VERILOG PROGRAMS

GATE LEVEL MODELLING

Submitted by

MELVIN RIJOHN T

Basic Gates

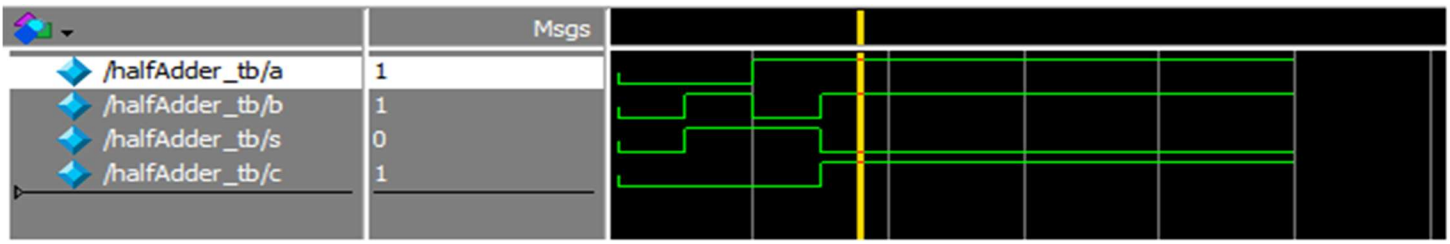
```
module basicGates(A,B,NY,AY,OY,NAY,NOY,XY,XNY);
        input A,B;
        output NY,AY,OY,NAY,NOY,XY,XNY;
        not(NY,A);
        and(AY, A, B);
        or(OY, A, B);
        nand(NAY, A, B);
        nor(NOY, A, B);
        xor(XY, A, B);
        xnor(XNY,A,B);
endmodule

//Test bench

module basicgates_tb;
        reg a,b;
        wire NY,AY,OY,NAY,NOY,XY,XNY;
        basicGates hh(a,b,NY,AY,OY,NAY,NOY,XY,XNY);
        initial begin
                a=0; b=0; #10ns;
                a=0; b=1; #10ns;
                a=1; b=0; #10ns;
                a=1; b=1; #10ns;
                #100ns;
        $finish;
        end
endmodule
```

Output:

Half Adder

```
module halfAdder(s,c,a,b);
        input a,b;
        output s,c;
        xor(s,a,b);
        and(c,a,b);
endmodule

//Test Bench

module halfAdder_tb;
        reg a,b;
        wire s,c;

        halfAdder dut(s,c,a,b);
        initial begin
                a=0; b=0; #10;
                a=0; b=1; #10;
                a=1; b=0; #10;
                a=1; b=1; #10;
        end
endmodule
```

Output:



Half Substractor

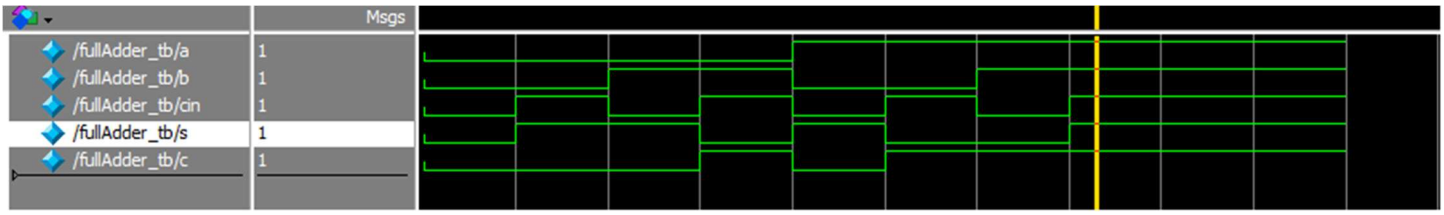
```
module halfsubstractor (a,b,d,br);
        input a,b;
        output d,br;
        wire t0;
        not(t0,a);
        xor(d,a,b);
        and(br,t0,b);
endmodule

//Test Bench
module halfsubstractor_tb();
        reg a,b;
        wire d,br;
        halfsubstractor dut(a,b,d,br);
        initial begin
        a=0; b=0; #10;
        a=0; b=1; #10;
        a=1; b=0; #10;
        a=1; b=1; #10;
        end
endmodule
```

Output:

Full Adder

```
module fullAdder(s,c,a,b,cin);
	input a,b,cin;
	output s,c;
	wire t0,t1,t2;
	xor(s,a,b,cin);
	and(t0,a,b);
	xor(t1,a,b);
	and(t2,t1,cin);
	or(c,t0,t2);
endmodule

//Test Bench

module  fullAdder_tb;
	reg a,b,cin;
	wire s,c;

	fullAdder dut(s,c,a,b,cin);
	initial begin
		a=0; b=0; cin=0; #10;
		a=0; b=0; cin=1; #10;
		a=0; b=1; cin=0; #10;
		a=0; b=1; cin=1; #10;
		a=1; b=0; cin=0; #10;
		a=1; b=0; cin=1; #10;
		a=1; b=1; cin=0; #10;
		a=1; b=1; cin=1; #10;
	end
endmodule
```

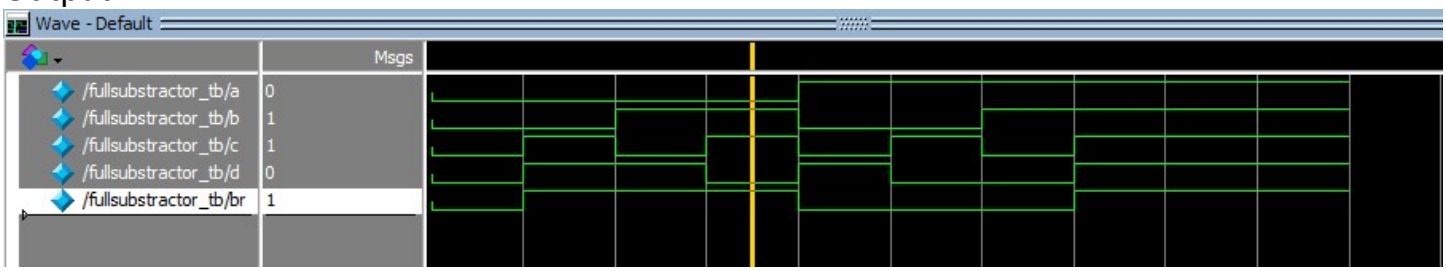
Output:



Full Substractor

```
module halfsubstractor (a,b,d,br);
   input a,b;
   output d,br;
   wire t0;
   not(t0,a);
   xor(d,a,b);
   and(br,t0,b);
endmodule

module fullsubstractor (a,b,c,d,br);
   input a,b,c;
   output d,br;
   wire t0,t1,t2;
   halfsubstractor hs0(a,b,t0,t1);
   halfsubstractor hs1(t0,c,d,t2);
   or(br,t1,t2);
endmodule

module fullsubstractor_tb();
   reg a,b,c;
   wire d,br;
   fullsubstractor dut(a,b,c,d,br);
   initial begin
      a=0; b=0; c=0; #10;
      a=0; b=0; c=1; #10;
      a=0; b=1; c=0; #10;
      a=0; b=1; c=1; #10;
      a=1; b=0; c=0; #10;
      a=1; b=0; c=1; #10;
      a=1; b=1; c=0; #10;
      a=1; b=1; c=1; #10;
   end
endmodule
```

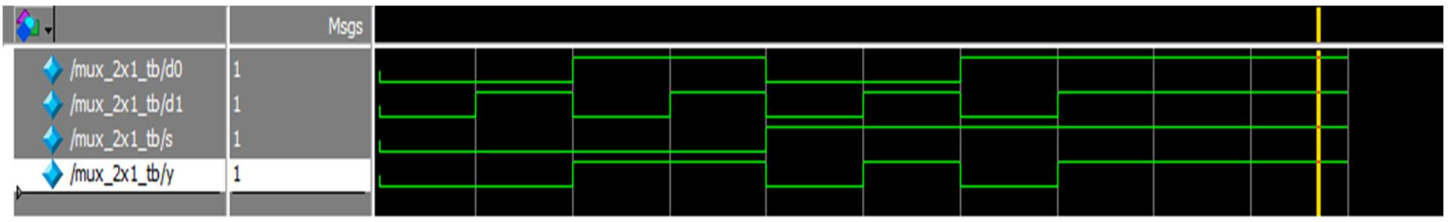
Output:



2:1 Mux

```
module mux_2x1(d0,d1,s,y);
	input d0,d1,s;
	output y;
	wire t0,t1,sBar;
	not(sBar,s);
	and(t0,d0,sBar);
	and(t1,d1,s);
	or(y,t0,t1);
endmodule

//TestBench

module mux_2x1_tb;
	reg d0,d1,s;
	wire y;

	mux_2x1 dut(d0,d1,s,y);

	initial begin
		s=0; d0=0; d1=0; #10;
		s=0; d0=0; d1=1; #10;
		s=0; d0=1; d1=0; #10;
		s=0; d0=1; d1=1; #10;
		s=1; d0=0; d1=0; #10;
		s=1; d0=0; d1=1; #10;
		s=1; d0=1; d1=0; #10;
		s=1; d0=1; d1=1; #10;
	end
endmodule
```

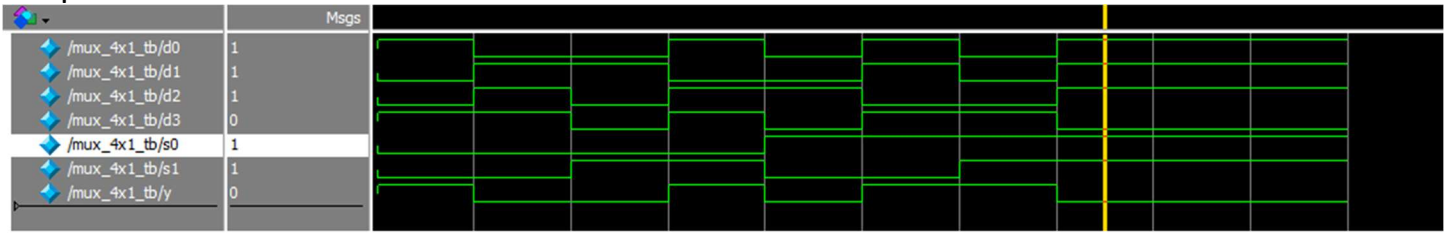
Output:



4:1 Mux using 2:1 Mux

```
module mux_2x1(d0,d1,s,y);
	input d0,d1,s;
	output y;
	wire sBar,t1,t2;
	not(sBar,s);
	and(t1,d0,sBar);
	and(t2,d1,s);
	or(y,t1,t2);
endmodule

module mux_4x1(d0,d1,d2,d3,s0,s1,y);
	input d0,d1,d2,d3,s0,s1;
	output y;
	wire t1,t2;
	mux_2x1 m0(d0,d1,s0,t1);
	mux_2x1 m1(d2,d3,s0,t2);
	mux_2x1 m2(t1,t2,s1,y);
endmodule

module mux_4x1_tb;
	reg d0,d1,d2,d3,s0,s1;
	wire y;
	mux_4x1 dut(d0,d1,d2,d3,s0,s1,y);
	initial begin
		s0=0; s1=0; d0=1; d1=0; d2=0; d3=1; #10;
		s0=0; s1=0; d0=0; d1=1; d2=1; d3=1; #10;
		s0=0; s1=1; d0=0; d1=1; d2=0; d3=0; #10;
		s0=0; s1=1; d0=1; d1=0; d2=1; d3=1; #10;
		s0=1; s1=0; d0=0; d1=0; d2=1; d3=0; #10;
		s0=1; s1=0; d0=1; d1=1; d2=0; d3=1; #10;
		s0=1; s1=1; d0=0; d1=0; d2=0; d3=1; #10;
		s0=1; s1=1; d0=1; d1=1; d2=1; d3=0; #10;
	end
endmodule
```

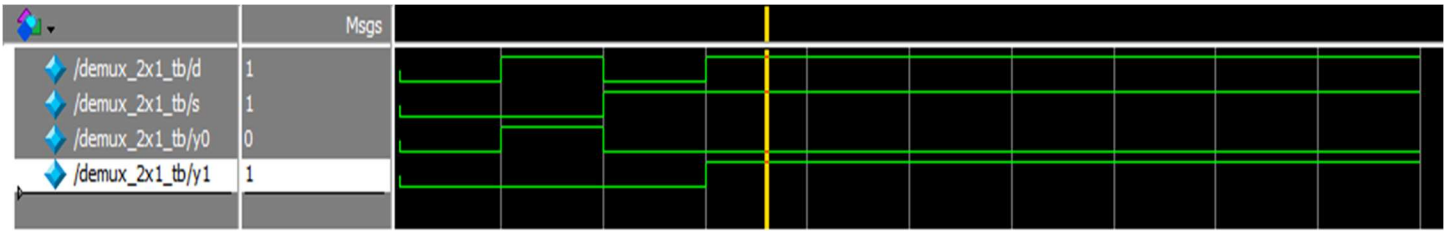
Output:



