Name : ATHISH S A

Roll No: B21EC040

GATES IMPLEMENTATION USING VERILOG

1. Not Gate

Gate Level Modeling

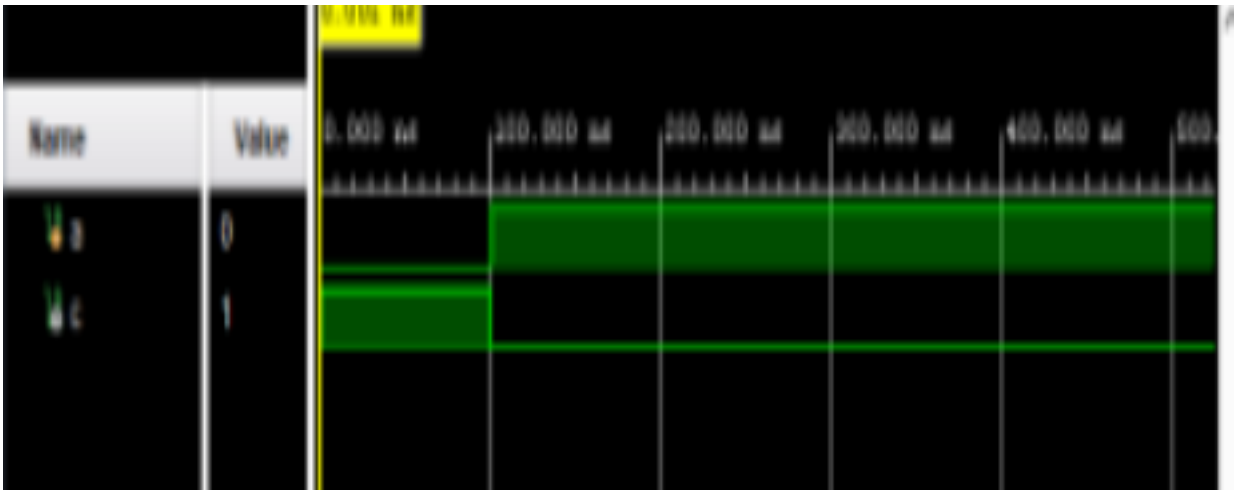
```
module not_gate(c,a);
input a;
output c;
not (c,a);
endmodule
```

Data Flow Modeling

```
module not_data(c,a);
input a;
output c;
assign c=~a;
endmodule
```

Behavioral Modeling

```
module not_beh(c,a);
input a;
output c;
reg c;
always@(a)
begin
if (a==0)
c=1;
else
c=0;
end
endmodule
```



2. NOR Gate

Gate Level Modeling

```
module nor_gate(c,a,b);
input a,b;
output c;
nor (c,a,b);
endmodule
```

Data Flow Modeling

```
module nor_data(c,a,b);
input a,b;
output c;
assign c=~(a|b);
endmodule
```

Behavioral Modeling

```
module nor_beh(c,a,b);
input a,b;
output c;
```

```
reg c;
always@(a,b)

begin
if (a==0 & b==0)
c=1;
else
c=0;
end
endmodule
```

3. And Gate

Gate Level Modeling

```
module
and_gate(c,a,b); input
a,b;
output c;
and (c,a,b);
endmodule
```

Data Flow Modeling

```
module
and_data(c,a,b); input
a,b;
output c;
assign c = a & b;
```

```
endmodule
```

Behavioral Modeling

```
module
and_beh(c,a,b); input
a,b;
output c;
reg c;
always@(a,b)

begin
if (a==0|b==0)
c=0;
else
if(a==0|b==1)
c=0;
else
if(a==1|b==0)
c=0;
else
c=1;
end
endmodule
```

