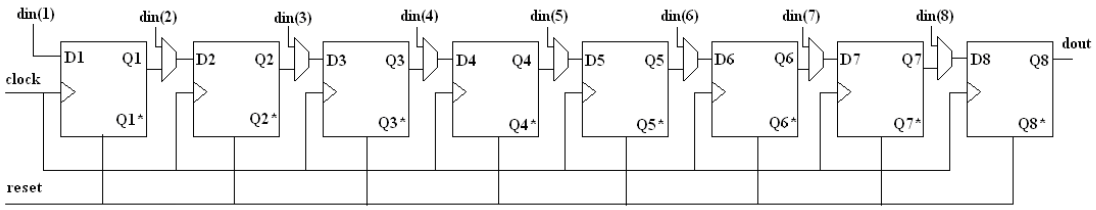
VERILOG SOURCE CODE:

Behavioral Modeling:

```
module pipo(clk, rst, din, dout);
   input clk;
   input rst;
   input [7:0] din;
   output [7:0] dout;
          reg [7:0] dout;
          always @ (posedge(clk) or posedge(rst)) begin
          if (rst==1'b1)
          begin
          dout=8'hzz;
          end
          else
          begin
          dout=din;
          end
          end
endmodule
```

PARALLEL-IN SERIAL-OUT SHIFT REGISTER:

LOGIC DIAGRAM :



VERILOG SOURCE CODE:

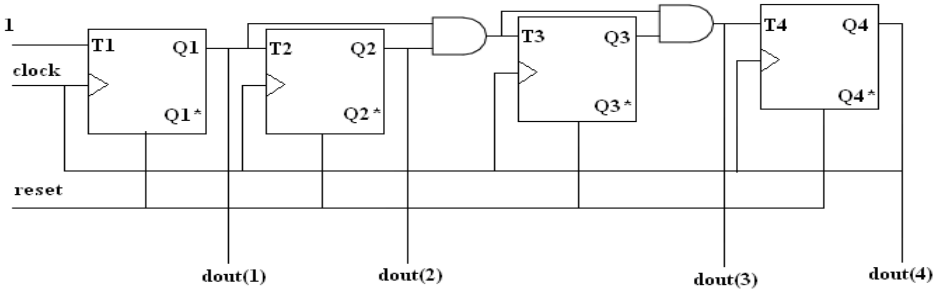
Behavioral Modeling:

```
module piso(din, clk, rst, load, dout);
  input [7:0] din;
  input clk;
  input rst;
  input load;
  output dout;
        reg dout;
        reg [8:0]x;
        always @(posedge(clk) or posedge(rst)) begin
        if (rst==1'b1)
        begin
        dout=1'bz;
        end
        else
        begin
        if (load==1'b0)
        begin
        x=din;
        end
        else
        x={x[7:0],1'hz};
        dout=x[8];
        end
        end
        endmodule
```

SYNCHRONOUS AND ASYNCHRONOUS COUNTER

SYNCHRONOUS COUNTER:

LOGIC DIAGRAM:



VERILOG SOURCE CODE:

Behavioral Modeling:

```
module syncntr(clk, rst, q);
  input clk;
  input rst;
  output [3:0]q;
        reg [3:0]q;
        reg [3:0]x=0;
        always @ (posedge(clk) or posedge(rst))
        begin
        if (rst==1'b1)
        begin
        q=4'b0;
        end
        else
        begin
        x=x+1'b1;
        end
        q=x;
        end
        endmodule
```

RESULT:

Thus the Output's of of 8:1 Mux/Demux, Full Adder, 8-bit Magnitude comparator, Encoder/decoder, Priority encoder, D-FF, 4-bit Shift registers, 3-bit Synchronous Counters, Binary to Gray converter, Parity generator are verified by synthesizing and simulating the VERILOG code.

Exp.2-DESIGN OF MOORE AND MEALY FSM

AIM:

To develop the source code for moore and melay FSM by using VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3

ALGORITM:

Step1: Define the specifications and initialize the design.

Step2: Write the source code in VERILOG.

Step3: Check the syntax and debug the errors if found, obtain the synthesis report.

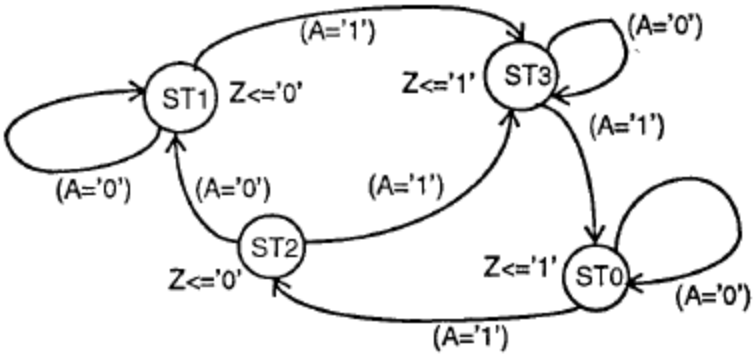
Step4: Verify the output by simulating the source code.

Step5: Write all possible combinations of input using the test bench.

Step6: Obtain the place and route report.

MOORE FSM:

LOGIC DIAGRAM:



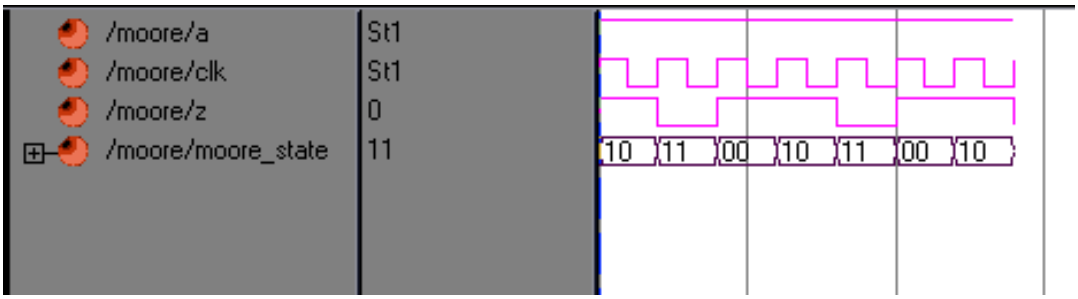
VERILOG SOURCE CODE:

Behavioral Modeling:

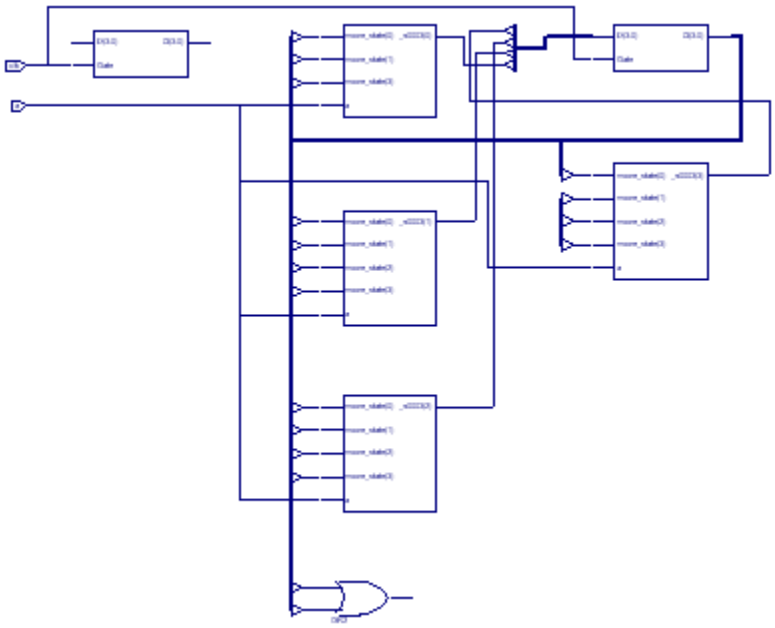
```
module moorefsm(a,clk,z);
  input a;
  input clk;
  output z;
        reg z;

        parameter st0=0,st1=1,st2=2,st3=3;
        reg[0:1]moore_state;
        initial
        begin
        moore_state=st0;
        end
        always @ (posedge(clk))
        case(moore_state)
        st0:
        begin
        z=1;
        if(a)
        moore_state=st2;
        end

        st1:
        begin
        z=0;
        if(a)
        moore_state=st3;
        end

        st2:
        begin
        z=0;
        if(~a)
        moore_state=st1;
        else
        moore_state=st3;
        end

        st3:
```

```
begin
z=1;
if(a)
moore_state=st0;
end
endcase
endmodule
```

Simulation output:



Synthesis RTL Schematic:



Synthesis report:

```
=========================================================================
*                  Final Report                    *
=========================================================================
Device utilization summary:
---------------------------
Selected Device : 3s400tq144-5
 Number of Slices:              3  out of   3584     0%
 Number of Slice Flip Flops:    5  out of   7168     0%
 Number of 4 input LUTs:        5  out of   7168     0%
 Number of bonded IOBs:         3  out of     97     3%
 Number of GCLKs:               1  out of      8    12%
=========================================================================
TIMING REPORT
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE. FOR ACCURATE TIMING
INFORMATION PLEASE REFER TO THE TRACE REPORT GENERATED AFTER PLACE-and-ROUTE.
Clock Information:
------------------------------------+------------------------+-------+
Clock Signal                        | Clock buffer(FF name)  | Load  |
------------------------------------+------------------------+-------+
clk                                 | BUFGP                  | 5     |
------------------------------------+------------------------+-------+
Timing Summary:
```