1:2 Demux

```
module demux_2x1(d,s,y0,y1);
	input d,s;
	output y0,y1;
	wire sBar;
	not(sBar,s);
	and(y0,d,sBar);
	and(y1,d,s);
endmodule

//TestBench

module demux_2x1_tb;
	reg d,s;
	wire y0,y1;

	demux_2x1 dut(d,s,y0,y1);

	initial begin
		d=0; s=0; #10;
		d=1; s=0; #10;
		d=0; s=1; #10;
		d=1; s=1; #10;
	end
endmodule
```

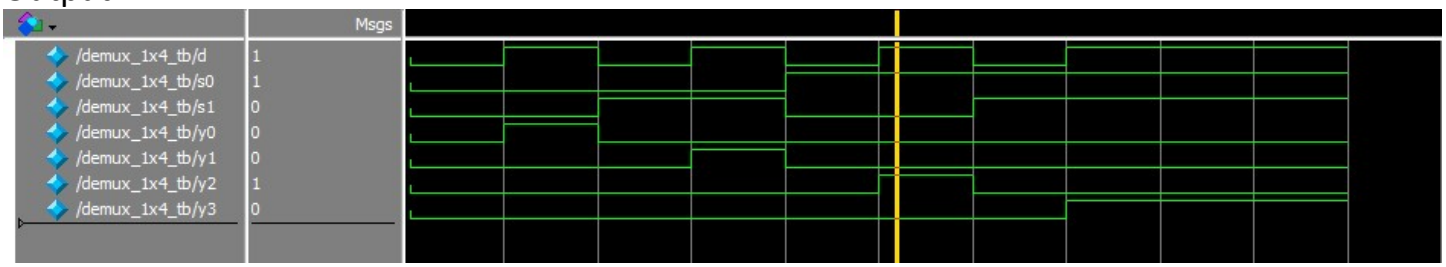
Output:



1:4 Demux using 1:2 Demux

```
module demux_1x2 (d,s,y0,y1);
   input d,s;
   output y0,y1;
   wire sBar;
   not(sBar,s);
   and(y0,d,sBar);
   and(y1,d,s);
endmodule

module demux_1x4(d,s0,s1,y0,y1,y2,y3);
   input d,s0,s1;
   output y0,y1,y2,y3;
   wire t0,t1;
   demux_1x2 d0(d,s0,t0,t1);
   demux_1x2 d1(t0,s1,y0,y1);
   demux_1x2 d2(t1,s1,y2,y3);
endmodule

module demux_1x4_tb ();
   reg d,s0,s1;
   wire y0,y1,y2,y3;
   demux_1x4 dut(d,s0,s1,y0,y1,y2,y3);
   initial begin
      d=0; s0=0; s1=0; #10;
      d=1; s0=0; s1=0; #10;
      d=0; s0=0; s1=1; #10;
      d=1; s0=0; s1=1; #10;
      d=0; s0=1; s1=0; #10;
      d=1; s0=1; s1=0; #10;
      d=0; s0=1; s1=1; #10;
      d=1; s0=1; s1=1; #10;
   end
endmodule
```

Output:



VERILOG PROGRAMS
DATA FLOW MODELLING

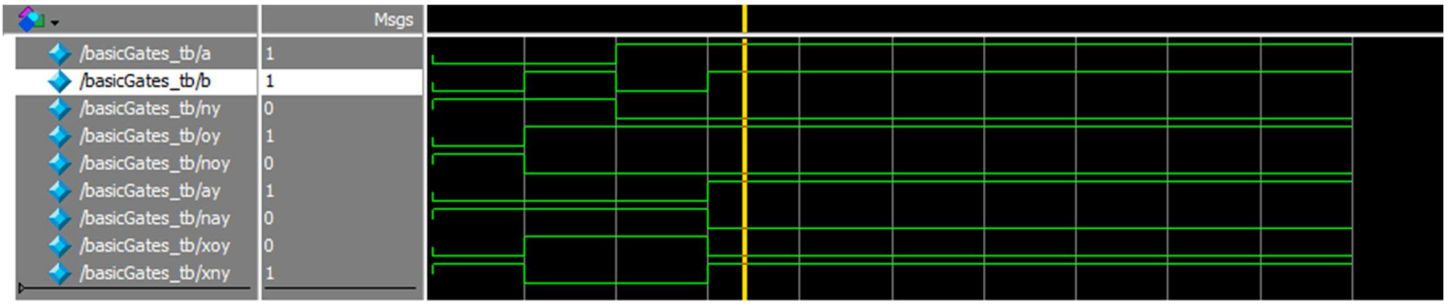
Submitted by

MELVIN RIJOHN T

Basic Gates

```
module basicGates(a,b,ny,oy,ay,nay,noy,xoy,xny);
	input a,b;
	output ny,oy,noy,ay,nay,xoy,xny;
	assign ny = ~a;
	assign oy = a|b;
	assign noy = ~(a|b);
	assign ay = a&b;
	assign nay = ~(a&b);
	assign xoy = a^b;
	assign xny = ~(a^b);
endmodule

//Test Bench
module basicGates_tb;
	reg a,b;
	wire ny,oy,noy,ay,nay,xoy,xny;

	basicGates dut(a,b,ny,oy,ay,nay,noy,xoy,xny);
	initial begin
		a=0; b=0; #10;
		a=0; b=1; #10;
		a=1; b=0; #10;
		a=1; b=1; #10;
	end
endmodule
```

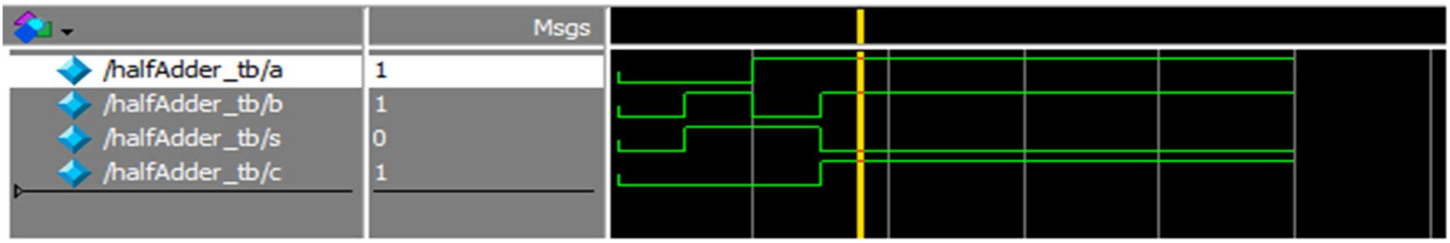
Output:



Half Adder

```
module halfAdder(s,c,a,b);
        input a,b;
        output s,c;
        assign s = a^b;
        assign c = a&b;
endmodule

//Test Bench

module halfAdder_tb;
        reg a,b;
        wire s,c;

        halfAdder dut(s,c,a,b);
        initial begin
                a=0; b=0; #10;
                a=0; b=1; #10;
                a=1; b=0; #10;
                a=1; b=1; #10;
        end
endmodule
```

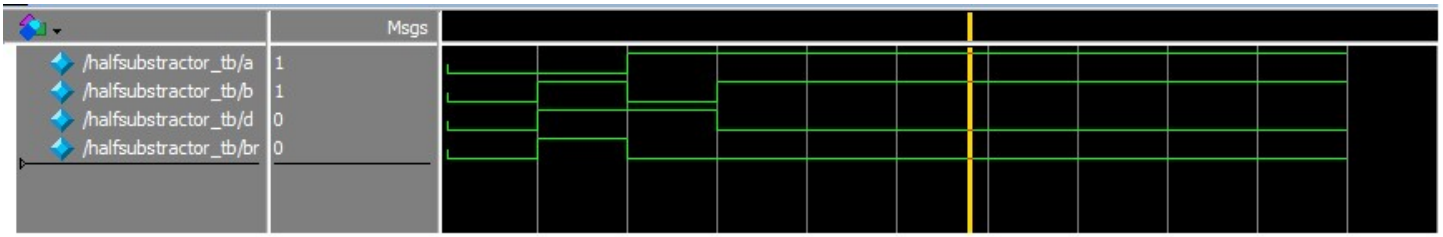
Output:



Half Substractor

```
module halfsubstractor (a,b,d,br);
	input a,b;
	output d,br;
	assign d = a^b;
	assign br = (~a)&b;
endmodule

//Test Bench
module halfsubstractor_tb();
	reg a,b;
	wire d,br;
	halfsubstractor dut(a,b,d,br);
	initial begin
		a=0; b=0; #10;
		a=0; b=1; #10;
		a=1; b=0; #10;
		a=1; b=1; #10;
	end
endmodule
```

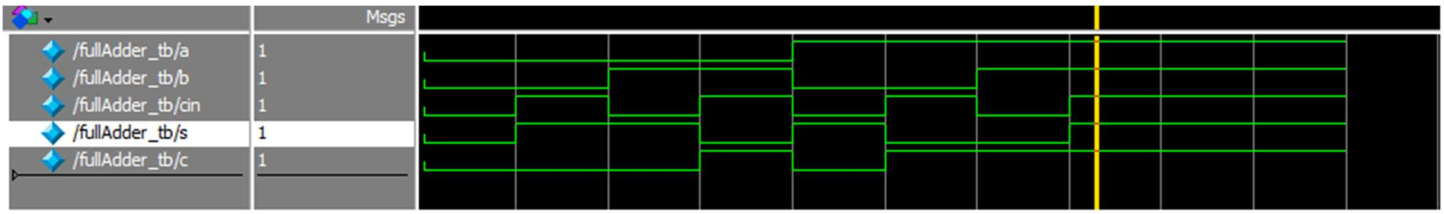
Output:



Full Adder

```
module fullAdder(s,c,a,b,cin);
	input a,b,cin;
	output s,c;

	assign s = (a^b)^cin;
	assign c = (a&b)|((a^b)&cin);
endmodule

//Test Bench
module  fullAdder_tb;
	reg a,b,cin;
	wire s,c;

	fullAdder dut(s,c,a,b,cin);
	initial begin
		a=0; b=0; cin=0; #10;
		a=0; b=0; cin=1; #10;
		a=0; b=1; cin=0; #10;
		a=0; b=1; cin=1; #10;
		a=1; b=0; cin=0; #10;
		a=1; b=0; cin=1; #10;
		a=1; b=1; cin=0; #10;
		a=1; b=1; cin=1; #10;
	end
endmodule
```

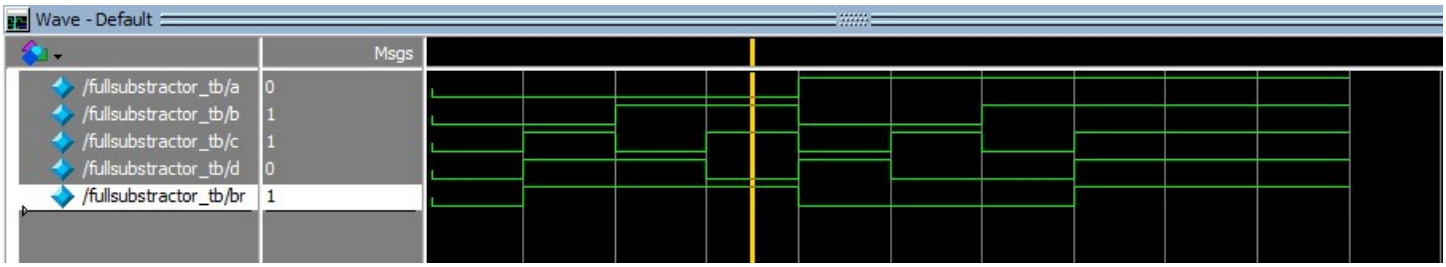
Output:



Full Substractor

```
module halfsubstractor (a,b,d,br);
	input a,b;
	output d,br;
	assign d = a^b;
	assign br = (~a)&b;
endmodule

module fullsubstractor (a,b,c,d,br);
	input a,b,c;
	output d,br;
	wire t0,t1,t2;
	halfsubstractor hs0(a,b,t0,t1);
	halfsubstractor hs1(t0,c,d,t2);
	assign br = t1 | t2;
endmodule

module fullsubstractor_tb();
	reg a,b,c;
	wire d,br;
	fullsubstractor dut(a,b,c,d,br);
	initial begin
		a=0; b=0; c=0; #10;
		a=0; b=0; c=1; #10;
		a=0; b=1; c=0; #10;
		a=0; b=1; c=1; #10;
		a=1; b=0; c=0; #10;
		a=1; b=0; c=1; #10;
		a=1; b=1; c=0; #10;
		a=1; b=1; c=1; #10;
	end
endmodule
```