4. Or Gate

Gate Level Modeling

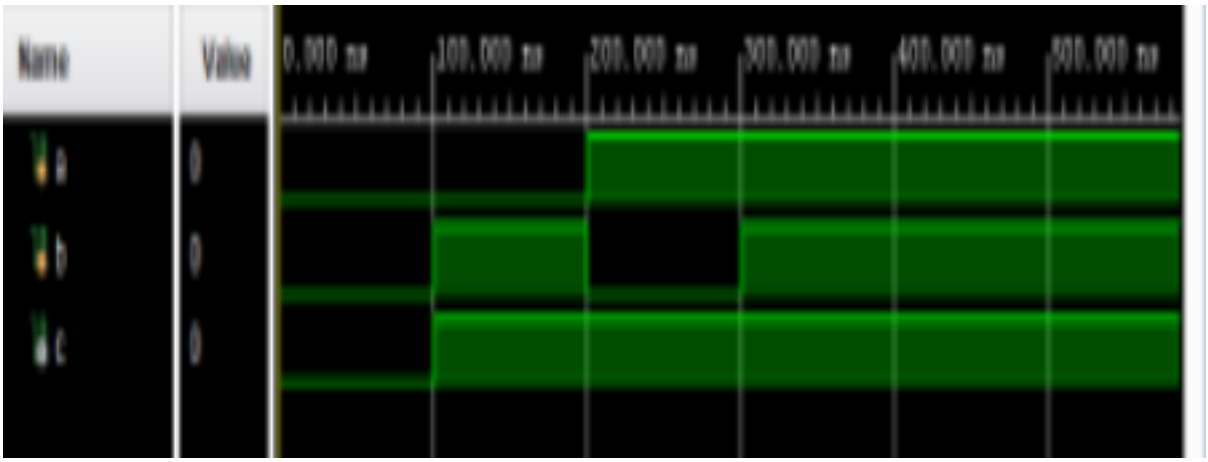
```
module or_gate(c,a,b);
input a,b;
output c;
or (c,a,b);
endmodule
```

Data Flow Modeling

```
module or_data(c,a,b);
input a,b;
output c;
assign c=a|b;
endmodule
```

Behavioral Modeling

```
module or_beh(c,a,b);
input a,b;
output c;
reg c;
always@(a,b);

initial

begin
if (a==0|b==0)
c=0;
else if(a==0|b==1)
c=0;
else if(a==1|b==0)
c=0;
else
c=1;
end
endmodule
```



5. NAND Gate

Gate Level Modeling

```
module nand_gate(c,a,b);
input a,b;
output c;
nand (c,a,b);
endmodule
```

Data Flow Modeling

```
module
nand_data(c,a,b); input
a,b;
output c;
assign c =~(a&b);
endmodule
```

Behavioral Modeling

```
module
nand_beh(c,a,b); input
a,b;
output c;
reg c;
always@(a,b)

begin
```

if (a==1 & b==1)

c=0;

else

c=1;

end

endmodule

6. XOR Gate

Gate Level Modeling

module xor_gate(c,a,b);

input a,b;

output c;

xor (c,a,b);

endmodule

Data Flow Modeling

module xor_data(c,a,b);

input a,b;

output c;

assign c=(a^b);

endmodule

Behavioral Modeling

module xor_beh(c,a,b);

input a,b;

output c;

reg c;

always@(a,b)
begin

if (a==0 & b==0)

c=0;

else if (a==1 & b==1)

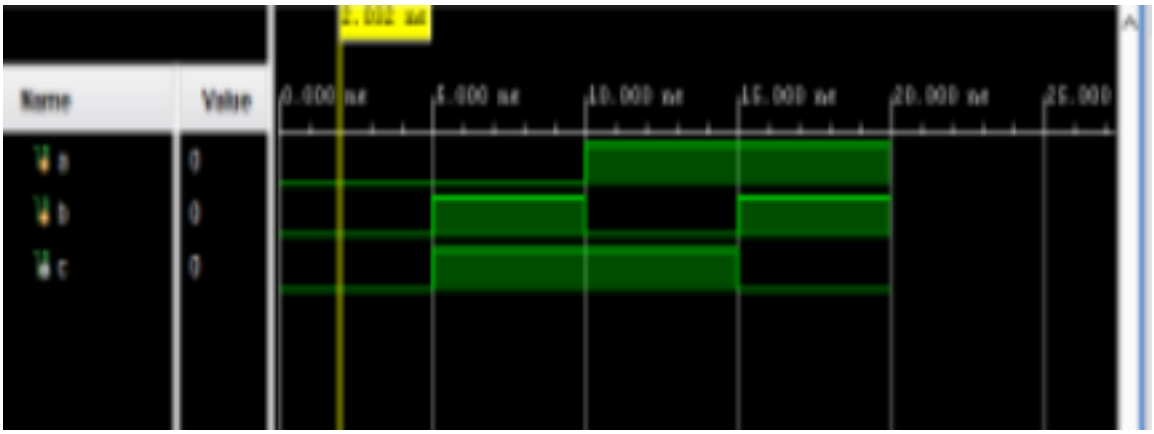
c=0;