Speed Grade: -5

Minimum period: 2.910ns (Maximum Frequency: 343.648MHz)

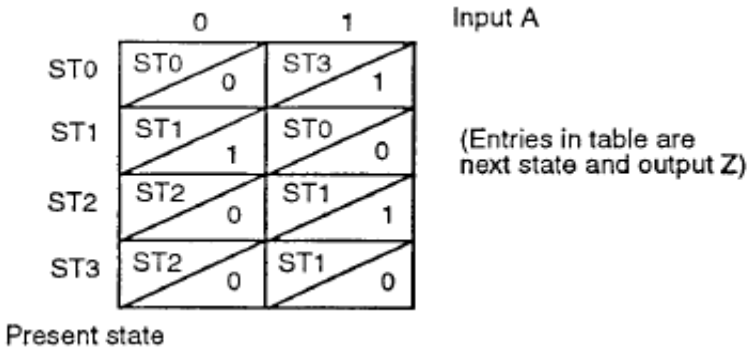
Minimum input arrival time before clock: 2.444ns

Maximum output required time after clock: 6.141ns path found

Maximum combinational path delay: No

MEALY FSM:

TRUTH TABLE:



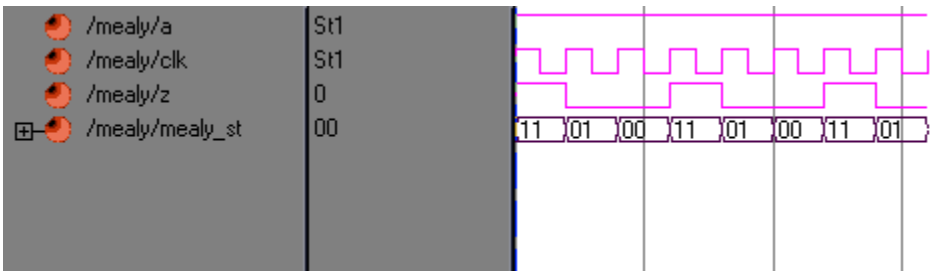
VERILOG SOURCE CODE:

Behavioral Modeling:

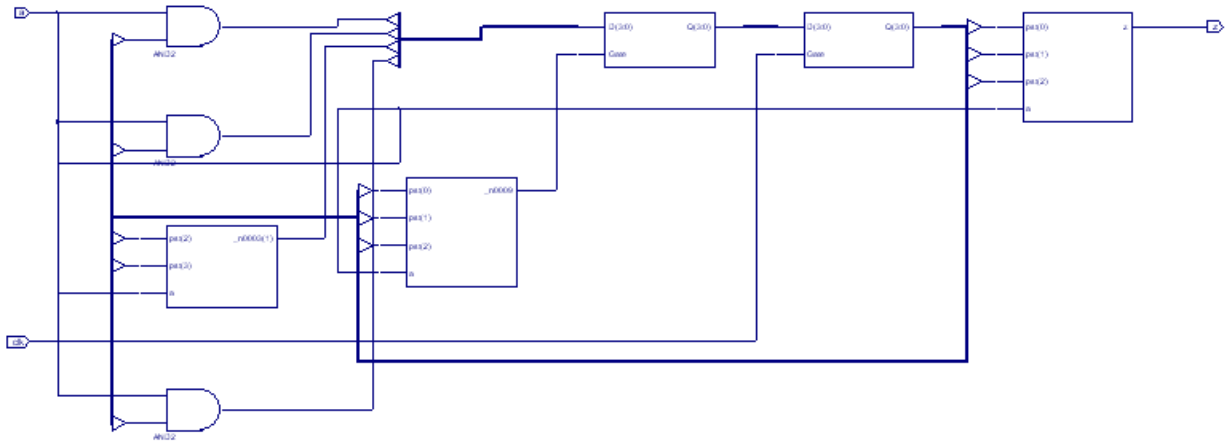
```
module mealayfsm(a, clk, z);
  input a;
  input clk;
  output z;
        reg z;

        parameter st0=0,st1=1,st2=2,st3=3;
        reg[0:1]mealy_state;
        initial
        begin
        mealy_state=st0;
        end
        always @ (posedge(clk))
        case(mealy_state)
        st0:
        begin
        if(a) begin
        z=1;
        mealy_state=st3; end
```

```
else
z=0;
end

st1:
begin
if(a) begin
z=0;
mealy_state=st0; end
else
z=1;
end

st2:
begin
if(a) begin
z=1;
mealy_state=st1; end
else
z=0;
end

st3:
begin
z=0;
if(a) begin
mealy_state=st1;        end
else
mealy_state=st2;
end

endcase
endmodule
```

Simulation output:



Synthesis RTL Schematic:



Synthesis report:

==========

* Final Report *

==========

Device utilization summary:

Selected Device : 3s400tq144-5

Number of Slices: 5 out of 3584 0%

Number of Slice Flip Flops: 8 out of 7168 0%

Number of 4 input LUTs: 6 out of 7168 0%

Number of bonded IOBs: 3 out of 97 3%

Number of GCLKs: 1 out of 8 12%

==========

TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.

FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT

GENERATED AFTER PLACE-and-ROUTE.

Clock Information:

Clock Signal | Clock buffer(FF name) | Load |

---+---+---+

_n0009(_n00091:O) | NONE(*)(nst_3) | 4 |

clk | BUFGP | 4 |

---+---+---+

(*) This 1 clock signal(s) are generated by combinatorial logic,

and XST is not able to identify which are the primary clock signals.

Please use the CLOCK_SIGNAL constraint to specify the clock signal(s) generated by combinatorial logic.

INFO:Xst:2169 - HDL ADVISOR - Some clock signals were not automatically buffered by XST with BUFG/BUFR resources. Please use the buffer_type constraint in order to insert these buffers to the clock signals to help prevent skew problems.

Timing Summary:

Speed Grade: -5

Minimum period: No path found

Minimum input arrival time before clock: 2.518ns

Maximum output required time after clock: 7.561ns

Maximum combinational path delay: 7.931ns

RESULT:

Thus the output of Synchronous FSM – Mealy and Moore machines are verified by synthesizing and simulating the VERILOG code.

Exp.3-TRAFFIC LIGHT CONTROLLER

AIM:

To develop the source code for traffic light controller by using VHDL/VEILOG and obtain the simulation, place and route and implementation into FPGA.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3

ALGORITM:

Step1: Define the specifications and initialize the design.

Step2: Write the source code in VERILOG.

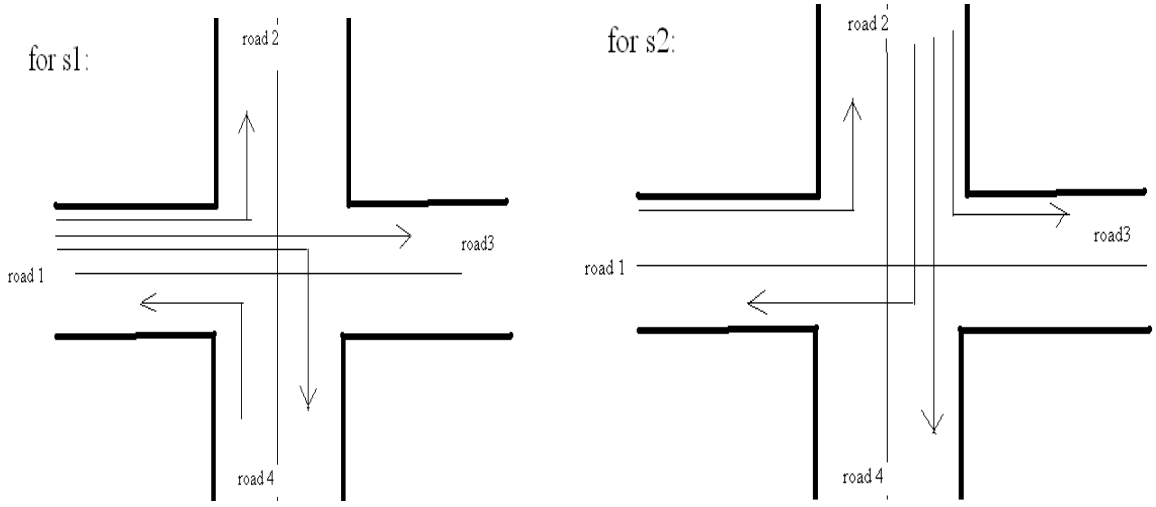
Step3: Check the syntax and debug the errors if found, obtain the synthesis report.

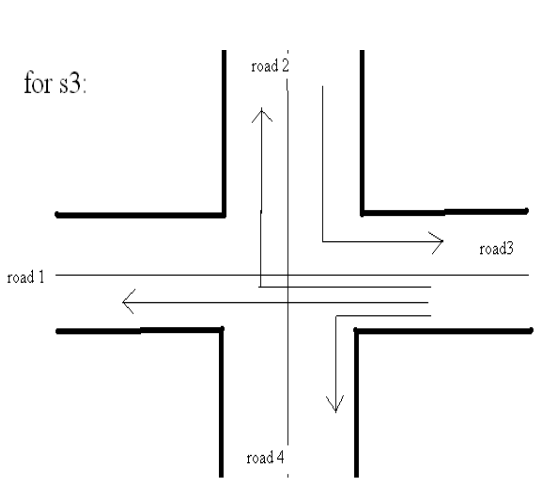
Step4: Verify the output by simulating the source code.