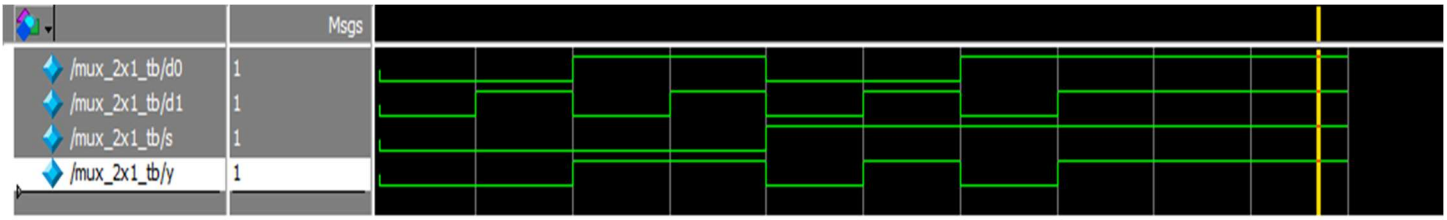
Output:



2:1 Mux

```
module mux_2x1(d0,d1,s,y);
	input d0,d1,s;
	output y;
	assign y = (d0&(~s))| (d1&s);
endmodule

//TestBench

module mux_2x1_tb;
	reg d0,d1,s;
	wire y;

	mux_2x1 dut(d0,d1,s,y);

	initial begin
		s=0; d0=0; d1=0; #10;
		s=0; d0=0; d1=1; #10;
		s=0; d0=1; d1=0; #10;
		s=0; d0=1; d1=1; #10;
		s=1; d0=0; d1=0; #10;
		s=1; d0=0; d1=1; #10;
		s=1; d0=1; d1=0; #10;
		s=1; d0=1; d1=1; #10;
	end
endmodule
```

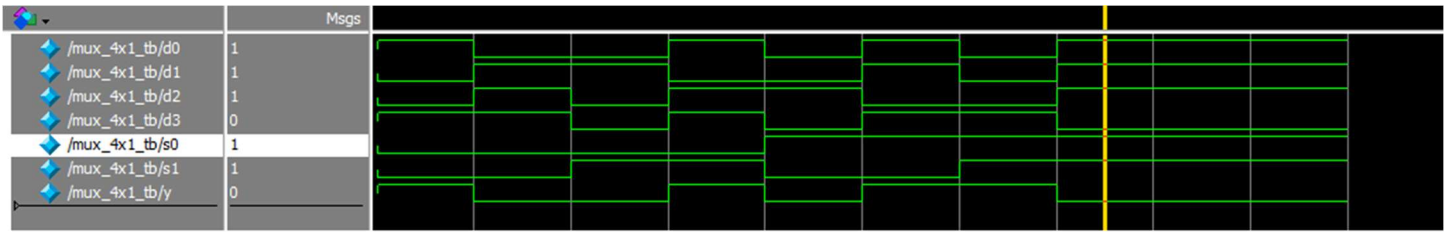
Output:



4:1 Mux using 2:1 Mux

```
module mux_2x1(d0,d1,s,y);
	input d0,d1,s;
	output y;
	assign y = (d0&(~s))| (d1&s);
endmodule

module mux_4x1(d0,d1,d2,d3,s0,s1,y);
	input d0,d1,d2,d3,s0,s1;
	output y;
	wire t1,t2;
	mux_2x1 m0(d0,d1,s0,t1);
	mux_2x1 m1(d2,d3,s0,t2);
	mux_2x1 m2(t1,t2,s1,y);
endmodule
//TestBench

module mux_4x1_tb;
	reg d0,d1,d2,d3,s0,s1;
	wire y;
	mux_4x1 dut(d0,d1,d2,d3,s0,s1,y);
	initial begin
		s0=0; s1=0; d0=1; d1=0; d2=0; d3=1; #10;
		s0=0; s1=0; d0=0; d1=1; d2=1; d3=1; #10;
		s0=0; s1=1; d0=0; d1=1; d2=0; d3=0; #10;
		s0=0; s1=1; d0=1; d1=0; d2=1; d3=1; #10;
		s0=1; s1=0; d0=0; d1=0; d2=1; d3=0; #10;
		s0=1; s1=0; d0=1; d1=1; d2=0; d3=1; #10;
		s0=1; s1=1; d0=0; d1=0; d2=0; d3=1; #10;
		s0=1; s1=1; d0=1; d1=1; d2=1; d3=0; #10;
	end
endmodule
```

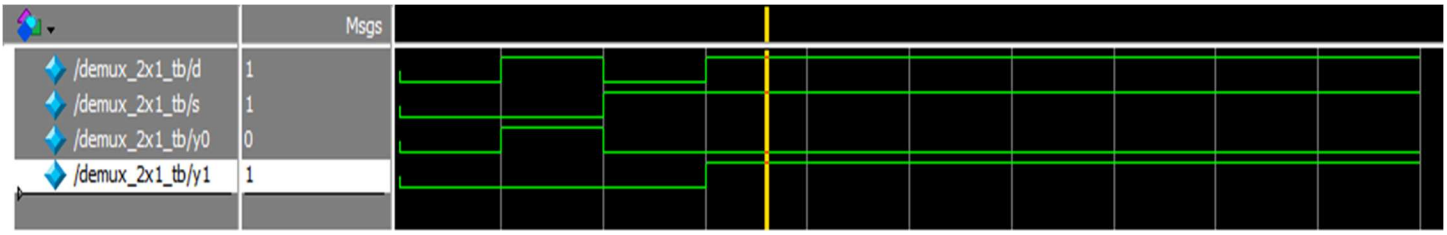
Output:



1:2 Demux

```
module demux_2x1(d,s,y0,y1);
	input d,s;
	output y0,y1;
	assign y0 = d&(~s);
	assign y1 = d&s;
endmodule

//TestBench

module demux_2x1_tb;
	reg d,s;
	wire y0,y1;

	demux_2x1 dut(d,s,y0,y1);

	initial begin
		d=0; s=0; #10;
		d=1; s=0; #10;
		d=0; s=1; #10;
		d=1; s=1; #10;
	end
endmodule
```

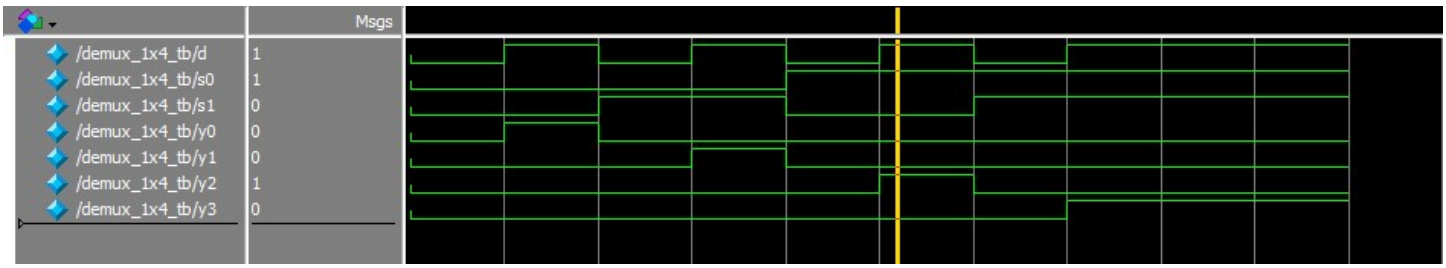
Output:



1:4 Demux using 1:2 Demux

```
module demux_1x2(d,s,y0,y1);
	input d,s;
	output y0,y1;
	assign y0 = d&(~s);
	assign y1 = d&s;
endmodule

module demux_1x4(d,s0,s1,y0,y1,y2,y3);
	input s0,s1,d;
	output y0,y1,y2,y3;
	wire t1,t2;
	demux_1x2 d0(d,s0,t1,t2);
	demux_1x2 d1(t1,s1,y0,y1);
	demux_1x2 d2(t2,s1,y2,y3);
endmodule
//TestBench

module demux_1x4_tb;
	reg d,s0,s1;
	wire y0,y1,y2,y3;
	demux_1x4 dut(d,s0,s1,y0,y1,y2,y3);
	initial begin
		d=0; s0=0; s1=0; #10;  d=1; s0=0; s1=0; #10;
		d=0; s0=0; s1=1; #10;  d=1; s0=0; s1=1; #10;
		d=0; s0=1; s1=0; #10;  d=1; s0=1; s1=0; #10;
		d=0; s0=1; s1=1; #10;  d=1; s0=1; s1=1; #10;
	end
endmodule
```

Output:



VERILOG PROGRAMS

SEQUENTIAL CIRCUITS

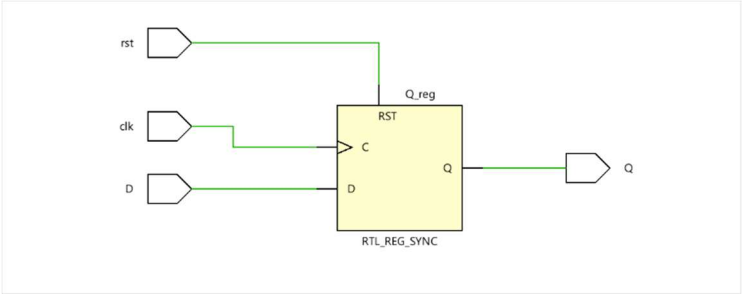
Submitted by

MELVIN RIJOHN T

DFF

```
module dff (
    input wire D,
    input wire clk,
    input wire rst,
    output reg Q;
);
    always @(posedge clk) begin
        if(rst)
            Q <= 0;
        else
            Q <= D;
    end
endmodule

module tb();
    reg d,rst,clk=0;
    wire q;
    always #5 clk=~clk;
    initial begin
        rst=1; d=0; #10;
        rst=1; d=1; #10;
        rst=0; d=0; #10;
        rst=0; d=1; #10;
    end
endmodule
```

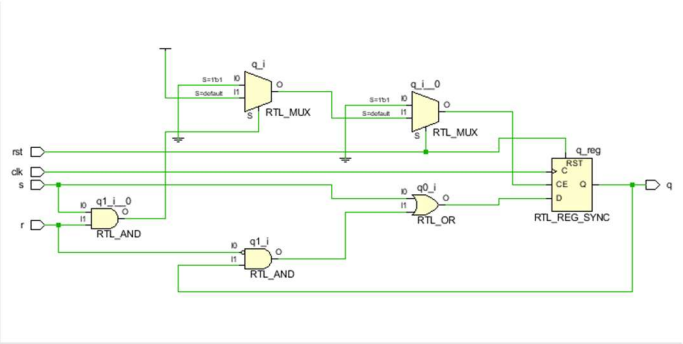
OUTPUT

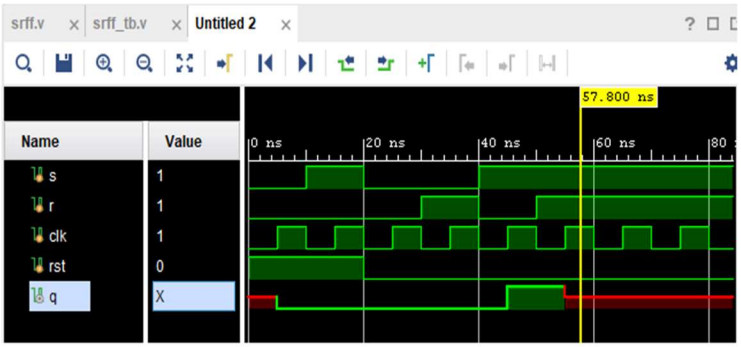


SRFF

```
module srff (
    input wire s,
    input wire r,
    input wire clk,
    input wire rst,
    output reg q
);
    always @(posedge clk) begin
        if(rst)
            q <= 0;
        else
            if(s&r)
                q <= 1'bx;
            else
                q = s | (~r & q);
    end
endmodule

module  srff_tb();
    reg s,r,clk=0,rst;
    wire q;
    srff dut(s,r,clk,rst,q);
    always #5 clk=~clk;

    initial begin
        rst = 1; s=0; r=0; #10;
        rst = 1; s=1; r=0; #10;
        rst = 0; s=0; r=0; #10;
        rst = 0; s=0; r=1; #10;
        rst = 0; s=1; r=0; #10;
        rst = 0; s=1; r=1; #10;
        #10;
    end
endmodule
```