else

c=1;

end

endmodule



7. XNOR Gate

Gate Level Modeling

module xnor_gate(c,a,b);

input a,b;

output c;

xnor (c,a,b);
endmodule

Data Flow Modeling

module

xnor_data(c,a,b); input

a,b;

output c;

assign c=~(a^b);

endmodule

Behavioral Modeling

module

xnor_beh(c,a,b); input

a,b;

output c;

reg c;

always@(a,b)

begin

if (a==0 & b==0)

c=1;

else if (a==1 &

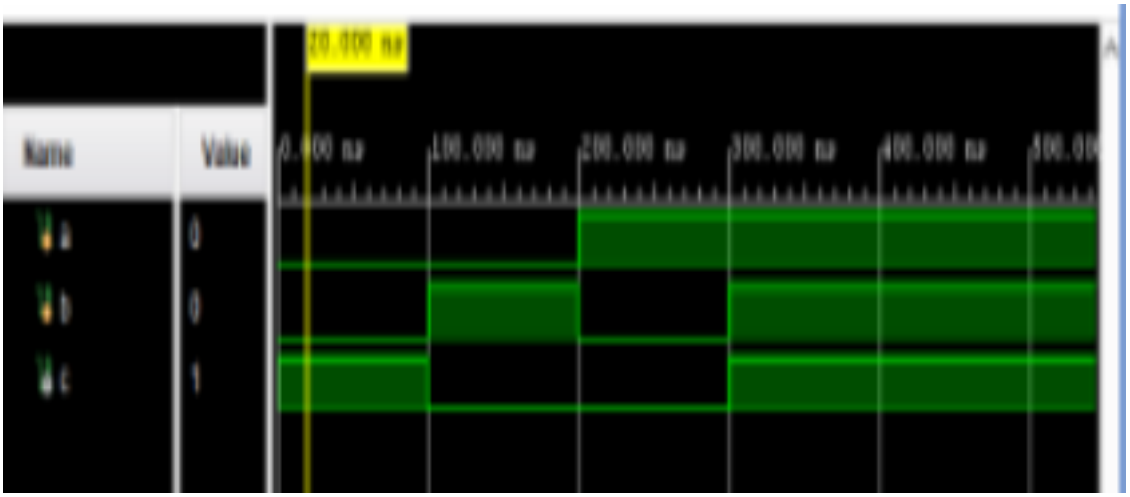
b==1) c=1;

else

c=0;

end

endmodule



8. Half Adder

Gate Level Modeling

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

Data Flow Modeling

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

Behavioral Modeling

```
module halfadder_beh(s,c,a,b);
input a,b;
output s,c;
reg s,c;
always@(a,b)

begin
if (a==b)
begin
s=0;
c=b;
end
```

else

begin

s=1;

c=0;

end

end

endmodule

9. Full Adder

Gate Level Modeling

module

fulladder_gate(s,c,a,b,cin); input

a,b,cin;

output s,c;

wire i1,i2,i3;

xor (i1,a,b);

and (i2,a,b);

xor (s,i1,cin);

and (i3,i1,cin);

or (c,i2,i3);

endmodule

Data Flow Modeling

module

fulladder_data(s,c,a,b,cin); input

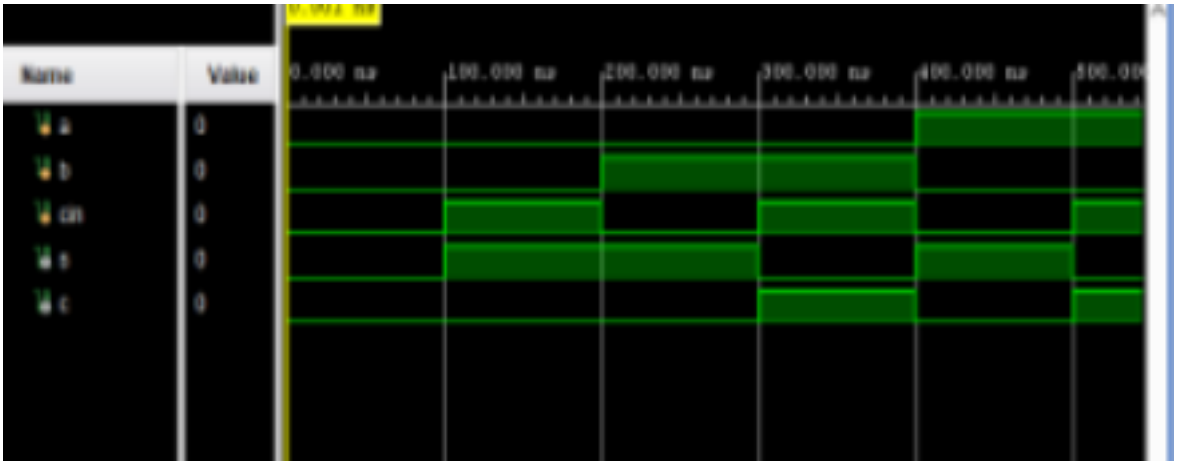
a,b,cin;

