

## Realization of Logic Gates and Familiarization of FPGAs

### AND GATE DESIGN(BEHAVIORAL)

```
//Achutha Aswin Naick
//Roll no. 03
module and_gate_design(
    input A,
    input B,
    output reg Y
);
    always @(*)
    begin
        Y= (A&B);
    end
endmodule
```

### AND GATE DESIGN(DATAFLOW)

```
//Achutha Aswin Naick
//Roll no. 03
module and_gate(
    input A,
    input B,
    output Y
);
    assign Y = A & B;
endmodule
```

### AND GATE DESIGN(STRUCTURAL)

```
//Achutha Aswin Naick
//Roll no. 03
module and_gate(
    input A,
    input B,
    output Y
);
    and G1(Y,A,B);
endmodule
```

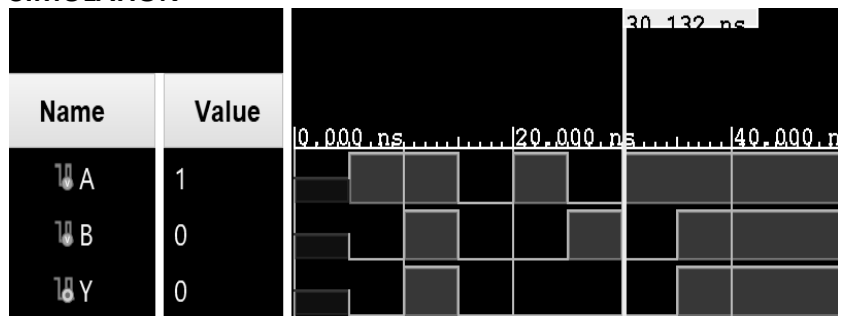
### TESTBENCH FOR AND GATE

```
//Achutha Aswin Naick
//Roll no. 03
module and_gate_testbench();
    reg A,B;
    wire Y;
    and_gate_design DUT(A,B,Y);

    initial
    begin
        $monitor("A=%b,B=%b,Y=%b", A,B,Y);

        #5 A=1'b1; B=1'B0;
        #5 A=1'b1; B=1'B1;
        #5 A=1'b0; B=1'B0;
        #5 A=1'b1; B=1'B0;
        #5 A=1'b0; B=1'B1;
        #5 A=1'b1; B=1'B0;
        #5 A=1'b1; B=1'B1;
        #5 A=1'b1; B=1'B1;
        #10 $finish;
    end
endmodule
```

### SIMULATION



### OR GATE DESIGN(BEHAVIORAL)

```
//Achutha Aswin Naick
//Roll no. 03
module or_gate_design(
    input A,
    input B,
    output reg Y
);

    always@(*)
    begin

        Y= (A|B);
    end
endmodule
```

### OR GATE DESIGN(DATAFLOW)

```
//Achutha Aswin Naick
//Roll no. 03
module or_gate(
    input A,
    input B,
    output Y
);
    assign Y = A | B;
endmodule
```

### OR GATE DESIGN(STRUCTURAL)

```
//Achutha Aswin Naick
//Roll no. 03
module or_gate(
    input A,
    input B,
    output Y
);
    or G2(Y,A,B);
endmodule
```

### TESTBENCH FOR OR GATE

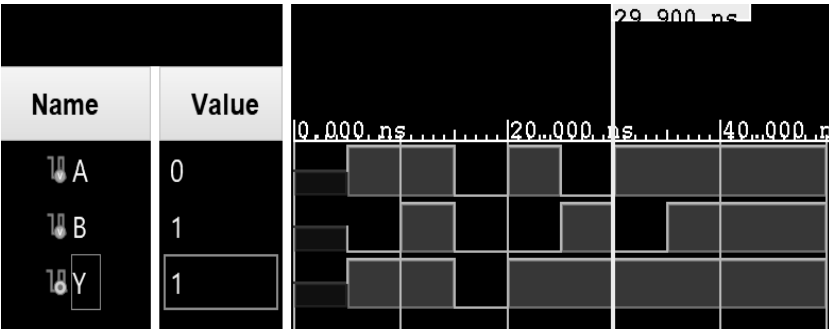
```
//Achutha Aswin Naick
//Roll no. 03
module or_gate_testbench();
    reg A,B;
    wire Y;
    or_gate_design DUT(A,B,Y);

    initial
        begin

            $monitor("A=%b,B=%b,Y=%b", A,B,Y);

            #5 A=1'b1; B=1'B0;
            #5 A=1'b1; B=1'B1;
            #5 A=1'b0; B=1'B0;
            #5 A=1'b1; B=1'B0;
            #5 A=1'b0; B=1'B1;
            #5 A=1'b1; B=1'B0;
            #5 A=1'b1; B=1'B1;
            #5 A=1'b1; B=1'B1;
            #10 $finish;
        end
endmodule
```