OUTPUT

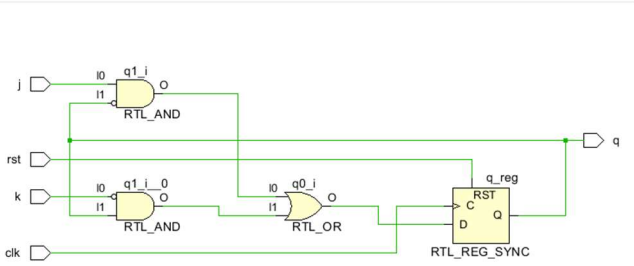




JKFF

```
module jkff(
    input j,
    input k,
    input clk,
    input rst,
    output reg q
);
    always @(posedge clk) begin
        if(rst)
            q <= 0;
        else
            q = (j & (~q)) | (~k & q);
    end
endmodule

module jkff_tb();
    reg j,k,clk=0,rst;
    wire q;
    jkff dut(j,k,clk,rst,q);
    always #5 clk=~clk;

    initial begin
        rst = 1; j=0; k=0; #10;
        rst = 1; j=1; k=0; #10;
        rst = 0; j=0; k=0; #10;
        rst = 0; j=0; k=1; #10;
        rst = 0; j=1; k=0; #10;
        rst = 0; j=1; k=1; #10;
    end

endmodule
```

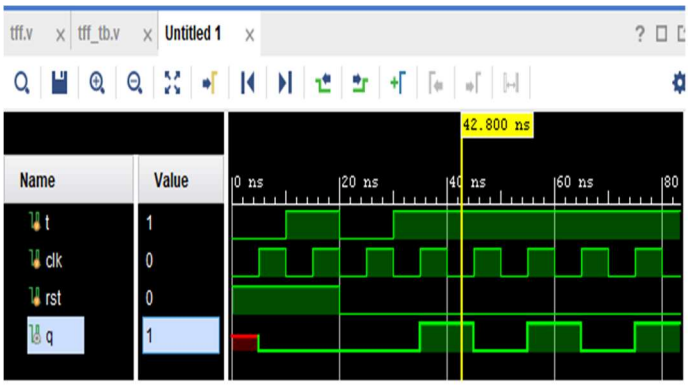
OUTPUT

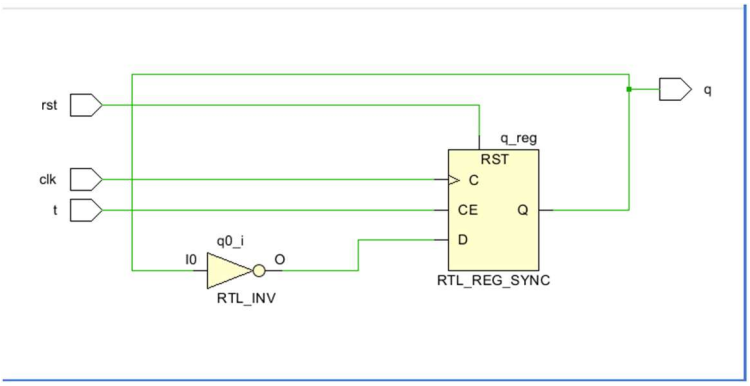


TFF

```
module tff (
    input wire t,
    input wire clk,
    input wire rst,
    output reg q
);
    always @(posedge clk) begin
        if(rst)
            q <= 0;
        else
            q <= t ? ~q : q;
    end
endmodule

module tff_tb ();
    reg t, clk = 0,rst;
    wire q;
    tff dut(t,clk,rst,q);
    always #5 clk=~clk;

    initial begin
        rst = 1; t=0; #10;
        rst = 1; t=1; #10;
        rst = 0; t=0; #10;
        rst = 0; t=1; #10;
        #10;
    end
endmodule
```

OUTPUT



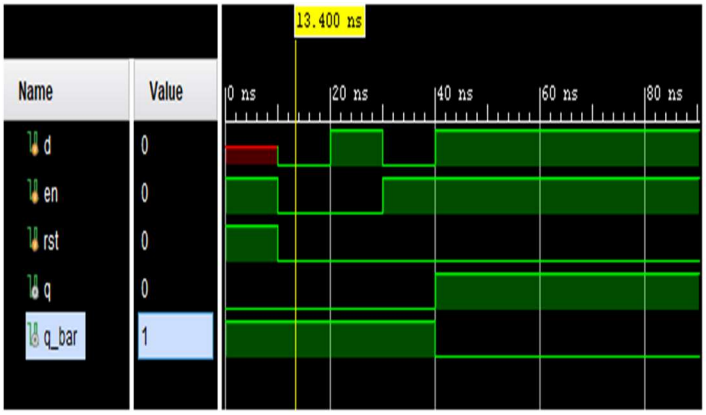


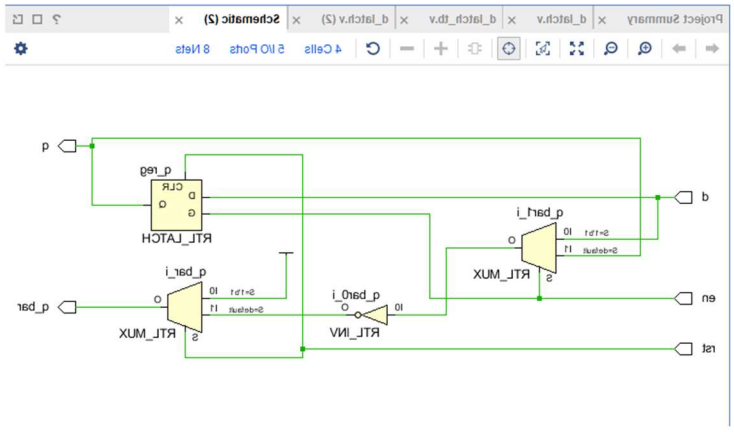
D LATCH

```
module d_latch (
    input d,
    input en,
    input rst,
    output reg q,
    output reg q_bar
);
    always @(*) begin
        if(rst) begin
            q = 0;
            q_bar = 1;
        end
        else begin
            q = en ? d : q;
            q_bar = ~q;
        end
    end
endmodule

module d_latch_tb ();
    reg d,en,rst;
    wire q,q_bar;

    d_latch dut(d,en,rst,q,q_bar);
    initial begin
        rst = 1; en=1; #10;
        rst = 0; en=0; d=0; #10;
        rst = 0; en=0; d=1; #10;
        rst = 0; en=1; d=0; #10;
        rst = 0; en=1; d=1; #10;
```

```
    end
endmodule
```

OUTPUT



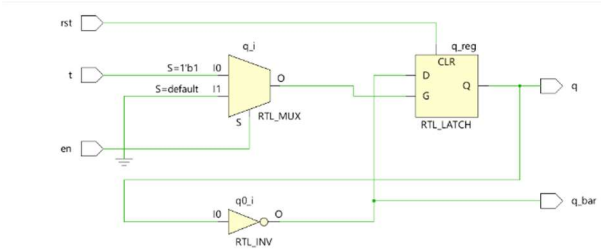


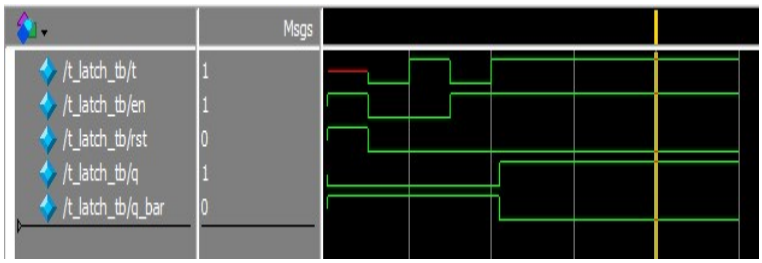
T LATCH

```
module t_latch (
    input t,
    input en,
    input rst,
    output reg q,
    output reg q_bar
);
    always @(*) begin
        if(rst) begin
            q = 0;
        end
        else begin
            if(en) begin
                case (t)
                    0 : q = q;
                    1 : q = #2 ~q;
                endcase
            end
        end
    end
    always @(*) begin
        q_bar = ~q;
    end
endmodule

module t_latch_tb ();
    reg t,en,rst;
    wire q,q_bar;

    t_latch dut(t,en,rst,q,q_bar);
    initial begin
```

```
        rst = 1; en=1; #10;
        rst = 0; en=0; t=0; #10;
        rst = 0; en=0; t=1; #10;
        rst = 0; en=1; t=0; #10;
        rst = 0; en=1; t=1; #10;
        rst = 0; en=1; t=1; #10;
    end
endmodule
```

OUTPUT



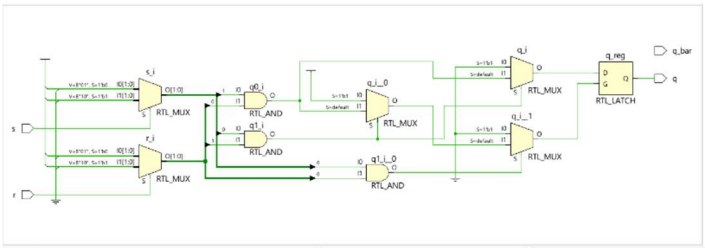


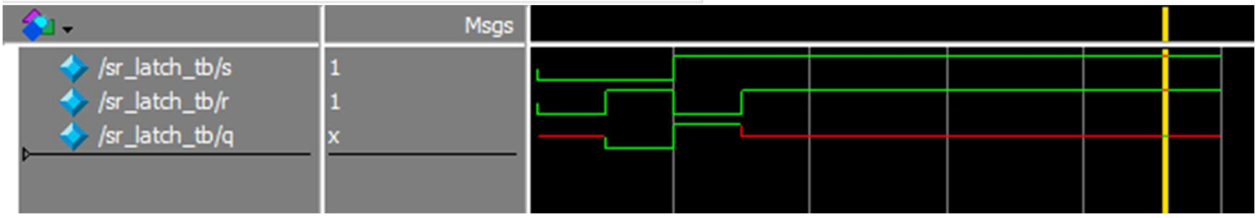
SR LATCH

```
module sr_latch (
    input s,
    input r,
    output reg q,
);
    always @(*) begin
        if(s==0 && r==0) begin
            q = q;
        end
        else if(s==0 && r==1) begin
            q = 0;
        end
        else if(s==1 && r==0) begin
            q = 1;
        end
        else if(s==1 && r==1) begin
            q = 1'bx;
        end
    end
endmodule

module sr_latch_tb();
    reg s,r;
    wire q;
    sr_latch dut(s,r,q);
    initial begin
        s = 0; r = 0;
        #10 s = 0; r = 1;
        #10 s = 1; r = 0;
        #10 s = 1; r = 1;
    end
```

```
endmodule
```

OUTPUT



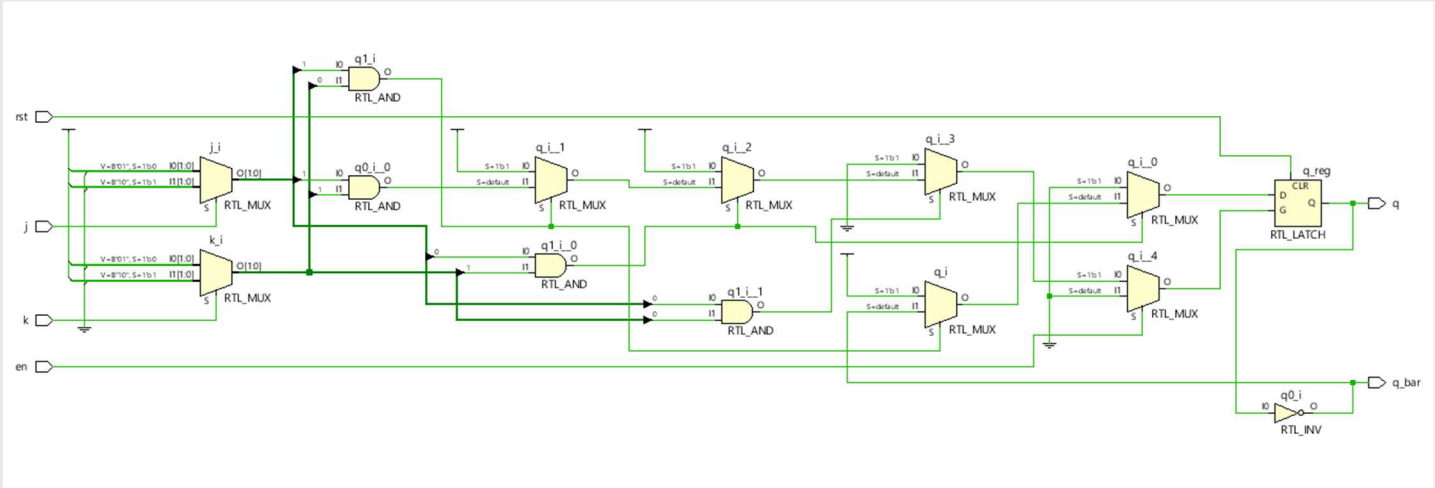


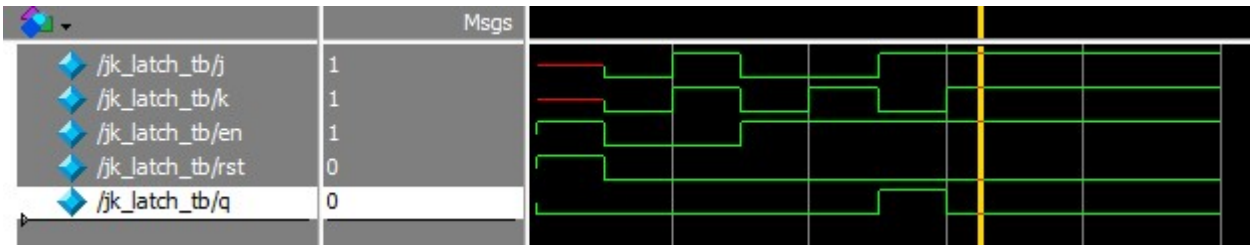
JK _LATCH

```
module jk_latch (
    input j,
    input k,
    input en,
    input rst,
    output reg q
);
    always @(*) begin
        if(rst) begin
            q = 0;
        end
        else begin
            if(en) begin
                if(j ==0 && k == 0) begin
                    q = q;
                end
                else if(j == 0 && k == 1) begin
                    q = 0;
                end
                else if(j == 1 && k == 0) begin
                    q = 1;
                end
                else if(j == 1 && k == 1) begin
                    q = ~q;
                end
            end
        end
    end
endmodule

module jk_latch_tb ();
```

```
    reg j,k,en,rst;
    wire q;

    jk_latch dut(j,k,en,rst,q);
    initial begin
        rst = 1; en=1; #10;
        rst = 0; en=0; j=0; k=0;#10;
        rst = 0; en=0; j=1; k=1;#10;
        rst = 0; en=1; j=0; k=0;#10;
        rst = 0; en=1; j=0; k=1;#10;
        rst = 0; en=1; j=1; k=0;#10;
        rst = 0; en=1; j=1; k=1;#10;
    end
endmodule
```