output s,c;

```
wire i1,i2,i3;
assign s=i1^cin;
assign i2=a&b;
assign i1=a^b;
assign i3=cin&i1;
assign c=i2|i3;
endmodule
```

Behavioral Modeling

```
module
fulladder_beh(s,c,a,b,cin); input
a,b,cin;
output s,c;
reg s,c;
always@(a,b,cin)

begin
if (a==b)
begin
s=cin;
c=b;
end
else
begin
s=~cin;
c=cin;
end
end
endmodule
```



10. 4:1 MUX

Gate Level Modeling

```
module mux_gate(a,b,c,d,select0,select1,y);
input a,b,c,d,select0,select1;
output y;
wire i1,i2,i3,i4,i5,i6;
not (i1,select0);
not (i2,select1);
and (i3,i1,i2,a);
and (i4,i1,select1,b);
and (i5,select0,i2,c);
and (i6,select0,select1,d);
or (y,i3,i4,i5,i6);
endmodule
```

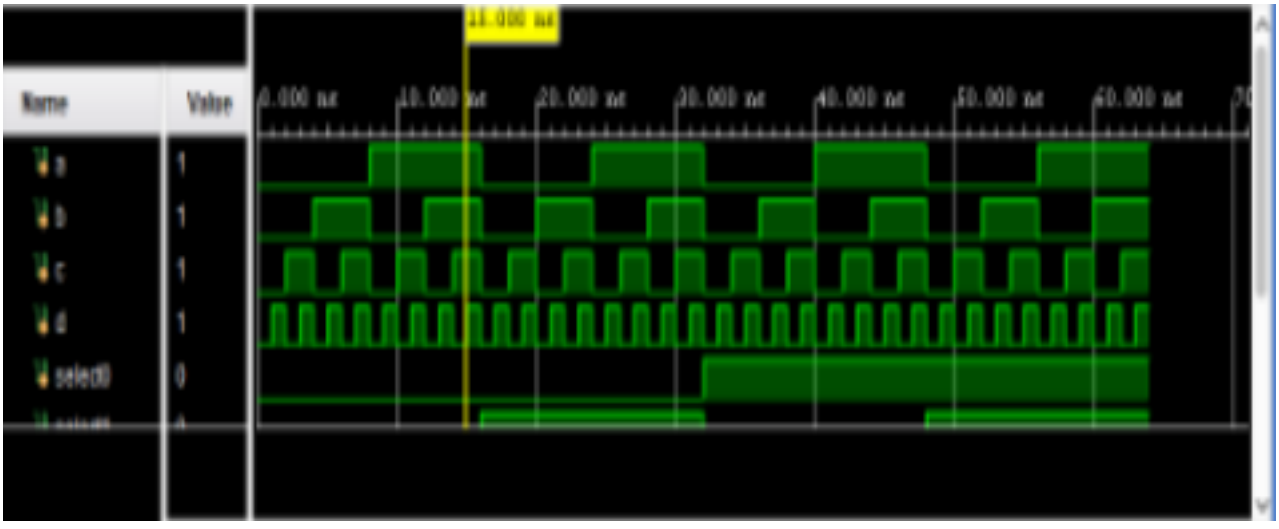
Data Flow Modeling

```
module
mux_data(a,b,c,d,select0,select1,y); input
a,b,c,d,select0,select1;
output y;
wire i1,i2,i3,i4,i5,i6;
assign i1=~select0;
assign i2=~select1;
assign i3=i1&i2&a;
```

```
assign i4=i1&select1&b;
assign i5=select0&i2&c;
assign i6=select0&select1&d;
assign y=i3|i4|i5|i6;
endmodule
```

