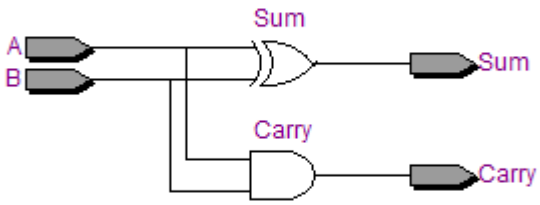
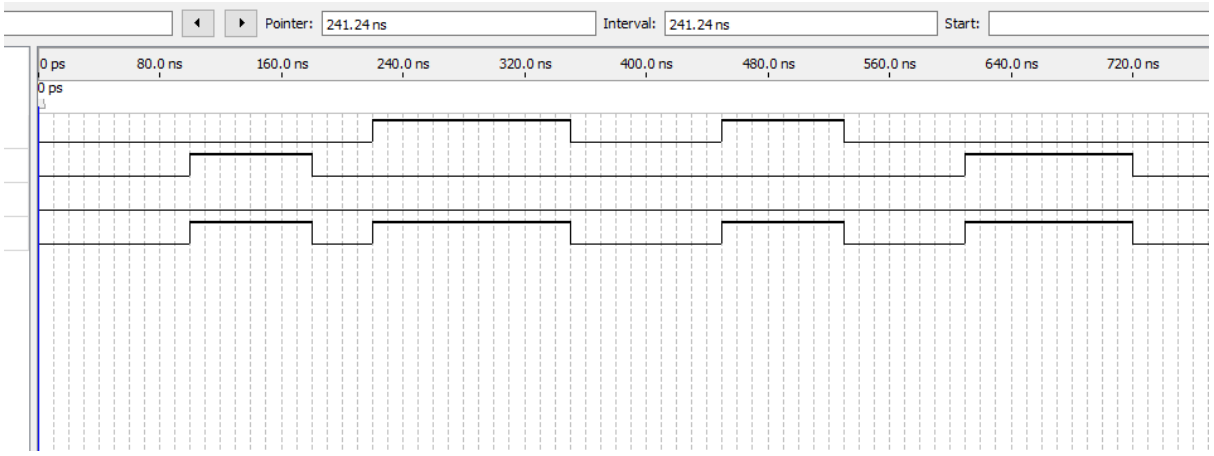
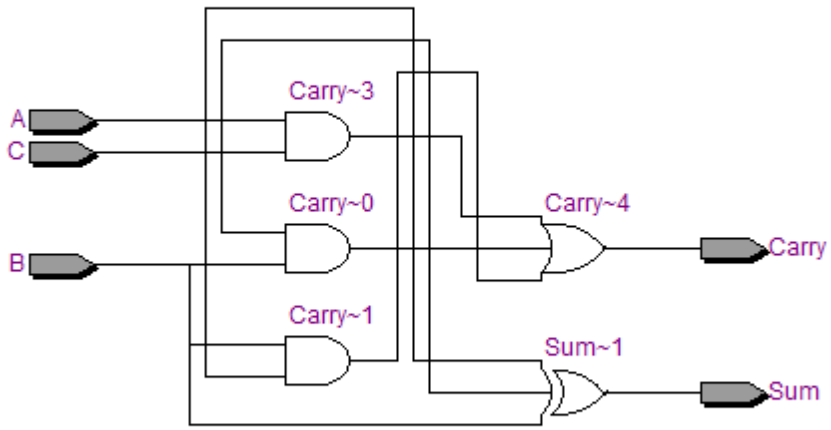
HALF ADDER



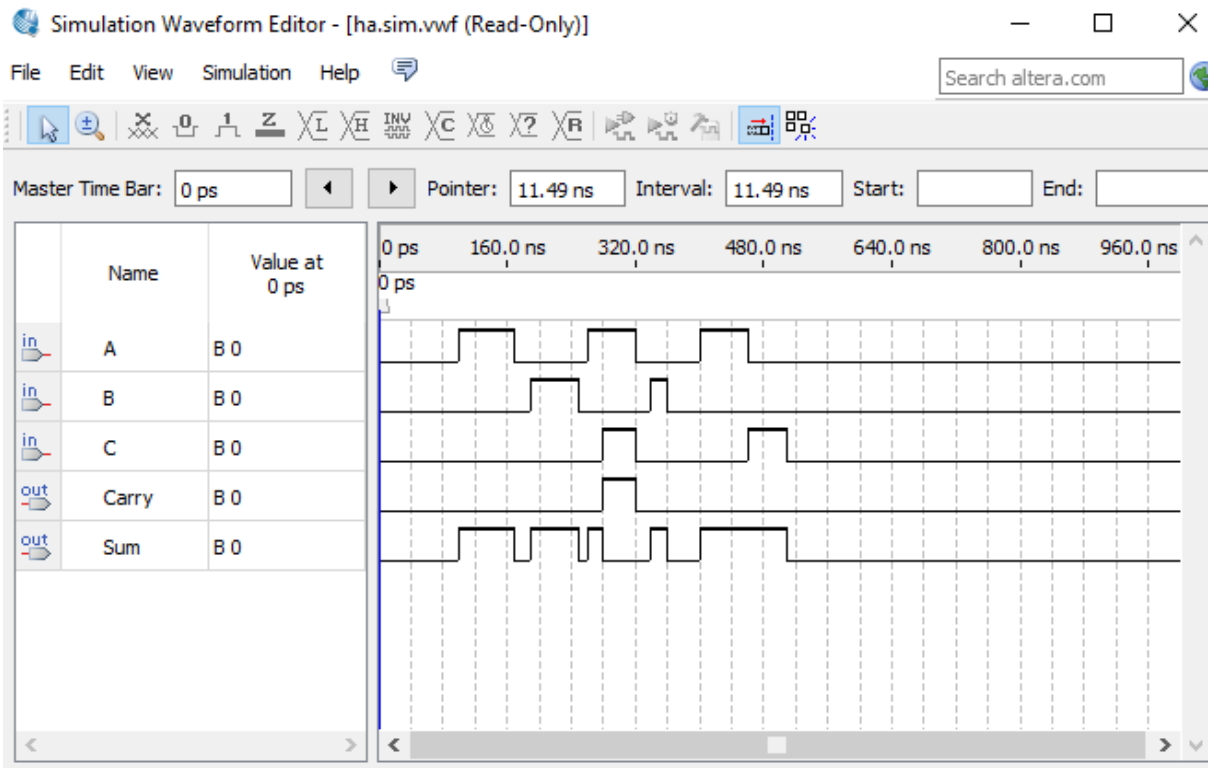

```
module ha(A,B,Sum,Carry);
input A,B;
output Sum,Carry;
assign Sum=A^B;
assign Carry=A&B;
endmodule
```



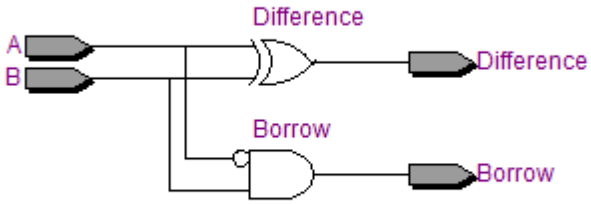
FULL ADDER



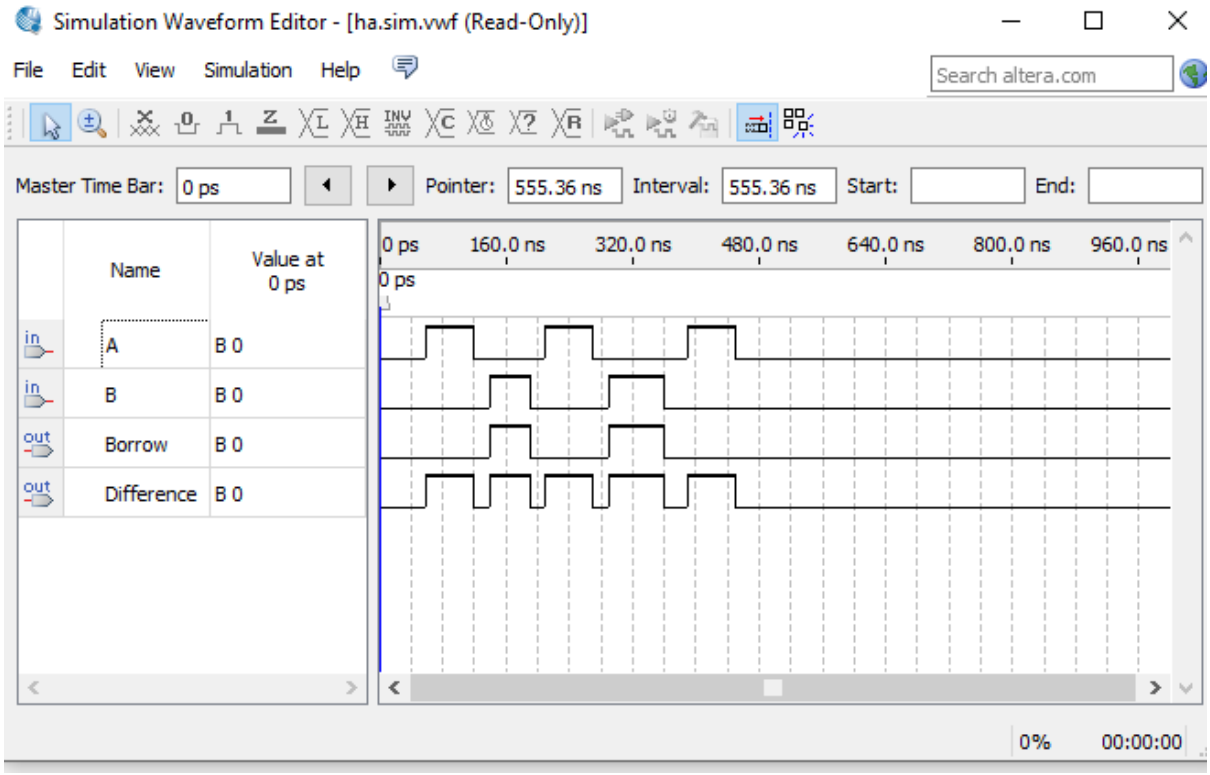

```
module ha(A,B,C,Sum,Carry);
input A,B,C;
output Sum,Carry;
assign Sum=A^B^C;
assign Carry=(A&B)|(B&C)|(C&A);
endmodule
```