### SIMULATION



### NOT GATE DESIGN(BEHAVIORAL)

```
//Achutha Aswin Naick
//Roll no. 03
module not_gate_design(
input A,
output reg Y
);

always@(*)
begin
Y= ~A;

end
endmodule
```

### NOT GATE DESIGN(DATAFLOW)

```
//Achutha Aswin Naick
//Roll no. 03
module not_gate(
input A,
output Y
);
assign Y =~ A;
endmodule
```

### NOT GATE DESIGN(STRUCTURAL)

```
//Achutha Aswin Naick
//Roll no. 03
module not_gate(
input A,
output Y
);
not G3(Y,A,B);
endmodule
```

### TESTBENCH FOR NOT GATE

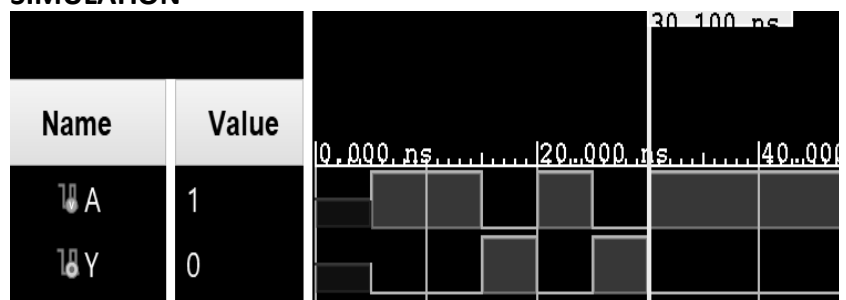
```
//Achutha Aswin Naick
//Roll no. 03
module not_gate_testbench();

reg A;
wire Y;

not_gate_design DUT(A,Y);

initial
begin
$monitor("A=%b,Y=%b",A,Y);
#5 A=1'b1;
#5 A=1'b1;
#5 A=1'b0;
#5 A=1'b1;
#5 A=1'b0;
#5 A=1'b1;
#5 A=1'b1;
#5 A=1'b1;
#5 A=1'b1;
#10 $finish;
end
endmodule
```

### SIMULATION



## NAND GATE DESIGN(BEHAVIORAL)

```
//Achutha Aswin Naick
//Roll no. 03
module nand_gate_design(
    input A,
    input B,
    output reg YNAND
);

    always@(*)
    begin

        YNAND=~ (A&B);
    end
endmodule
```

## NAND GATE DESIGN(DATAFLOW)

```
//Achutha Aswin Naick
//Roll no. 03
module nand_gate(
    input A,
    input B,
    output Y
);
    assign Y = ~( A & B );
endmodule
```

## NAND GATE DESIGN(STRUCTURAL)

```
//Achutha Aswin Naick
//Roll no. 03
module nand_gate(
    input A,
    input B,
    output Y
);
    nand G4(Y,A,B);
endmodule
```

