VLSI & E-CAD

IV YEAR I SEM

ANURAG COLLEGE OF ENGINEERING

AUSHAPUR (V), GHATKESAR (M), MEDCHAL (D)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Digital System Design Labs

Developed By

EASI Team

Entuple Technologies

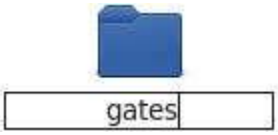
INDEX

1. Tool flow
   - Digital Simulation flow procedure ---------- 3
   - Digital Synthesis flow procedure ---------- 12
2. HDL Code to realize all the logic gates ----------16
3. Design and simulation of adder, serial binary adder ---------- 18
4. Design and simulation of carry lookahead adder---------- 22
5. Design of 2 to 4 decoder ---------- 25
6. Design of 8 to 3 encoder ---------- 26
7. Design of 8 to 1 multiplexer ---------- 28
8. Design of 4 bit binary to gray converter----------30
9. Design of Demultiplexer, Comparator ---------- 32
10. Design of Full adder using 3 modeling styles----------36
11. Design of flip flops: SR, D, JK, T ---------- 39
12. Design 4-bit binary counter ---------- 43
13. Design a N-bit Register of SISO, SIPO ---------- 45
14. Design a N-bit Register of PISO, PIPO ---------- 48
15. Design of sequence detector (Finite State Machine) ----------52
16. Design of 4-Bit Multiplier, Divider ---------- 54
17. Design of ALU perform- ADD, SUB, AND, OR, 1's & 2's compliment, -- 57
    Multiplication, Division

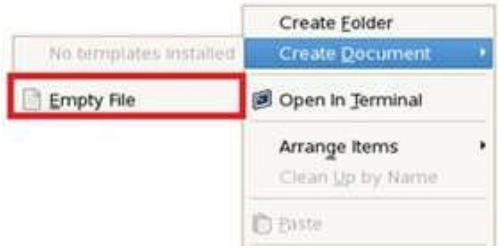
1. Digital Simulation flow

Create a folder in any location. To create a folder right click and select the Create Folder option.
Note: Do not use space in folder name or filename.

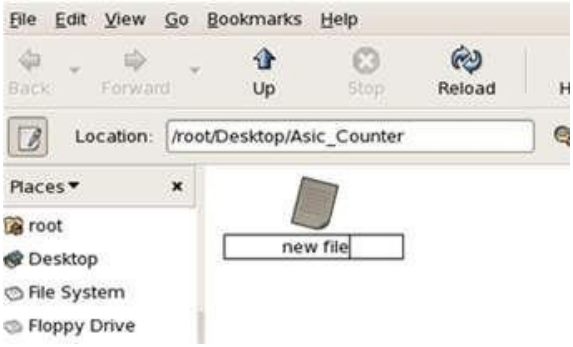
It will create a folder like below and rename it to your requirement.



After creating the folder enter into the location and create a Verilog file specified below or copy the file to the location.



This creates a file shown below



Name the file as gates.v

Double click on gates.v file (or open it with gedit) and type your Verilog code specified below

*gates.v

```
`timescale 1ns/1ps

module gates (input a,b,
              output c,d,e,f,g,h,i);

assign c = ~a; //NOT gate

assign d = a|b; //OR gate

assign e = a&b; //AND gate

assign f = a^b; // EX-OR gate

assign g = ~(a|b); //NOR gate

assign h = ~(a&b); // NAND gate

assign i = ~(a^b); // EX-NOR gate

endmodule
```

Follow similar procedure to create testbench file

*gates.v gates_tb.v

```
`timescale 1ns/1ps

module tb();

reg a,b;
wire c,d,e,f,g,h,i;

gates uut (.a(a),.b(b),.c(c),.d(d),.e(e),.f(f),.g(g),.h(h),.i(i));

initial begin

a=0;b=0;#100;
a=0;b=1;#100;
a=1;b=0;#100;
a=1;b=1;#100;
a=0;b=0;#100;
end
endmodule
```

Save the files and they should look like below window

Right click in the same folder and give open in terminal

Re check the location using “pwd “ command.

Now invoke the Cadence tools using below commands from the terminal

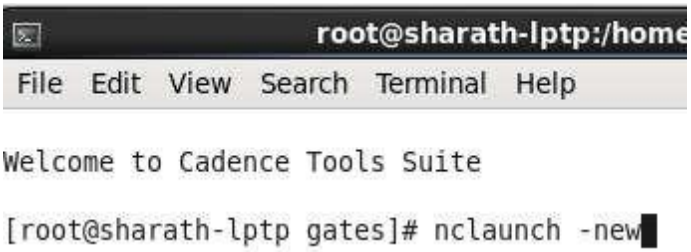
csh

source /cad/cshrc

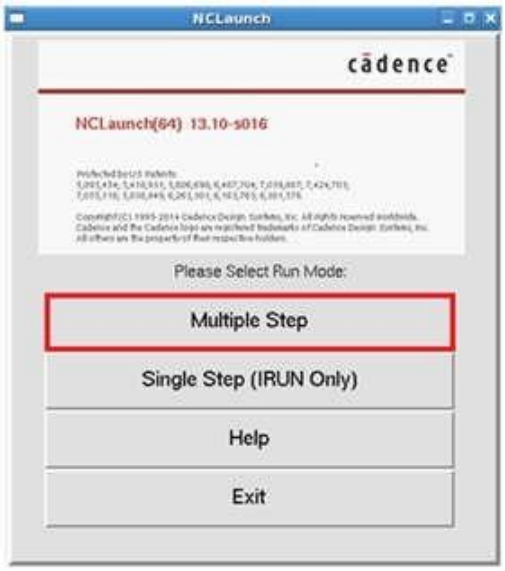
Now a welcome note appears on the terminal (if the commands are properly executed).

Now we will launch the Incisive tool for performing Simulation. Use the below command to invoke the tool.

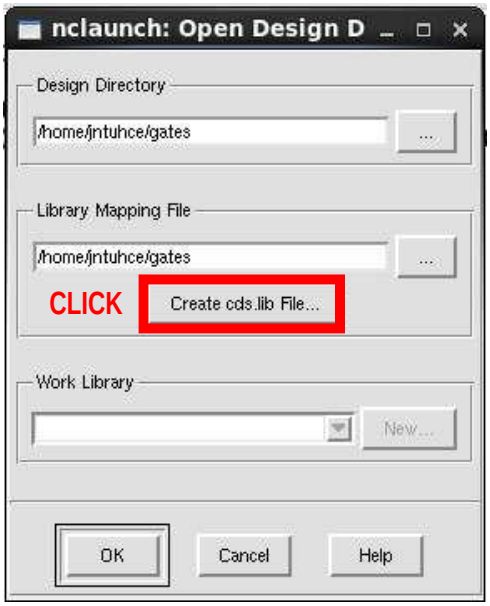
nclaunch -new

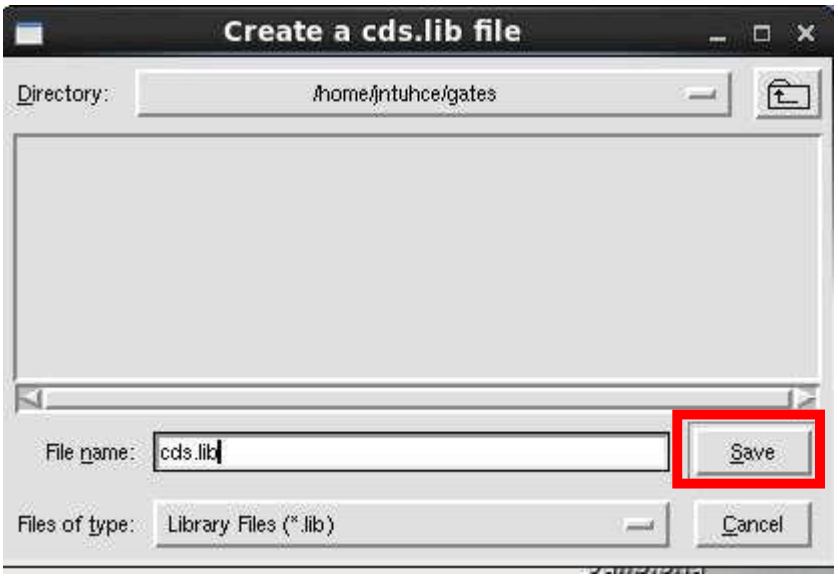


It will invoke the nclaunch window for functional simulation we can compile, elaborate and simulate the design using Multistep option.

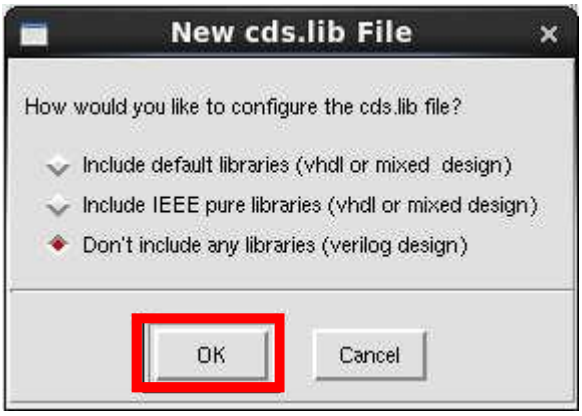


Create the cds.lib and hdl.var files for to Compile, elaborate and simulate the design and test bench, Click on Create cds.lib File option as shown below.

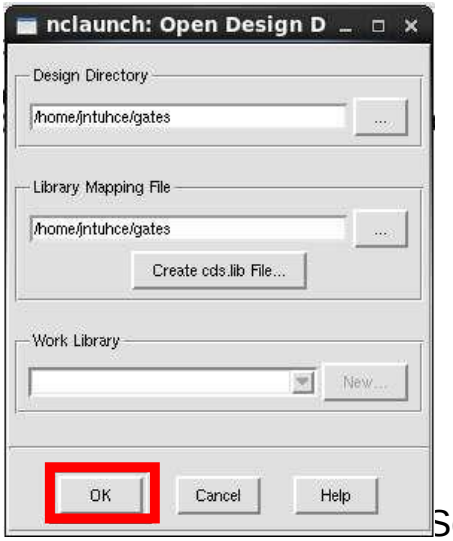




Click save option

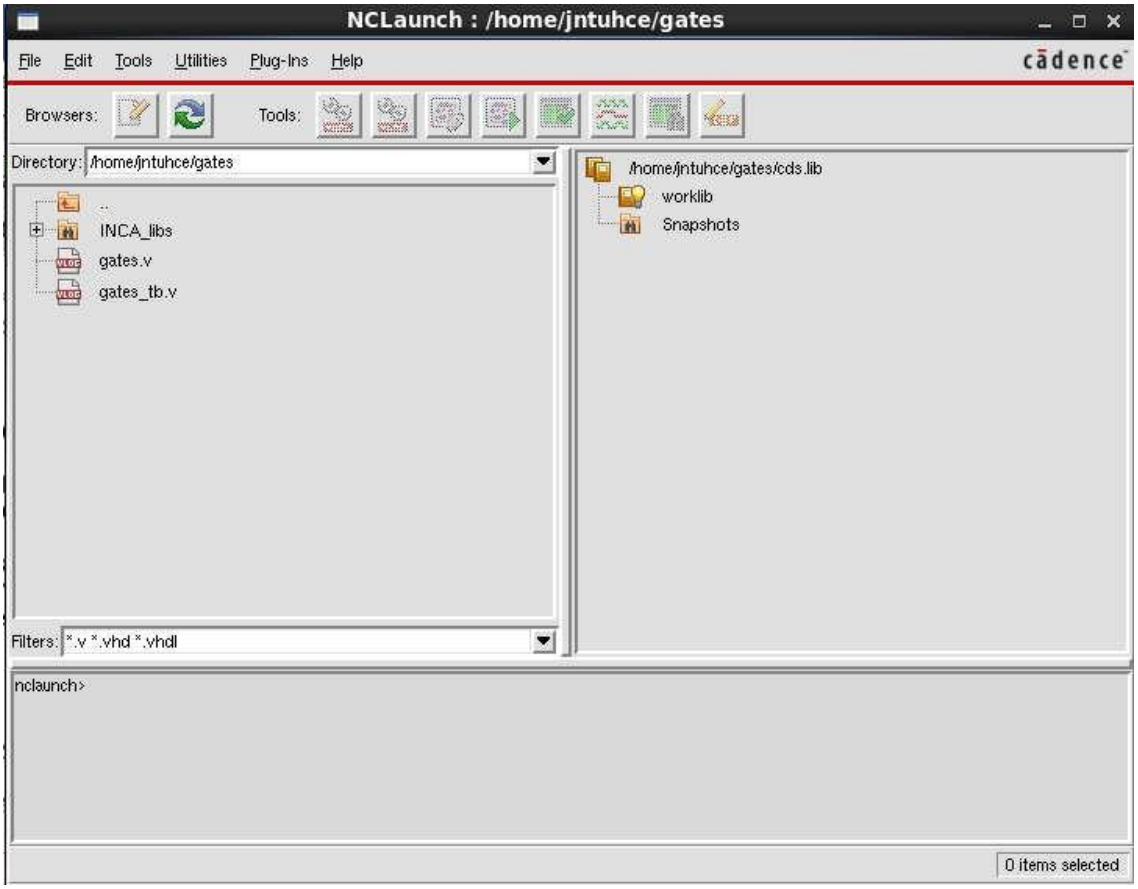


Select OK



Select ok

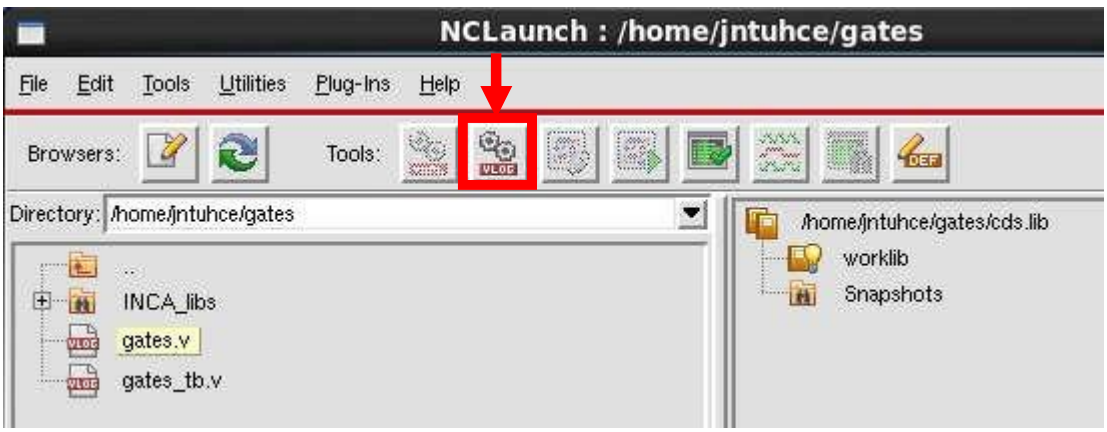
You can see the below window after giving ok