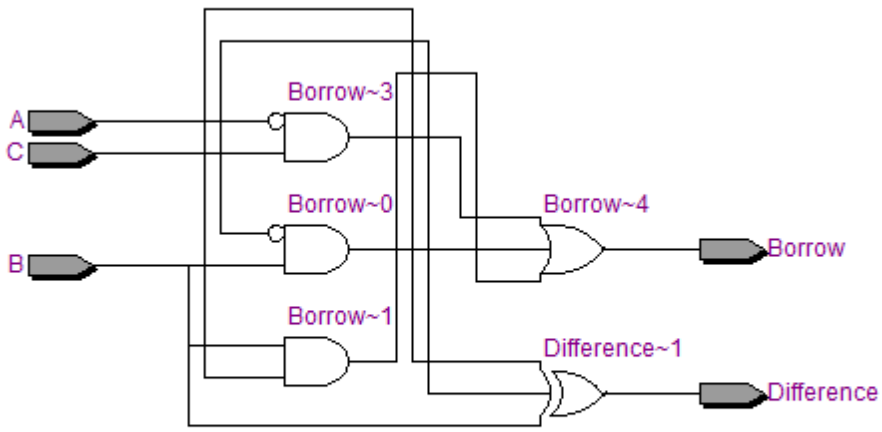
HALF SUBTRACTOR




```
module ha(A,B,Difference,Borrow);
input A,B;
output Difference,Borrow;
assign Difference=A^B;
assign Borrow= ~A&B;
endmodule
```

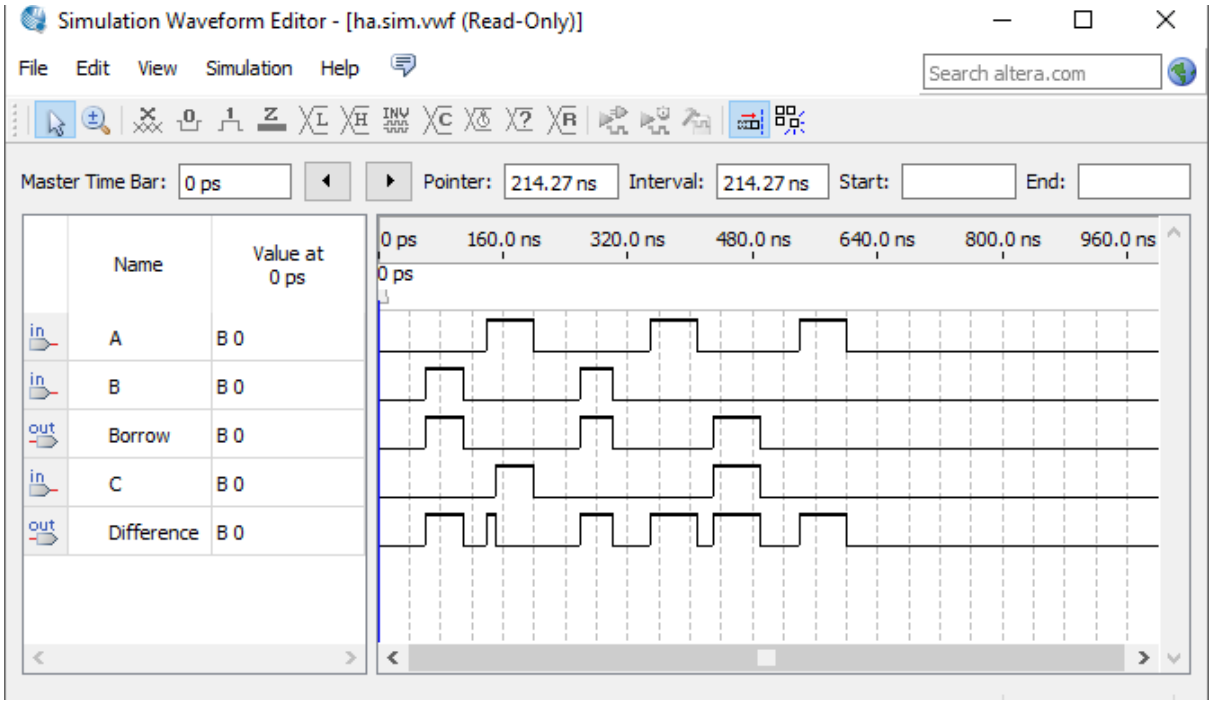


FULL SUBTRACTOR

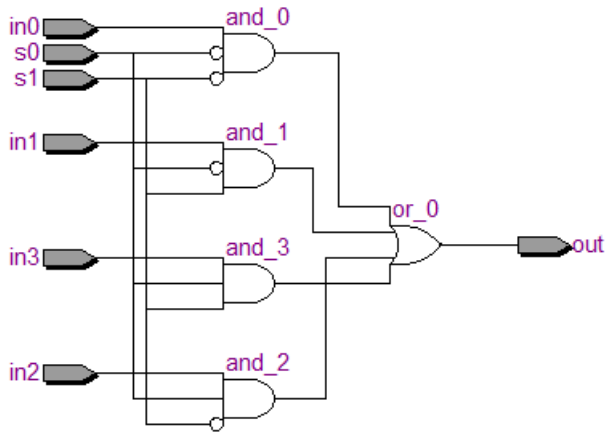


module ha(A,B,C,Difference,Borrow);

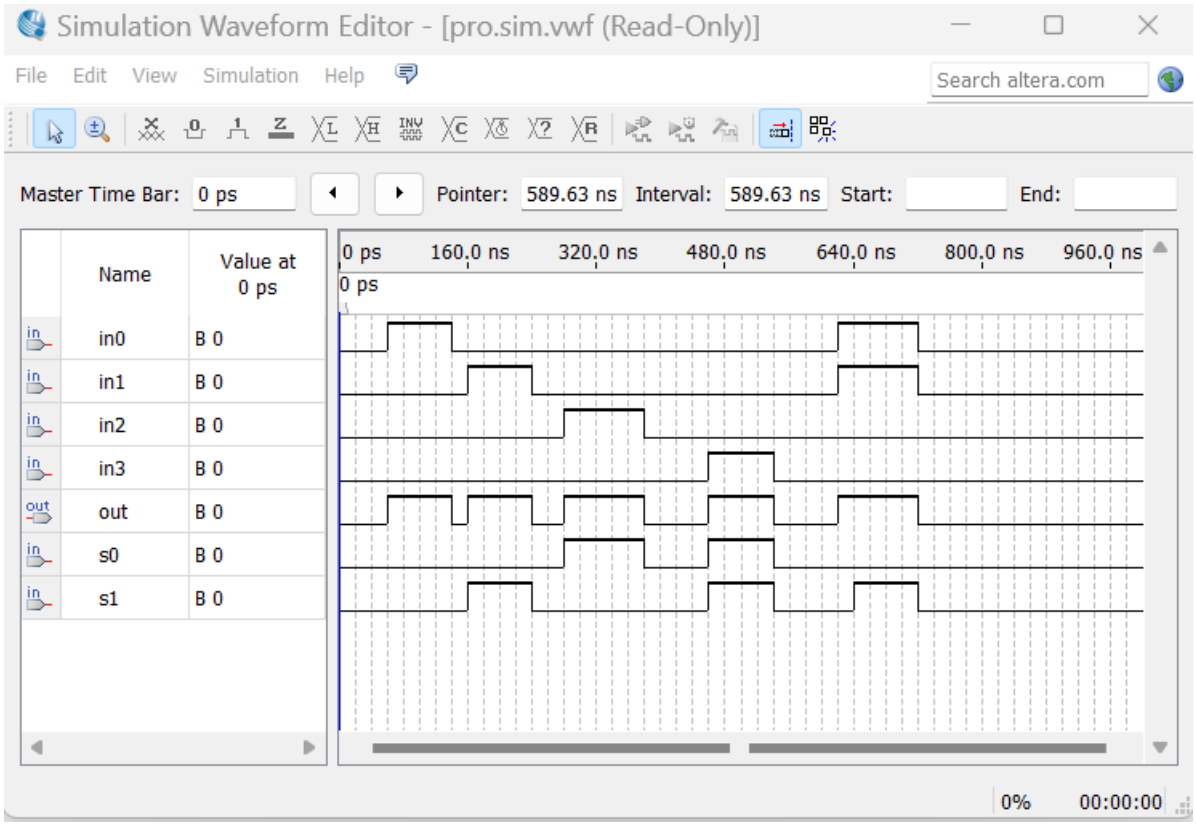
```
input A,B,C;
output Difference,Borrow;
assign Difference=A^B^C;
assign Borrow= (~A&B)|(B&C)|(C&~A);
endmodule
```