## TESTBENCH FOR NAND GATE

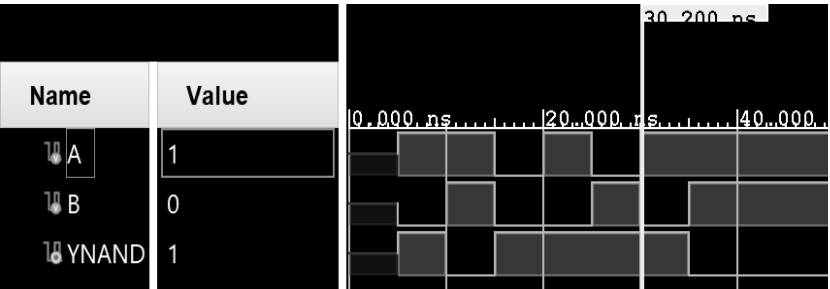
```
//Achutha Aswin Naick
//Roll no. 03
module nand_gate_testbench();
    reg A,B;
    wire YNAND;

    nand_gate_design DUTNAND(A,B,YNAND);

    initial
    begin
        $monitor("A=%b,B=%b,YNAND=%b",A,B,YNAND);

        #5 A=1'b1; B=1'b0;
        #5 A=1'b1; B=1'b1;
        #5 A=1'b0; B=1'b0;
        #5 A=1'b1; B=1'b0;
        #5 A=1'b0; B=1'b1;
        #5 A=1'b1; B=1'b0;
        #5 A=1'b1; B=1'b1;
        #5 A=1'b1; B=1'b1;
        #10 $finish;
    end
endmodule
```

## SIMULATION



### NOR GATE DESIGN(BEHAVIORAL)

```
//Achutha Aswin Naick
//Roll no. 03
module nor_gate_design(
input C,
input D,
output reg YOR
);

always@ (*)
begin

YOR= ~(C|D) ;
end
endmodule
```

### NOR GATE DESIGN(DATAFLOW)

```
//Achutha Aswin Naick
//Roll no. 03
module nor_gate(
input A,
input B,
output Y
);
assign Y = ~( A | B );
endmodule
```

### NOR GATE DESIGN(STRUCTURAL)

```
//Achutha Aswin Naick
//Roll no. 03
module nor_gate(
input A,
input B,
output Y
);
nor G5(Y,A,B);
endmodule
```

### TESTBENCH FOR NOR GATE

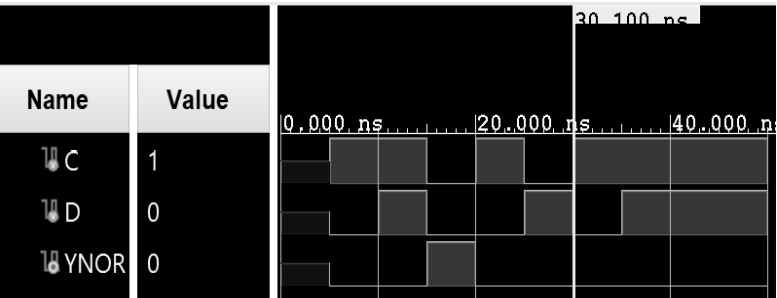
```
//Achutha Aswin Naick
//Roll no. 03
module nor_gate_testbench();
reg C,D;
wire YNOR;

nor_gate_design DUTNOR(C,D,YNOR);

initial
begin
$monitor("C=%b,D=%b,YNOR=%b",C,D,YNOR);

#5 C=1'b1; D=1'b0;
#5 C=1'b1; D=1'b1;
#5 C=1'b0; D=1'b0;
#5 C=1'b1; D=1'b0;
#5 C=1'b0; D=1'b1;
#5 C=1'b1; D=1'b0;
#5 C=1'b1; D=1'b1;
#5 C=1'b1; D=1'b1;
#10 $finish;
end
endmodule
```

### SIMULATION



### XOR GATE DESIGN(BEHAVIORAL)

```
//Achutha Aswin Naick
//Roll no. 03
module xor_gate_design(
input E,
input F,
output reg YXOR
);

always@ (*)
begin

YXOR= (E&(~F) + F&(~E));
end
endmodule
```

### XOR GATE DESIGN(DATAFLOW)

```
//Achutha Aswin Naick
//Roll no. 03
module xor_gate(
input A,
input B,
output Y
);
assign Y = ( A ^ B );
endmodule
```

### XOR GATE DESIGN(STRUCTURAL)

```
//Achutha Aswin Naick
//Roll no. 03
module xor_gate(
input A,
input B,
output Y
);
xor G6(Y,A,B);
endmodule
```

### TESTBENCH FOR XOR GATE

```
//Achutha Aswin Naick
//Roll no. 03
module xor_gate_testbench();
reg E,F;
wire YXOR;

xor_gate_design DUTXOR(E,F,YXOR);

initial
begin
$monitor("E=%b,F=%b,YXOR=%b",E,F,YXOR)

#5 E=1'b1; F=1'b0;
#5 E=1'b1; F=1'b1;
#5 E=1'b0; F=1'b0;
#5 E=1'b1; F=1'b0;
#5 E=1'b0; F=1'b1;
#5 E=1'b1; F=1'b0;
#5 E=1'b1; F=1'b1;
#5 E=1'b1; F=1'b1;
#10 $finish;
end
endmodule
```