OUTPUT



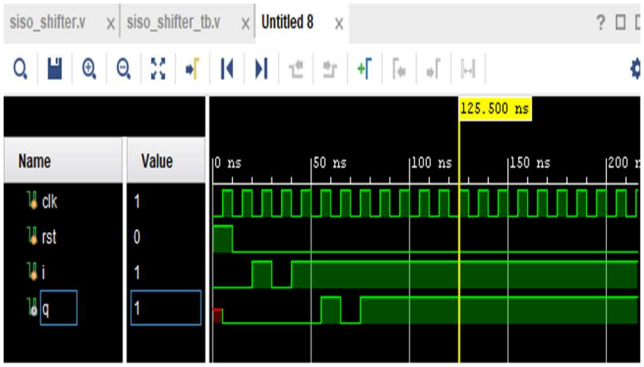


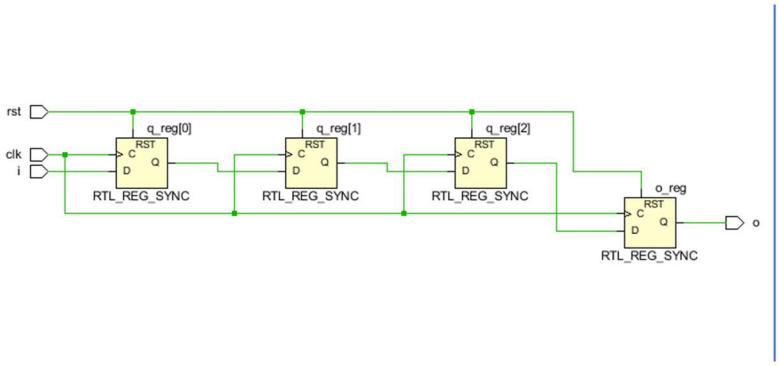
SISO REGISTER

```
module siso_shifter (
    input i,
    input clk,
    input rst,
```

```
    output reg o
);
    reg[2:0] q;
    always @(posedge clk) begin
        if(rst)begin
            o <= 0;
            q <= 0;
         end
        else begin
            q[0] <= i;
            q[1] <= q[0];
            q[2] <= q[1];
            o <= q[2];
        end
    end
endmodule

module siso_shifter_tb ();
    reg clk=0, rst;
    reg i;
    wire q;
    siso_shifter dut (i, clk, rst, q);

    always #5 clk=~clk;

    initial begin
        rst = 1; i = 0;  #10;
        rst = 0; i = 0; #10;
        rst = 0; i = 1; #10;
        rst = 0; i = 0; #10;
        rst = 0; i = 1; #10;
    end
endmodule
```

OUTPUT

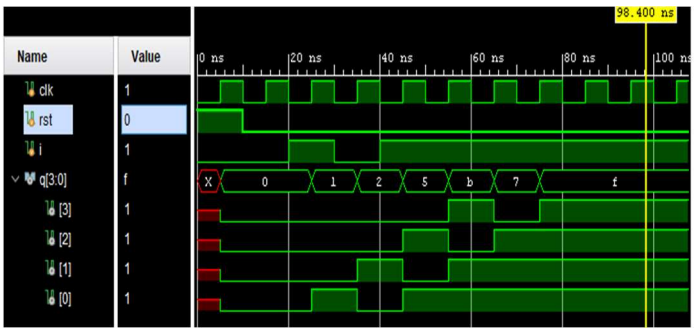


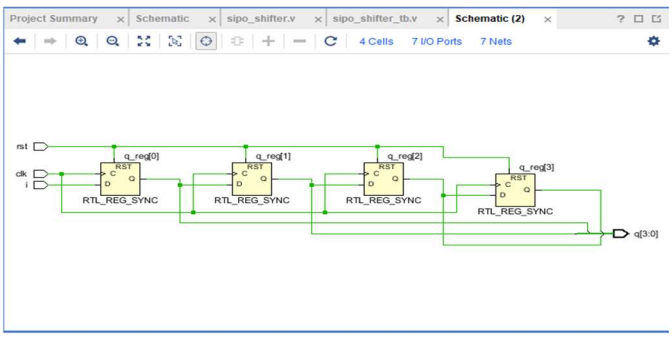


SIPO REGISTER

```
module sipo_shifter (
    input i,
    input clk,
    input rst,
    output reg[3:0] q
);
    reg[2:0] q;
    always @(posedge clk) begin
        if(rst)begin
            q <= 0;
         end
        else begin
            q
            q[0] <= i;
            q[1] <= q[0];
            q[2] <= q[1];
            q[3] <= q[2];
        end
    end
endmodule

module sipo_shifter_tb ();
    reg clk=0, rst;
    reg i;
    wire[3:0] q;
    sipo_shifter dut (i, clk, rst, q);

    always #5 clk=~clk;

    initial begin
        rst = 1; i = 0; #10;
        rst = 0; i = 0; #10;
        rst = 0; i = 1; #10;
        rst = 0; i = 0; #10;
        rst = 0; i = 1; #10;
    end
endmodule
```

OUTPUT

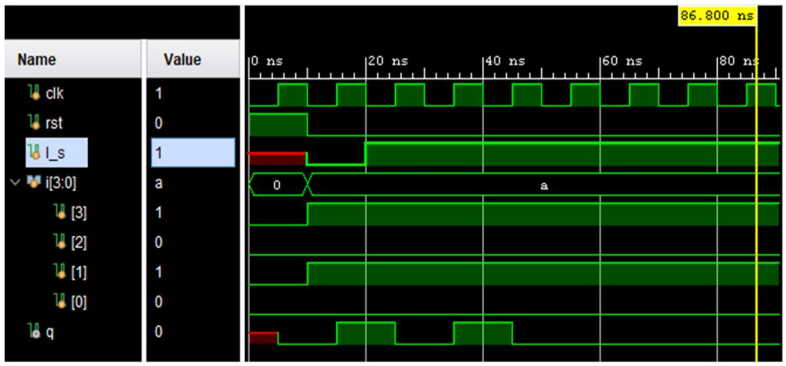


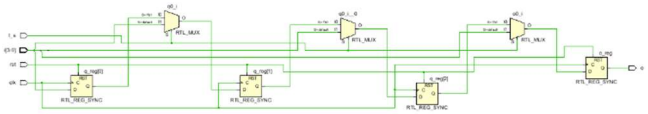


PISO REGISTER

```
module piso_shifter(
    input[3:0] i,
    input clk,
    input rst,
    input l_s,
    output reg o
    );
    reg[2:0] q;
    always @(posedge clk) begin
        if(rst)begin
            q <= 0;
            o <= 0;
         end
        else begin
            q[0] <= i[0];
            q[1] <= l_s ? q[0] : i[1];
            q[2] <= l_s ? q[1] : i[2];
            o <= l_s ? q[2] : i[3];
        end
    end
endmodule

module piso_shifter_tb();
    reg clk=0, rst, l_s;
    reg[3:0] i;
    wire q;
    piso_shifter dut (i, clk, rst, l_s, q);
    always #5 clk=~clk;
    initial begin
        rst = 1; i = 0;  #10;
        rst = 0; i = 4'b1010; l_s = 0; #10;
        rst = 0; i = 4'b1010; l_s = 1; #10;

    end
endmodule
```

OUTPUT

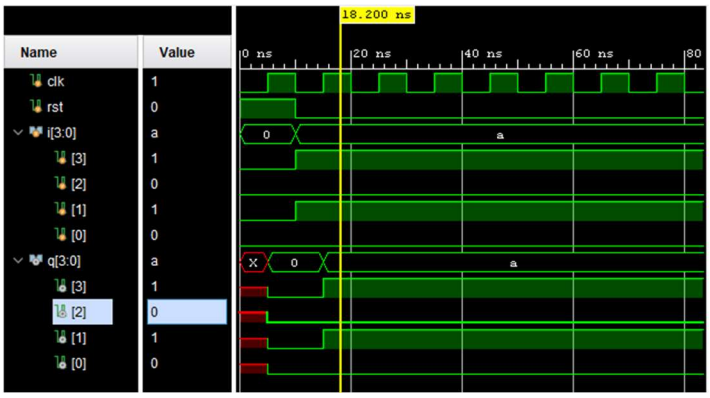


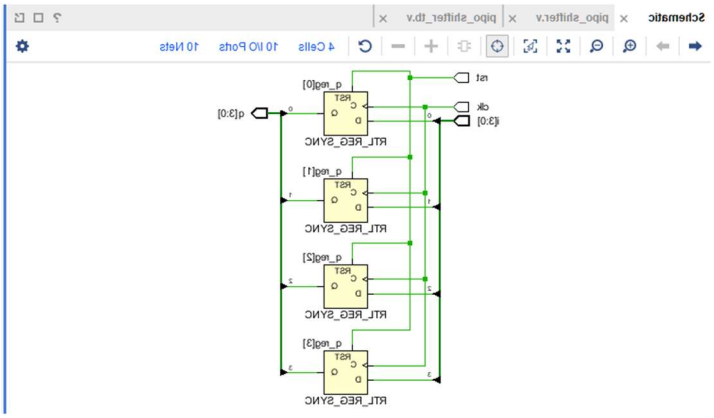


PIPO REGISTER

```
module pipo_shifter(
    input[3:0] i,
    input clk,
    input rst,
    output reg[3:0] q
    );
    always @(posedge clk) begin
        if(rst)begin
            q <= 0;
         end
        else begin
            q[0] <= i[0];
            q[1] <= i[1];
            q[2] <= i[2];
            q[3] <= i[3];
        end
    end
endmodule

module pipo_shifter_tb();
    reg clk=0, rst;
    reg[3:0] i;
    wire[3:0] q;
    pipo_shifter dut (i, clk, rst, q);
    always #5 clk=~clk;
    initial begin
        rst = 1; i = 0;  #10;
        rst = 0; i = 4'b1010; #10;
    end
endmodule
```