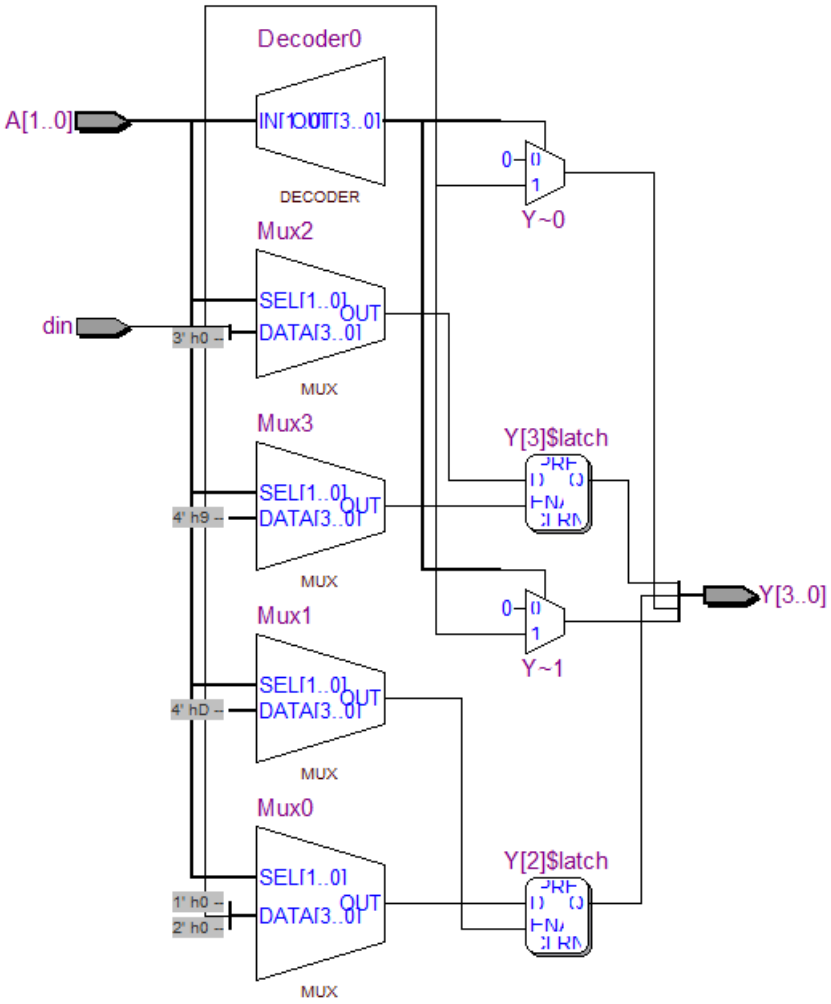
4:1 MULTIPLEXER




```
module m31(out,in0,in1,in2,in3,s0,s1);
output out;//output port
input in0,in1,in2,in3;//Input ports
input s0,s1;//Input ports:select lines
wire inv0,inv1;//Inverter Outputs
wire a0,a1,a2,a3;//AND gate outputs
not not_0(inv0,s0);
not not_1(inv1,s1);
and and_0(a0,in0,inv0,inv1);
and and_1(a1,in1,inv0,s1);
and and_2(a2,in2,s0,inv1);
and and_3(a3,in3,s0,s1);
or or_0(out,a0,a1,a2,a3);
endmodule
```

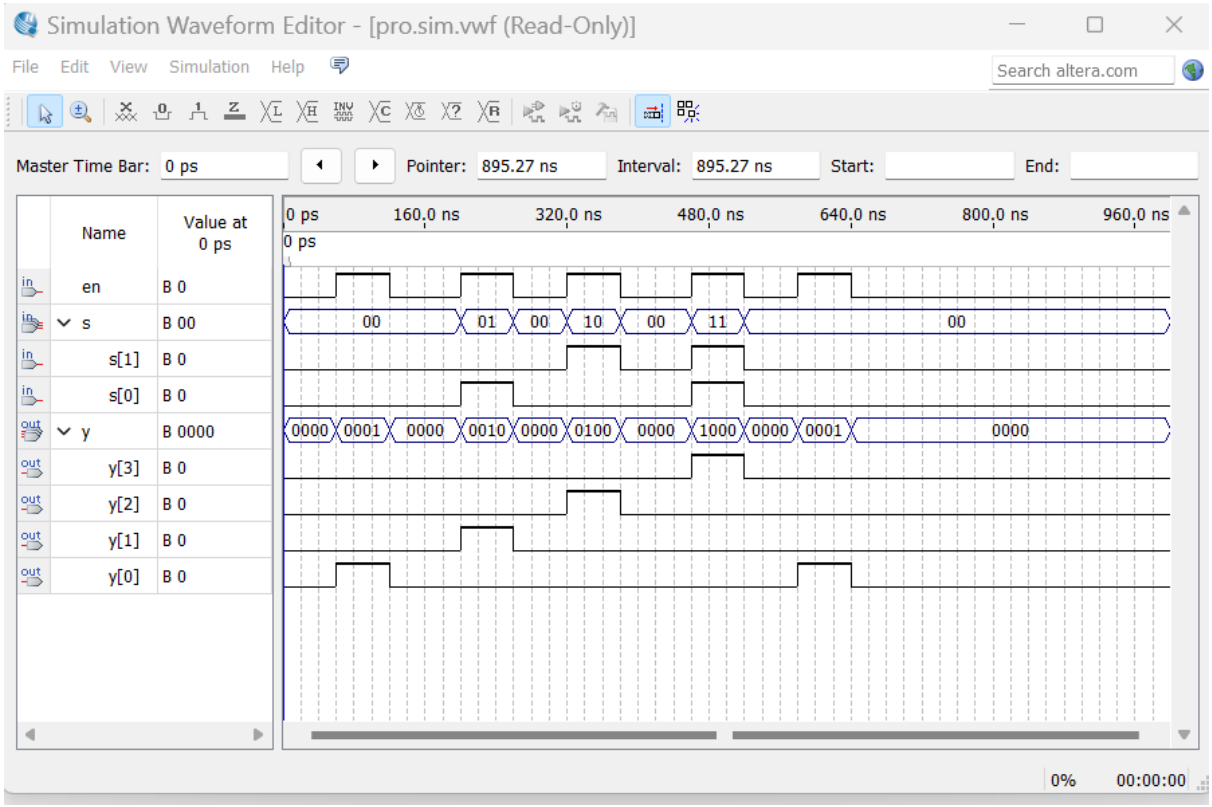


DEMULTIPLEXER

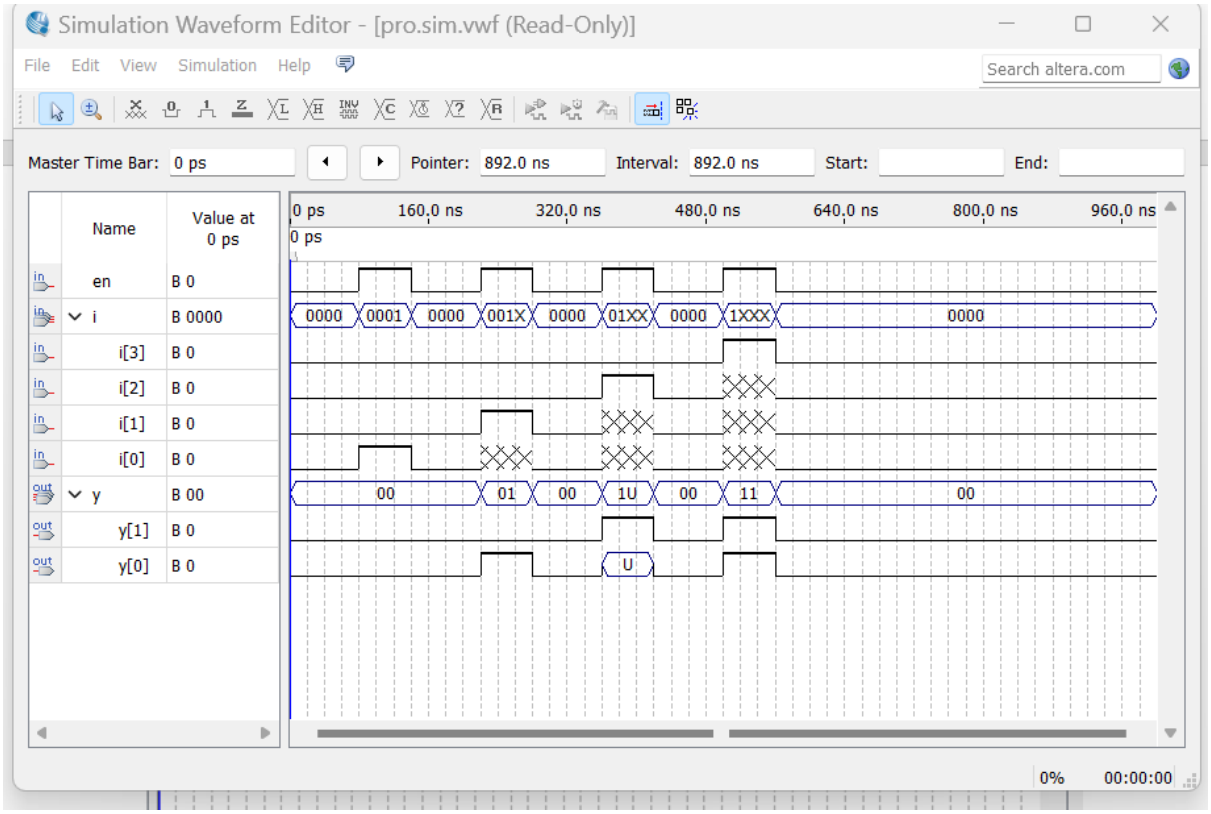



```
module pro (output reg [3:0] Y, input [1:0] A, input din);
always @(Y, A) begin
   case (A)
      2'b00 : begin Y[0] = din; Y[3:1] = 0; end
      2'b01 : begin Y[1] = din; Y[0] = 0;   end
      2'b10 : begin Y[2] = din; Y[1:0] = 0; end
      2'b11 : begin Y[3] = din; Y[2:0] = 0; end
   endcase
end
```