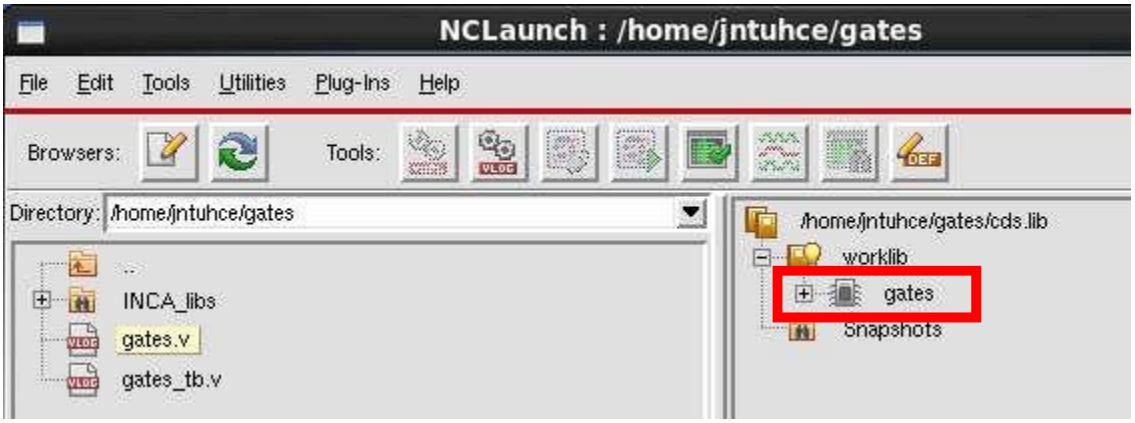
Left side you can see the HDL files, Right side of the window has worklib and snapshots directories listed.

Worklib is the directory where all the compiled codes are stored while Snapshot will have output of elaboration which in turn goes for simulation

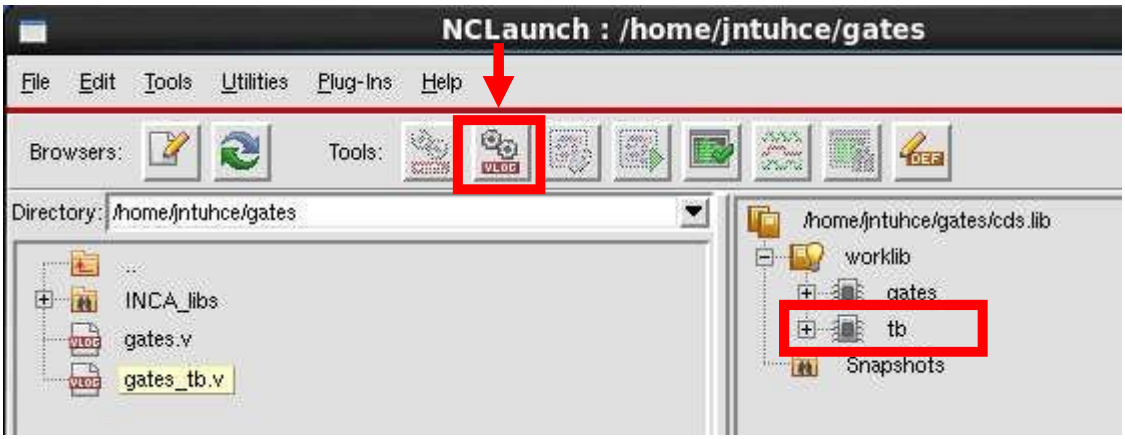
Compilation:



left side select the file and in Tools : launch verilog compiler with current selection will get enable. Click it to compile the code



After compilation it will come under worklib you can see in right side window.

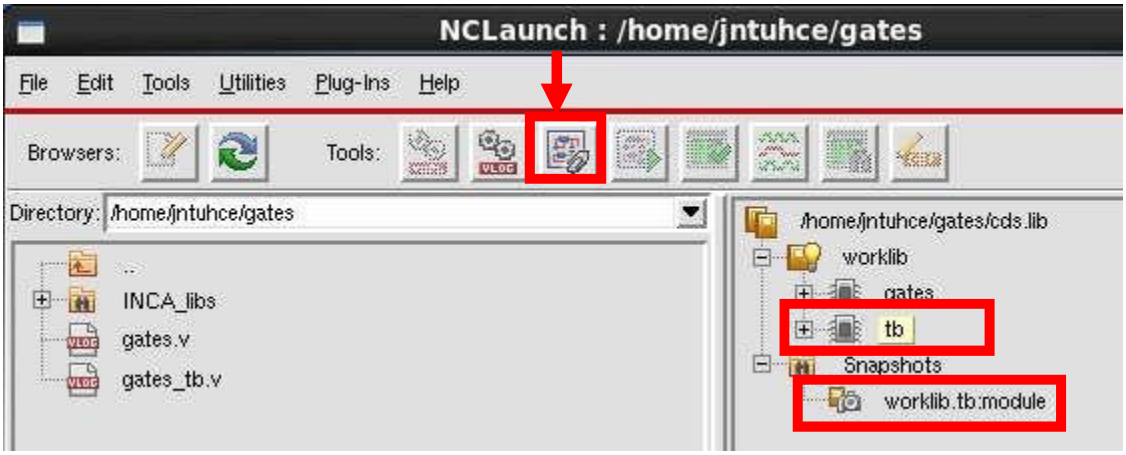


Select the test bench and compile it. It will come under worklib. Under Worklib you can see the module and testbench. Next is to elaborate the design.

Elaboration:

select the testbench file under worklib and in Tools : launch elaborator with current selection will get enable. select the elaborator to elaborate the design.

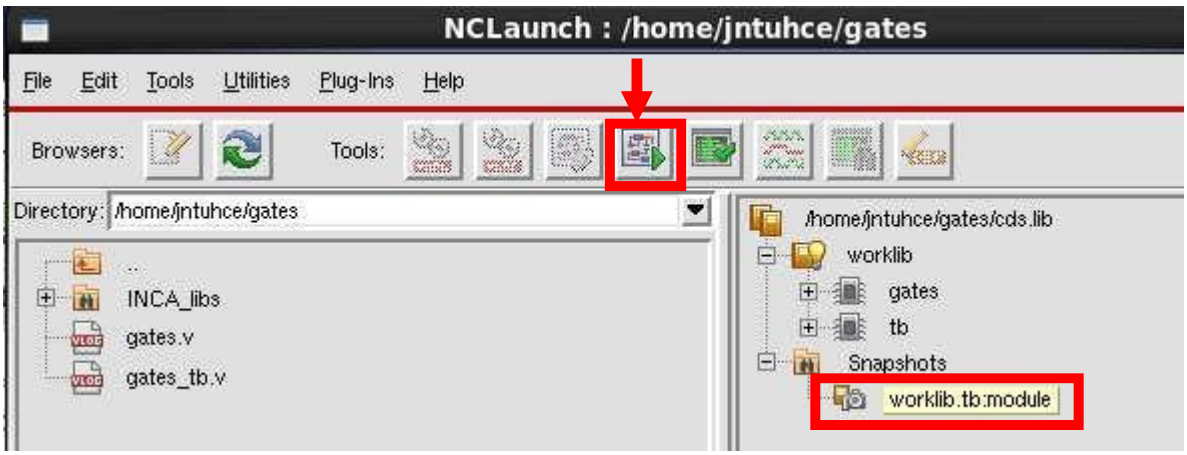
Choose the test bench and elaborate the design



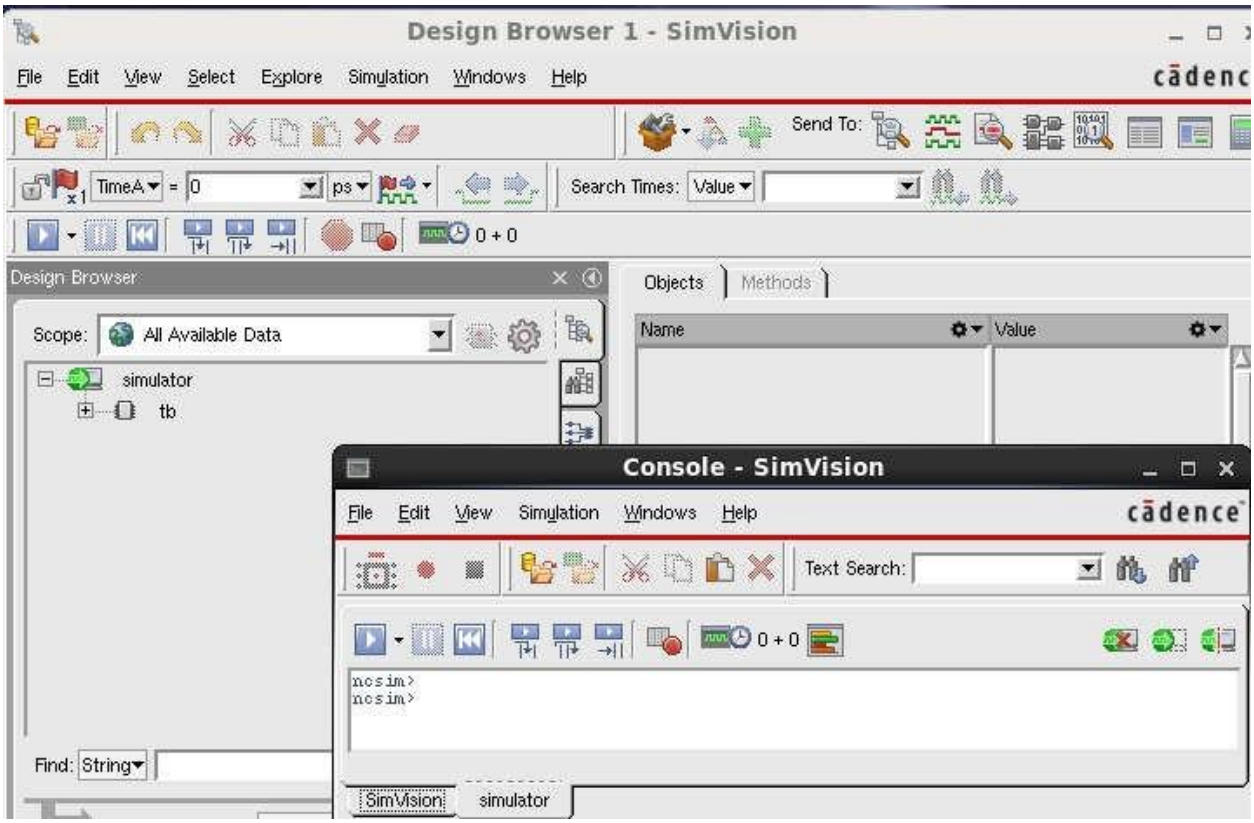
After elaboration the file will come under snapshots.

Simulatiuon:

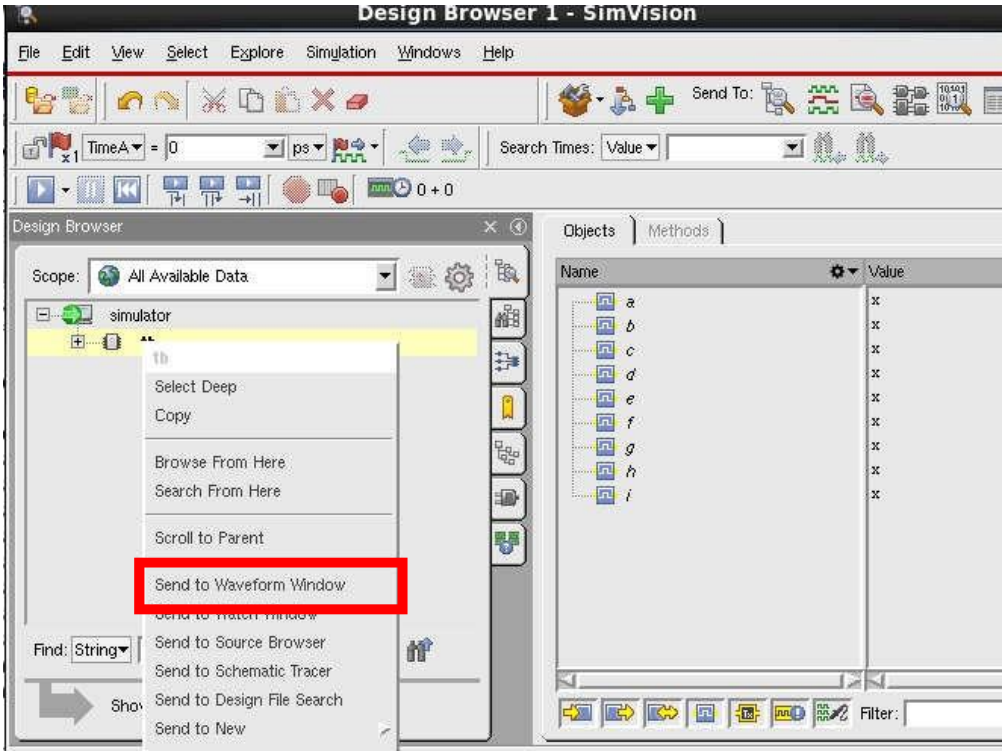
Select the testbench file under snapshot and in Tools : Launch simulator with current selection will get enable.



select simulator to simulate the design. After simulation you will get the two windows like below image.