### SIMULATION



### XNOR GATE DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module xnor_gate_design(
    input G,
    input H,
    output reg YXNOR
);

always@ (*)
begin

    YXNOR=~ (G& (~H) + H& (~G));
end

endmodule
```

### XNOR GATE DESIGN(DATAFLOW)

```
//Achutha Aswin Naick
//Roll no. 03
module xnor_gate(
    input A,
    input B,
    output Y
);
    assign Y = ~( A ^ B );
endmodule
```

### XNOR GATE DESIGN(STRUCTURAL)

```
//Achutha Aswin Naick
//Roll no. 03
module xnor_gate(
    input A,
    input B,
    output Y
);
    xnor G7(Y,A,B);
endmodule
```

### TESTBENCH FOR XNOR GATE

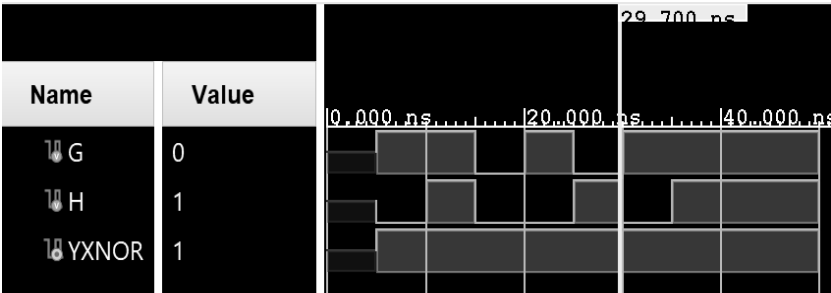
```
//Achutha Aswin Naick
//Roll no. 03
module xnor_gate_testbench();
    reg G,H;
    wire YXNOR;

    xnor_gate_design DUTXNOR(G,H,YXNOR);

    initial
    begin
        $monitor("G=%b,H=%b,YXNOR=%b",G,H,YXNOR);

        #5 G=1'b1; H=1'b0;
        #5 G=1'b1; H=1'b1;
        #5 G=1'b0; H=1'b0;
        #5 G=1'b1; H=1'b0;
        #5 G=1'b0; H=1'b1;
        #5 G=1'b1; H=1'b0;
        #5 G=1'b1; H=1'b1;
        #5 G=1'b1; H=1'b1;
        #10 $finish;
    end
endmodule
```

### SIMULATION



## Realization of Functions using Basic and Universal Gates

### SOP ( $AB'C+AC'+B'C'$ ) using NAND Gates

```
//Achutha Aswin Naick
//Roll no. 03
module SOP_nand(
    input A,
    input B,
    input C,
    output I,
    wire D,E,F,G,H
);
    nand G1(D,B,B);
    nand G2(E,C,C);
    nand G3(F,E,A);
    nand G4(G,A,D,C);
    nand G5(H,D,E);
    nand G6(I,F,G,H);
endmodule
```

### SOP ( $AB'C+AC'+B'C'$ ) using Basic Gates

```
//Achutha Aswin Naick
//Roll no. 03
module SOP_basic(
    input A,
    input B,
    input C,
    wire D,E,F,G,H,I,
    output J
);
    not G1(H,B);
    not G2(E,C);
    and G3(D,A,H);
    or G4(F,A,H);
    and G5(I,C,D);
    and G6(G,E,F);
    or G7(J,I,G);
endmodule
```










### TESTBENCH FOR SOP

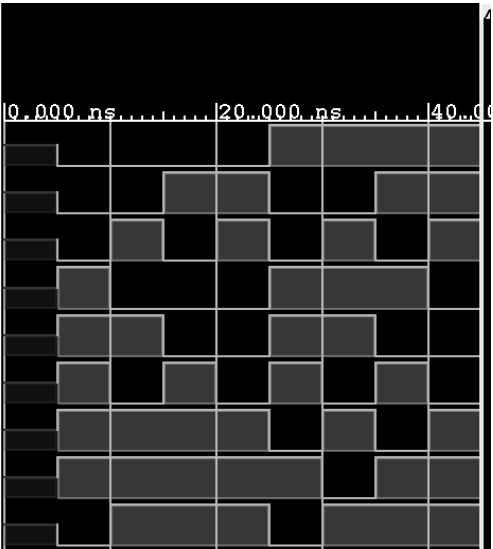
```
//Achutha Aswin Naick
//Roll no. 03
module tb();
    reg A,B,C;
    wire D,E,F,G,H,I;