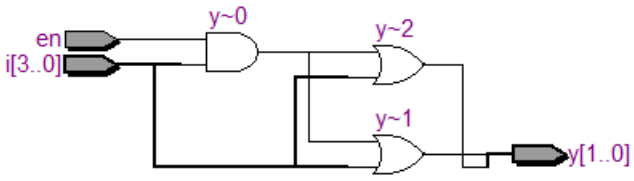
endmodule



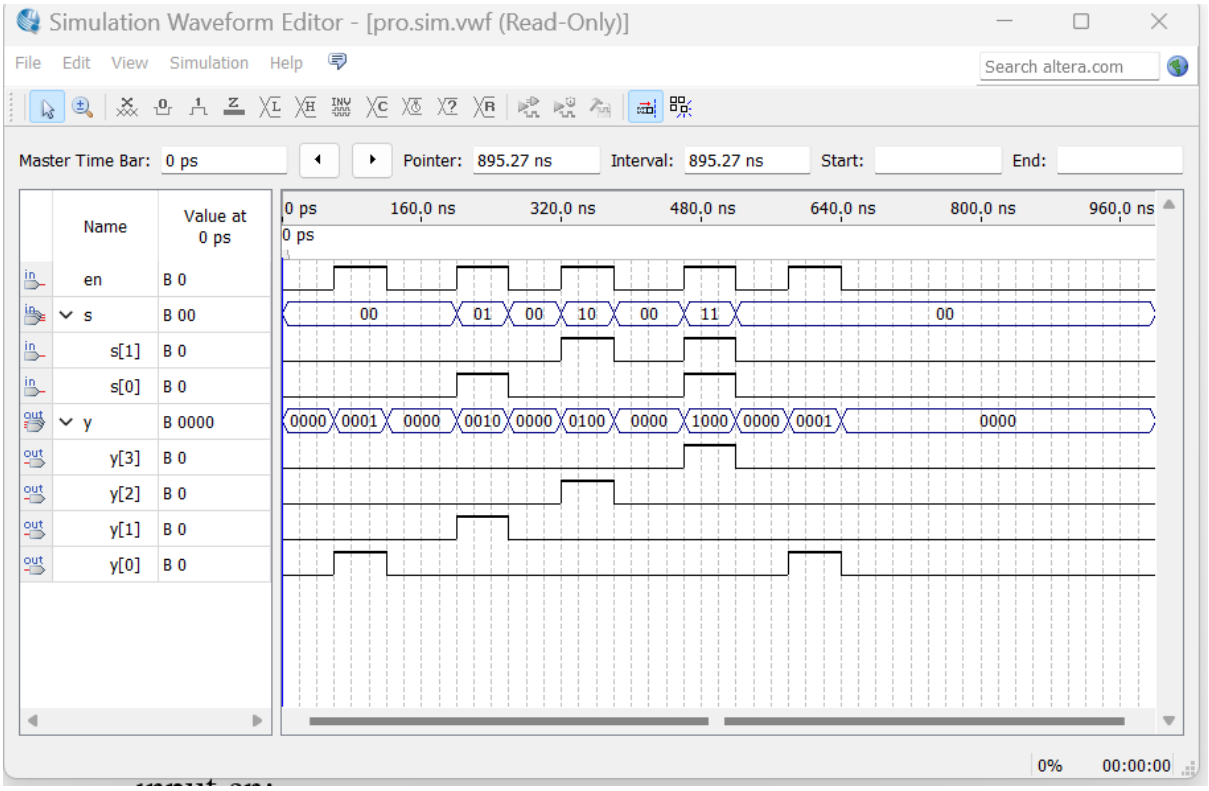
ENCODER




```
module priorityenc(en,i,y);
input en;
input [3:0]i;
output[1:0]y;
assign y[1]=i[2]|i[3]&en;
assign y[0]=i[1]|i[3]&en;
endmodule
```

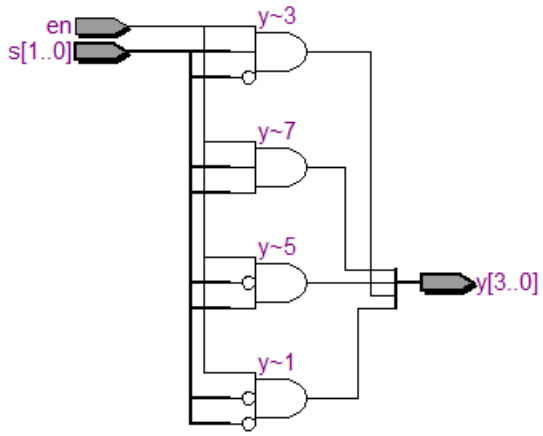


DECODER

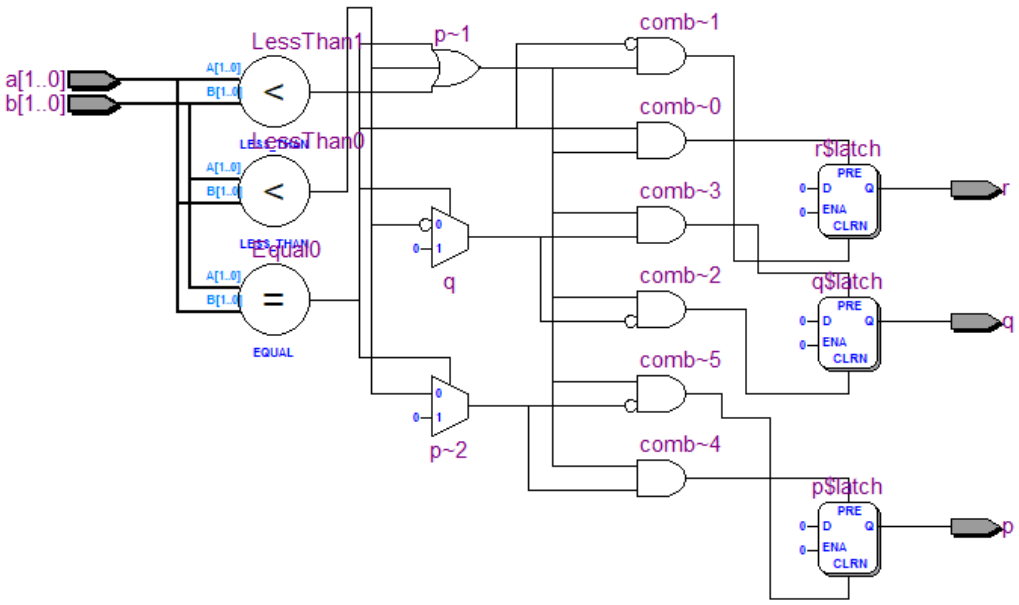



```
module decode(en,s,y);
input en;
input [1:0]s;
output[3:0]y;
```

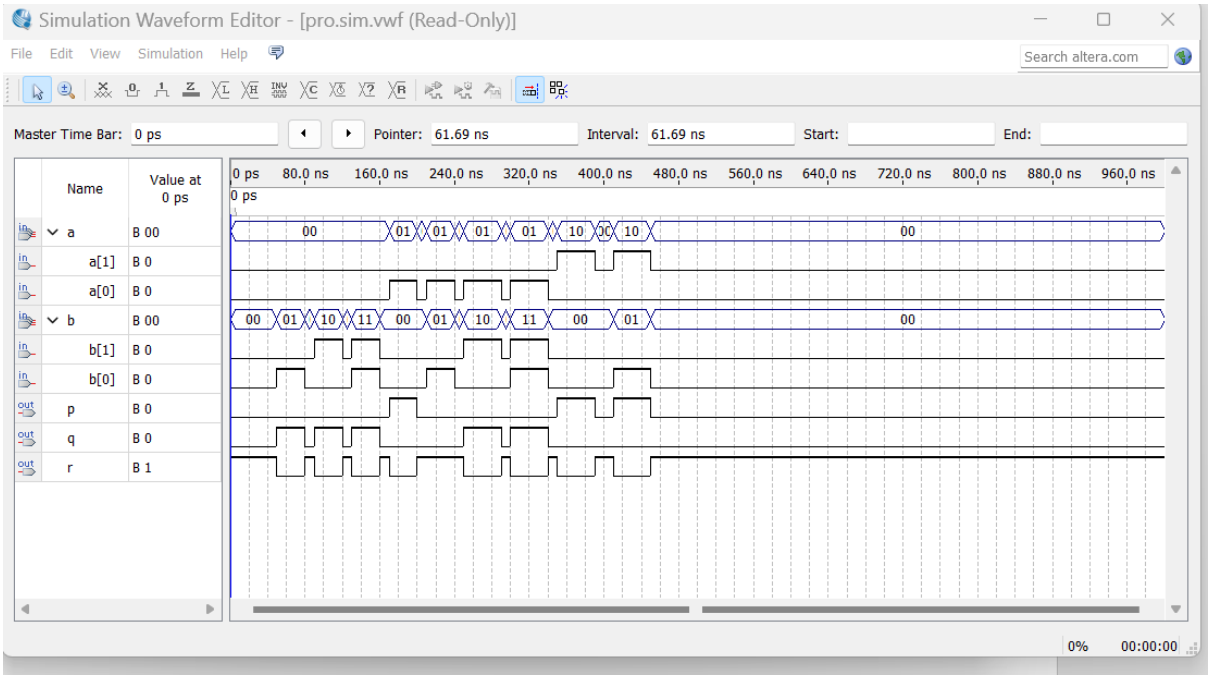
```
assign y[0]=(~s[1])&(~s[0])&en;
assign y[1]=(~s[1])&(s[0])&en;
assign y[2]=(s[1])&(~s[0])&en;
assign y[3]=(s[1])&(s[0])&en;
endmodule
```