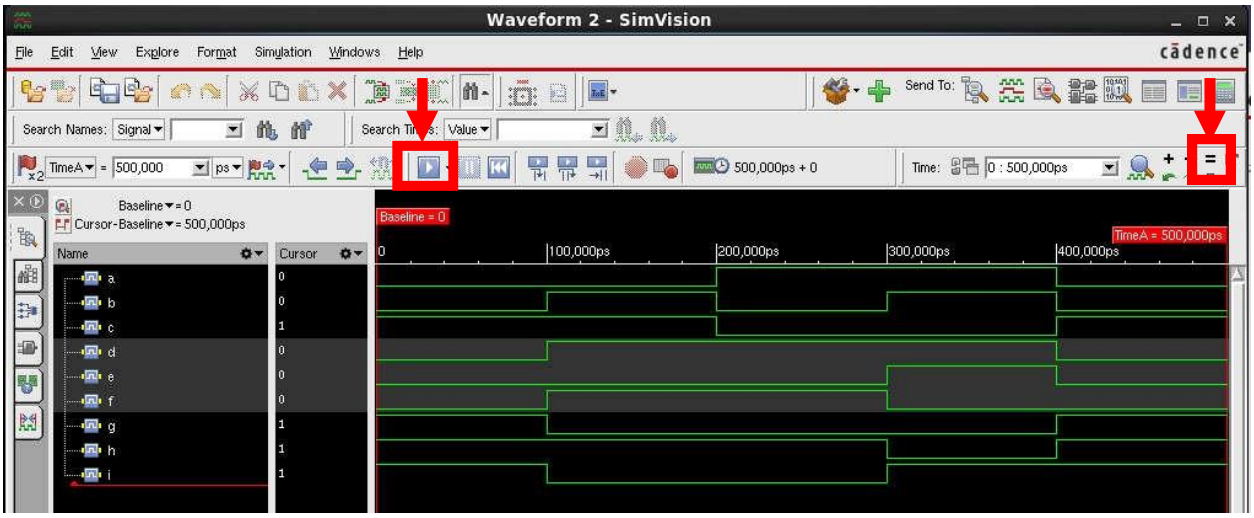


you will get the two windows Design Browser and Simvision .In design browser you can see the test bench in left side window.



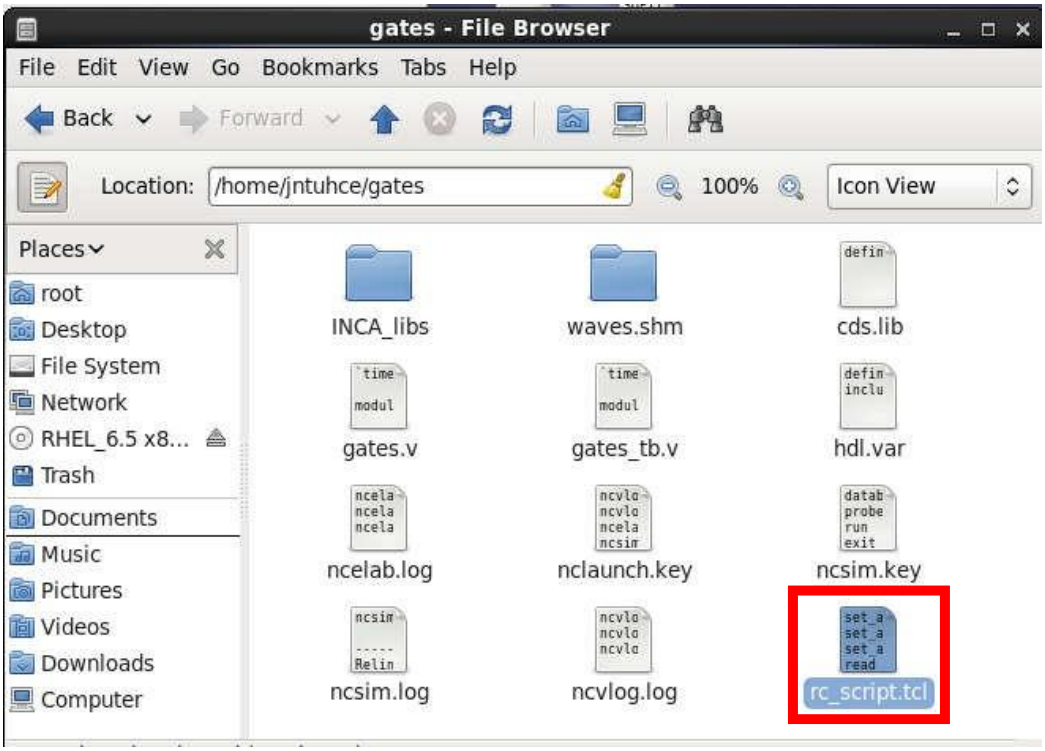
select the test bench for the gates and Right click it. Select the send to waveform window or select the waveform icon

you can see the waveform window after that click the run tool to see the functional simulation for the gates

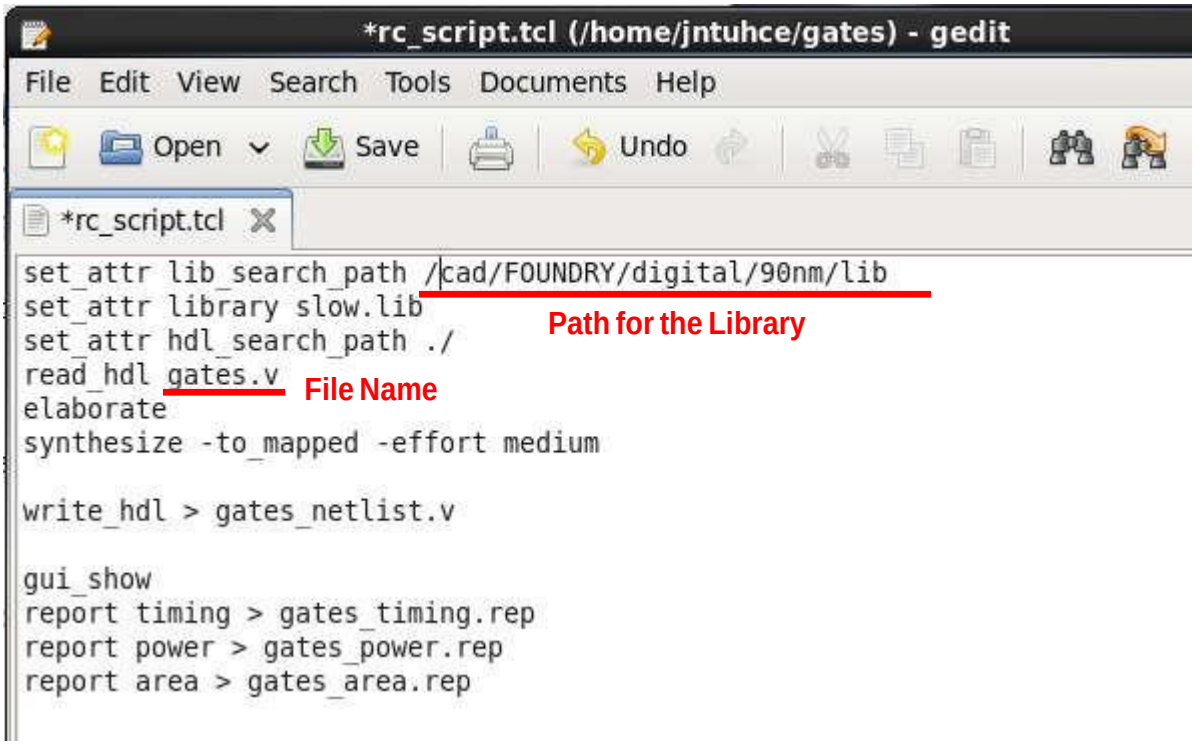


Synthesis Flow:

Synthesis will be done using RTL Compiler. It is a script language called Tool Command Language(TCL)



Inside the run.tcl file we have to mention the commands like below image.



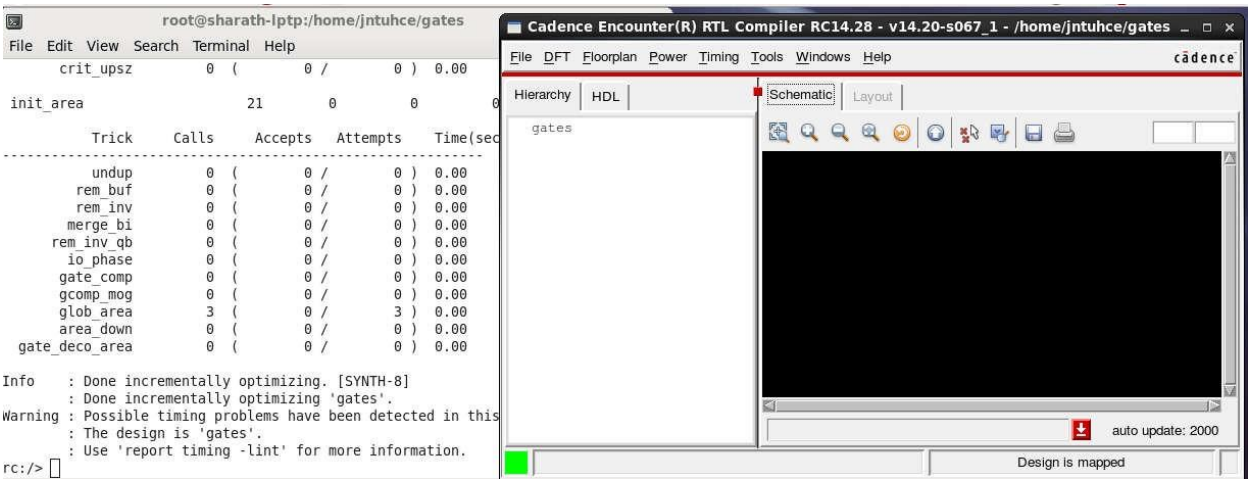
Script file explained below

- Give the path of the library w.r.t to the directory you are in using the command: set_attribute lib_search_path
- Give the path of the RTL files with respect to the directory you are in using the below command: set_attribute hdl_search_path
- Read the library from the directory specified in giving the path for the library files in First line using the command: set_attribute library (slow.lib) is the name of the library file in the directory --library.
- Read the RTL files from the directory specified in the second line. The RTL files are in the directory name : read_hdl gates.v
- Now Elaborate the design using : elaborate command.
- Synthesize the circuit using the command: synthesize -to_mapped -effort medium.

- Timing could be check using : report timing. - Similarly for Gates : report gates.
- Check area using : report area.
- Check Power dissipation using : report power. It will generate the reports
- Write the hdl code in terms of library components for the synthesized circuit using the command: write_hdl > gates_netlist.v

Invoke RTL Compiler by typing below command on your terminal window. The below picture can be seen after typing the above command

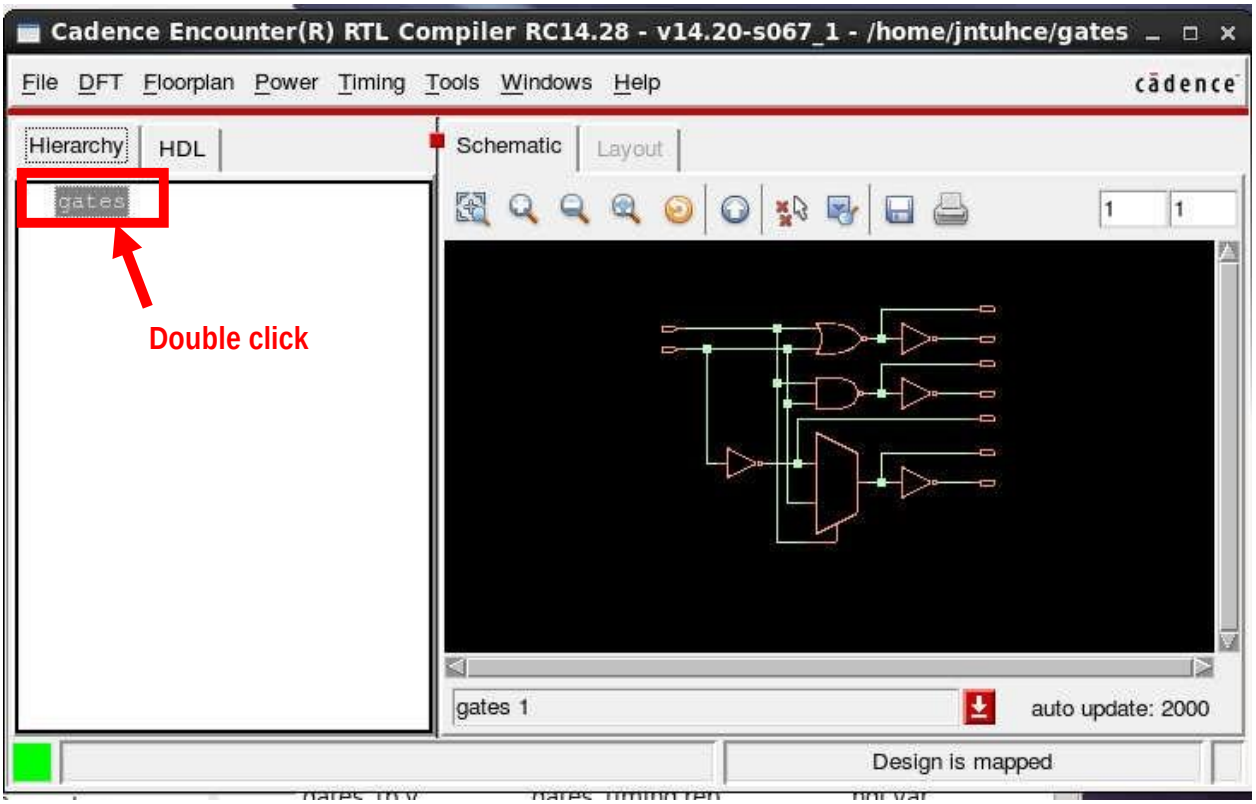
rc -f rc_script.tcl -gui



Now open gates_timing.rep file to observe the timing information

gates_power.rep file to observe the power

gates_area.rep file to observe the area information.