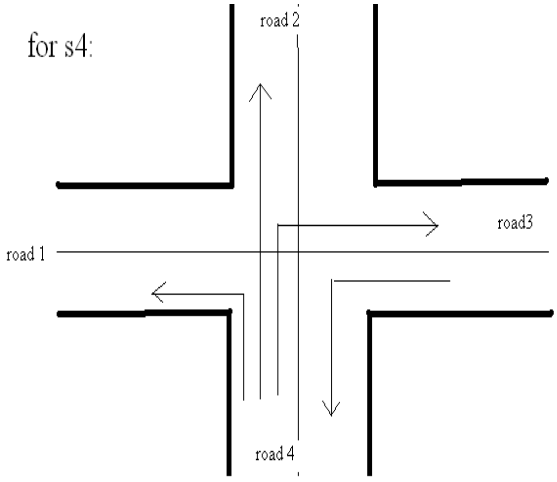
Step5: Write all possible combinations of input using the test bench.

Step6: Obtain the place and route report.

LOGIC DIAGRAM:







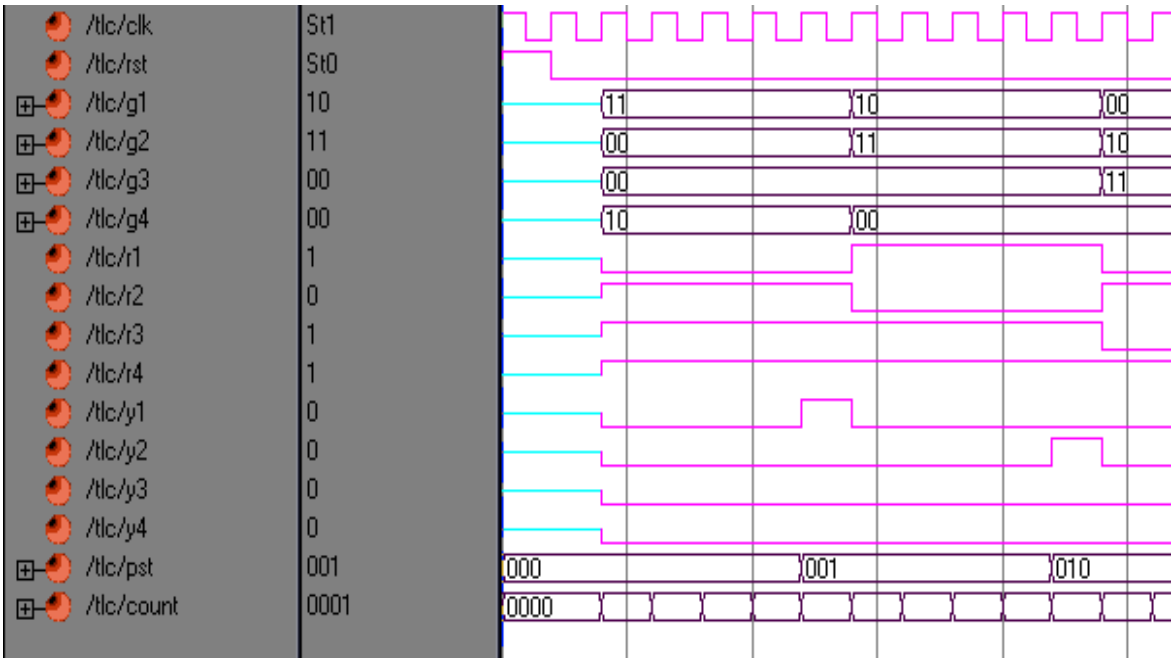
VERILOG SOURCE CODE:

Behavioral Modeling:

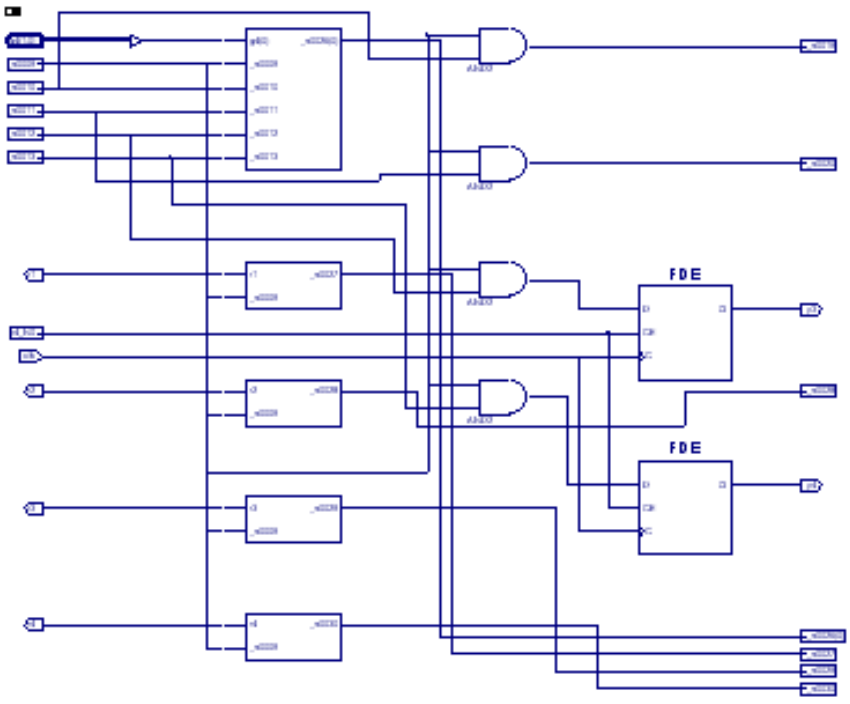
```
module tlc(clk,rst, g1,g2,g3,g4, r1,r2,r3,r4,y1,y2,y3,y4);
   input clk,rst;
   output [1:0] g1,g2,g3,g4;
   output r1,r2,r3,r4,y1,y2,y3,y4;
          reg[1:0]g1,g2,g3,g4;
          reg r1,r2,r3,r4,y1,y2,y3,y4;
                    parameter st0=0,st1=1,st2=2,st3=3;
                                        reg [2:0]pst;
                    reg[3:0] count=0;
                                        always@(posedge clk )
                    begin
                    if(rst==1'b1)
                    pst=st0;
                     else
                    case(pst)

                    st0:
                   if(count==4)
                    begin
                    y1=1'b1;y2=1'b0;y3=1'b0;y4=1'b0;
                    count=0;
                    pst=st1;
                    end
                    else
                    begin
                    g1=2'b11;g2=2'b00;g3=2'b00;g4=2'b10;
                    r1=1'b0;r2=1'b1;r3=1'b1;r4=1'b1;
                    y1=1'b0;y2=1'b0;y3=1'b0;y4=1'b0;
                    pst=st0;
                    count=count+1;
                    end
                     st1:
```

```
                if(count==4)
                begin
                y1=1'b0;y2=1'b1;y3=1'b0;y4=1'b0;
                count=0;
                pst=st2;
                end
                else
                begin
                g1=2'b10;g2=2'b11;g3=2'b00;g4=2'b00;
                r1=1'b1;r2=1'b0;r3=1'b1;r4=1'b1;
                y1=1'b0;y2=1'b0;y3=1'b0;y4=1'b0;
                pst=st1;
                count=count+1;
                end
                        st2:
                        if(count==4)
                begin
                y1=1'b0;y2=1'b0;y3=1'b1;y4=1'b0;
                count=0;
                pst=st3;
                end
                else
                begin
                g1=2'b00;g2=2'b10;g3=2'b11;g4=2'b00;
                r1=1'b0;r2=1'b1;r3=1'b0;r4=1'b1;
                y1=1'b0;y2=1'b0;y3=1'b0;y4=1'b0;
                pst=st2;
                count=count+1;
                end

                st3:
                                if(count==4)
                begin
                y1=1'b0;y2=1'b0;y3=1'b0;y4=1'b1;
                count=0;
                pst=st1;
                end
                else
                begin
                g1=2'b00;g2=2'b00;g3=2'b10;g4=2'b11;
                r1=1'b0;r2=1'b1;r3=1'b1;r4=1'b0;
                y1=1'b0;y2=1'b0;y3=1'b0;y4=1'b0;
                pst=st3;
                count=count+1;
                end
                endcase
                end
endmodule
```

Simulation output:



Synthesis RTL Schematic:



Synthesis report:

```
=====================================================================
*                  Final Report                     *
=====================================================================
```

Device utilization summary:

Selected Device : 3s400tq144-5

Number of Slices:	45	out of 3584	1%
Number of Slice Flip Flops:	50	out of 7168	0%
Number of 4 input LUTs:	64	out of 7168	0%
Number of bonded IOBs:	18	out of 97	18%
Number of GCLKs:	1	out of 8	12%

```
=====================================================================
```

TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.

FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT

GENERATED AFTER PLACE-and-ROUTE.

Clock Information:

Clock Signal	Clock buffer(FF name)	Load
clk	BUFGP	50

Timing Summary:

Speed Grade: -5

Minimum period: 6.698ns (Maximum Frequency: 149.305MHz)

Minimum input arrival time before clock: 5.509ns

Maximum output required time after clock: 6.280ns

Maximum combinational path delay: No path found

RESULT:

Thus the output of Traffic Light controller are verified by synthesizing and simulating the VERILOG code.

Exp.4-PCI Bus & arbiter

AIM:

To develop the source code for PCI Bus & arbiter by using VEILOG and obtain the simulation, place and route and implementation into FPGA.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3

ALGORITM:

Step1: Define the specifications and initialize the design.

Step2: Write the source code in VERILOG.

Step3: Check the syntax and debug the errors if found, obtain the synthesis report.

Step4: Verify the output by simulating the source code.