2 bit MAGNITUDE COMPARATOR



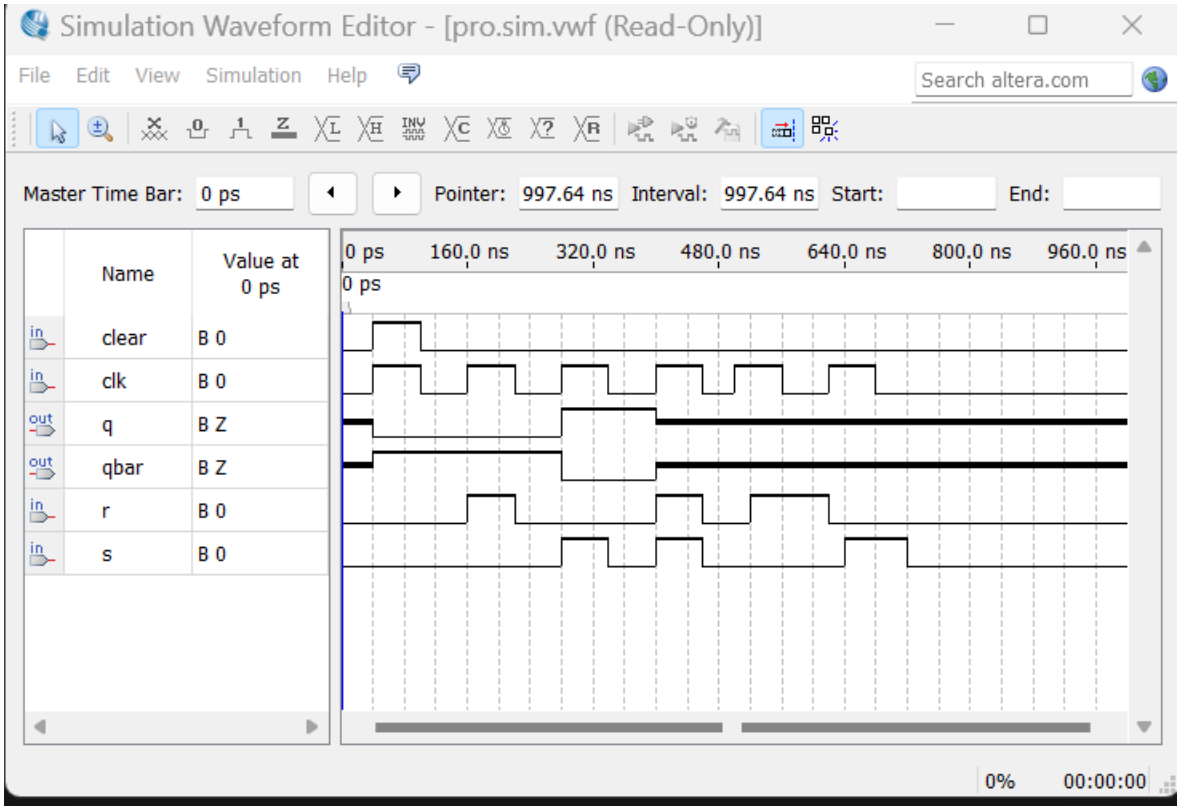

```
module pro(p,q,r,a,b);
input [1:0]a;
input [1:0]b;
output reg p,q,r;
always@(a or b)
begin
if(a===b) {p,q,r}=1'b001;
else if(a>b)  {p,q,r}=3'b100;
else if(a<b)  {p,q,r}=3'b010;
end
endmodule
```

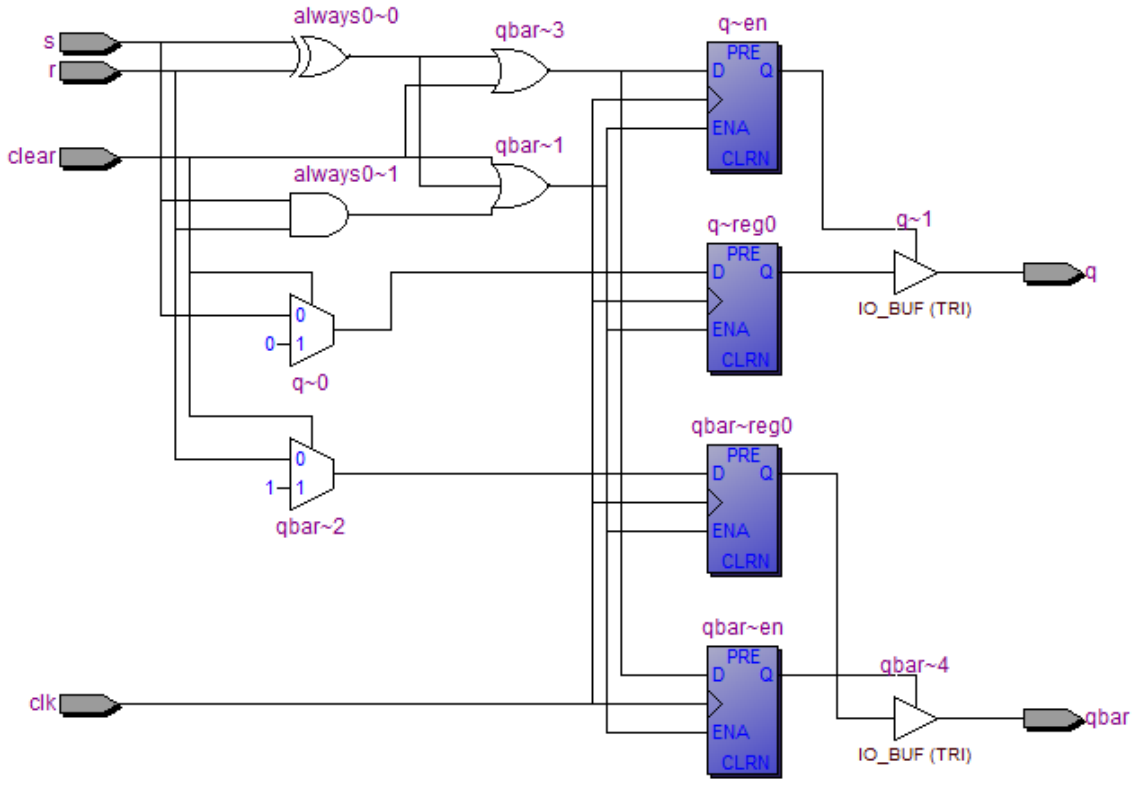


SR FLIP FLOP

```
module pro(s,r,clk,clear,q,qbar);
input s,r,clk,clear;
output reg q,qbar;
always@(posedge clk)
begin
if(clear)
begin
q=0;
qbar=1;
end
else
begin
```

```
if(s!=r)
begin
q=s;
qbar=r;
end
else if(s==1&&r==1)
begin
q=1'bz;
qbar=1'bz;
end
end
end
endmodule
```

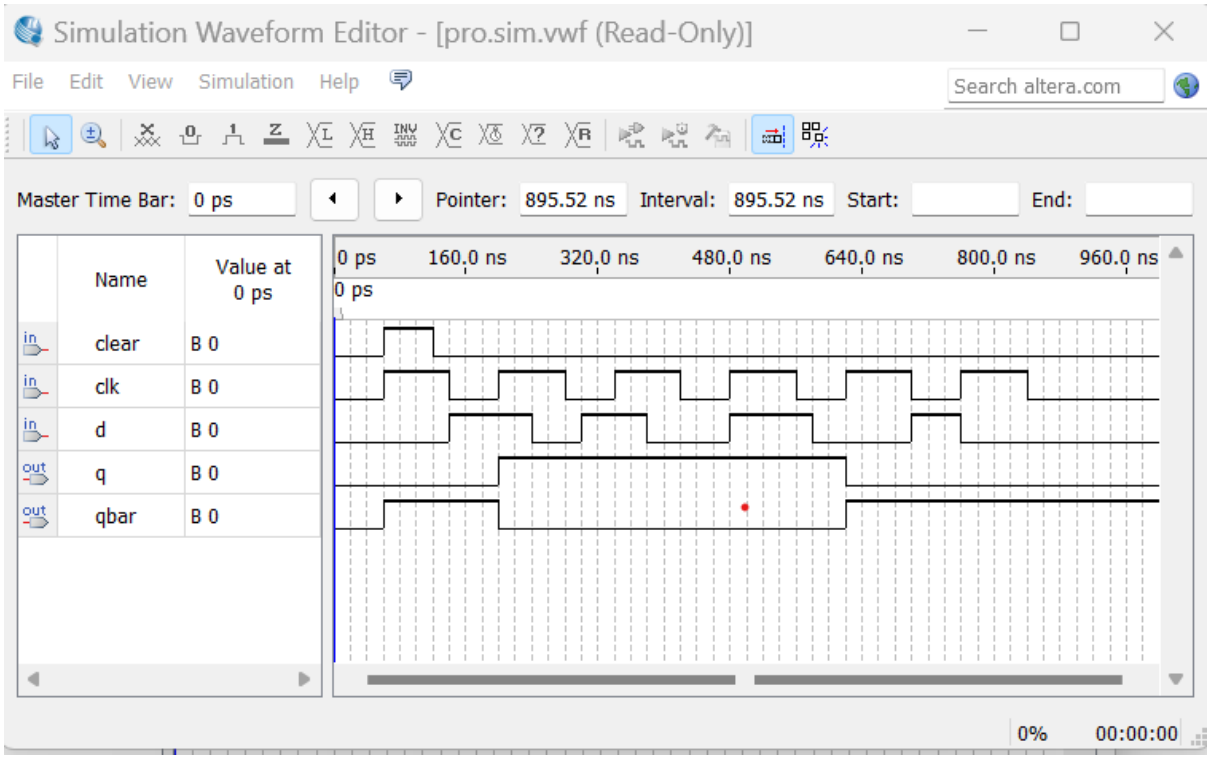

Simulation Waveform Editor - [pro.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 997.64 ns
Interval: 997.64 ns
Start:
End:
Name
Value at 0 ps
clear B 0
clk B 0
q B Z
qbar B Z
r B 0
s B 0
0 ps
160.0 ns
320.0 ns
480.0 ns
640.0 ns
800.0 ns
960.0 ns
0%
00:00:00