This will show the RTL schematic of the netlist generated.

2. HDL Code to realize all the logic gates

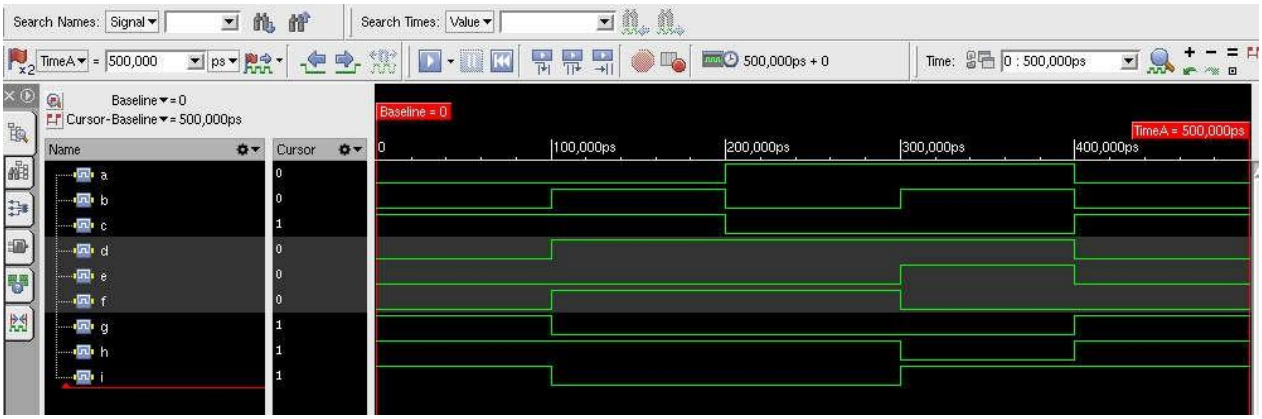
Verilog Design:

```
`timescale 1ns/1ps

module gates (input a,b,
            output c,d,e,f,g,h,i);

assign c = ~a; //NOT gate

assign d = a|b; //OR gate

assign e = a&b; //AND gate

assign f = a^b; // EX-OR gate

assign g = ~(a|b); //NOR gate

assign h = ~(a&b); // NAND gate

assign i = ~(a^b); // EX-NOR gate

endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps

module tb();

reg a,b;
wire c,d,e,f,g,h,i;

gates uut (.a(a),.b(b),.c(c),.d(d),.e(e),.f(f),.g(g),.h(h),.i(i));

initial begin

a=0;b=0;#100;
a=0;b=1;#100;
a=1;b=0;#100;
a=1;b=1;#100;
a=0;b=0;#100;

end

endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib

set_attr library slow.lib

set_attr hdl_search_path ./

read_hdl gates.v

elaborate

synthesize -to_mapped -effort medium

write_hdl >  gates_netlist.v

gui_show

report timing > gates_timing.rep

report power > gates_power.rep

report area > gates_area.rep
```

3. Design and simulation of adder, serial binary adder

Verilog Design:

Adder:

```
`timescale  1ns/1ps
module   full_adder_4bit(
   input  cin,
   input [3:0]in_a,
   input [3:0]in_b,
   output [3:0]sum,
   output cout
    );

    assign    {cout,sum} = in_a + in_b + cin;

endmodule
```

Serial binary adder:

```
`timescale 1ns/1ps

module serial_adder
    (   input clk,reset,  //clock and reset
        input a,b,cin,  //note that cin is used for only first
iteration.
        output reg s,cout  //note that s comes out at every clock
cycle and cout is valid only for last clock cycle.
        );
reg c,flag;

always@(posedge clk or posedge reset)
begin
    if(reset == 1) begin //active high reset
        s = 0;
        cout = c;
        flag = 0;
    end else begin
        if(flag == 0) begin
            c = cin;  //on first iteration after rst assign cin to c.
            flag = 1;  //then make flag 1, so that this if statement
isnt executed any more.
        end
        cout = 0;
        s = a ^ b ^ c;  //SUM
        c = (a & b) | (c & b) | (a & c);  //CARRY
    end
end

endmodule
```

Verilog Testbench:

Adder Testbench:

```
`timescale 1ns/1ps

module   adder_tb ();

reg cin;
reg [3:0]in_a;
reg [3:0]in_b;
wire [3:0]sum;
wire cout;


full_adder_4bit uut (
   .cin(cin),
   .in_a(in_a),
   .in_b(in_b),
   .sum(sum),
   .cout(cout)
    );

initial begin

in_a = 4'h0;
in_b = 4'h0;
cin = 1;
#100;
in_a = 4'h3;
in_b = 4'h4;
cin = 1;
#100;
in_a = 4'h7;
in_b = 4'h8;
cin = 0;
#100;
in_a = 4'h9;
in_b = 4'h9;
cin = 0;
#100;
in_a = 4'hA;
in_b = 4'hB;
cin = 1;
#100;

end
endmodule
```

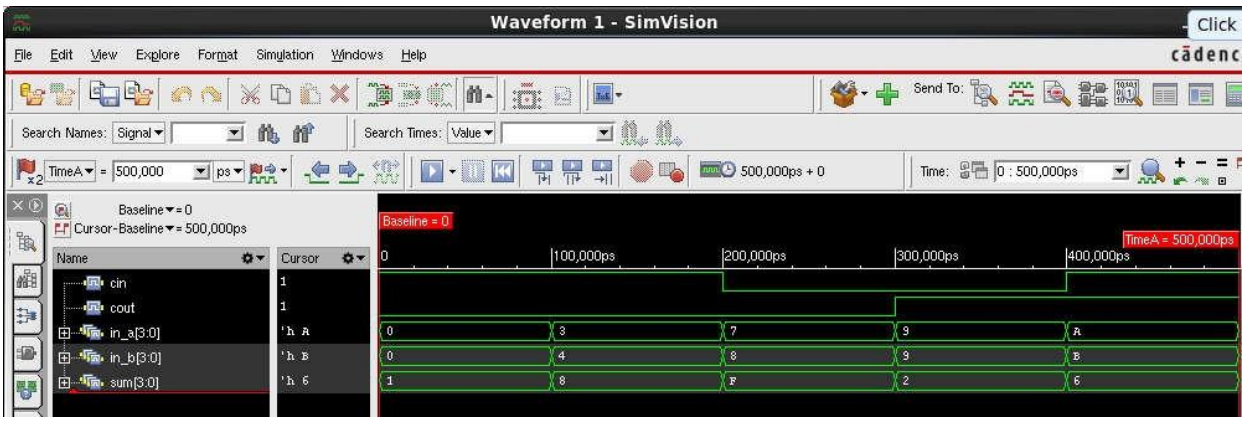
Serial binary adder testbench:

```
`timescale 1ns/1ps
module tb;

    // Inputs
    reg clk;
    reg reset;
    reg  a;
    reg  b;
    reg cin;

    // Outputs
    wire s;
    wire cout;

    // Instantiate the Unit Under Test (UUT)
    serial_adder uut (
        .clk(clk),
        .reset(reset),
        .a(a),
        .b(b),
        .cin(cin),
        .s(s),
        .cout(cout)
    );

//generate clock with 10 ns clock period.
    initial begin
      clk=0;
      forever  #5 clk = ~clk;
    end

    initial begin
        // Initialize Inputs

        reset = 0;
        a = 0;
        b = 0;
        cin = 0;
        reset = 1;
        #100;
        reset = 0;
        //add two 4 bit numbers, 1111 + 1101 = 11101
        a = 1; b = 1; cin = 1;    #10;
        a = 1; b = 0; cin = 0;  #10;
        a = 1; b = 1; cin = 0;  #10;
        a = 1; b = 1; cin = 0;  #10;
        reset = 1;
        #100;
        reset = 0;
```

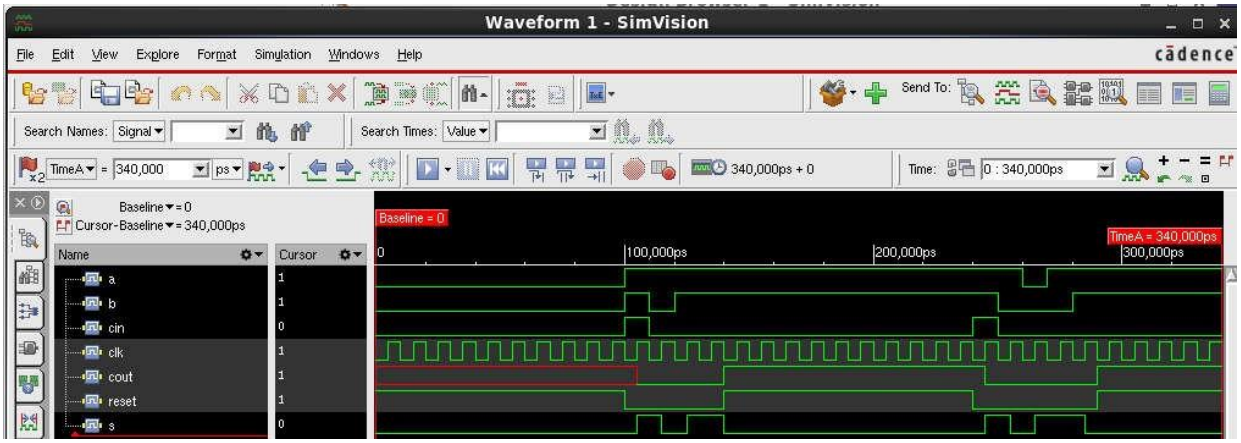
```
        //add two 5 bit numbers, 11011 + 10001 = 101101
        a = 1; b = 1; cin = 1;    #10;
        a = 1; b = 0; cin = 0;  #10;
        a = 0; b = 0; cin = 0;  #10;
        a = 1; b = 0; cin = 0;  #10;
        a = 1; b = 1; cin = 0;  #10;
        reset = 1;
        #50 $finish;

    end

endmodule
```

Simulation Results:

Adder results:



Serial binary adder results:



Synthesis tcl Script:

Adder tcl:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl adder.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > adder_netlist.v

gui_show
report timing > adder_timing.rep
report power > adder_power.rep
report area > adder_area.rep
```

Serial binary adder tcl:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl serial_adder.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > serial_adder_netlist.v

gui_show
report timing > serial_adder_timing.rep
report power > serial_adder_power.rep
report area > serial_adder_area.rep
```

4. Design of Carry lookahead adder

Verilog Design:

```
`timescale 1ns/1ps

module CLA_Adder(a,b,cin,sum,cout);
      input[3:0] a,b;
      input  cin;
      output [3:0] sum;
      output cout;
      wire p0,p1,p2,p3,g0,g1,g2,g3,c1,c2,c3,c4;
      assign p0=(a[0]^b[0]),
                  p1=(a[1]^b[1]),
```

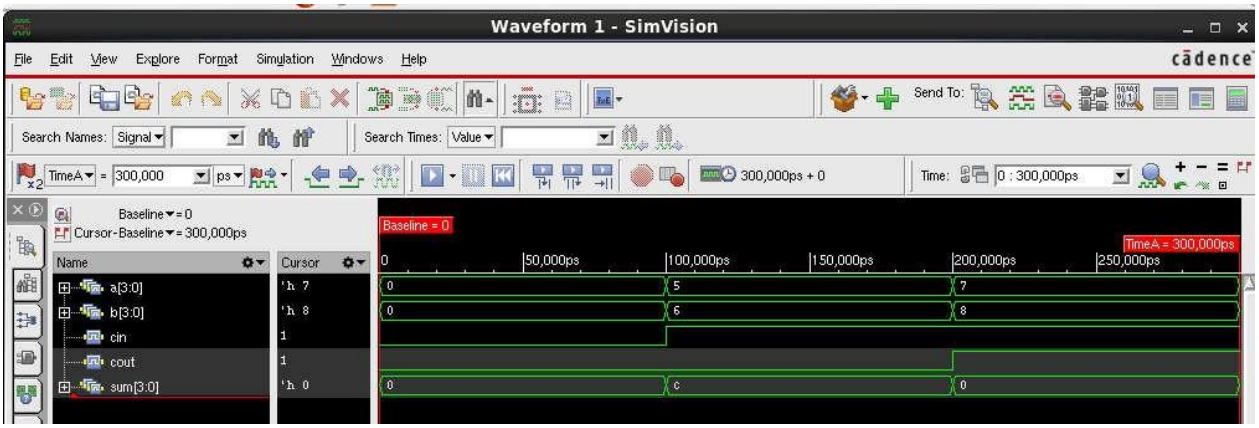
```
                  p2=(a[2]^b[2]),
                  p3=(a[3]^b[3]);
      assign g0=(a[0]&b[0]),
                  g1=(a[1]&b[1]),
                  g2=(a[2]&b[2]),
                  g3=(a[3]&b[3]);
      assign c0=cin,
                  c1=g0|(p0&cin),
                  c2=g1|(p1&g0)|(p1&p0&cin),
                  c3=g2|(p2&g1)|(p2&p1&g0)|(p1&p1&p0&cin),

c4=g3|(p3&g2)|(p3&p2&g1)|(p3&p2&p1&g0)|(p3&p2&p1&p0&cin);
      assign sum[0]=p0^c0,
                  sum[1]=p1^c1,
                  sum[2]=p2^c2,
                  sum[3]=p3^c3;
      assign cout=c4;

endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps
module TestModule;

      // Inputs
      reg [3:0] a;
      reg [3:0] b;
      reg cin;

      //  Outputs
      wire [3:0] sum;
      wire cout;

      // Instantiate the Unit Under Test (UUT)
      CLA_Adder uut (
            .a(a),
            .b(b),
            .cin(cin),
            .sum(sum),
            .cout(cout)
      );

      initial begin
            // Initialize Inputs
            a = 0;
            b = 0;
            cin = 0;

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