Step5: Write all possible combinations of input using the test bench.

Step6: Obtain the place and route report.

LOGIC DIAGRAM:

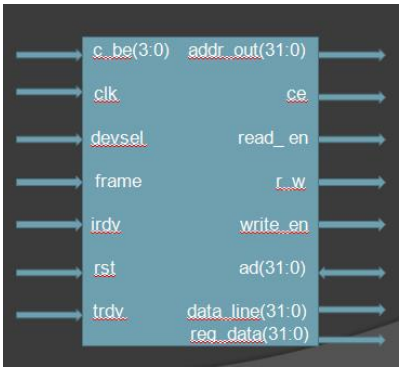


Fig. 1 Interface Block

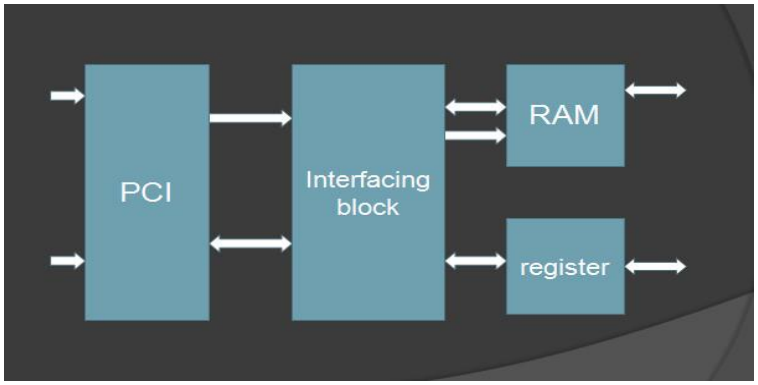


Fig. 2 Simulation Environment

VERILOG SOURCE CODE:

```
input wire clk,GNT,start;
input wire GLOBAL_IRDY;
output reg REQ = 1,FRAME = 1,I_AM_OWNER = 0,IRDY =1  ;
integer counter = 0;
wire IDLE ;
assign IDLE = FRAME & GLOBAL_IRDY ;
reg pipeline1 =1,pipeline2 =1,pipeline3 =1,pipeline4 =1,pipeline5 = 1 ;
always @(posedge clk)
begin
if(start & (GNT) & FRAME)
begin
        @(negedge clk)
        REQ<=0;
end
else if ((~GNT)  && IDLE & (~ REQ) )
begin
        @(negedge clk)
        REQ<=1;
        FRAME <= 0 ;
end
else if(~ pipeline4 && pipeline5)
begin
@(negedge clk)
```

```
FRAME <=1 ;
end
end
always @(posedge clk)
begin
if(~ pipeline1 && pipeline2)
begin
@(negedge clk)
IRDY <= 0;
end
else if (~ pipeline5)
begin
@(negedge clk)
IRDY <=1;
end
end
always @(posedge clk)
begin
 if ((~GNT)  && IDLE & (~ REQ) )
begin
        @(negedge clk)
        I_AM_OWNER<=1;
end
else if (~ pipeline5)
begin
```

```
@(negedge clk)
I_AM_OWNER<=0;
end
end
always @(posedge clk)
begin
pipeline1 <= FRAME ;
pipeline2 <= pipeline1;
pipeline3 <= pipeline2;
pipeline4 <= pipeline3;
pipeline5 <= pipeline4;
end
endmodule
```

PCI-arbiter/ SimpleInitiator.v

```
input [7:0] I_AM_OWNERS,IRDYS,FRAMES;
output wire GLOBAL_IRDY,GLOBAL_FRAME;
assign {GLOBAL_IRDY,GLOBAL_FRAME} =
(I_AM_OWNERS[0]) ? {IRDYS[0],FRAMES[0]} :
(I_AM_OWNERS[1]) ? {IRDYS[1],FRAMES[1]} :
(I_AM_OWNERS[2]) ? {IRDYS[2],FRAMES[2]} :
(I_AM_OWNERS[3]) ? {IRDYS[3],FRAMES[3]} :
(I_AM_OWNERS[4]) ? {IRDYS[4],FRAMES[4]} :
(I_AM_OWNERS[5]) ? {IRDYS[5],FRAMES[5]} :
(I_AM_OWNERS[6]) ? {IRDYS[6],FRAMES[6]} :
(I_AM_OWNERS[7]) ? {IRDYS[7],FRAMES[7]} : {1'b1,1'b1};
module SimpleInitiator (start,clk,REQ,GNT,FRAME,IRDY,I_AM_OWNER,GLOBAL_IRDY);
input wire clk,GNT,start;
input wire GLOBAL_IRDY;
output reg REQ = 1,FRAME = 1,I_AM_OWNER = 0,IRDY =1  ;
integer counter = 0;
wire IDLE ;
```

```
assign IDLE = FRAME & GLOBAL_IRDY ;
reg pipeline1 =1,pipeline2 =1,pipeline3 =1,pipeline4 =1,pipeline5 = 1 ;
always @(posedge clk)
begin
if(start & (GNT) & FRAME)
begin
       @(negedge clk)
       REQ<=0;
end
else if ((~GNT)  && IDLE & (~ REQ) )
begin
       @(negedge clk)
       REQ<=1;
       FRAME <= 0 ;
end
else if(~ pipeline4 && pipeline5)
begin
@(negedge clk)
FRAME <=1 ;
end
end
always @(posedge clk)
begin
if(~ pipeline1 && pipeline2)
begin
@(negedge clk)
IRDY <= 0;
end
else if (~ pipeline5)
begin
@(negedge clk)
IRDY <=1;
end
end
always @(posedge clk)
begin
 if ((~GNT)  && IDLE & (~ REQ) )
begin
       @(negedge clk)
       I_AM_OWNER<=1;
end
else if (~ pipeline5)
```