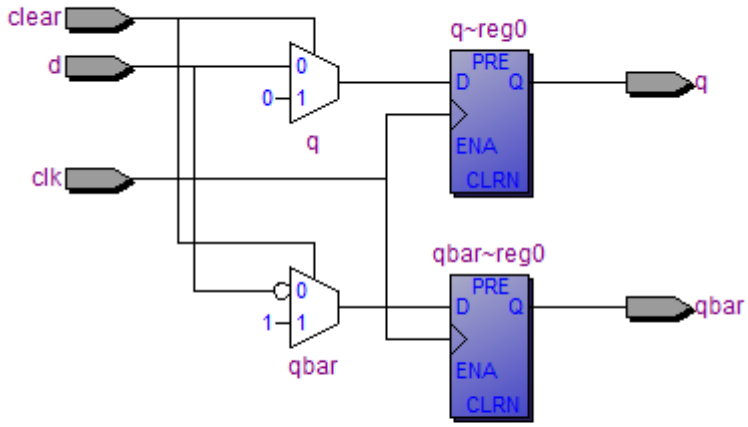


DFF

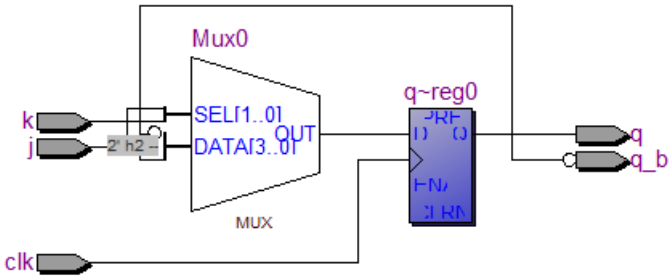
```
module pro(d,clk,clear,q,qbar);
input d,clk,clear;
output reg q,qbar;
always@(posedge clk)
begin
if(clear==1)
begin
```

```
q=0;
qbar=1;
end
else
begin
q=d;
qbar=!d;
end
end
endmodule
```





JK FLIP FLOP

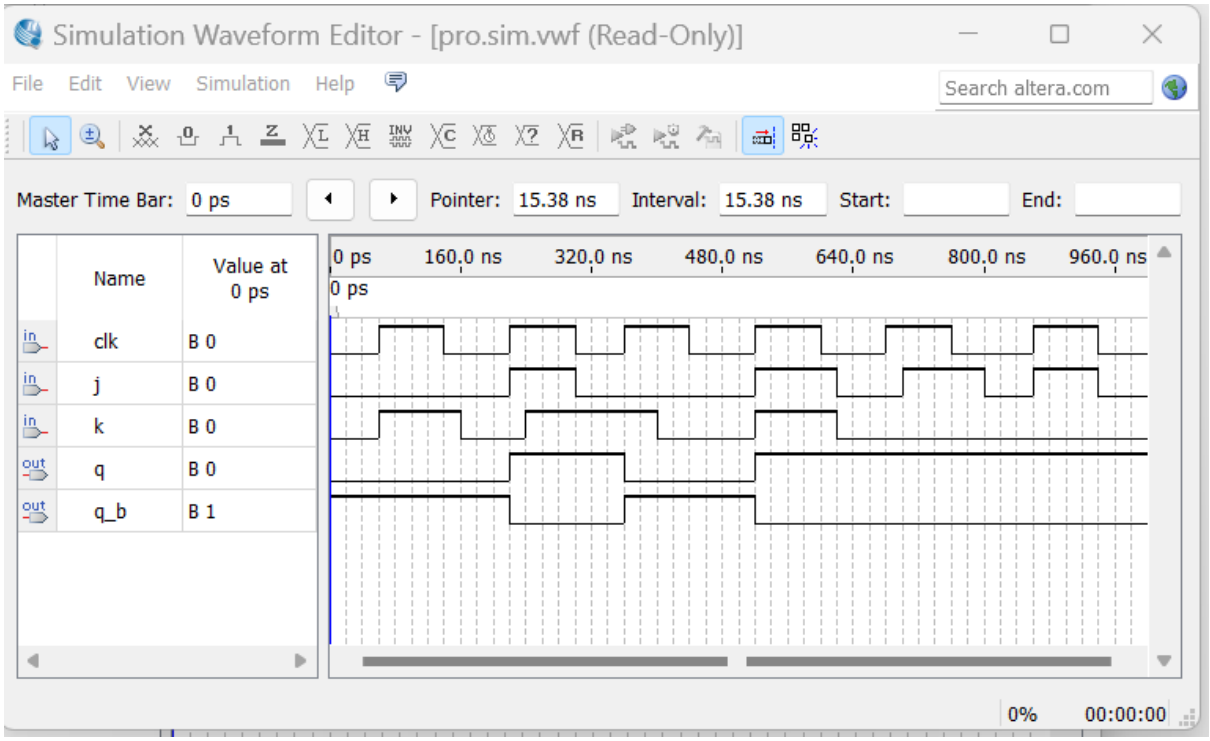



```
module pro(input j,k,clk,output reg
q,output q_b);

assign q_b=~q;

always @(posedge clk)
```

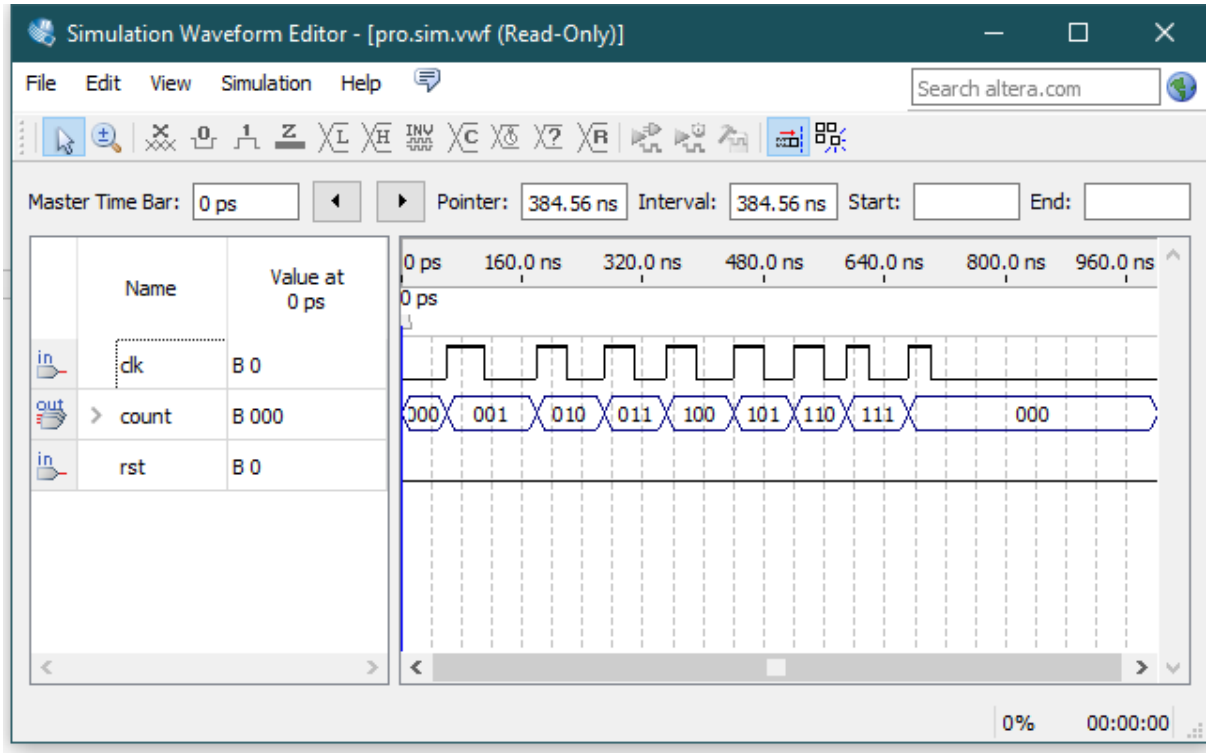
```
case ({j,k})
2'b00:q<=q;
2'b01:q<=1'b0;
2'b10:q<=1'b1;
2'b11:q<=!q;
endcase
endmodule
```

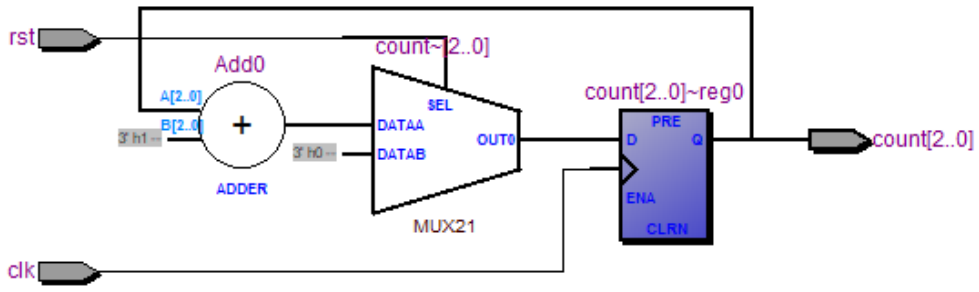


SYNCHRONOUS COUNTER

```
module  counter(clk,rst,count);
input clk,rst;
output reg[2:0]count;
```

```
always @(posedge clk)
begin
if(rst==1)
begin
count=0;
end
else
begin
count=count+1;
end
end
endmodule
```