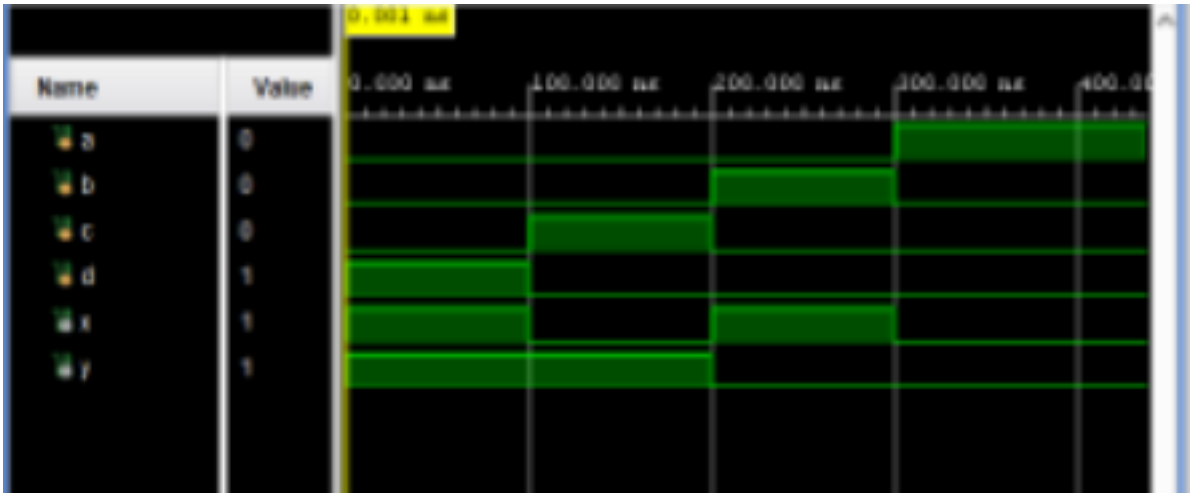
Behavioral Modeling

```
module mux_beh(a,b,c,d,select0,select1,y);
input a,b,c,d,select0,select1;
output y;
reg y;
always @ (a or b or c or d or select0 or select1)

begin
if (select0 == 0 & select1 ==0)
y = a;
else if (select0 == 0 & select1 ==1)
y = b;
else if (select0 == 1 & select1 ==0)
y = c;
else
begin
y = d;
end
end
endmodule
```



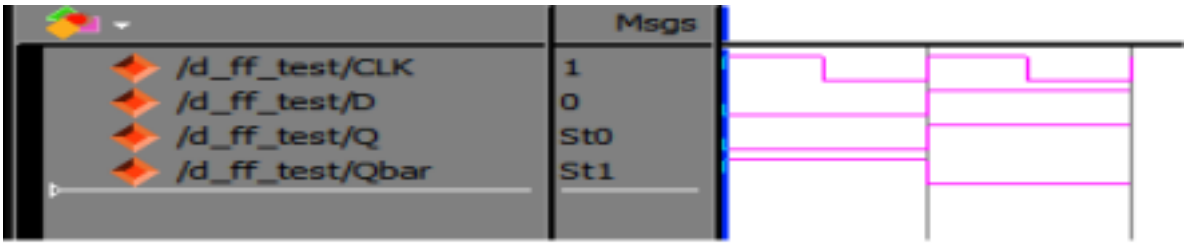
11. 4:2 Encoder

```
module encoder_4_2(a,b,c,d,x,y);
output x,y;
input a,b,c,d;
assign x = b | d;
assign y = c | d;
endmodule
```