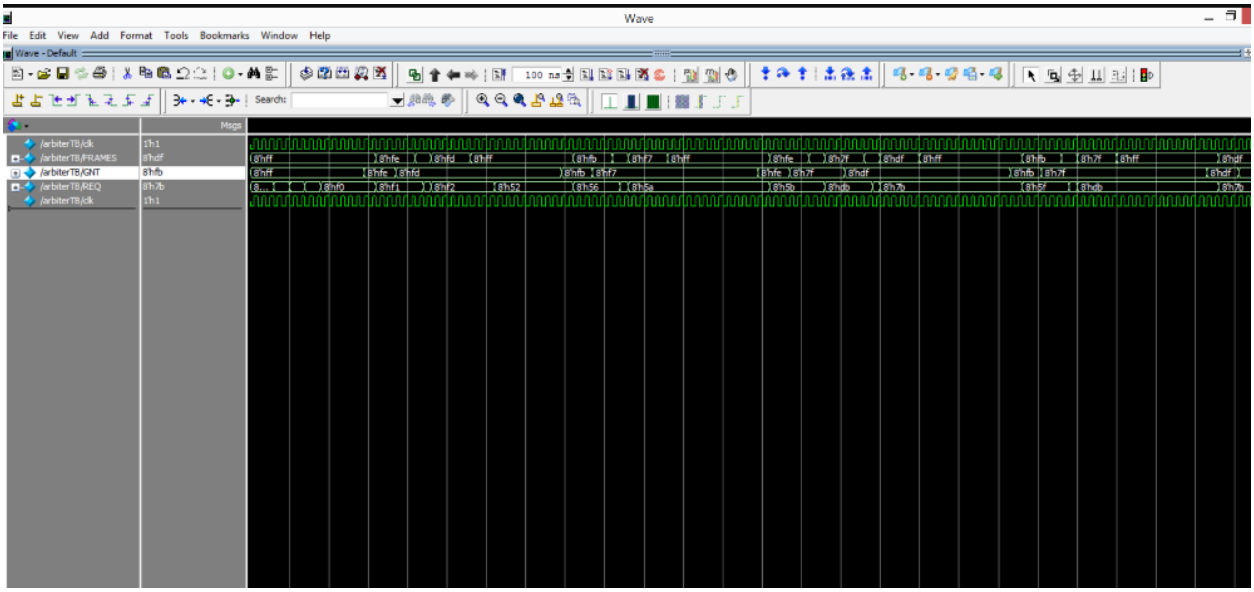
```
begin
@(negedge clk)
I_AM_OWNER<=0;
end
end
always @(posedge clk)
begin
pipeline1 <= FRAME ;
pipeline2 <= pipeline1;
pipeline3 <= pipeline2;
pipeline4 <= pipeline3;
pipeline5 <= pipeline4;
end
endmodule
module SpecializedMux(I_AM_OWNERS,IRDYS,FRAMES,GLOBAL_IRDY,GLOBAL_FRAME);
input [7:0] I_AM_OWNERS,IRDYS,FRAMES;
output wire GLOBAL_IRDY,GLOBAL_FRAME;
assign {GLOBAL_IRDY,GLOBAL_FRAME} =
(I_AM_OWNERS[0]) ? {IRDYS[0],FRAMES[0]} :
(I_AM_OWNERS[1]) ? {IRDYS[1],FRAMES[1]} :
(I_AM_OWNERS[2]) ? {IRDYS[2],FRAMES[2]} :
(I_AM_OWNERS[3]) ? {IRDYS[3],FRAMES[3]} :
(I_AM_OWNERS[4]) ? {IRDYS[4],FRAMES[4]} :
(I_AM_OWNERS[5]) ? {IRDYS[5],FRAMES[5]} :
(I_AM_OWNERS[6]) ? {IRDYS[6],FRAMES[6]} :
(I_AM_OWNERS[7]) ? {IRDYS[7],FRAMES[7]} : {1'b1,1'b1};
endmodule
module arbiter (clk,req,gnt,frame,irdy,trdy);
input wire clk ,irdy,frame,trdy;
input wire [7:0] req;
output reg [7:0] gnt;
reg [7:0] fifo[7:0];
integer pointer;
reg lastframe;
reg trflag;
reg [7:0]lastreq;
initial
begin
lastframe=1'b1;
trflag = 1'b0 ;
gnt = 8'hff ;
pointer= 0;
```

```
lastreq = 8'hff ;
end
always@(posedge clk)
begin
if(lastframe && ~frame)
begin
trflag=1'b1;lastframe =frame;
end
end
if(trflag==1)
begin
@(negedge clk)
gnt=fifo[0]; pointer = pointer - 1;
fifo[0]=fifo[1];fifo[1]=fifo[2];fifo[2]=fifo[3];fifo[3]=fifo[4];fifo[4]=fifo[5];fifo[5]=fifo[6
];fifo[6]=fifo[7];trflag=1'b0;
end
if (lastreq != req)
begin
@(negedge clk)
req=fifo[pointer];pointer = pointer + 1;
end
endmodule
module arbiterTB;
reg clk;
wire [7:0] GNT;
wire [7:0] FRAMES , IRDYS, I_AM_OWNERS;
wire GLOBAL_IRDY , GLOBAL_FRAME;
wire [7:0] REQ;
reg [7:0] STARTS = 8'h00;
always
begin
#5
clk <= ~ clk;
end
initial
begin
clk <= 0;
#20
STARTS[0] <= 1;
#20
STARTS[1] <= 1;
#20
```

```
STARTS[2] <= 1;
#20
STARTS[3] <= 1;
#200
STARTS[3] <= 0;
#20
STARTS[5] <= 1;
STARTS[7] <= 1;
STARTS[1] <= 0;
STARTS[0] <= 0;
end
SpecializedMux myMux(I_AM_OWNERS,IRDYS,FRAMES,GLOBAL_IRDY,GLOBAL_FRAME);
arbiter A(clk,REQ,GNT,GLOBAL_FRAME,GLOBAL_IRDY, 1'b1);
SimpleInitiator simple0 (STARTS[0],clk,REQ[0],GNT[0],FRAMES[0],IRDYS[0],I_AM_OWNERS[0] ,
GLOBAL_IRDY);
SimpleInitiator simple1 (STARTS[1],clk,REQ[1],GNT[1],FRAMES[1],IRDYS[1],I_AM_OWNERS[1] ,
GLOBAL_IRDY);
SimpleInitiator simple2 (STARTS[2],clk,REQ[2],GNT[2],FRAMES[2],IRDYS[2],I_AM_OWNERS[2] ,
GLOBAL_IRDY);
SimpleInitiator simple3 (STARTS[3],clk,REQ[3],GNT[3],FRAMES[3],IRDYS[3],I_AM_OWNERS[3] ,
GLOBAL_IRDY);
SimpleInitiator simple4 (STARTS[4],clk,REQ[4],GNT[4],FRAMES[4],IRDYS[4],I_AM_OWNERS[4] ,
GLOBAL_IRDY);
SimpleInitiator simple5 (STARTS[5],clk,REQ[5],GNT[5],FRAMES[5],IRDYS[5],I_AM_OWNERS[5] ,
GLOBAL_IRDY);
SimpleInitiator simple6 (STARTS[6],clk,REQ[6],GNT[6],FRAMES[6],IRDYS[6],I_AM_OWNERS[6] ,
GLOBAL_IRDY);
SimpleInitiator simple7 (STARTS[7],clk,REQ[7],GNT[7],FRAMES[7],IRDYS[7],I_AM_OWNERS[7] ,
GLOBAL_IRDY);
endmodule
          module arbiterTB;
          wire [7:0] GNT;
          wire [7:0] FRAMES , IRDYS, I_AM_OWNERS;
          wire GLOBAL_IRDY , GLOBAL_FRAME;
          wire [7:0]REQ;
          reg [7:0] STARTS = 8'h00;
          always
          begin
          #5
          clk <= ~ clk;
          end
          initial
```

```
begin
clk <= 0 ;
# 20
STARTS[0] <= 1 ;
#20
STARTS[1] <=1;
#20
STARTS[2] <=1;
#20
STARTS[3] <=1;
# 200
STARTS[3] <=0;
#20
STARTS[5] <=1;
STARTS[7] <=1;
STARTS[1] <=0;
STARTS[0] <=0;
end
SpecializedMux myMux(I_AM_OWNERS,IRDYS,FRAMES,GLOBAL_IRDY,GLOBAL_FRAME);
arbiter A(clk,REQ,GNT,GLOBAL_FRAME,GLOBAL_IRDY, 1'b1);
SimpleInitiator simple0
(STARTS[0],clk,REQ[0],GNT[0],FRAMES[0],IRDYS[0],I_AM_OWNERS[0] , GLOBAL_IRDY);
SimpleInitiator simple1
(STARTS[1],clk,REQ[1],GNT[1],FRAMES[1],IRDYS[1],I_AM_OWNERS[1] , GLOBAL_IRDY);
SimpleInitiator simple2
(STARTS[2],clk,REQ[2],GNT[2],FRAMES[2],IRDYS[2],I_AM_OWNERS[2] , GLOBAL_IRDY);
SimpleInitiator simple3
(STARTS[3],clk,REQ[3],GNT[3],FRAMES[3],IRDYS[3],I_AM_OWNERS[3] , GLOBAL_IRDY);
SimpleInitiator simple4
(STARTS[4],clk,REQ[4],GNT[4],FRAMES[4],IRDYS[4],I_AM_OWNERS[4] , GLOBAL_IRDY);
SimpleInitiator simple5
(STARTS[5],clk,REQ[5],GNT[5],FRAMES[5],IRDYS[5],I_AM_OWNERS[5] , GLOBAL_IRDY);
SimpleInitiator simple6
(STARTS[6],clk,REQ[6],GNT[6],FRAMES[6],IRDYS[6],I_AM_OWNERS[6] , GLOBAL_IRDY);
SimpleInitiator simple7
(STARTS[7],clk,REQ[7],GNT[7],FRAMES[7],IRDYS[7],I_AM_OWNERS[7] , GLOBAL_IRDY);
endmodule
```