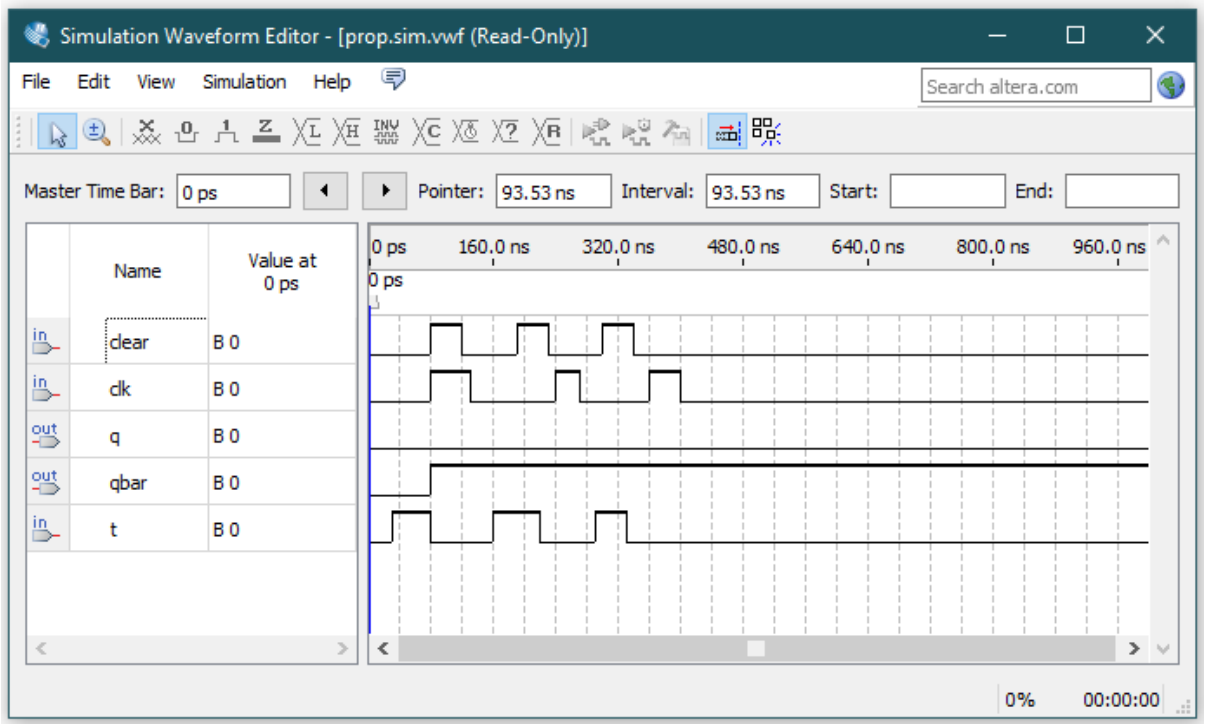


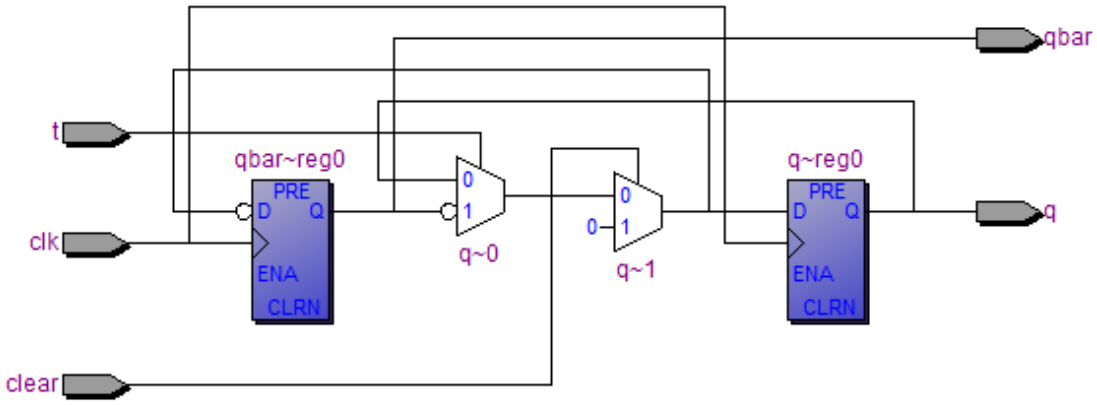


TFF

```
module pro(t,clk,clear,q,qbar);
input t,clk,clear;
output reg q,qbar;
```

```
always@(posedge clk)
begin
if(clear==1)
begin
q=0;
qbar=1;
end
else if(t==1)
begin
q=~qbar;
end
else
begin
q=q;
end
qbar=~q;
end
endmodule
```

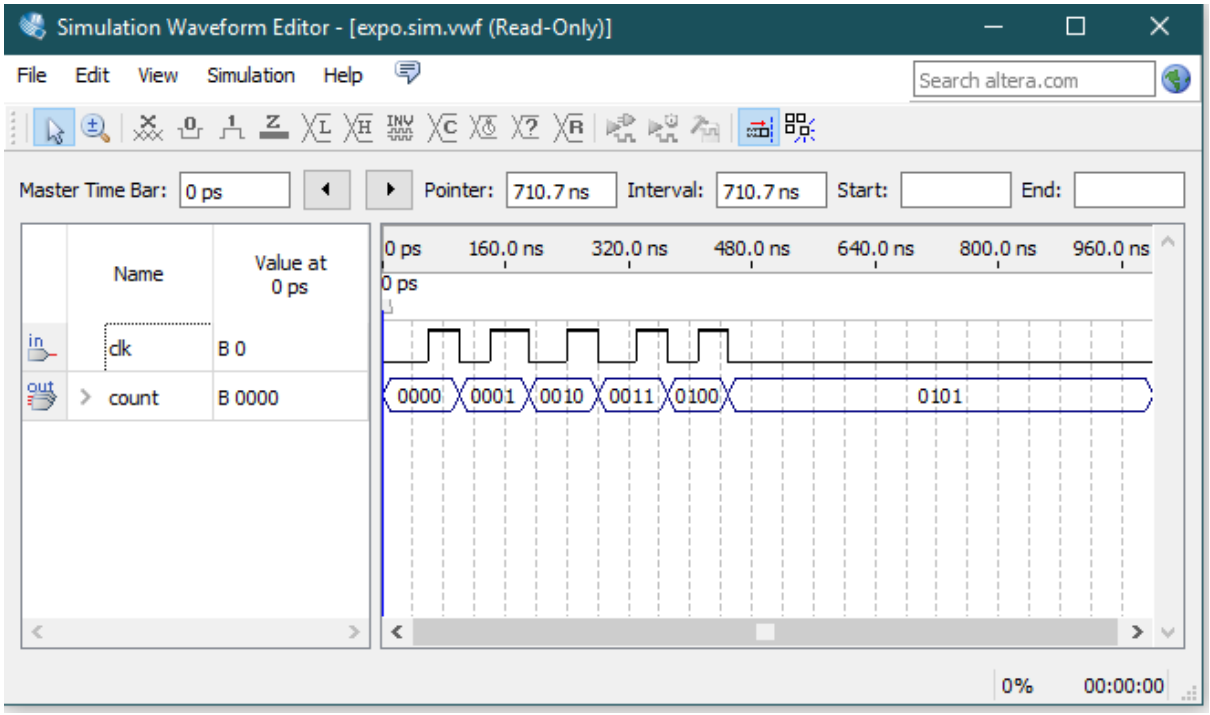


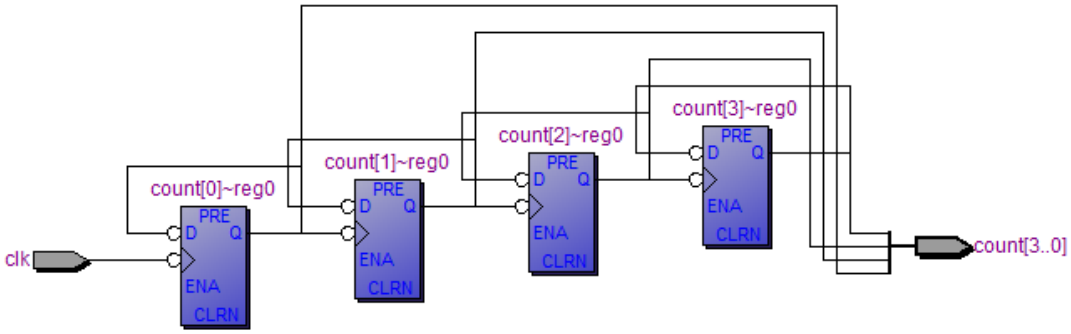


RIPPLE COUNTER

module expo(clk,count);

```
input clk;
output [3:0]count;
reg[3:0]count;
wire clk;
initial
count =4'b0000;
always@(negedge clk)
count[0]<=~count[0];
always@(negedge count[0])
count[1]<=~count[1];
always@(negedge count[1])
count[2]<=~count[2];
always@(negedge count[2])
count[3]<=~count[3];
endmodule
```