    SOP_nand DUT(A,B,C,D,E,F,G,H,I);

    initial
    begin
        $monitor("A=%b,B=%b,C=%b,D=%b,E=%b,F=%b,G=%b,H=%b,I=%b",A,B,C,D,E,F,G,H,I);
        #5 A=1'b0; B=1'b0; C=1'b0;
        #5 A=1'b0; B=1'b0; C=1'b1;
        #5 A=1'b0; B=1'b1; C=1'b0;
        #5 A=1'b0; B=1'b1; C=1'b1;
        #5 A=1'b1; B=1'b0; C=1'b0;
        #5 A=1'b1; B=1'b0; C=1'b1;
        #5 A=1'b1; B=1'b1; C=1'b0;
        #5 A=1'b1; B=1'b1; C=1'b1;
        #5 $finish;
    end
endmodule
```

### SIMULATION

Name		Value
 A	1	
 B	1	
 C	1	
 D	0	
 E	0	
 F	0	
 G	1	
 H	1	
 I	1	





## POS ((A+B'+C)(A+C')(B'+C')) using NOR Gates

```
//Achutha Aswin Naick
//Roll no. 03
module POS_nor(
    input A,
    input B,
    input C,
    wire D,E,F,G,H,I,J,K,L,
    output M
);
    nor G1(D,B,B);
    nor G2(E,C,C);
    nor G3(F,A,D);
    nor G4(G,D,E);
    nor G5(H,A,E);
    nor G6(I,D,D);
    nor G7(J,I,C);
    nor G8(K,G,H);
    nor G9(L,K,K);
    nor G10(M,L,J);
endmodule
```

## POS ((A+B'+C)(A+C')(B'+C')) using Basic Gates

```
//Achutha Aswin Naick
//Roll no. 03
module POS_basic(
    input A,
    input B,
    input C,
    output J,
    wire D,E,F,G,H,I
);
    not G1(D,B);
    not G2(E,C);
    or G3(F,E,A);
    or G4(G,A,D);
    or G5(H,G,C);
    or G6(I,E,D);
    or G7(J,I,F,H);
endmodule
```

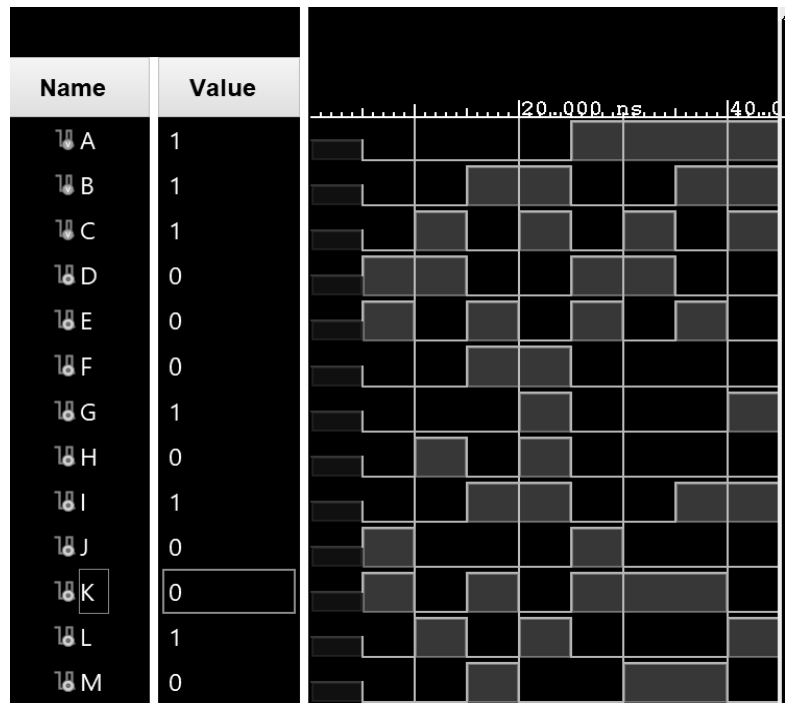
## TESTBENCH FOR POS