OUTPUT WAVE FORM



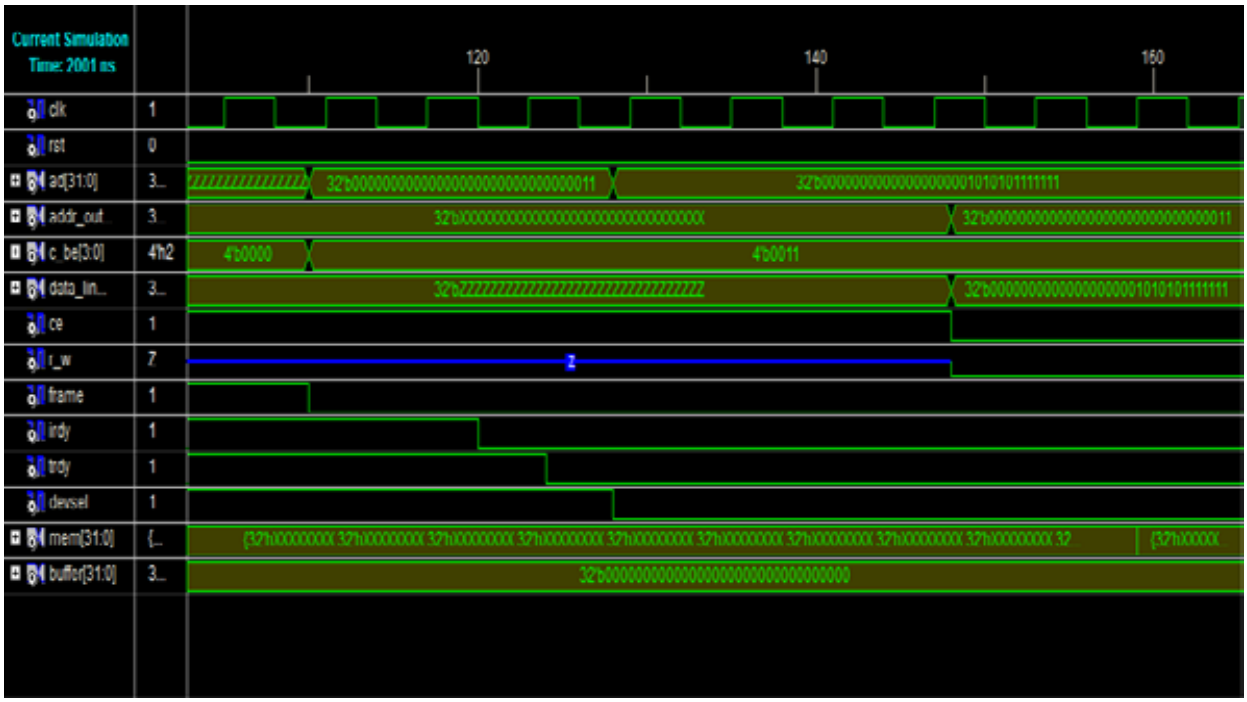


Fig. 3. Simulation waveform of write data to RAM

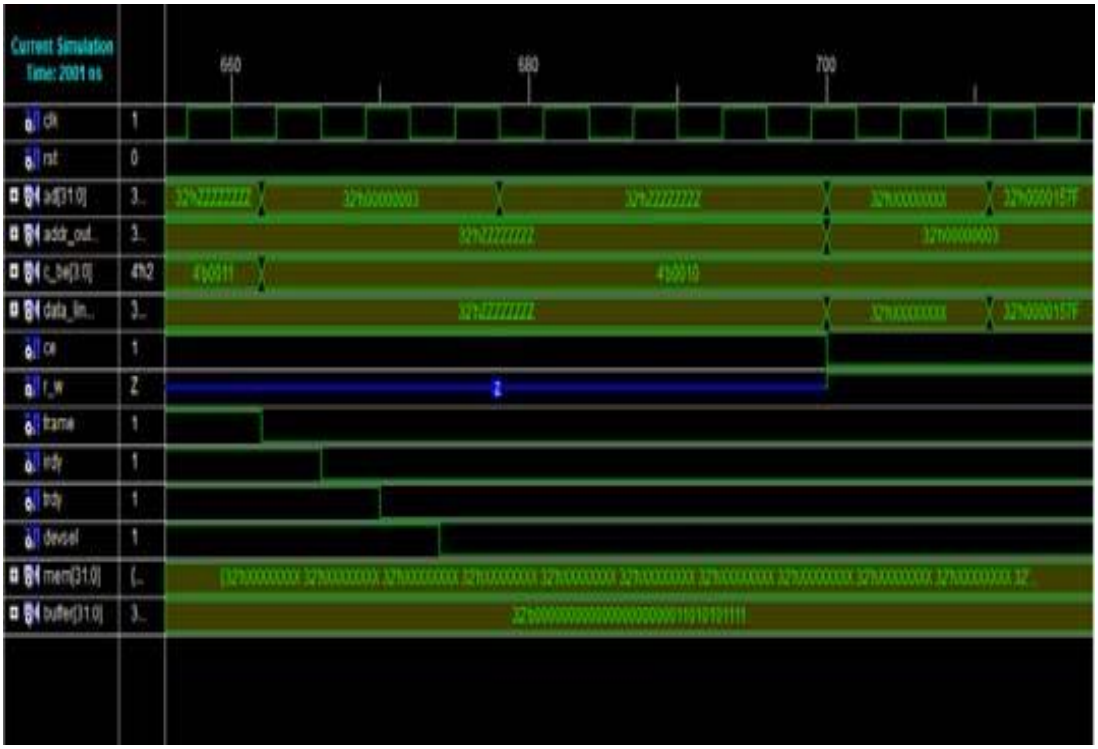


Fig. 4. Simulation waveform of Read data from RAM

RESULT:

Thus the output of PCI Bus & arbiter are verified by synthesizing and simulating the VERILOG code.

Exp.5-UART/ USART implementation in Verilog.

AIM:

To develop the source code for UART/USART by using VEILOG and obtain the simulation, place and route and implementation into FPGA.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3

ALGORITM:

Step1: Define the specifications and initialize the design.

Step2: Write the source code in VERILOG.

Step3: Check the syntax and debug the errors if found, obtain the synthesis report.

Step4: Verify the output by simulating the source code.

Step5: Write all possible combinations of input using the test bench.

Step6: Obtain the place and route report.

UART MODULE:

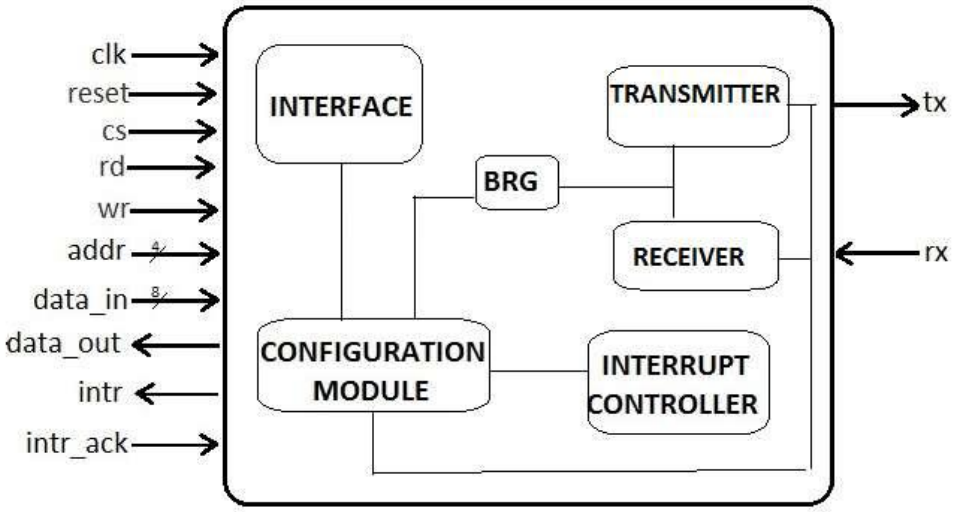


Fig 1.UART Module

Verilog Source code:

```
/*
 * Static clock divider. Displays deviation from target output frequency during synthesis.
 *
 * Author: whitequark@whitequark.org (2016)
 *
 * Parameters:
 *  FREQ_I:  input frequency
 *  FREQ_O:  target output frequency
 *  PHASE:   polarity of the output clock after reset
 *  MAX_PPM: maximum frequency deviation; produces an error if not met
 *
 * Signals:
 *  reset:   active-low reset
 *  clk_i:   input clock
 *  clk_o:   output clock
 */
module ClockDiv #(
      parameter FREQ_I  = 2,
      parameter FREQ_O  = 1,
      parameter PHASE   = 1'b0,
      parameter MAX_PPM = 1_000_000
   ) (
      input  reset,
      input  clk_i,
      output clk_o
   );

   // This calculation always rounds frequency up.
   localparam INIT = FREQ_I / FREQ_O / 2 - 1;
   localparam ACTUAL_FREQ_O = FREQ_I / ((INIT + 1) * 2);
   localparam PPM = 64'd1_000_000 * (ACTUAL_FREQ_O - FREQ_O) / FREQ_O;
   initial $display({"ClockDiv #(.FREQ_I(%d), .FREQ_O(%d),\n",
                     "          .INIT(%d), .ACTUAL_FREQ_O(%d), .PPM(%d))"},
                    FREQ_I, FREQ_O, INIT, ACTUAL_FREQ_O, PPM);
   generate
      if(INIT < 0)
         _ERROR_FREQ_TOO_HIGH_ error();
      if(PPM > MAX_PPM)
         _ERROR_FREQ_DEVIATION_TOO_HIGH_ error();
   endgenerate

   reg [$clog2(INIT):0] cnt = 0;
   reg                  clk = PHASE;
   always @(posedge clk_i or negedge reset)
      if(!reset) begin
         cnt <= 0;
```

```
        clk <= PHASE;
      end else begin
        if(cnt == 0) begin
          clk <= ~clk;
          cnt <= INIT;
        end else begin
          cnt <= cnt - 1;
        end
      end

    assign clk_o = clk;

endmodule
```

UART Transmitter.

```
1.       module UART_Transmitter (Serial_out, Data_Bus, Byte_ready,
   Load_XMT_datareg, T_byte, Clock, reset);
2.       parameter word_size = 8;                  //size of data word
3.       parameter one_count = 3;                 //number of states
4.       parameter state_count = one_count;     //number of bits in
   state register
5.       parameter size_bit_count = 3;             //size of bit
   counter
6.       parameter idle = 3'b001;
7.       parameter waiting = 3'b010;
8.       parameter sending = 3'b100;
9.       parameter all_ones = 9'b1_1111_1111;      //word + extra bit
10.      output serial_out                         //serial output to
   data channel
11.      input [word_size - 1:0] Data_Bus;       //data bus containing
   data word
12.      input Byte_ready;                       //used by host to
   signal ready
13.      input Load_XMT_datareg;              //used to load the data
   register
14.      input T_byte;               //used to signal the start of
   transmission
15.      input Clock;                 //bit clock of the transmitter
16.      input reset_;                //resets internal registers
17.      reg [word_size - 1:0] XMT_datareg;          //transmit data
   register
18.      reg [word_size:0] XMT_shftreg;              //transmit shift
   register
19.      reg Load_XMT_shfteg; //flag to load
20.      reg [state_count - 1:0] state, next_state; //state machine
   controller
21.      reg [size_bit_count:0] bit_count; //counts the bits that are
   transmitted
```

```
22.       reg clear; //clears bit_count after last bit is sent
23.       reg shift; //causes shift of data in XMT_shftreg
24.       reg start; //signals start of transmission
25.       assign Serial_out = XMT_shftreg[0]; //LSB of shift register
26.       always@(state or Byte_ready or bit_count or T_byte)
27.       begin: Output_and_next_state
28.       Load_XMT_shftreg = 0;
29.       clear = 0;
30.       shift = 0;
31.       start = 0;
32.       next_state = state;
33.       case(state)
34.       idle: if(Byte_ready == 1) begin
35.       Load_XMT_shftreg = 1;
36.       next_state = waiting;
37.       end
38.       waiting: if(T_byte == 1)begin
39.       start = 1;
40.       next_state = sending;
41.       end
42.       sending: if(bit_count != word_size + 1)
43.       shift = 1;
44.       else begin
45.       clear = 1;
46.       next_state = idle;
47.       end
48.       default: next_state = idle;
49.       endcase
50.       end
51.       always@(posedge Clock or negedge reset_) begin:
   State_Transitions
52.       if(reset_ == 0) state <= idle; else state <= next_state: end
53.       always@(posedge Clock or negedge reset_) begin:
   Register_Transfers
54.       if(reset_ == 0) begin
55.       XMT_shftreg <= all_ones;
56.       bit_count <= 0;
57.       end
58.       else begin
59.       if(Load_XMT_datareg == 1)
60.       XMT_datareg <= Data_Bus; //get the data bus
61.       if(Load_XMT_shftreg == 1)
62.       XMT_shftreg <= (XMT_datareg, 1'b1); //load shift reg
63.       if(start == 1)
64.       XMT_shftreg[0] <= 0; //signal start of transmission
65.       if(clear == 1) bit_count <= 0;
66.       else if(shift == 1) bit_count <= bit_count + 1;
67.       if(shift ==1) XMT_shftreg <= {1'b1,
   XMT_shftreg[word_size:1]}; //shift right, fill with 1's
```

```
68.       end end endmodule
```