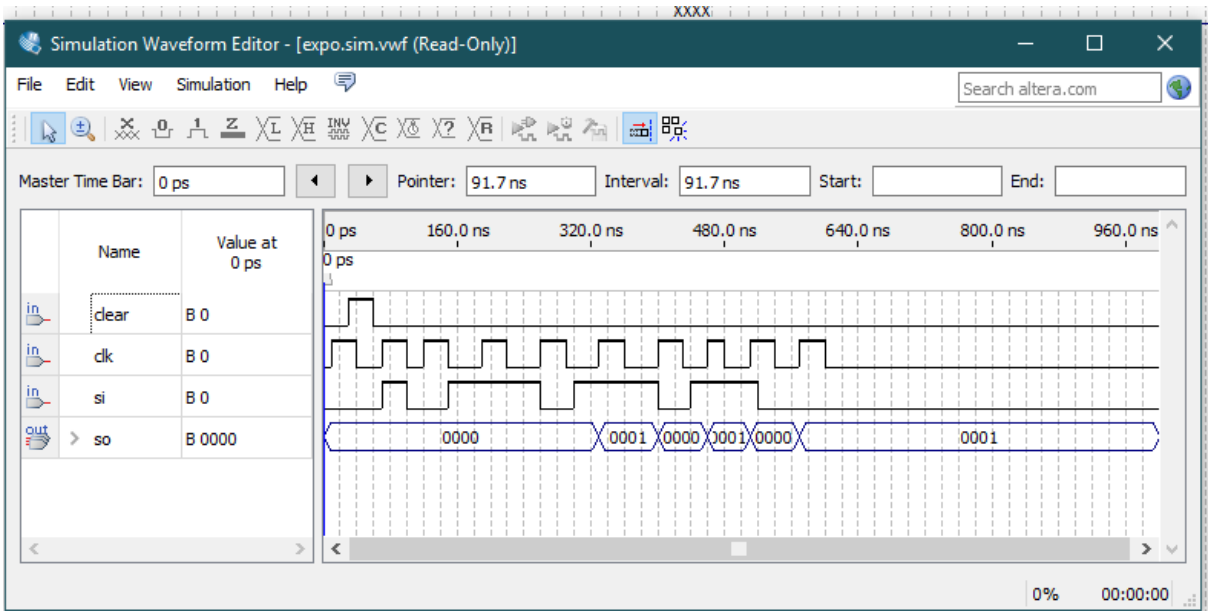


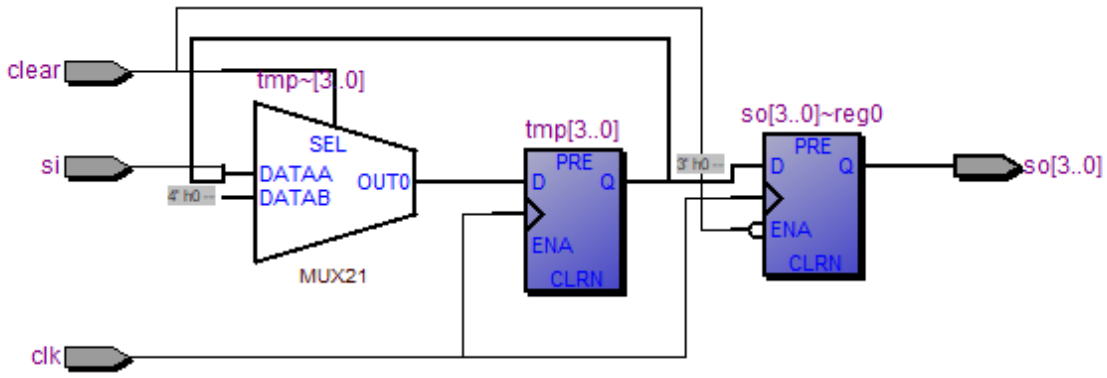
SISO Shift register

module expo (input clk,clear,si,output reg [3:0]so);

reg [3:0]tmp;

```
always @(posedge clk)
begin
if (clear)
begin
tmp<=4'b0000;
end
else
begin
tmp<=tmp<<1;
tmp[0]<=si;
so=tmp[3];
end
end
endmodule
```

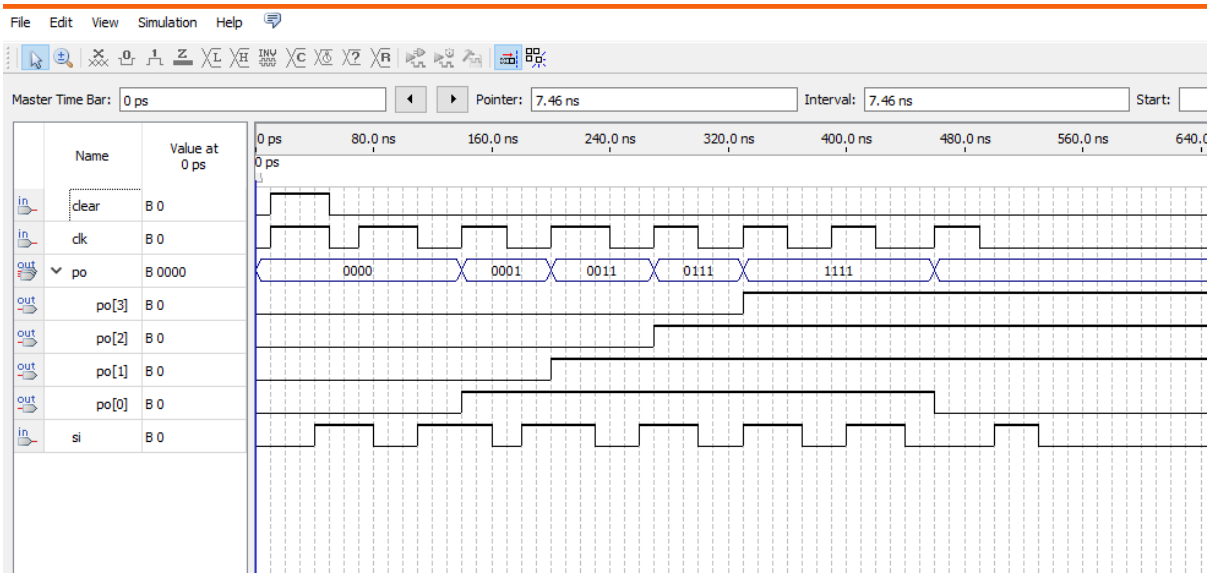


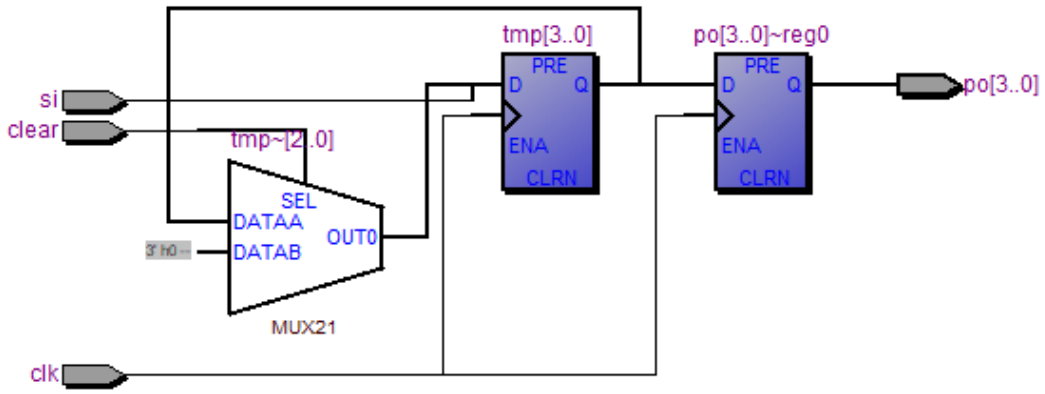


SIPO

```
module propt(clk,clear,si,po);
input clk,si,clear;
output [3:0]po;
reg [3:0]tmp;
```

```
reg [3:0]po;
always @(posedge clk)
begin
if (clear)
tmp<=4'b0000;
else
tmp<=tmp<<1;
tmp[0]<=si;
po=tmp;
end
endmodule
```

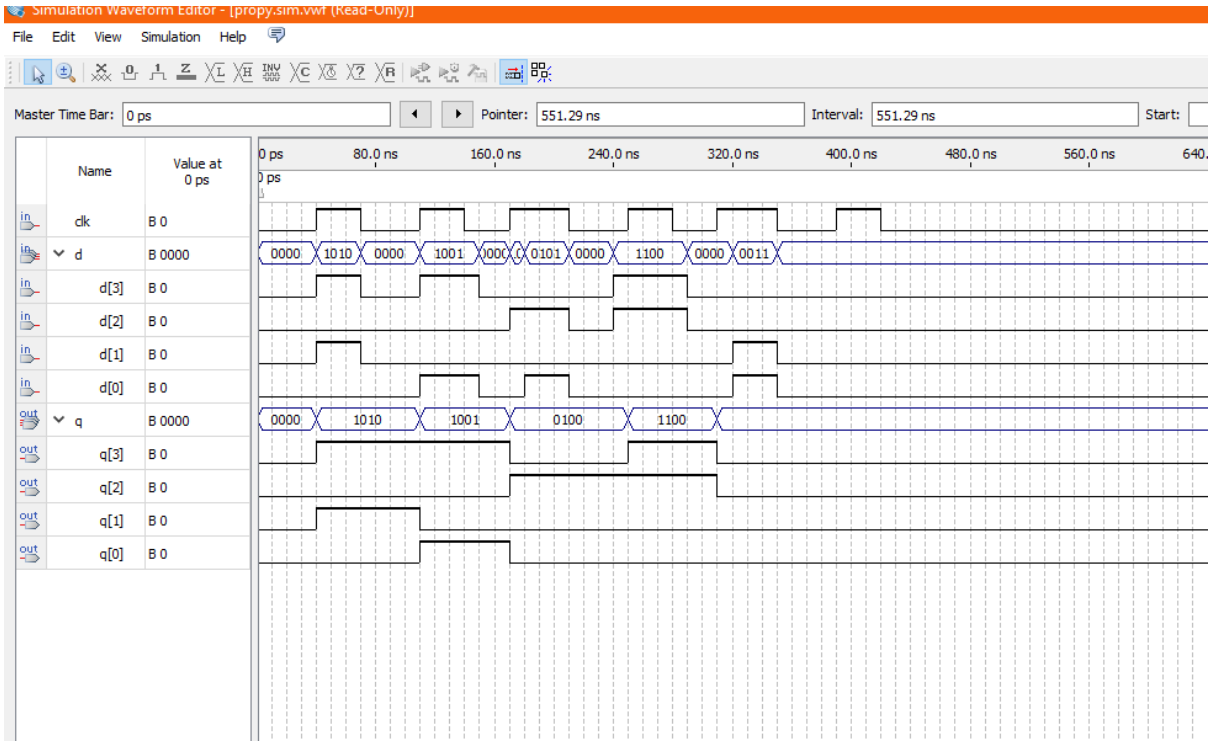


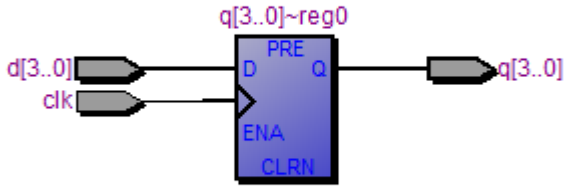


PIPO

```
module propy(clk,d,q);
input clk;
input [3:0]d;
output reg [3:0]q;
always @(posedge clk)
begin
q[3]=d[3];
q[2]=d[2];
```

```
q[1]=d[1];
q[0]=d[0];
end
endmodule
```





SEQUENCE DETECTOR