OUTPUT

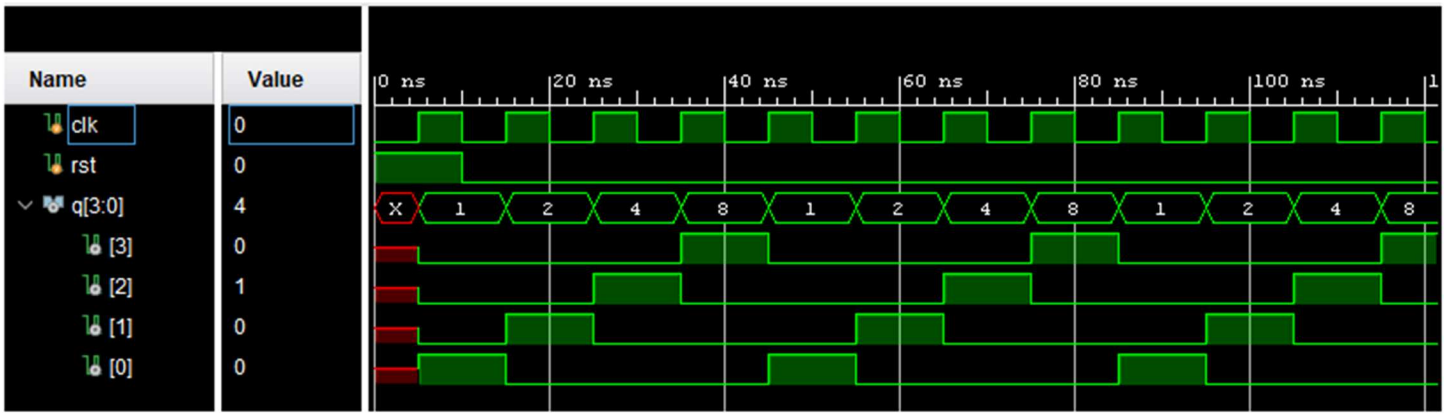


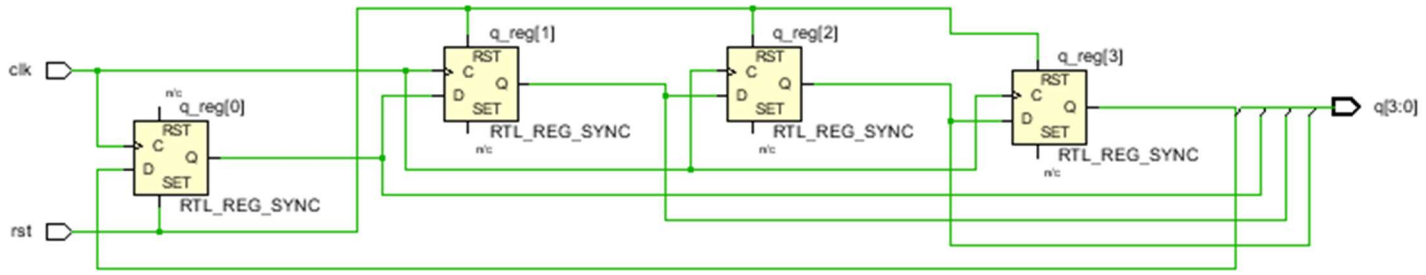


RING COUNTER

```
module ring_counter (
    input clk,
    input rst,
    output reg[3:0] q
);
    always @(posedge clk ) begin
        if (rst)
            q <= 4'b0001;
        else
            q <= {q[2:0], q[3]};
    end
endmodule

module ring_counter_tb ();
    reg clk=0, rst;
    wire[3:0] q;
    ring_counter dut (clk, rst, q);
    always #5 clk = ~clk;
    initial begin
        rst = 1; #10;
        rst = 0;
    end
endmodule
```

OUTPUT

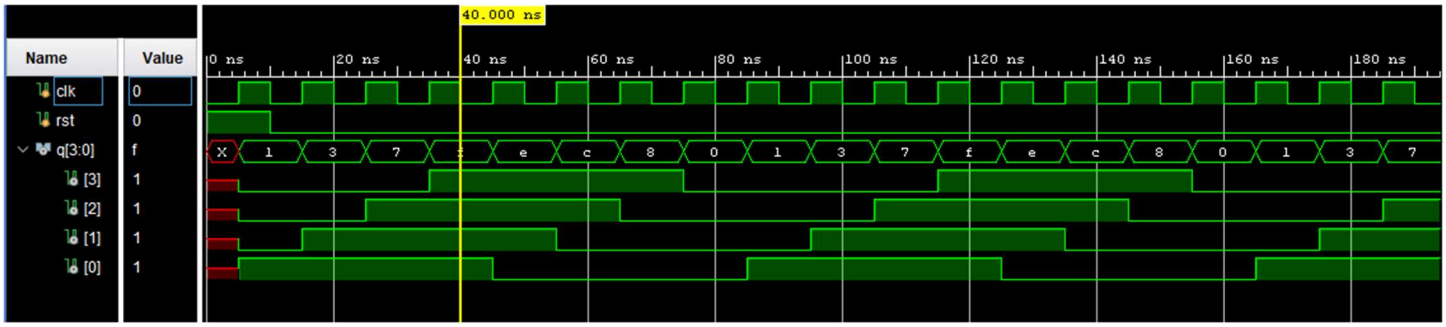


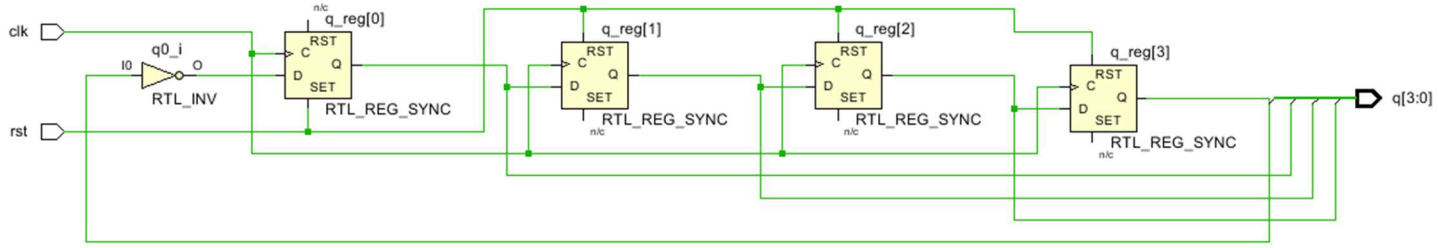


JOHNSON COUNTER

```
module johnson_counter (
    input clk,
    input rst,
    output reg[3:0] q
);
    always @(posedge clk ) begin
        if (rst)
            q <= 4'b0000;
        else
            q <= {q[2:0], ~q[3]};
    end
endmodule

module johnson_counter_tb ();
    reg clk=0, rst;
    wire[3:0] q;
    johnson_counter dut (clk, rst, q);
    always #5 clk = ~clk;
    initial begin
        rst = 1; #10;
        rst = 0;
    end
endmodule
```

OUTPUT

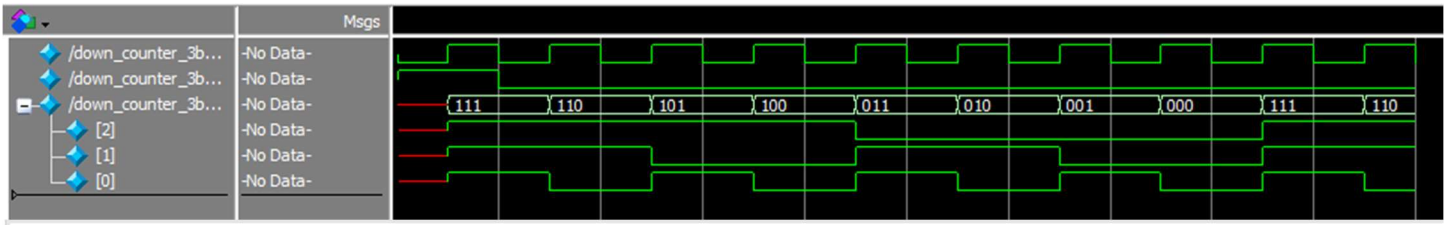


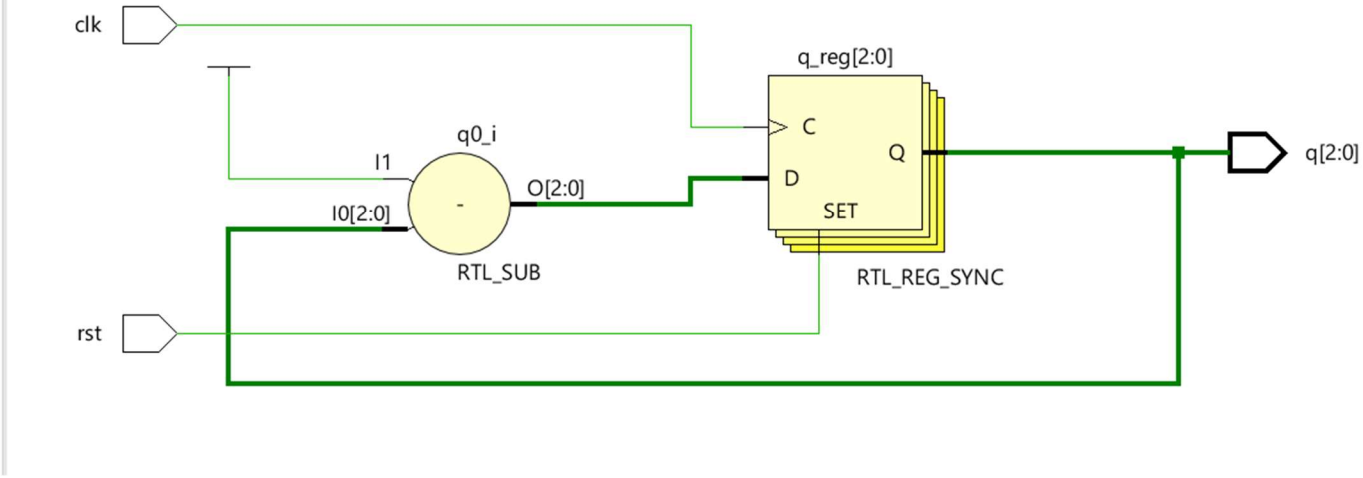


DOWN COUNTER

```
module down_counter_3b (
    input clk,
    input rst,
    output reg[2:0] q
);
    always @(posedge clk) begin
        if(rst)
            q=7;
        else
            q = q-1;
    end
endmodule

module down_counter_3b_tb ();
    reg clk=0,rst;
    wire [2:0] q;
    down_counter_3b dut (clk,rst,q);
    always #5 clk = ~clk;
    initial begin
        rst = 1; #10;
        rst = 0;
    end
endmodule
```

OUTPUT



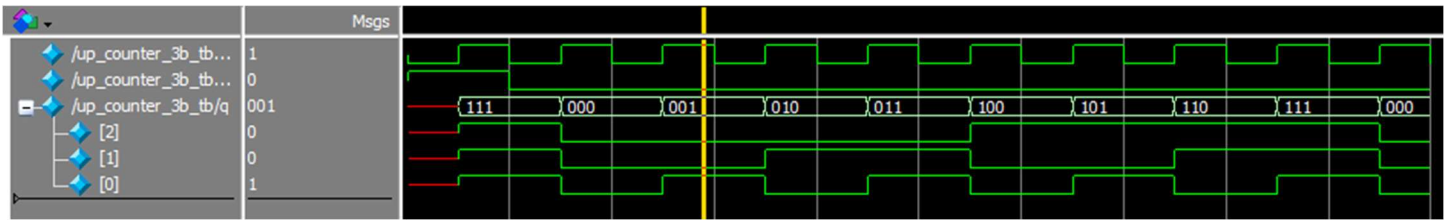


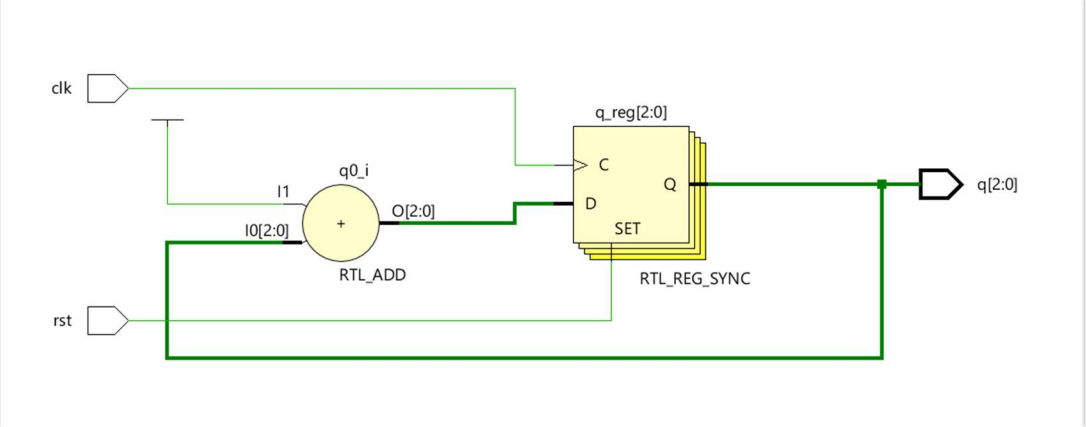
UP COUNTER

```
module up_counter_3b (
    input clk,
    input rst,
    output reg[2:0] q
);
    always @(posedge clk) begin
        if(rst)
            q=7;
        else
            q = q + 1;
    end
endmodule

module up_counter_3b_tb();
    reg clk=0,rst;
    wire [2:0] q;
    up_counter_3b dut (clk,rst,q);
    always #5 clk = ~clk;
    initial begin
        rst = 1; #10;
        rst = 0;
```

```
    end
endmodule
```