UART Receiver

```
1.        module UART8_Receiver
2.        (RCV_datareg, read_not_ready_out, Error1 ,Error2, Serial_in,
  read_no_ready_in, Sample_clk, reset_); //Sample_clk is 8x Bit_clk
3.        parameter word_size = 8;
4.        parameter half_word = word_size/2;
5.        parameter Num_count_bits = 4; //to hold count of word_size
6.        parameter Num_state_bits = 2; //number of bits in states
7.        parameter idle = 2'b00;
8.        parameter starting = 2'b01;
9.        parameter receiving = 2'b10;
10.       output [word_size - 1:0] RCV_datareg;
11.       output read_not_ready_out, Error1, Error2;
12.       input Serial_in, Sample_clk, reset_, read_not_ready_in;
13.       reg RCV_datareg;
14.       reg [word_size - 1:0] RCV_shftreg;
15.       reg [Num_counter_bits - 1:0] Sample_counter;
16.       reg [Num_counter_bits:0] Bit_counter;
17.       reg [Num_state_bits - 1:0] state, next_state;
18.       reg inc_Bit_counter, clr_Bit_counter;
19.       reg inc_Sample_counter, clr_Sample_counter;
20.       reg shift, load, read_not_ready_out;
21.       reg Error1, Error2;
22.       always @ (state or Serial_in or read_not_ready_in or
  Sample_counter or Bit_counter) begin
23.       read_not_ready_out = 0;
24.       clr_Sample_counter = 0;
25.       clr_Bit_counter = 0;
26.       inc_Sample_counter = 0;
27.       inc_Bit_counter = 0;
28.       shift = 0;
29.       Error1 = 0;
30.       Error2 = 0;
31.       load = 0;
32.       next_state = state;
33.       case (state)
34.       idle: if (Serial_in = = 0) next_state = starting;
35.       starting: if (Serial_in = = 1) begin
36.       next_state = idle;
37.       clr_Sample_counter = 1 ;
38.       end else
39.       if (Sample_counter = = half_word - 1) begin
40.       next_state = receiving;
41.       clr_Sample_counter = 1 ;
42.       end else inc_Sample_counter = 1 ;
```

```
43.       receiving: if (Sample_counter < word_ size - 1)
   inc_Sample_counter = 1 ;
44.       else begin
45.       clr_Sample_counter = 1 ;
46.       if (Bit_counter != word_size) begin
47.       shift = 1 ;
48.       inc_Bit_counter = 1 ;
49.       end
50.       else begin
51.       next_state = idle;
52.       read_not_ready_out = 1 ;
53.       clr_Bit_counter = 1 ;
54.       if (read_not_ready_in = = 1 ) Error1 = 1 :
55.       else if (Serial_in = = 0) Error2 = 1 ;
56.       else load = 1 ;
57.       end end
58.       default: next_state = idle;
59.       endcase
60.       end
61.       always @ (posedge Sample_clk) begin
62.       if (reset_ = = 0) begin
63.       state <= idle;
64.       Sample_counter <= 0;
65.       Bit_counter <= 0;
66.       RCV_datareg <= 0;
67.       RCV_shftreg <= 0;
68.       end
69.       else begin
70.       state <= next_state;
71.       if (clr_Sample_counter = = 1 ) Sample_counter <= 0;
72.       else if (inc_Sample_counter = = 1 ) Sample_counter <=
   Sample_counter + 1 ;
73.       if (clr_Bit_counter = = 1 ) Bit_counter <= 0;
74.       else if (inc_Bit_counter == 1) Bit_counter <= Bit_counter + 1
   ;
75.       if (shift = = 1 ) RCV_shftreg <= (Serial_in,
   RCV_shftreg[word_size - 1 : 1)};
76.       if (load = = 1) RCV_datareg <= RCV_shftreg;
77.       end
78.       end endmodule
```

UART Test bench:

```
module UARTLoopback(
    input       clk_12mhz,
    output [7:0] leds,
    input       uart_rx,
    output      uart_tx,
    output      debug1,
    output      debug2
  );

  wire [7:0] rx_data;
  wire       rx_ready;
  wire       rx_ack;
  wire       rx_error;
  wire [7:0] tx_data;
  wire       tx_ready;
  wire       tx_ack;
  UART #(
    .FREQ(12_000_000),
    .BAUD(115200)
  ) uart (
    .reset(1'b1),
    .clk(clk_12mhz),
    .rx_i(uart_rx),
    .rx_data_o(rx_data),
    .rx_ready_o(rx_ready),
    .rx_ack_i(rx_ack),
    .rx_error_o(rx_error),
    .tx_o(uart_tx),
    .tx_data_i(tx_data),
    .tx_ready_i(tx_ready),
    .tx_ack_o(tx_ack)
  );

  reg       empty    = 1'b1;
  reg  [7:0] data     = 8'h00;
  wire      rx_strobe = (rx_ready && empty);
  wire      tx_strobe = (tx_ack && !empty);
  always @(posedge clk_12mhz) begin
    if(rx_strobe) begin
      data <= rx_data;
      empty <= 1'b0;
    end
    if(tx_strobe)
      empty <= 1'b1;
  end
```

```
  assign rx_ack   = rx_strobe;
  assign tx_data  = data;
  assign tx_ready = tx_strobe;

  assign leds = {rx_error, rx_data[6:0]};
  assign debug1 = uart_rx;
  assign debug2 = uart_tx;

endmodule
```

Output Waveform:

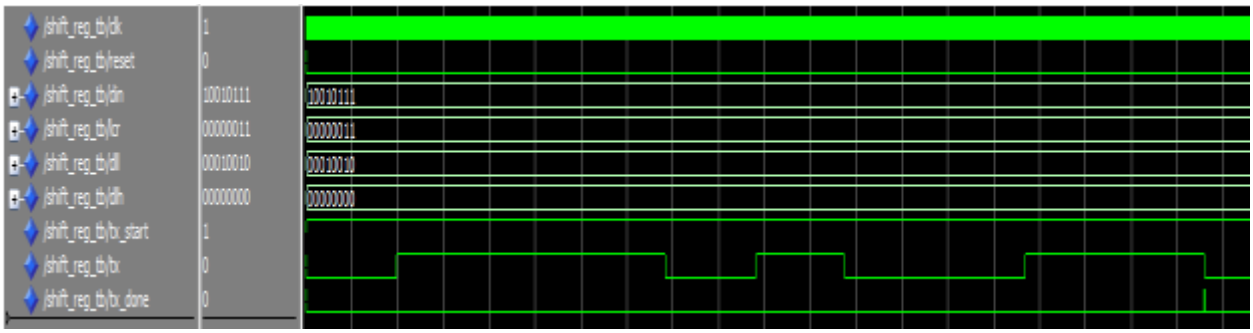


Fig.2-Simulation Waveform of Transmission

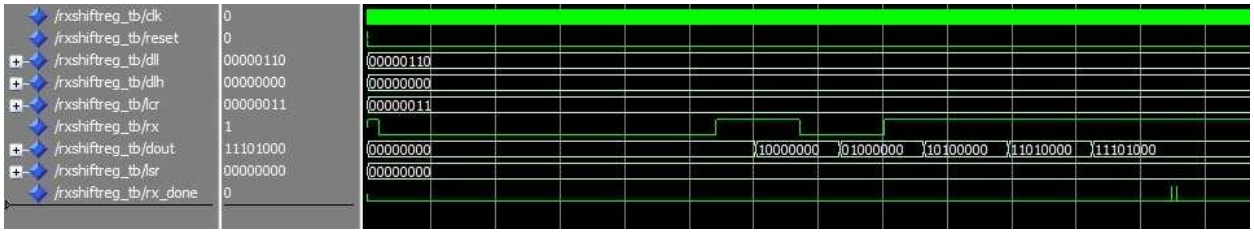


Fig.3-Simulation Waveform of Receiver

RESULT:

Thus the output of UART/ USART are verified by synthesizing and simulating the VERILOG code.

Exp.6-DESIGN OF MEMORIES

AIM:

To develop the source code for memories by using VEILOG and obtain the simulation, place and route and implementation into FPGA.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3

ALGORITM:

Step1: Define the specifications and initialize the design.

Step2: Write the source code in VERILOG.

Step3: Check the syntax and debug the errors if found, obtain the synthesis report.

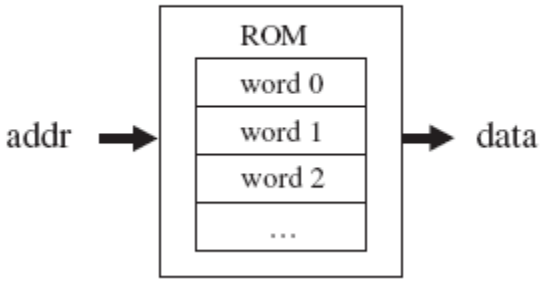
Step4: Verify the output by simulating the source code.