```
            a = 5;
            b = 6;
            cin = 1;

            #100;

            a = 7;
            b = 8;
            cin = 1;

            // Wait 100 ns for global reset to finish
            #100 $finish;
      end

endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl cl_adder.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > cl_adder_netlist.v

gui_show
report timing > cl_adder_timing.rep
report power > cl_adder_power.rep
report area > cl_adder_area.rep
```

5. Design of 2 to 4 Decoder

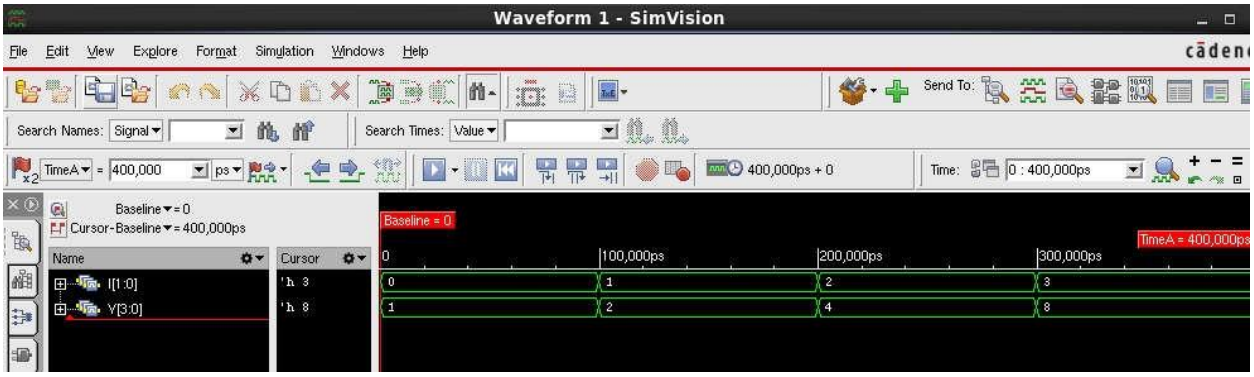
Verilog Design:

```
`timescale 1ns/1ps

module decode(
input [1:0]I,
output reg[3:0] Y
);
always @ (I)
case (I)
2'b00 : Y <= 4'h1;
2'b01 : Y <= 4'h2;
2'b10 : Y <= 4'h4;
2'b11 : Y <= 4'h8;
default : Y <= 4'h0;
endcase
endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps

module decode_tb;

// Inputs
reg [1:0] I;

// Outputs
wire [3:0] Y;
// Instantiate the Unit Under Test (UUT)

decode uut (
.I(I),
.Y(Y) );

initial begin
// Initialize Inputs
I = 2'b00;
// Wait 100 ns for global reset to finish
#100; I = 2'b01;
#100; I = 2'b10;
#100; I = 2'b11;
#100 $finish;

End

endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl decoder.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > decoder_netlist.v
gui_show
report timing > decoder_timing.rep
report power > decoder_power.rep
report area > decoder_area.rep
```

6. Design of 8 to 3 encoder

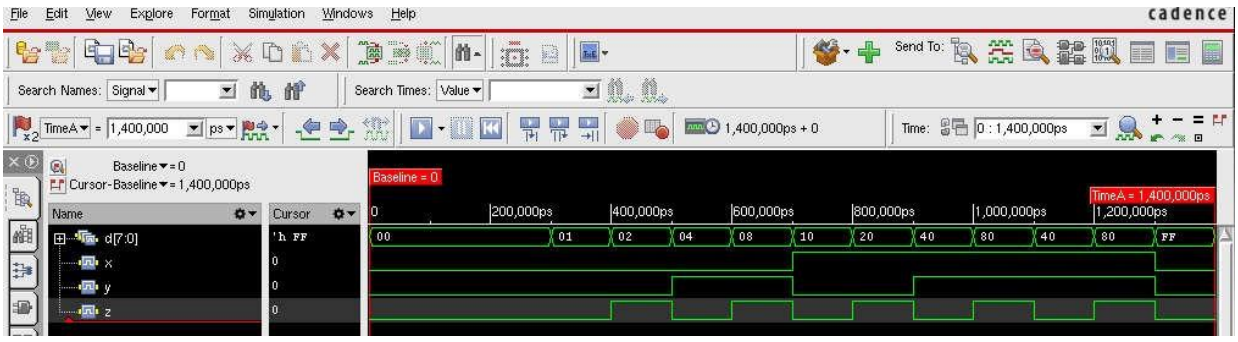
Verilog Design:

```
`timescale 1ns/1ps
module encoder1(
output reg x,y,z,
input [7:0] d);
always @(d)
case (d)
8'h01 : {x,y,z} <= 3'b000;
8'h02 : {x,y,z} <= 3'b001;
8'h04 : {x,y,z} <= 3'b010;
8'h08 : {x,y,z} <= 3'b011;
8'h10 : {x,y,z} <= 3'b100;
8'h20 : {x,y,z} <= 3'b101;
8'h40 : {x,y,z} <= 3'b110;
8'h80 : {x,y,z} <= 3'b111;
default : {x,y,z} <= 3'b000;
endcase
endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps

module encoder_tb;
// Inputs
reg [7:0] d;
// Outputs
wire x; wire y; wire z;
// Instantiate the Unit Under Test (UUT)
encoder1 uut (
.x(x),
.y(y),
.z(z),
.d(d)
);
initial begin
// Initialize Inputs
d = 8'h00;
// Wait 100 ns for global reset to finish
#100; d = 8'h00;
#100; d = 8'h00;
#100; d = 8'h01;
#100; d = 8'h02;
#100; d = 8'h04;
#100; d = 8'h08;
#100; d = 8'h10;
#100; d = 8'h20;
#100; d = 8'h40;
#100; d = 8'h80;
#100; d = 8'h40;
#100; d = 8'h80;
#100; d = 8'hFF;
#100 $finish;
// Add stimulus here
end
endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl encoder.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > encoder_netlist.v

gui_show
report timing > encoder_timing.rep
report power > encoder_power.rep
report area > encoder_area.rep
```

7. Design of 8 to 1 multiplexer

Verilog Design:

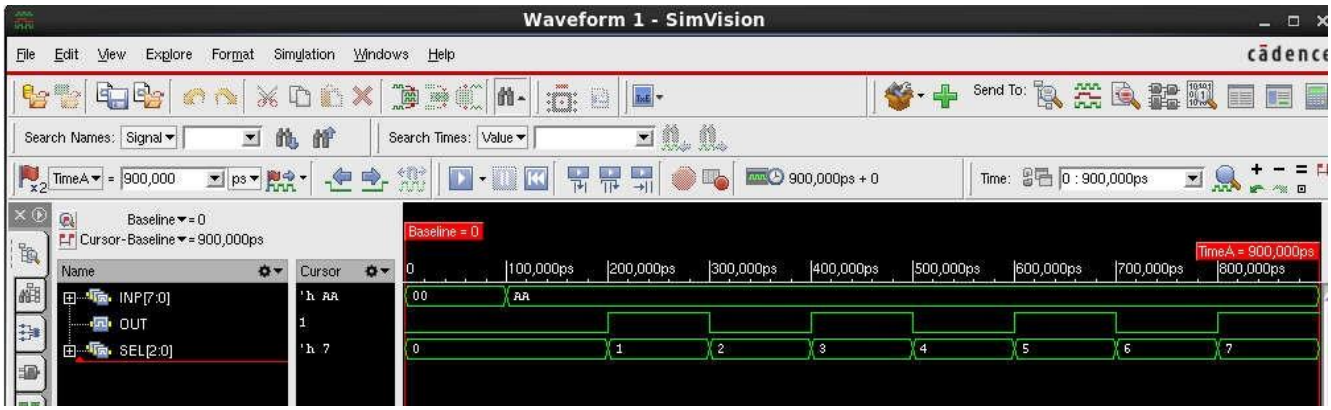
```
`timescale 1ns/1ps
module mux_8_1(
input [7:0] INP ,
input [2:0] SEL,
output reg OUT
);

always @ (SEL)
case (SEL)
3'b000 : OUT <= INP[0];
3'b001 : OUT <= INP[1];
3'b010 : OUT <= INP[2];
3'b011 : OUT <= INP[3];
3'b100 : OUT <= INP[4];
3'b101 : OUT <= INP[5];
3'b110 : OUT <= INP[6];
3'b111 : OUT <= INP[7];
default : OUT <= 0;
endcase
endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps

module tb;
// Inputs
reg [2:0] SEL;
reg [7:0] INP;
// Outputs
```

```
wire OUT;
// Instantiate the Unit Under Test (UUT)
mux_8_1 uut (
.INP(INP),
.SEL(SEL),
.OUT(OUT)
);
initial begin
// Initialize Inputs
INP = 0;
SEL = 0;
// Wait 100 ns for global reset to finish
#100;
SEL = 3'b000;
INP = 8'hAA;
#100;
SEL = 3'b001;
#100;
SEL = 3'b010;
#100;
SEL = 3'b011;
#100;
SEL = 3'b100;
#100;
SEL = 3'b101;
#100;
SEL = 3'b110;
#100;
SEL = 3'b111;
#100 $finish;
// Add stimulus here
end
endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl mux.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > mux_netlist.v

gui_show
report timing > mux_timing.rep
report power > mux_power.rep
report area > mux_area.rep
```

8. Design of 4 bit binary to gray converter

Verilog Design:

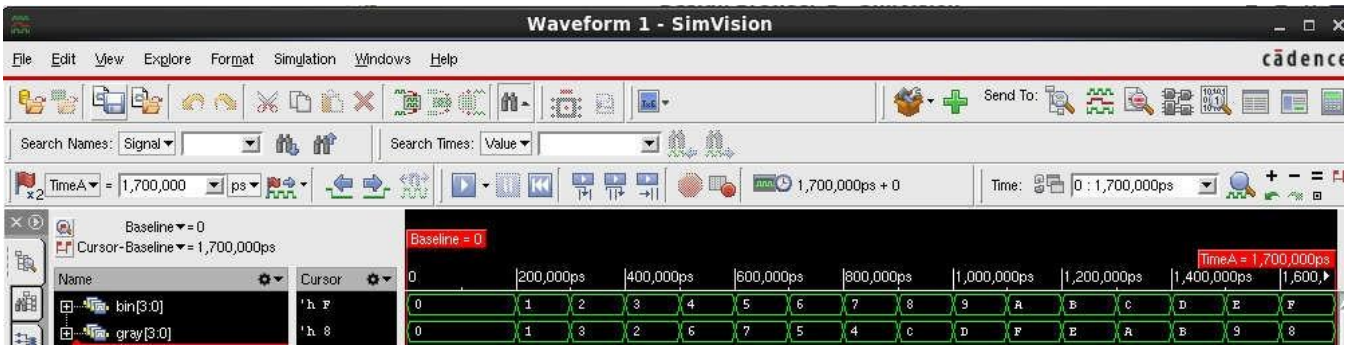
```
`timescale 1ns/1ps

module bin2gray (gray, bin);
output [3:0] gray;
input [3:0] bin;
assign gray = (bin>>1) ^ bin;
endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps

module bin2gray_tb;
// Inputs
reg [3:0] bin;
// Outputs
wire [3:0] gray;
// Instantiate the Unit Under Test (UUT)
bin2gray uut (
.gray(gray),
.bin(bin) );
initial begin
// Initialize Inputs
bin =4'h0;
// Wait 100 ns for global reset to finish
#100; bin =4'h0;
#100; bin =4'h1;
#100; bin =4'h2;
#100; bin =4'h3;
```

```
#100; bin =4'h4;
#100; bin =4'h5;
#100; bin =4'h6;
#100; bin =4'h7;
#100; bin =4'h8;
#100; bin =4'h9;
#100; bin =4'hA;
#100; bin =4'hB;
#100; bin =4'hC;
#100; bin =4'hD;
#100; bin =4'hE;
#100; bin =4'hF;
#100$finish;
end
endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl bin2gray.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > bin2gray_netlist.v

gui_show
report timing > bin2gray_timing.rep
report power > bin2gray_power.rep
report area > bin2gray_area.rep
```

9. Design of Demultiplexer, Comparator

Verilog Design:

Demultiplexer:

```
`timescale 1ns/1ps

module demux1to4(
     Data_in,
     sel,
    Data_out
    );

//list the inputs and their sizes
    input Data_in;
    input [1:0] sel;
//list the outputs and their sizes
    output [3:0] Data_out;

//Internal variables
    reg [3:0]Data_out;

//always block with Data_in and sel in its sensitivity list
    always @(Data_in or sel)
    begin
        case (sel)  //case statement with "sel"
        //multiple statements can be written inside each case.
        //you just have to use 'begin' and 'end' keywords as shown
below.
            2'b00 : begin
                        Data_out[0] = Data_in;
                        Data_out[1] = 0;
                        Data_out[2] = 0;
                        Data_out[3] = 0;
                    end
            2'b01 : begin
                        Data_out[0] = 0;
                        Data_out[1] = Data_in;
                        Data_out[2] = 0;
                        Data_out[3] = 0;
                    end
            2'b10 : begin
                        Data_out[0] = 0;
                        Data_out[1] = 0;
                        Data_out[2] = Data_in;
                        Data_out[3] = 0;
                    end
            2'b11 : begin
                        Data_out[0] = 0;
                        Data_out[1] = 0;
```

```
                        Data_out[2] = 0;
                        Data_out[3] = Data_in;
                    end
        endcase
    end

endmodule
```

Comparator:

```
`timescale 1ns/1ps

module comparator_4_bit (a_gt_b, a_lt_b, a_eq_b, a,b);
input [3 : 0] a,b;
output a_gt_b, a_lt_b, a_eq_b;

     assign a_gt_b = (a > b);
     assign a_lt_b = (a < b);
     assign a_eq_b = (a == b);
endmodule
```

Verilog Testbench:

Demultiplexer:

```
`timescale 1ns/1ps

module tb_demux;

    // Inputs
    reg Data_in;
    reg [1:0] sel;