```
//Achutha Aswin Naick
//Roll no. 03
module tb();
    reg A,B,C;
    wire D,E,F,G,H,I,J,K,L,M;

    POS_nor DUT(A,B,C,D,E,F,G,H,I,J,K,L,M);

    initial
    begin
        $monitor("A=%b,B=%b,C=%b,D=%b,E=%b,F=%b,G=%b,H=%b,I=%b,J=%b,K=%b,L=%b,M=%b",A,B,C,D,E,F,G,H,I,J,K,L,M);
        #5 A=1'b0; B=1'b0; C=1'b0;
        #5 A=1'b0; B=1'b0; C=1'b1;
        #5 A=1'b0; B=1'b1; C=1'b0;
        #5 A=1'b0; B=1'b1; C=1'b1;
        #5 A=1'b1; B=1'b0; C=1'b0;
        #5 A=1'b1; B=1'b0; C=1'b1;
        #5 A=1'b1; B=1'b1; C=1'b0;
        #5 A=1'b1; B=1'b1; C=1'b1;
        #5 $finish;
    end
endmodule
```

## SIMULATION



## Realization of Half Adder, Full Adder, Half Subtractor, Full Subtractor

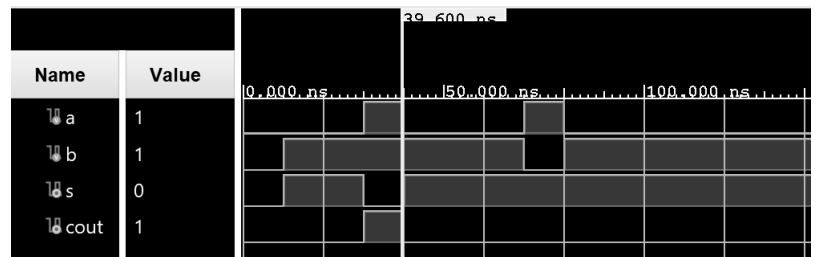
### HALF ADDER DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module half_adder_behavioral(
    input a,
    input b,
    output reg s,
    output reg cout
);
    always@(*)
    begin
        s = a ^ b;
        cout = a & b;
    end
endmodule
```

### TESTBENCH FOR HALF ADDER

```
//Achutha Aswin Naick
//Roll no. 03
module half_adder_testbench();
    reg a,b;
    wire s,cout;
    half_adder DUT (a,b,s,cout);
    initial
    begin
        repeat(10)
        begin
            {a,b}=$random;
            #10;
            $display("%t,a=%b,b=%b,s=%b,cout=%b", $time, a,b,s,cout);
        end
    end
endmodule
```

### SIMULATION



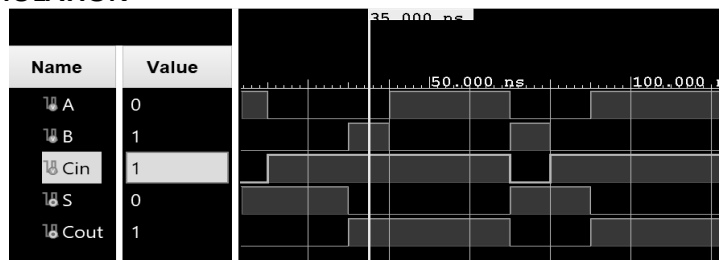
### FULL ADDER DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module full_adder(
    input A,B,Cin,
    output reg S,Cout
);
    always@(*)
    begin
        S =A ^ B ^ Cin;
        Cout= ((A&B) | (A ^ B)&Cin) ;
    end
endmodule
```

### TESTBENCH FOR FULL ADDER