Step5: Write all possible combinations of input using the test bench.

Step6: Obtain the place and route report.

BLOCK DIAGRAM:

ROM

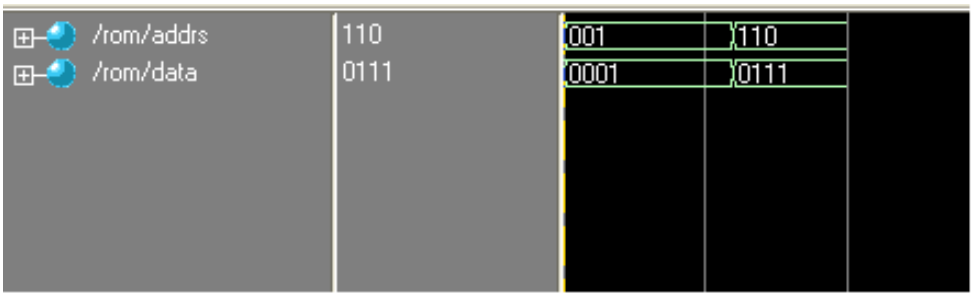


VERILOG SOURCE CODE:

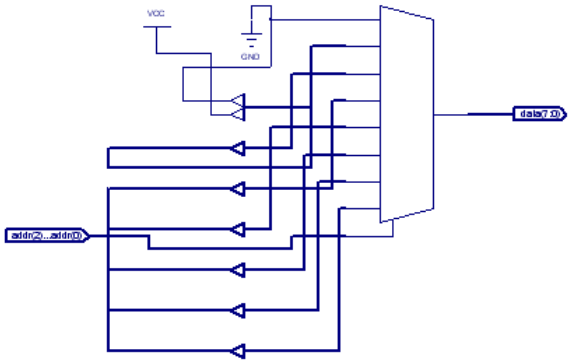
Behavioral Modeling:

```
module rom(addrs, data);
   input [2:0] addrs;
   output [3:0] data;
            reg[3:0]data;
            reg[3:0]mem[0:7];
            initial
            begin
            mem[0]=4'b0000;
            mem[1]=4'b0001;
            mem[2]=4'b0010;
            mem[3]=4'b0100;
            mem[4]=4'b1000;
            mem[5]=4'b0011;
            mem[6]=4'b0111;
            mem[7]=4'b1111;
            end
            always@(addrs)
            begin
            data=mem[addrs];
            end
endmodule
```

Simulation output:



Synthesis RTL Schematic:



Synthesis report:

==========

*** Final Report ***

==========

Device utilization summary:

Selected Device : 3s400tq144-5

Number of Slices: 4 out of 3584 0%

Number of 4 input LUTs: 7 out of 7168 0%

Number of bonded IOBs: 11 out of 97 11%

==========

TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.

FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT

GENERATED AFTER PLACE-and-ROUTE.

Clock Information:

No clock signals found in this design

Timing Summary:

Speed Grade: -5

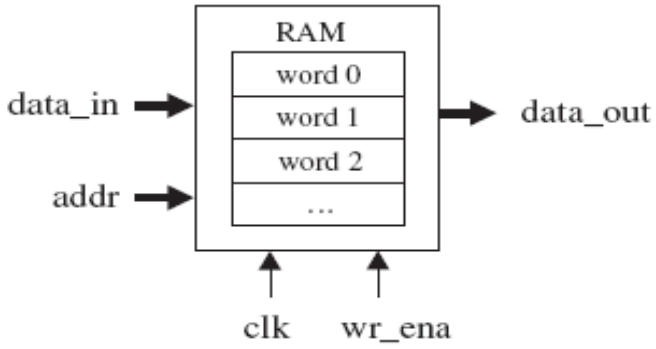
Minimum period: No path found

Minimum input arrival time before clock: No path found

Maximum output required time after clock: No path found

Maximum combinational path delay: 7.985ns

RAM

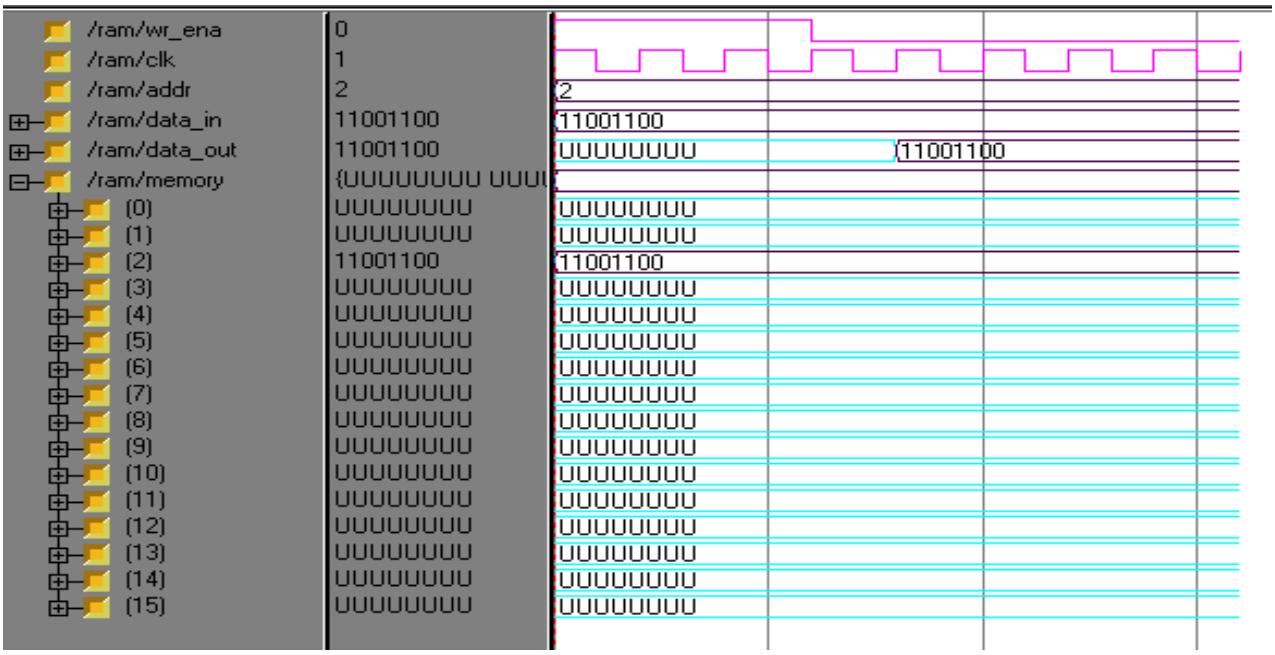


VERILOG SOURCE CODE:

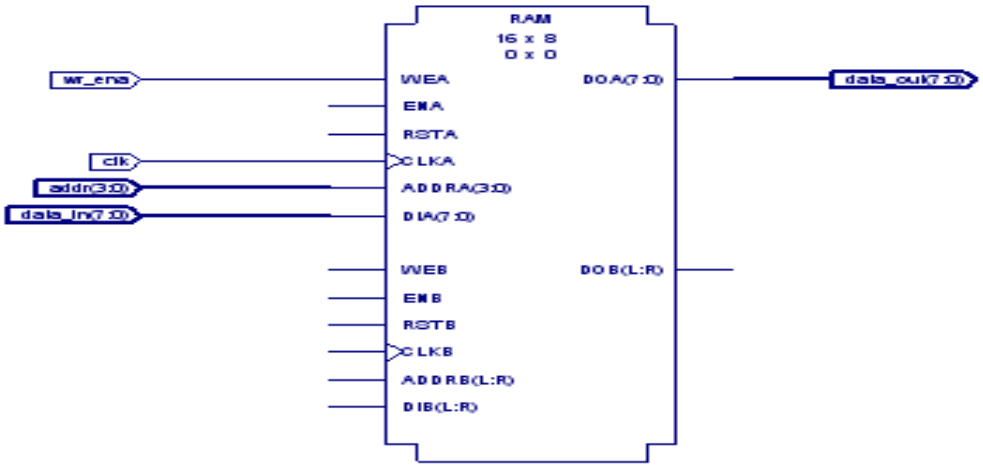
Behavioral Modeling:

```
module ram(clk,wr_en, addrs, data_in, data_out);
   input clk,wr_en;
   input [7:0] data_in;
         input [2:0] addrs;
   output [7:0] data_out;
         reg[7:0]data_out;
         reg[2:0] mem[7:0];
          always@(posedge clk,wr_en,data_in ,addrs)
                        if(clk)
                        begin
                        if(wr_en)
                        mem[addrs]=data_in;
                         else
                        data_out=mem[addrs];
                        end
endmodule
```

Simulation output:



Synthesis RTL Schematic:



Synthesis report:

```
=========================================================================
*                      Final Report                     *
=========================================================================
```

Device utilization summary:

```
---------------------------
Selected Device : 3s400tq144-5


 Number of bonded IOBs:             22  out of     97    22%
 Number of BRAMs:                   1  out of     16     6%
 Number of GCLKs:                   1  out of      8    12%
=========================================================================
```

TIMING REPORT

```
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
     FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
     GENERATED AFTER PLACE-and-ROUTE.
Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 1     |
-----------------------------------+------------------------+-------+
```

Timing Summary:

Speed Grade: -5

Minimum period: No path found

Minimum input arrival time before clock: 1.818ns

Maximum output required time after clock: 7.662ns

Maximum combinational path delay: No path found

RESULT:

Thus the outputs of memory structures are verified by synthesizing and simulating the VERILOG code.