    // Outputs
    wire [3:0]Data_out;

    // Instantiate the Unit Under Test (UUT)
    demux1to4 uut (
        .Data_in(Data_in),
        .sel(sel),
        .Data_out(Data_out)

    );

    initial begin
        //Apply Inputs
        Data_in = 1;
        sel = 0;    #100;
        sel = 1;    #100;
        sel = 2;    #100;
        sel = 3;    #100;
```

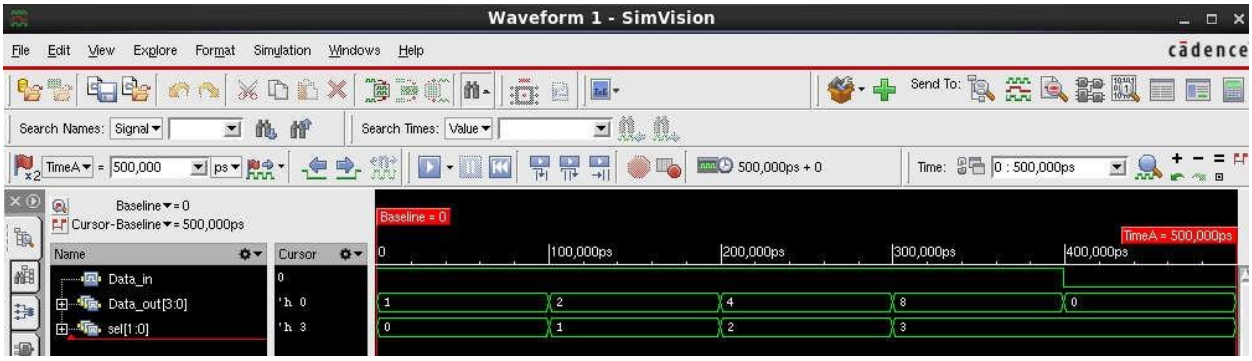
```
        Data_in = 0;
      #100 $finish;
    end

endmodule
```

Comparator:

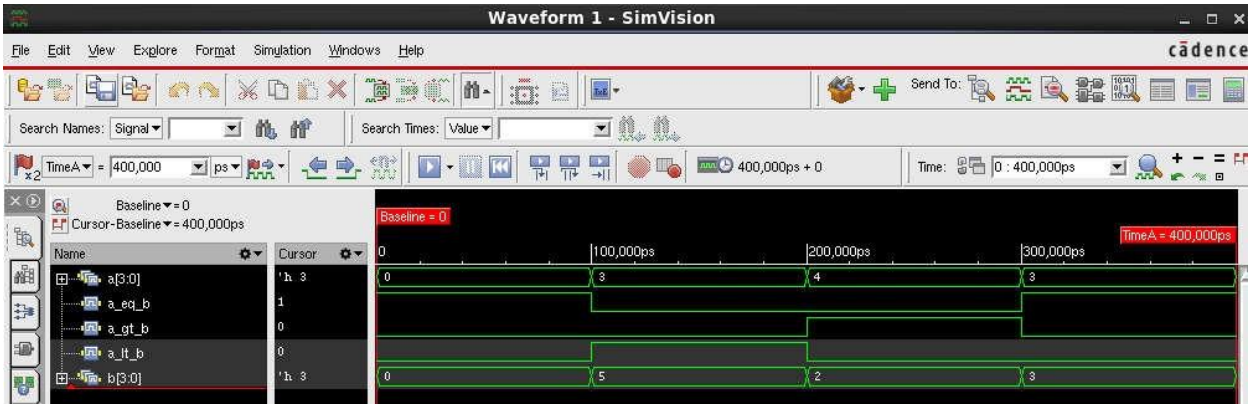
```
`timescale 1ns/1ps

module comparator_tb;
// Inputs
reg [3:0] a;
reg [3:0] b;
// Outputs
wire a_gt_b;
wire a_lt_b;
wire a_eq_b;
// Instantiate the Unit Under Test (UUT)
comparator_4_bit uut (
.a_gt_b(a_gt_b),
.a_lt_b(a_lt_b),
.a_eq_b(a_eq_b),
.a(a),
.b(b)
);
initial begin
// Initialize Inputs
a = 0;
b = 0;
// Wait 100 ns for global reset to finish
#100;
a = 4'h3;
b = 4'h5;
#100;
a = 4'h4;
b = 4'h2;
#100;
a = 4'h3;
b = 4'h3;
#100 $finish;
end
endmodule
```

Simulation Results:

Demultiplexer:



Comparator:



Synthesis tcl Script:

Demultiplexer:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl demux.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > demux_netlist.v

gui_show
report timing > demux_timing.rep
report power > demux_power.rep
report area > demux_area.rep
```

Comparator:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl comp.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > comp_netlist.v

gui_show
report timing > comp_timing.rep
report power > comp_power.rep
report area > comp_area.rep
```

10. Design of Full adder using 3 modeling styles

Verilog Design:

```
`timescale 1ns/1ps

//Dataflow Modeling
module fa(a, b, cin, sum, cout);
input a;
input b;
input cin;
output sum;
output cout;
assign sum = a ^ b ^ cin;
assign cout = (a& b) |(b & cin) |(a & cin);
endmodule

//Behavioral Modeling
module fa1(a, b, cin, sum, cout);
input a;
input b;
input cin;
output sum;
output cout;
reg sum,cout;
always @(a or b or cin)
begin
case ({a,b,cin})
3'b001: begin
sum <= 1'b1;
cout <= 1'b0;
end
3'b010: begin
sum <= 1'b1;
```

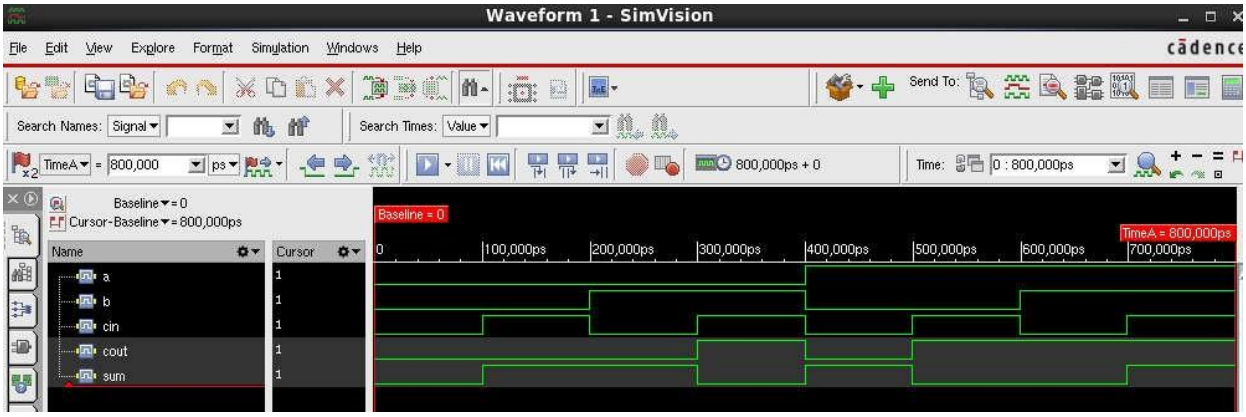
```
cout <= 1'b0;
end
3'b011: begin
sum <= 1'b0;
cout <= 1'b1;
end
3'b100: begin
sum <= 1'b1;
cout <= 1'b0;
end
3'b101: begin
sum <= 1'b0;
cout <= 1'b1;
end
3'b110: begin
sum <= 1'b0;
cout <= 1'b1;
end
3'b111: begin
sum <= 1'b1;
cout <= 1'b1;
end
default: begin
sum <= 1'b0;
cout <= 1'b0;
end
endcase
end
endmodule

//Structural modeling

module ha (a,b,s,co);
input a,b;
output s,co;

assign s= a^b;
assign co= a&b;

endmodule

module fa2(a, b, cin, sum, cout);
input a;
input b;
input cin;
output sum;
output cout;
wire w1, w2, w3;
ha ha_i1 (.a(a),
.b(b),
.s(w1),
```

```
.co(w3)
);
ha ha_i2 (.a(w1),
.b(cin),
.s(sum),
.co(w2)
);
or org_i (cout,w2,w3);
endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps

module fa_tst_v;
reg a;
reg  b;
reg cin;
wire sum;
wire cout;

fa uut (
.a(a),
.b(b),
.cin(cin),
.sum(sum),
.cout(cout)
);

initial begin
a= 0;
b=0;
cin = 0;
#100 a = 0; b = 0;
cin = 1;
#100 a = 0; b = 1;
cin = 0;
#100 a = 0;
b = 1; cin = 1;
#100 a = 1; b = 0;
cin = 0;
#100 a = 1; b = 0;
cin = 1;
#100 a = 1; b = 1;
cin = 0;
#100 a = 1; b = 1; cin = 1;
#100 $finish;

End

endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl fa.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > fa_netlist.v

gui_show
report timing > fa_timing.rep
report power > fa_power.rep
report area > fa_area.rep
```

11. Design of flip flops: SR, D, JK, T

Verilog Design:

```
`timescale 1ns/1ps

module FF_All(
input clk,
// Inputs for Flip-Flops
input J,K,
input R,S,
input D,
input set, reset,
input T_En,
// Outputs for Flip-Flops
output Q_JK, Q_JK_bar, // JK
output Q_RS,Q_RS_bar, // RS
output Q_D,Q_D_bar, // D
output Q_T,Q_T_bar // T
```

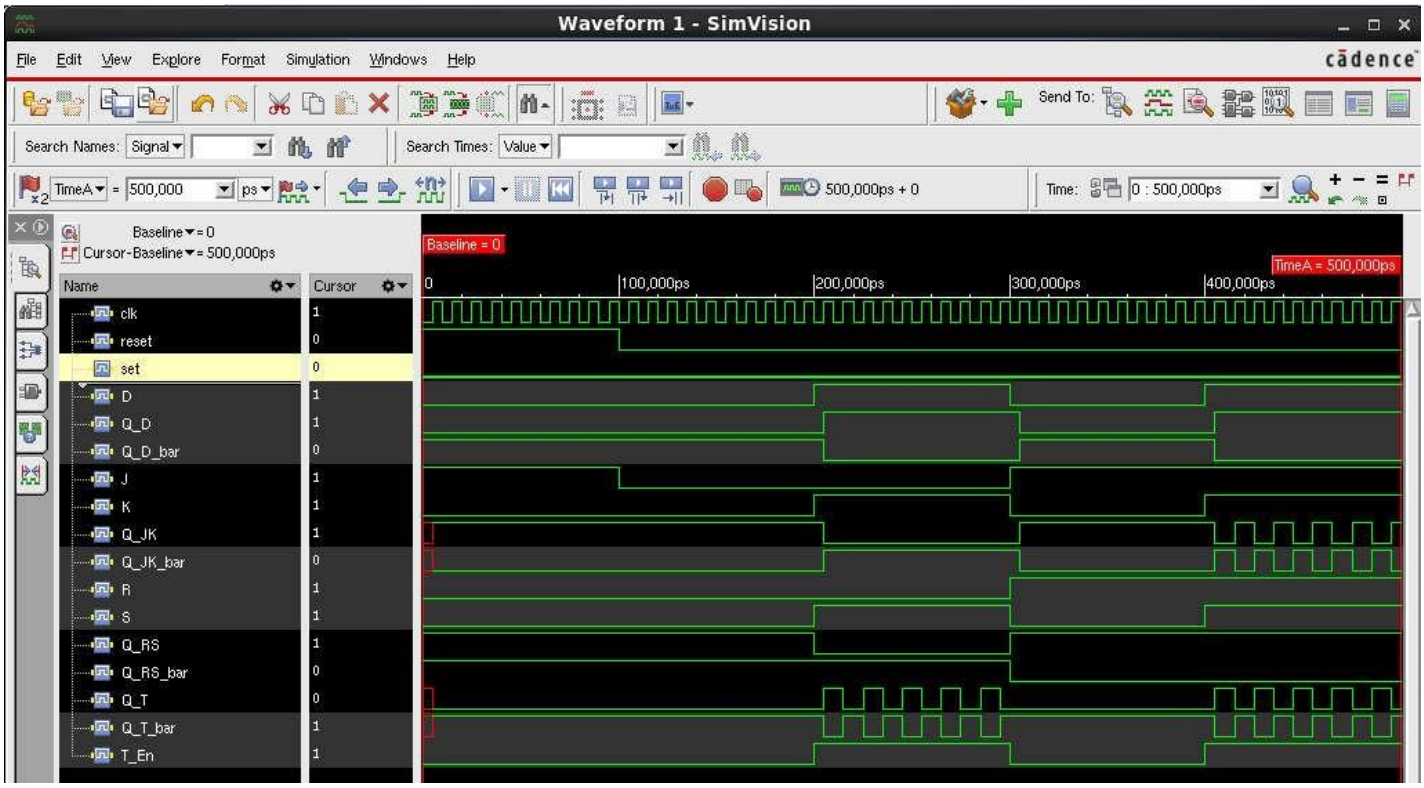
```
);
// Internal Variables
reg q_jk_temp;
reg q_rs_temp,q_rs_bar_temp;
reg q_d_temp;
reg q_t_temp;

// JK Flip-Flop description
always@(posedge clk)
begin
if (J==1'b0 && K==1'b1)
q_jk_temp = 1'b0;
else if(J==1'b1 && K==1'b0)
q_jk_temp = 1'b1;
else if(J==1'b1 && K==1'b1)
q_jk_temp =  ~q_jk_temp;
end
// JK Outputs assignments
assign Q_JK = q_jk_temp;
assign Q_JK_bar = ~q_jk_temp;


// RS Flip-Flop description
always@(R,S)
begin
q_rs_temp = ~(S && q_rs_bar_temp);
q_rs_bar_temp = ~(R && q_rs_temp);
end
assign Q_RS = q_rs_temp;
assign Q_RS_bar = q_rs_bar_temp;