OUTPUT



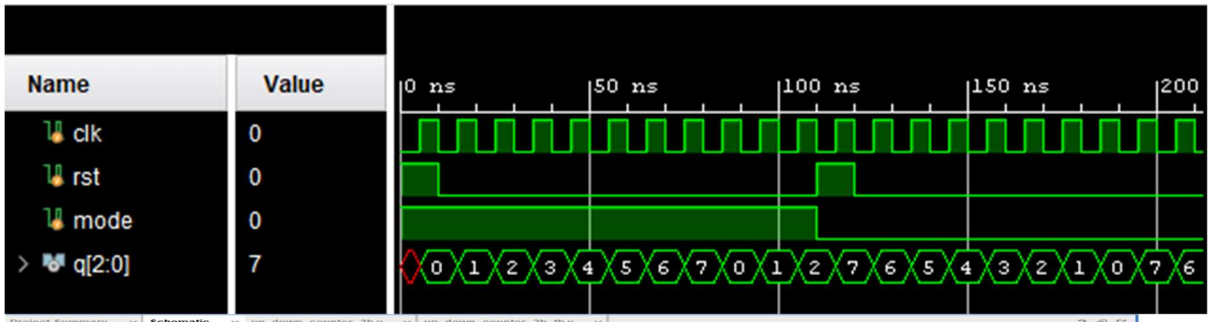


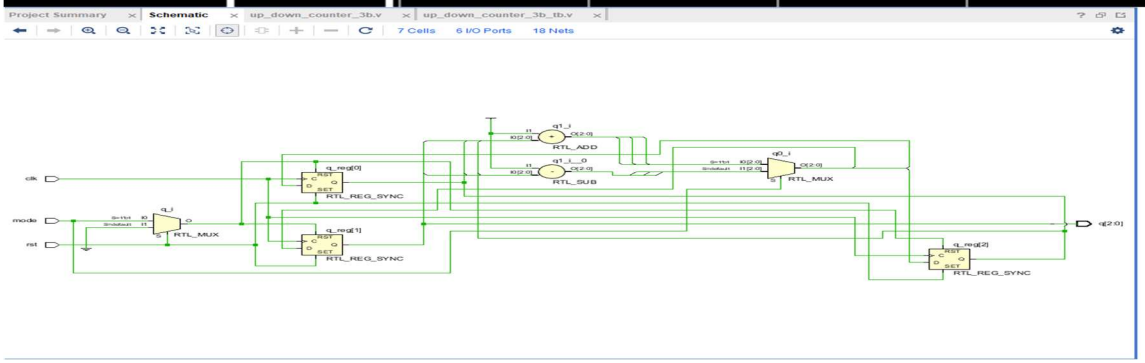
UP DOWN COUNTER

```
module up_down_counter_3b (
    input clk,
    input rst,
    input mode,
    output reg[2:0] q
);
    always @(posedge clk) begin
        if(rst)
            q = mode ? 0 : 7;
        else
            q = mode ? q + 1 : q - 1;
    end
endmodule

module up_down_counter_3b_tb();
```

```
    reg clk=0,rst,mode=1;
    wire [2:0] q;
    up_down_counter_3b dut (clk,rst,mode,q);
    always #5 clk = ~clk;
    initial begin
        rst = 1; #10;
        rst = 0; mode=1; #100;
        rst = 1; mode=0; #10;
        rst = 0; mode=0; #100;
    end
endmodule
```

OUTPUT



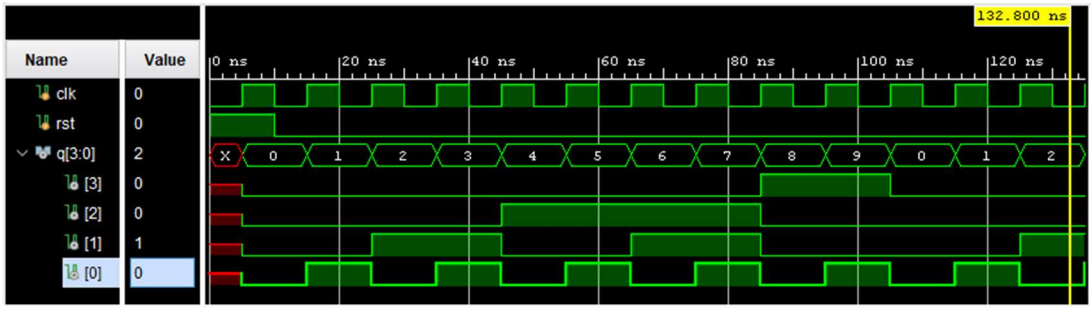


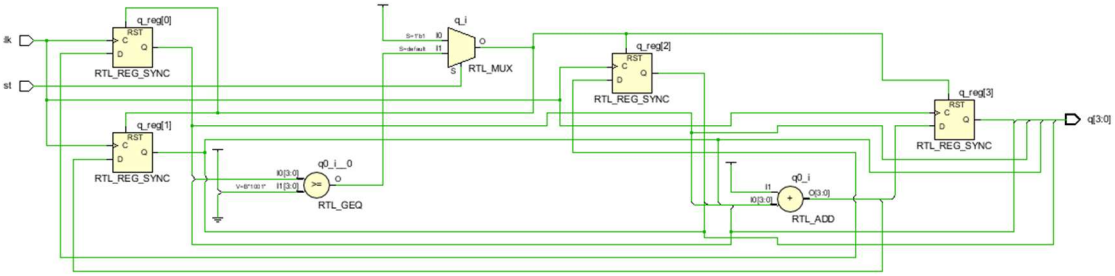
DECADE COUNTER

```
module decade_counter (
    input clk,
    input rst,
    output reg[3:0] q
);
    always @(posedge clk ) begin
        if (rst)
            q <= 0;
        else begin
            if(q > 4'b1001)
                q <= 0;
            else
                q <= q+1;
        end
    end
```

```
endmodule

module decade_counter_tb ();
    reg clk=0, rst;
    wire[3:0] q;
    decade_counter dut (clk,rst,q);
    always #5 clk = ~clk;
    initial begin
        rst = 1; #10;
        rst = 0; #10;
    end
endmodule
```

OUTPUT





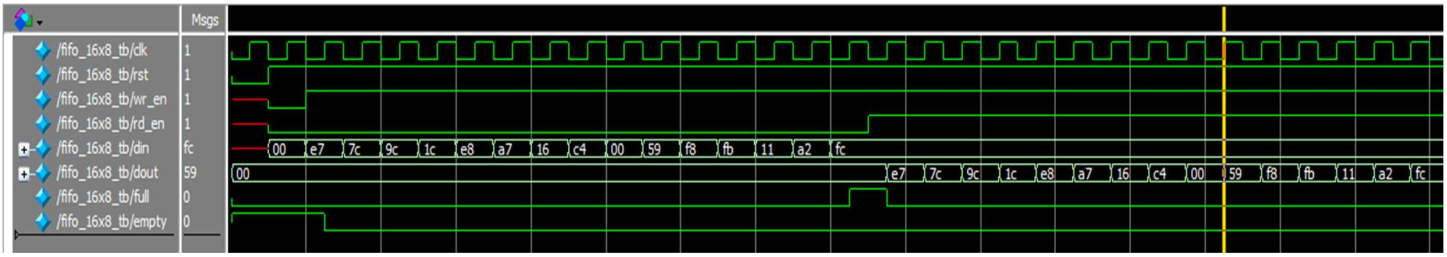
FIFO

```
module fifo_16x8(
input wire clk,
    input wire rst,
    input wire wr_en,
    input wire rd_en,
    input wire [7:0] din,
    output reg [7:0] dout,
    output wire full,
    output wire empty
);

    reg [7:0] fifo_mem [15:0];
    reg [3:0] wr_ptr = 0;
    reg [3:0] rd_ptr = 0;
    reg [4:0] count = 0;
```

```
    // Write Operation
    always @(posedge clk or negedge rst) begin
        if (!rst) begin
            wr_ptr <= 0;
            dout <= 8'b0;
        end else if (wr_en && !full) begin
            fifo_mem[wr_ptr] <= din;
            wr_ptr <= wr_ptr + 1;
        end
    end

    // Read Operation
    always @(posedge clk or negedge rst) begin
        if (!rst) begin
            rd_ptr <= 0;
            dout <= 8'b0;
        end else if (rd_en && !empty) begin
            dout <= fifo_mem[rd_ptr];
            rd_ptr <= rd_ptr + 1;
        end
    end

    assign full = (count == 15);
    assign empty = (count == 0);

    // Count Logic
    always @(posedge clk or negedge rst) begin
        if (!rst) begin
            count <= 0;
        end else if (wr_en && !full && rd_en && !empty) begin
            count <= count;   // When both read and write occur simultaneously
        end else if (wr_en && !full) begin
            count <= count + 1;  // Increment count on write
        end else if (rd_en && !empty) begin
            count <= count - 1;  // Decrement count on read
        end
    end
endmodule

module fifo_16x8_tb();
    reg clk=0,rst;
    reg wr_en,rd_en;
    reg [7:0] din;
    wire [7:0] dout;
    wire full,empty;

    fifo_16x8 dut(clk,rst,wr_en,rd_en,din,dout,full,empty);
    always #5 clk = ~clk;
    integer i,j;
```

```
    initial begin
        rst = 0; #10;
        rst = 1;
        wr_en = 0; rd_en = 0; din = 8'b0;
        #10;
        wr_en =1;
        for (i = 0; i<15; i=i+1) begin
            din = $urandom_range(0, 255);
          #10;
        end
        rd_en = 1;
        for (j = 0; j<15; j=j+1) begin
            $display("D[%d]: %h", j, dout);
          #10;
        end
    end
endmodule
```

OUTPUT



16x8 RAM SINGLE PORT

```
module ram_16x8_SP (
    input clk,
    input rst,
    input RW_en,
    input [3:0] addr,
    input [7:0] din,
    output reg [7:0] dout
);

    reg [7:0] mem [15:0];
    integer i;
    always @(posedge clk) begin
        if(!rst) begin
            for (i = 0; i<16; i=i+1) begin
                mem[i] = 0;
```

```
            end
        end
        else begin
            if(!RW_en) begin
                mem[addr] = din;
            end
            else begin
                dout = mem[addr];
            end
        end
    end
endmodule

module ram_16x8_SP_tb ();
    reg clk=0,rst;
    reg RW_en;
    reg [3:0] addr;
    reg [7:0] din;
    wire [7:0] dout;
    ram_16x8_SP dut(clk,rst,RW_en,addr,din,dout);
    always #5 clk = ~clk;
    integer i;
    initial begin
        rst = 0; #10;
        rst = 1;
        RW_en = 0;
        for (i = 0; i<16; i=i+1) begin
            addr = i;
            din = $urandom_range(0, 255);
        #10;
        end
        RW_en = 1;
        for (i = 0; i<16; i=i+1) begin
            addr = i;
        #10;
        end
    end
endmodule
```

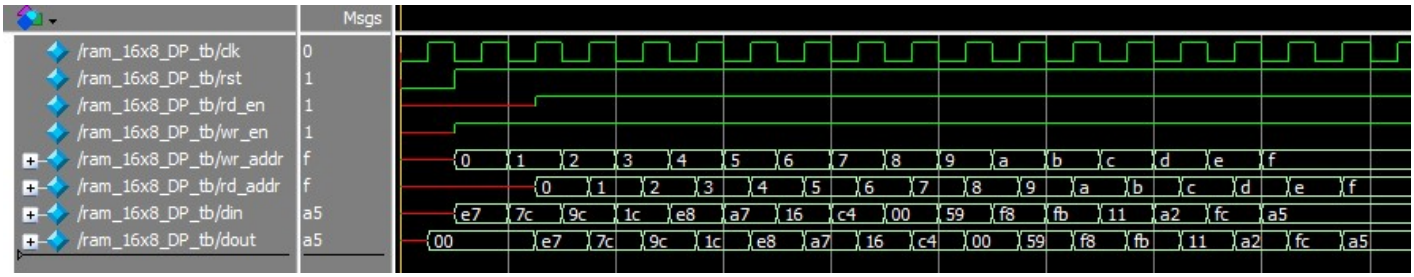
OUTPUT

16x8 RAM DUAL PORT

```
module ram_16x8_DP (
    input clk,
    input rst,
    input rd_en,
    input wr_en,
    input [3:0] wr_addr,
    input [3:0] rd_addr,
    input [7:0] din,
    output reg [7:0] dout
);
    integer i;
    reg [7:0] mem [15:0];

    always @(posedge clk) begin
        if(!rst) begin
            dout <= 0;
            for (i = 0; i<16; i=i+1) begin
                mem[i] <= 0;
            end
        end
        else  begin
            if(wr_en) begin
                mem[wr_addr] <= din;
            end
        end
    end
    always @(posedge clk) begin
        if (!rst) begin
            dout <= 0;
            for (i = 0; i<16; i=i+1) begin
                mem[i] <= 0;
            end
        end
        else begin
            if(rd_en) begin
                dout <= mem[rd_addr];
            end
        end
    end
endmodule

module ram_16x8_DP_tb ();
    reg clk=0,rst;
    reg rd_en,wr_en;
    reg [3:0] wr_addr, rd_addr;
    reg [7:0] din;
    wire [7:0] dout;
```

```
    ram_16x8_DP dut(clk,rst,rd_en,wr_en,wr_addr,rd_addr,din,dout);
    always #5 clk = ~clk;
    integer i,j;
    initial begin
        rst = 0; #10;
        rst = 1;
        wr_en = 1;
        for (i = 0; i<16; i=i+1) begin
            wr_addr = i;
            din = $urandom_range(0, 255);
            #10;
        end
    end
    initial begin
        #25;
        rd_en = 1;
        for (j = 0; j<16; j=j+1) begin
            rd_addr = j;
            #10;
            $display("mem[%0h]: %0h", rd_addr, dout);
        end
    end
endmodule
```