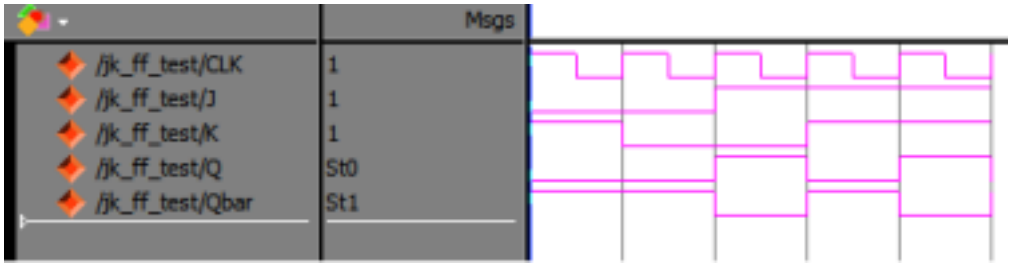
12. D-FF

```
module d_ff_beh(D,CLK,RST,Q,Qbar);
output Q,Qbar;
input D,CLK,RST;
reg Q;
always @(posedge CLK)

begin // positive-edge triggered
if (!RST) // synchronous reset, active low
Q <= 1'b0;
else
Q <= D; // characteristic equation
end
assign Qbar = ~ Q ;
endmodule
```



13. JK-FF

```
module jk_ff_beh(J,K,CLK,RST,Q,Qbar);
output Q,Qbar;
input J,K,CLK,RST;
reg Q;
always @(posedge CLK)

begin // positive-edge triggered
if (!RST) // synchronous reset, active low
Q <= 1'bo;
else
Q <= (J & ~Q)|(~K & Q); // characteristic equation
end
assign Qbar = ~ Q ;
Endmodule
```



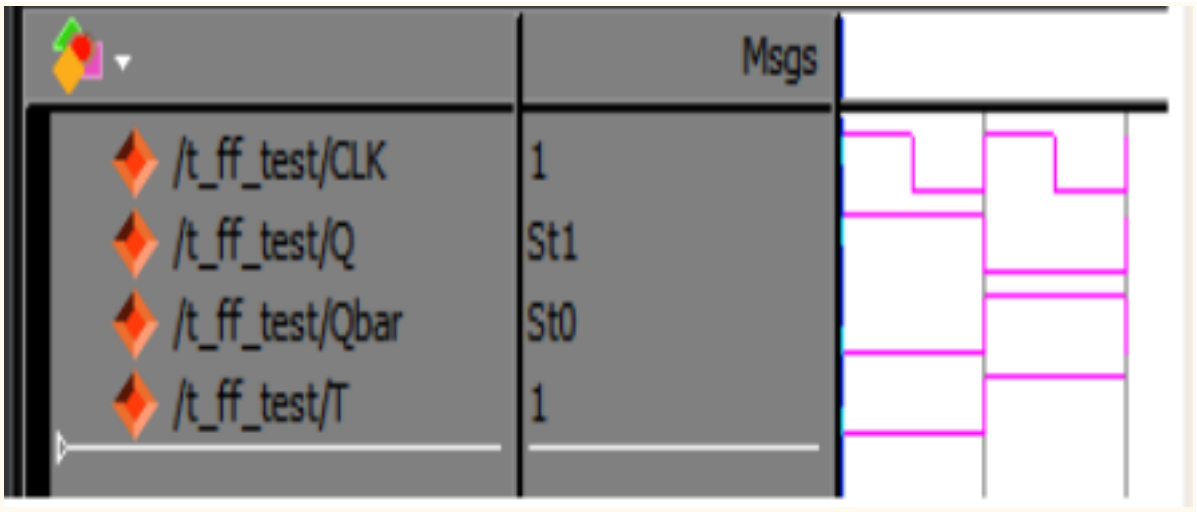
14. SR-FF

```
module sr_ff_beh(S,R,CLK,RST,Q,Qbar);
output Q,Qbar;
input S,R,CLK,RST;
reg Q;
always @(posedge CLK)
```

```
begin // positive-edge triggered
if (!RST) // synchronous reset, active low
Q <= 1'bo;
else
Q <= (S)|(~R & Q); // characteristic equation
end
assign Qbar = ~ Q ;
endmodule
```

15. T-FF

```
module t_ff_beh(T,CLK,RST,Q,Qbar);
output Q,Qbar;
input T,CLK,RST;
reg Q;
always @(posedge CLK)

begin // positive-edge triggered
if (!RST) // synchronous reset, active low
Q <= 1'bo;
else
Q <= T^Q;
//Q <= (~T&Q)|(T&~Q); // characteristic equation
end
assign Qbar = ~ Q ;
endmodule;
```


Msgs
/t_ff_test/CLK
/t_ff_test/Q
/t_ff_test/Qbar
/t_ff_test/T
1
St1
St0
1