// D Flip-Flop description
always@(posedge clk or posedge reset or posedge set)
begin
if (reset) // Asynchronous reset & set logic
q_d_temp = 1'b0;
else if(set)
q_d_temp = 1'b1;
else
q_d_temp = D;
end
// D-FF Outputs assignments
assign Q_D = q_d_temp;
assign Q_D_bar = ~q_d_temp;

// T Flip-Flop description
always@(posedge  clk)
begin
if (T_En)
q_t_temp = ~q_t_temp;
else
```

```
q_t_temp=0;
end
// T-FF Outputs assignments
assign Q_T = q_t_temp;
assign Q_T_bar = ~q_t_temp;

endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps

module flops_tb ();

reg clk;
// regs for Flip-Flops
reg J,K;
reg R,S;
reg D;
reg set, reset;
reg T_En;
// wires for Flip-Flops
wire Q_JK, Q_JK_bar; // JK
wire Q_RS,Q_RS_bar;// RS
wire Q_D,Q_D_bar; // D
wire Q_T,Q_T_bar; // T


FF_All  uut(.clk(clk), .J(J), .K(K), .R(R),.S(S), .D(D), .set(set),
.reset(reset), .T_En(T_En),
           .Q_JK(Q_JK), .Q_JK_bar(Q_JK_bar),
.Q_RS(Q_RS),.Q_RS_bar(Q_RS_bar),.Q_D(Q_D),.Q_D_bar(Q_D_bar),.Q_T(Q_T),
.Q_T_bar(Q_T_bar));


initial begin
clk=0;
forever #5 clk = ~clk;
end

initial begin
J=1; K=0; R=0; S=0;D=0; set=0; reset=1;T_En=0;
#100; J=0; K=0; R=0; S=0;D=0; set=0; reset=0;T_En=0;
#100; J=0; K=1; R=0; S=1;D=1; set=0; reset=0;T_En=1;
#100; J=1; K=0; R=1; S=0;D=0; set=0; reset=0;T_En=0;
#100; J=1; K=1; R=1; S=1;D=1; set=0; reset=0;T_En=1;
#100 $finish;
end
endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl flops.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > flops_netlist.v

gui_show
report timing > flops_timing.rep
report power > flops_power.rep
report area > flops_area.rep
```

12. Design 4-bit binary counter

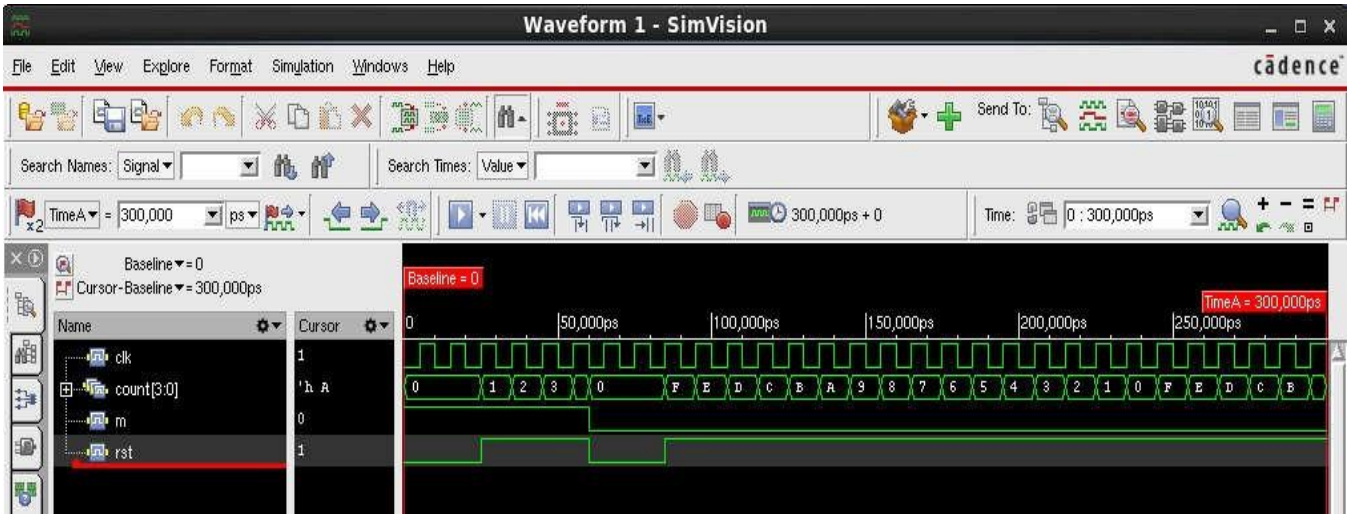
Verilog Design:

```
`timescale 1ns/1ps
module counter(clk,m,rst,count);
input clk,m,rst;
output reg [3:0] count;
always@(posedge clk or negedge rst)
begin
if(!rst)
count=0;
else if(m)
count=count+1;
else
count=count-1;
end
endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps
module counter_test;
reg clk,  rst,m;
wire [3:0] count;
initial
begin
clk=0;
rst=0;#25;
rst=1;
end
initial
begin
m=1;
#60 m=0;
rst=0;#25;
rst=1;
#50 m=0;
end

counter counter1(clk,m,rst, count);

always #5 clk=~clk;


initial
#300 $finish;

endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl counter.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > counter_netlist.v

gui_show
report timing > counter_timing.rep
report power > counter_power.rep
report area > counter_area.rep
```

13. Design a N-bit Register of SISO, SIPO

Verilog Design:

SISO

```
`timescale 1ns/1ps

module siso(clk,rst,a,q);
input a;
input clk,rst;
output q;
reg q;
always@(posedge clk,posedge rst)
begin
if(rst==1'b1)
q<=1'b0;
else
q<=a;
end
endmodule
```

SIPO:

```
`timescale  1ns/1ps
module sipo(a,clk,rst,q);
input clk,rst;
input a;
output [3:0]q;
wire [3:0]q;
reg [3:0]temp;
always@(posedge clk,posedge rst)
begin
if(rst==1'b1)
temp<=4'b0000;
else
begin
temp<=temp<<1'b1;
temp[0]<=a;
end
end
assign q=temp;
endmodule
```

Verilog Testbench:

SISO:

```
`timescale 1ns/1ps
module siso_tb();
reg clk,rst;
```

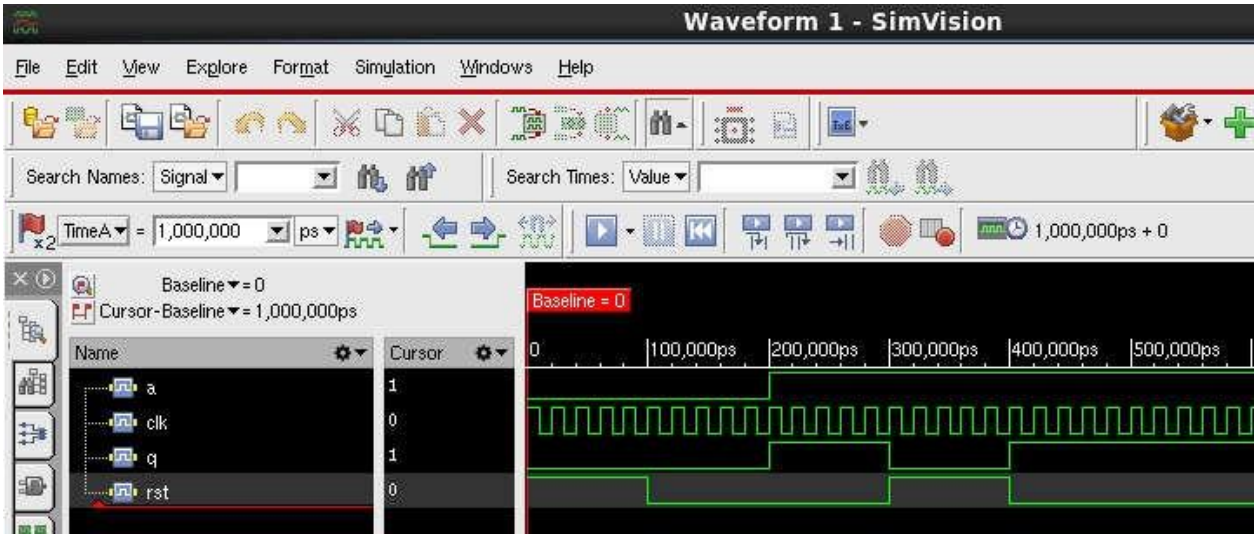
```
reg a;
wire q;

siso  uut(clk,rst,a,q);

initial
clk=1'b1;
always #10 clk=~clk;
initial begin
a=1'b0;rst=1'b1;
#100 rst=1'b0;
#100 a=1'b1;
#100 rst=1'b1;
#100 rst=1'b0;
end
initial
#1000 $stop;
endmodule
```

SIPO:

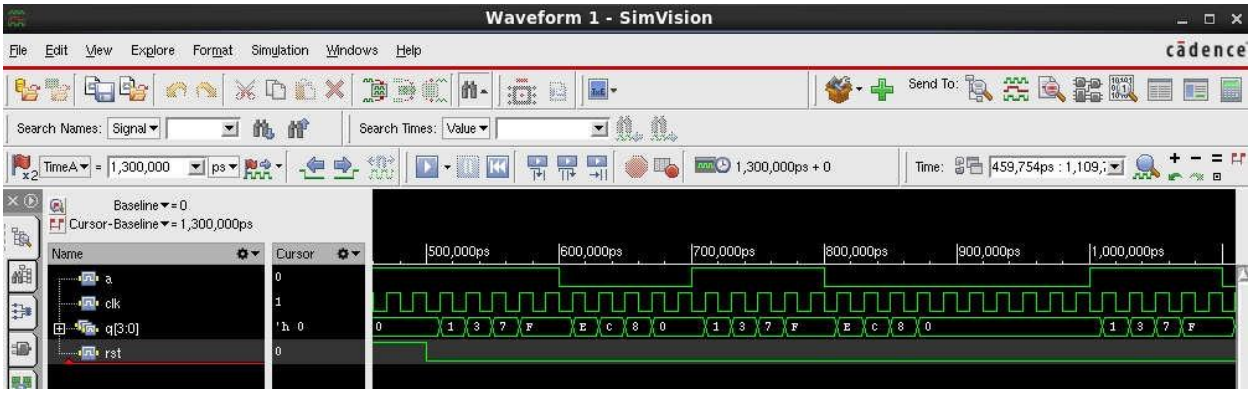
```
`timescale 1ns/1ps
module sipo_tb();
reg clk,rst;
reg a;
wire [3:0]q;
sipo  uut(a,clk,rst,q);

initial
clk=1'b0;
always #10 clk=~clk;
initial begin
rst=1'b1; a=1'b1;
#500 rst=1'b0;
#100 a=1'b0;
#100 a=1'b1;
#100 a=1'b0;
#100 a=1'b0;
#100 a=1'b1;
#100 a=1'b0;
end
initial
#1300 $stop;
 endmodule
```

Simulation Results:

SISO:



SIPO:



Synthesis tcl Script:

SISO:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl siso.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > siso_netlist.v
gui_show
report timing > siso_timing.rep
report power > siso_power.rep
report area > siso_area.rep
```

SIPO:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl sipo.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > sipo_netlist.v

gui_show
report timing > sipo_timing.rep
report power > sipo_power.rep
report area > sipo_area.rep
```

14. Design a N-bit Register of PISO, PIPO

Verilog Design:

PISO:

```
`timescale 1ns/1ps

module piso(clk,rst,a,q);
input clk,rst;
input [3:0]a;
output  q;
reg q;
reg [3:0]temp;
always@(posedge clk,posedge rst)
begin
if(rst==1'b1)
begin
q<=1'b0;
temp<=a;
end
else
begin
q<=temp[0];
temp <= temp>>1'b1;
end
end
endmodule
```

PIPO:

```
`timescale 1ns/1ps
```

```
module pipo(clk,rst,a,q);
input clk,rst;
input[3:0]a;
output[3:0]q;
reg[3:0]q;

always@(posedge clk,posedge rst)
begin
if (rst==1'b1)
q<=4'b0000;
else
q<=a;
end
endmodule
```

Verilog Testbench:

PISO:

```
`timescale 1ns/1ps

module piso_tb();

reg clk,rst;
reg [3:0]a;
wire q;
piso uut(clk,rst,a,q);
initial
clk=1'b1;
always #10 clk=~clk;
initial begin
rst=1'b1; a=4'b1101;
#300 rst=1'b0;
#200 rst=1'b1;
#200 rst=1'b0;
end
initial
#1000 $stop;
endmodule
```

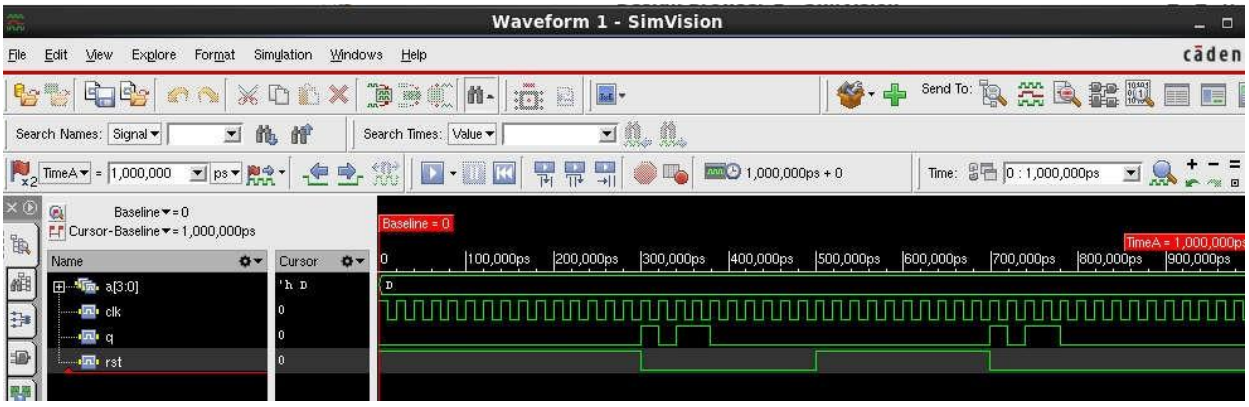
PIPO:

```
`timescale 1ns/1ps

module pipo_tb();

reg clk,rst;
reg[3:0]a;
wire[3:0]q;

pipo uut(clk,rst,a,q);
initial
```

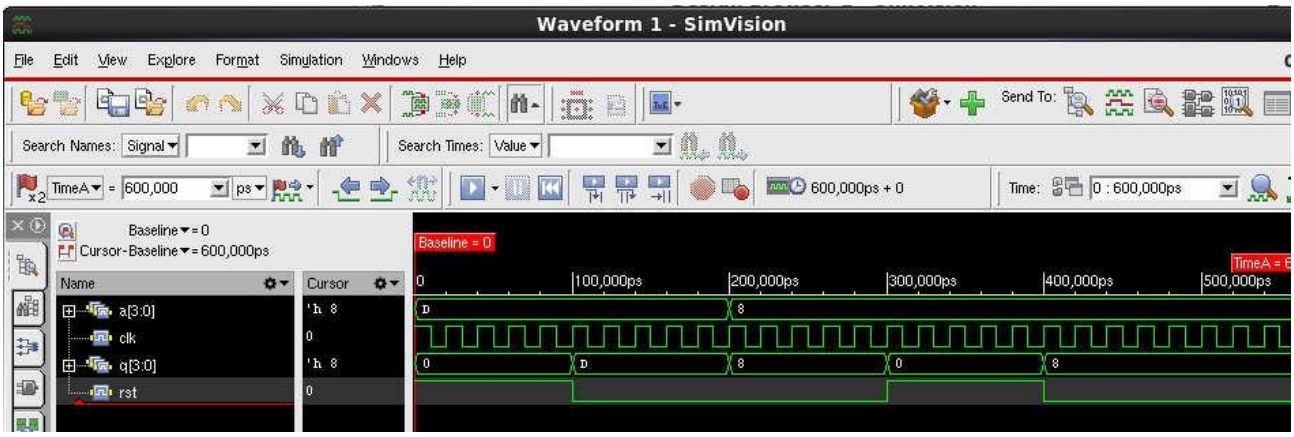
```
clk='b1;
always #10 clk=~clk;
initial begin
     a=4'b1101;rst=1'b1;
#100 rst=1'b0;
#100 a=4'b1000;
#100 rst=1'b1;
#100 rst=1'b0;
end
initial
#600 $stop;

endmodule
```

Simulation Results:

PISO:



PIPO:



Synthesis tcl Script:

PISO:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl piso.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > piso_netlist.v

gui_show
report timing > piso_timing.rep
report power > piso_power.rep
report area > piso_area.rep
```

PIPO:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl pipo.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > pipo_netlist.v

gui_show
report timing > pipo_timing.rep
report power > pipo_power.rep
report area > pipo_area.rep
```

15. Design of sequence detector (Finite State Machine)

Verilog Design:

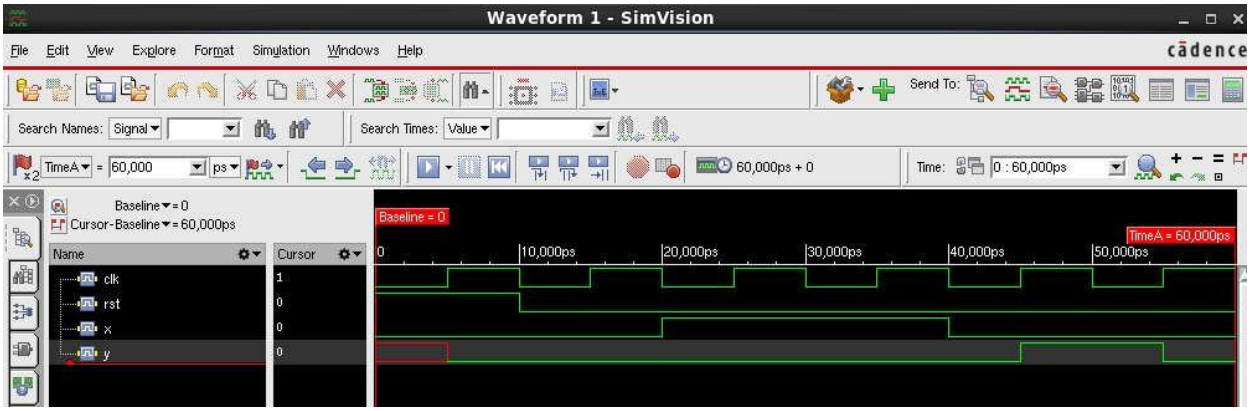
```
`timescale 1ns/1ps
module seq_det(x,clk,rst,y);
input x, clk, rst;
output y;
reg [2:0] state;
reg temp;
parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, S3 = 3'b011, S4 =
3'b100;
always @(posedge clk)
begin
if (rst)
state<=S0;
else
case (state)
S0: if(x)
state<=S3;
else
state<=S1;

S1:if(x)
state<=S2;
else
state<=S1;

S2:if(x)
state<=S3;
else
state<=S4;

S3:if(x)
state<=S3;
else
state<=S4;

S4:if(x)
state<=S2;
else
state<=S1;
endcase

end
always @(state)
begin
case (state)
S0:temp<=1'b0;

S1:temp<=1'b0;
```

```
S2:temp<=1'b0;

S3:temp<=1'b0;

S4:temp<=1'b1;
endcase
end
assign y = temp;
endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps
module seq_det_tb();

reg x, clk, rst;
wire y;

seq_det uut(x,clk,rst,y);

initial begin
clk=0;
forever #5 clk=~clk;
end
initial begin
rst=1; x=0;
#10;rst=0; x=0;
#10;rst=0; x=1;
#10;rst=0; x=1;
#10;rst=0; x=0;
#10;rst=0; x=0;
#10 $finish;
end
endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl fsm.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > fsm_netlist.v

gui_show
report timing > fsm_timing.rep
report power > fsm_power.rep
report area > fsm_area.rep
```

16. Design of 4-Bit Multiplier, Divider

Verilog Design:

Multiplier:

```
`timescale 1ns/1ps

module multipliermod(a, b, out);

input [3:0] a;
input [3:0] b;

output [7:0] out;

assign out=(a*b);

endmodule
```

Divider:

```
`timescale 1ns/1ps

module division(a, b, out);

input [3:0] a;
input [3:0] b;

output [3:0] out;

assign out=(a/b);

endmodule
```

Verilog Testbench:

Multiplier:

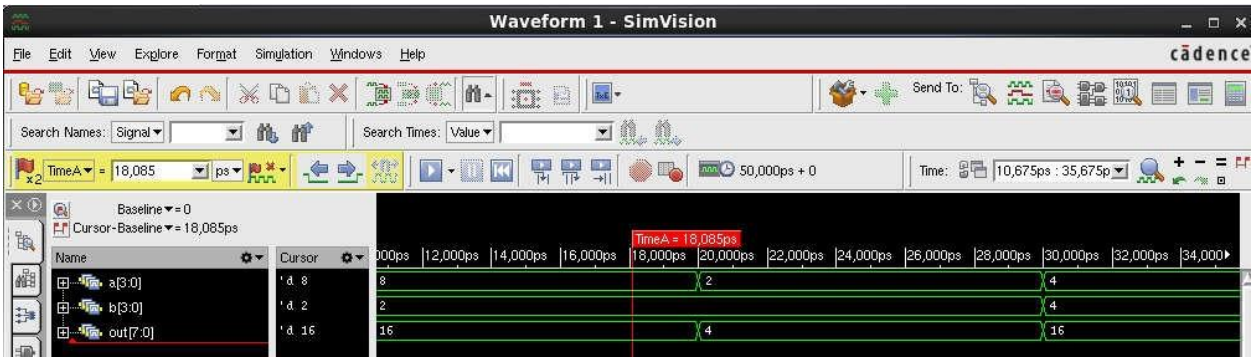
```
`timescale 1ns/1ps

module multiplier_tb;

reg [3:0] a;
reg [3:0] b;

wire [7:0] out;

multipliermod uut (.a(a),.b(b),.out(out) );

initial begin

#10 a=4'b1000;b=4'b0010;
#10 a=4'b0010;b=4'b0010;
#10 a=4'b0100;b=4'b0100;
#10 a=4'b1000;b=4'b0001;
#10;$finish;

end

endmodule
```

Divider:

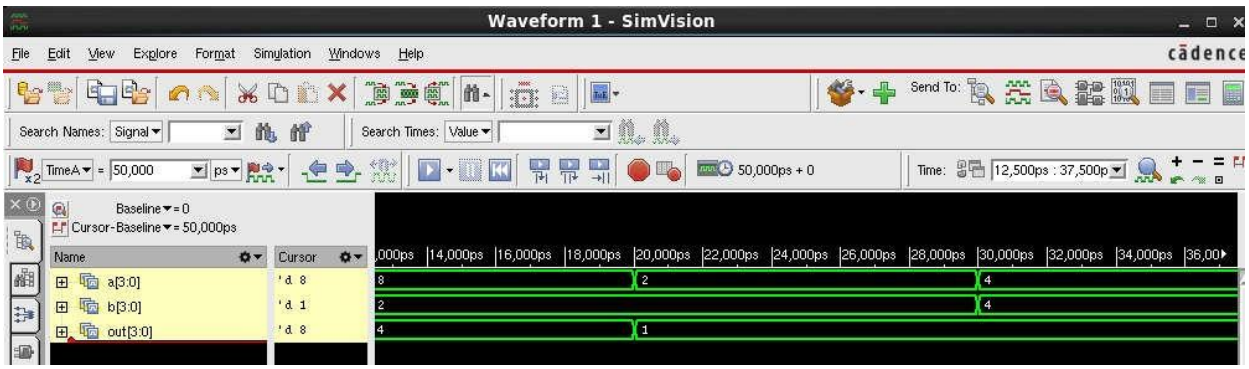
```
`timescale 1ns/1ps

module division_tb;

reg [3:0] a;
reg [3:0] b;

wire [3:0] out;

division uut (.a(a),.b(b),.out(out) );

initial begin

#10 a=4'b1000;b=4'b0010;
#10 a=4'b0010;b=4'b0010;
#10 a=4'b0100;b=4'b0100;
#10 a=4'b1000;b=4'b0001;
#10;$finish;

end

endmodule
```

Simulation Results:

Multiplier:



Divider:



Synthesis tcl Script:

Multiplier:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl mult.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > mult_netlist.v

gui_show
report timing > mult_timing.rep
report power > mult_power.rep
report area > mult_area.rep
```

Divider:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl div.v
elaborate
synthesize -to_mapped -effort medium

write_hdl > div_netlist.v

gui_show
report timing > div_timing.rep
report power > div_power.rep
report area > div_area.rep
```

17. Design of ALU perform- ADD, SUB, AND,OR,1's & 2's compliment, Multiplication, Division

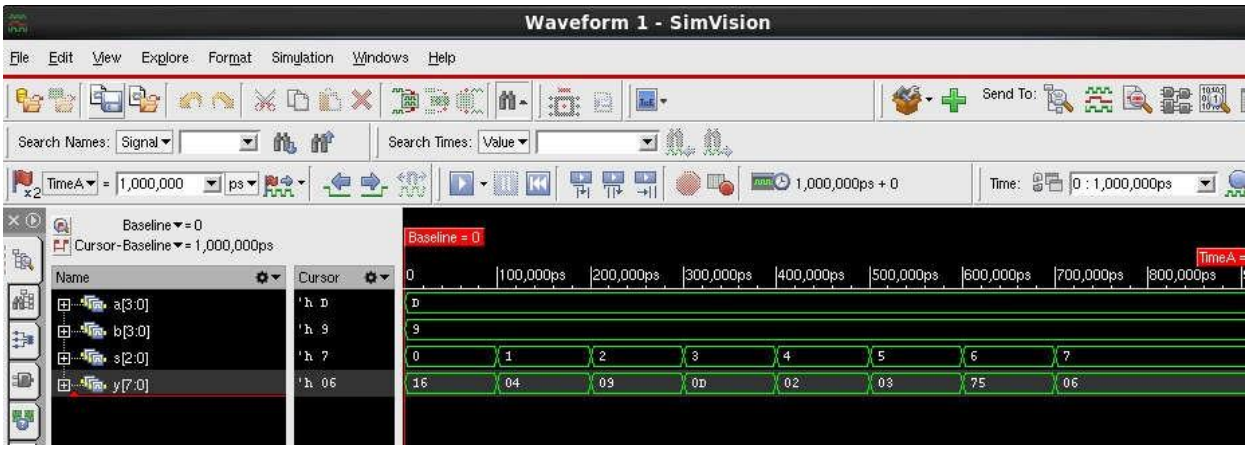
Verilog Design:

```
`timescale 1ns/1ps
module alu(a,b,s,y);
input[3:0]a;
input[3:0]b;
input[2:0]s;
output[7:0]y;
reg[7:0]y;
always@(a,b,s)
begin
case(s)
3'b000:y=a+b;
3'b001:y=a-b;
3'b010:y=a&b;
3'b011:y=a|b;
3'b100:y=4'b1111^a;
3'b101:y=(4'b1111^a)+1'b1;
3'b110:y=a*b;
3'b111:begin y=a;
y=y>>1'b1;
end
endcase
end
endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ps

module alu_tb();
reg[3:0]a;
reg[3:0]b;
reg[2:0]s;
wire[7:0]y;

alu uut(a,b,s,y);

initial begin
a=4'b1101;  b=4'b1001;
s=3'b000;
#100 s=3'b001;
#100 s=3'b010;
#100 s=3'b011;
#100 s=3'b100;
#100 s=3'b101;
#100 s=3'b110;
#100 s=3'b111;
end
initial
#1000 $stop;
endmodule
```

Simulation Results:



Synthesis tcl Script:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib
set_attr library slow.lib
set_attr hdl_search_path ./
read_hdl alu.v
elaborate
synthesize -to_mapped -effort medium
```

```
write_hdl > alu_netlist.v

gui_show
report timing > alu_timing.rep
report power > alu_power.rep
report area > alu_area.rep
```