OUTPUT



SQUARE OF A NUMBE (TASK)

```
module square_task;

  reg [31:0] num;
  reg [31:0] result;

  // Task to calculate square
  task square;
    input [31:0] in;
    output [31:0] out;
    begin
      out = in * in;
    end
  endtask
```

```
  initial begin
    num = 5;  // Change this number to calculate square of other values
    square(num, result);
    $display("The square of %0d is %0d", num, result);

  end
endmodule
```

OUTPUT

```
# Errors: 0, Warnings: 0
# vsim -gui work.square_task
# Start time: 20:14:08 on Sep 24,2024
# Loading work.square_task
VSIM 28> run
# The square of 5 is 25

VSIM 29>
```

FIBONACCI SERIES (FUNCTION)

```
module fibonacci_function;

  reg [31:0] n;
  reg [31:0] fib;
  integer i;

  // Function to calculate Fibonacci number
  function [31:0] fibonacci;
    input [31:0] num;
    integer j;
    reg [31:0] a, b, temp;
    begin
      a = 0;
      b = 1;
      if (num == 0) begin
        fibonacci = a;
      end else if (num == 1) begin
        fibonacci = b;
      end else begin
        for (j = 2; j <= num; j = j + 1) begin
          temp = a + b;
          a = b;
          b = temp;
        end
```

```
        fibonacci = b;
      end
    end
  endfunction

  initial begin
    // Number of Fibonacci terms to display
    n = 10;
    // Display Fibonacci series
    $display("Fibonacci series up to %0d terms:", n);
    for (i = 0; i < n; i = i + 1) begin
      fib = fibonacci(i);
      $display("Fib(%0d) = %0d", i, fib);
    end
  end
endmodule
```

OUTPUT

```
# Loading work.fibonacci_function
VSIM 30> run
# Fibonacci series up to 10 terms:
# Fib(0) = 0
# Fib(1) = 1
# Fib(2) = 1
# Fib(3) = 2
# Fib(4) = 3
# Fib(5) = 5
# Fib(6) = 8
# Fib(7) = 13
# Fib(8) = 21
# Fib(9) = 34
```

FACTORIAL OF A NUMBER (TASK)

```
module factorial_task;

  reg [31:0] num;
  reg [31:0] result;

  // Task to calculate factorial
  task factorial;
    input [31:0] in;
    output [31:0] out;
    integer i;
    begin
```

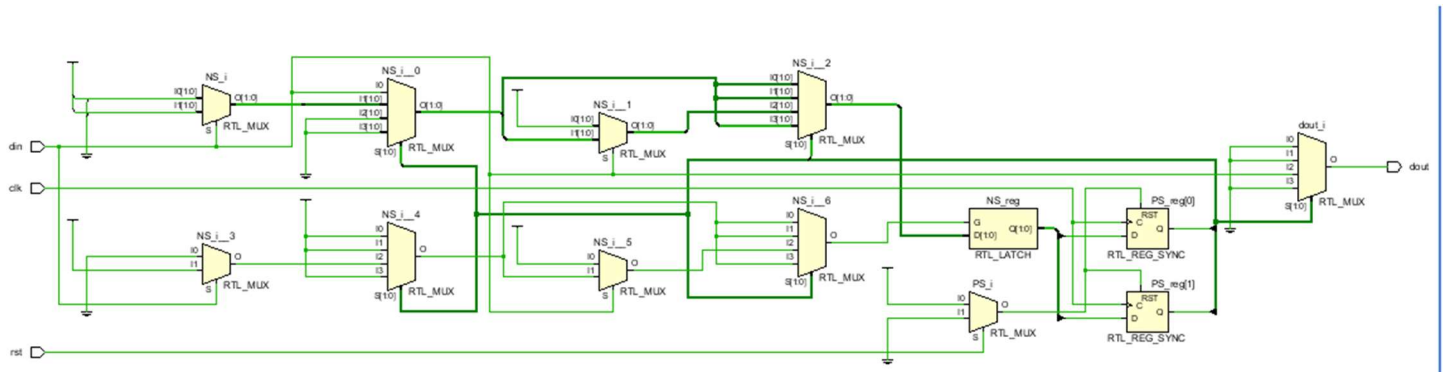
```
      out = 1;
      for (i = 1; i <= in; i = i + 1) begin
        out = out * i;
      end
    end
  endtask

  initial begin
    num = 5;
    factorial(num, result);
    $display("The factorial of %0d is %0d", num, result);
  end

endmodule
```

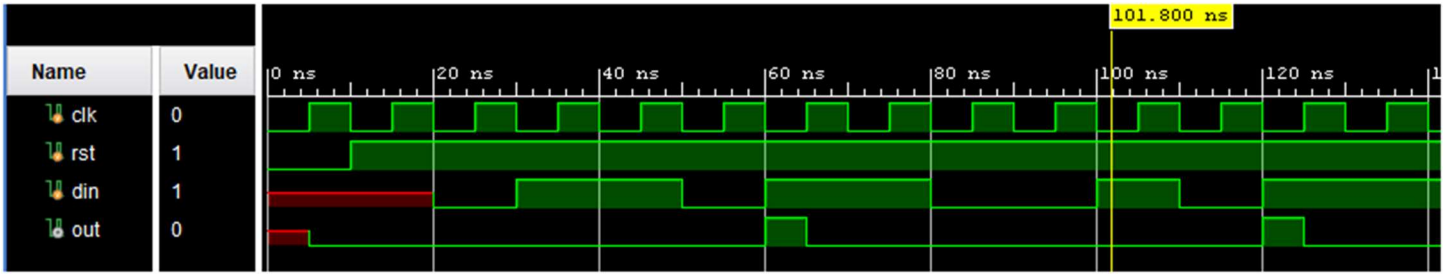
101 Sequence Detector

```
module seq_detector(
    input clk,
    input rst,
    input din,
    output reg dout
);
    parameter IDLE = 2'b00,
              S0 = 2'b01,
              S1 = 2'b10;
    reg[1:0] NS, PS;

    always @(posedge clk) begin
        if(!rst)
            PS = IDLE;
        else
            PS = NS;

    end

    always @(PS or din) begin
        dout = 0;
        case (PS)
            IDLE :  begin
                if(din)
                    NS = S0;
                else
                    NS = IDLE;
            end
            S0 : begin
                if(din)
                    NS = S0;
                else
                    NS = S1;
            end
            S1: begin
                if(din)begin
                    NS <= S0;
                    dout <= 1;
                end
                else
                    NS = IDLE;
            end
            default: NS = IDLE;
        endcase
    end
endmodule

module seq_detector_tb();
```

```
    reg clk=0,rst=1,din;
    wire out;
    seq_detector dut(clk,rst,din,out);
    always #5 clk = ~clk;
    initial begin
        rst = 0; #10;
        rst = 1; #10;

        din = 0; #10;
        din = 1; #10;
        din = 1; #10;
        din = 0; #10;
        din = 1; #10;
        din = 1; #10;
        din = 0; #10;
        din = 0; #10;
        din = 1; #10;
        din = 0; #10;
        din = 1; #10;
    end
endmodule
```

OUTPUT



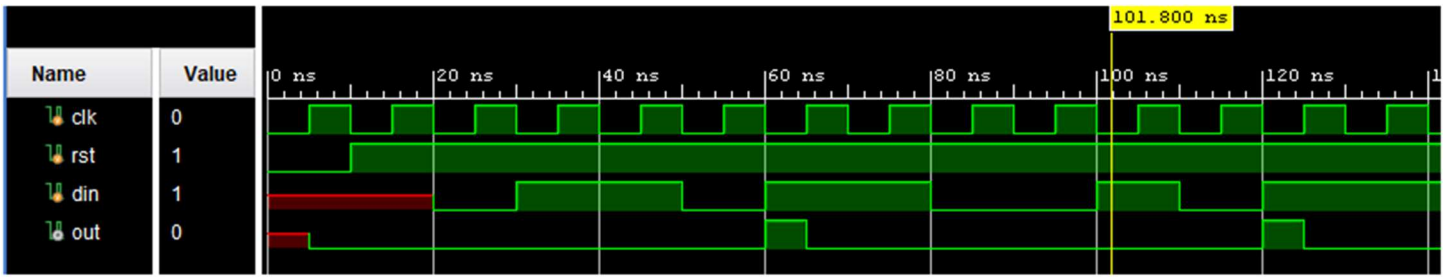


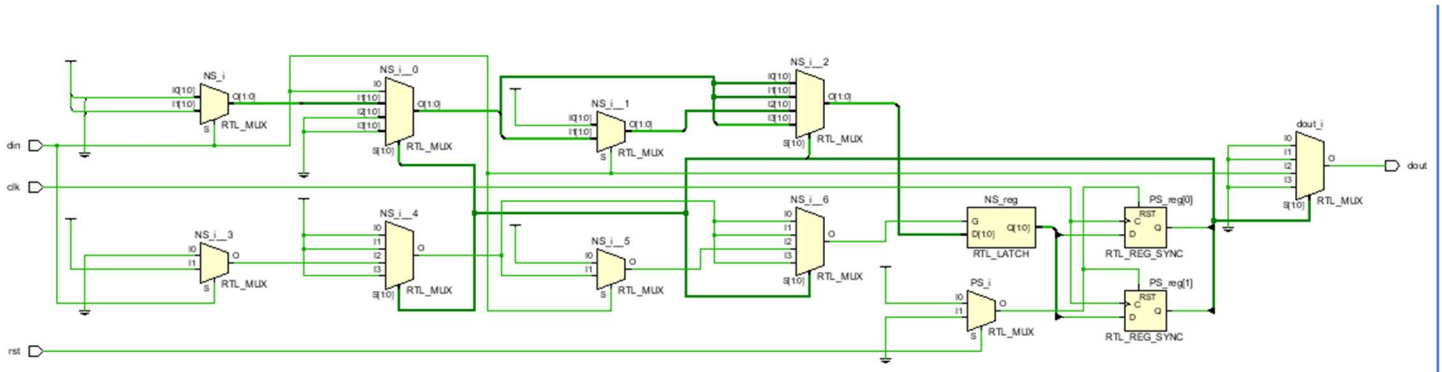
101 Sequence Detector (Moore)

```
module seq_detector_moore(
    input clk,
    input rst,
    input din,
    output reg dout
);
    parameter IDLE = 3'b000,
              S0 = 3'b001,
              S1 = 3'b010,
              S2 = 3'b101;
    reg[2:0] NS, PS;

    always @(posedge clk) begin
        if(!rst)
            PS = IDLE;
        else
            PS = NS;

    end

    always @(PS or din) begin
        dout = 0;
        case (PS)
            IDLE :  begin
                if(din)
                    NS = S0;
                else
                    NS = IDLE;
            end
            S0 : begin
                if(din)
                    NS = S0;
                else
                    NS = S1;
            end
            S1: begin
                if(din)begin
                    NS <= S2;
                end
                else
                    NS = IDLE;
            end
            S2: begin

                if(din)begin

                    NS <= IDLE;
                end
```

```
                else
                    NS = S1;
            end
            default: NS = IDLE;
        endcase
    end
    always @(PS) begin
        case (PS)
            S2: begin
                dout = 1;
            end
            default: dout = 0;
        endcase
    end
endmodule

module seq_detector_tb();
    reg clk=0,rst=1,din;
    wire out;
    seq_detector_moore dut(clk,rst,din,out);
    always #5 clk = ~clk;
    initial begin
        rst = 0; #10;
        rst = 1; #10;

        din = 0; #10;
        din = 1; #10;
        din = 1; #10;
        din = 0; #10;
        din = 1; #10;
        din = 1; #10;
        din = 0; #10;
        din = 0; #10;
        din = 1; #10;
        din = 0; #10;
        din = 1; #10;
    end
endmodule
```

OUTPUT




din
clk
rst
NS_i
RTL_MUX
NS_i__0
NS_i__1
NS_i__2
NS_i__3
NS_i__4
NS_i__5
NS_i__6
NS_reg
RTL_LATCH
PS_i
PS_reg[0]
PS_reg[1]
RTL_REG_SYNC
dout_i
dout


ACADEMY OF SCIENCE &
MAIN BLOCK