```
//Achutha Aswin Naick
//Roll no. 03
module fa_testbench();
    reg A,B,Cin;
    wire S,Cout;
    full_adder DUT (A,B,Cin,S,Cout);
    initial
    begin
        repeat(10)
        begin
            {A,B,Cin}=$random;
            #10 $display("%t,A=%b,B=%b,Cin=%b,S=%b,Cout=%b", $time,A,B,Cin,S,Cout);
        end
    end
endmodule
```

### SIMULATION



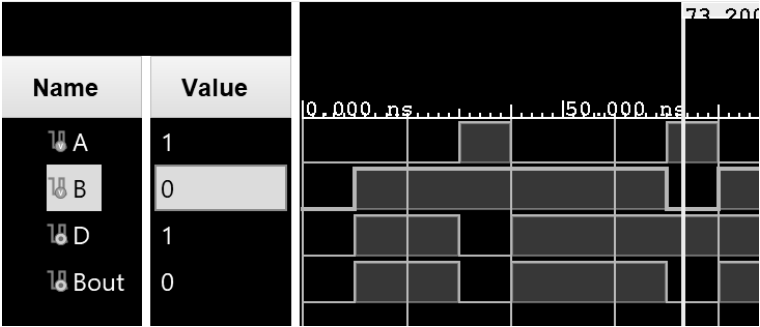
HALF SUBTRACTOR DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module half_subtractor(
    input A,
    input B,
    output D,
    output Bout
);
    assign D = A^B;
    assign Bout= (~(A) &B) ;
endmodule
```

TESTBENCH FOR HALF SUBTRACTOR

```
//Achutha Aswin Naick
//Roll no. 03
module halfsub_tb();
    reg A,B;
    wire D,Bout;
    half_subtractor DUT (A,B,D,Bout);
    initial
    begin
        repeat(10)
        begin
            {A,B}=$random;
            #10 $display ("%t,A=%b,B=%b,D=%b,Bout=%b", $time,A,B,D,Bout) ;
        end
    end
endmodule
```

SIMULATION



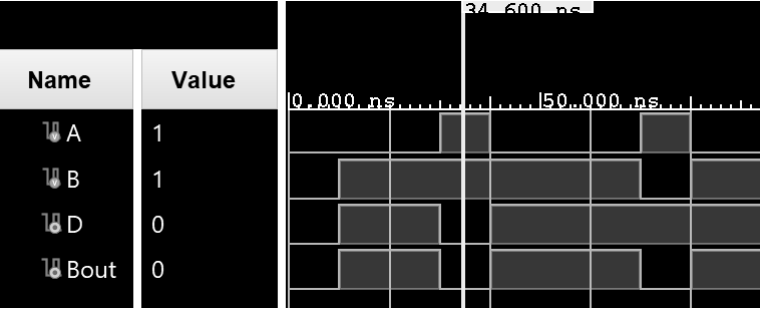
FULL SUBTRACTOR DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module full_subtractor(
    input A,
    input B,
    input Bin,
    output D,
    output Bout
);
    assign D = A^B^Bin;
    assign Bout= (~(A^B) &Bin) | (~(A) &B);
endmodule
```

TESTBENCH FOR FULL SUBTRACTOR

```
//Achutha Aswin Naick
//Roll no. 03
module fullsub_tb();
    reg A,B,Bin;
    wire D,Bout;
    full_subtractor DUT (A,B,Bin,D,Bout);
    initial
    begin
        repeat(10)
        begin
            {A,B,Bin}=$random;
            #10 $display ("%t,A=%b,B=%b,Bin=%b,D=%b,Bout=%b", $time,A,B,Bin,D,Bout) ;
        end
    end
endmodule
```

SIMULATION



Realization of 4 Bit Adder/Subtractor and BCD Adder

4 Bit ADDER DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module full_adder(
    input A,B,Cin,
    output S,Cout
);
    assign S=A^B^Cin;
    assign Cout=(A&B) | (A&Cin) | (B&Cin) ;
endmodule

module adder_4_bit(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0] S,
    output Cout
);
    full_adder fa0 (.A(A[0]),.B(B[0]),.Cin(Cin),.S(S[0]),.Cout(C1));
    full_adder fa1 (.A(A[1]),.B(B[1]),.Cin(C1),.S(S[1]),.Cout(C2));
    full_adder fa2 (.A(A[2]),.B(B[2]),.Cin(C2),.S(S[2]),.Cout(C3));
    full_adder fa3 (.A(A[3]),.B(B[3]),.Cin(Cin),.S(S[3]),.Cout(Cout));
endmodule
```

TESTBENCH FOR 4 Bit ADDER

```
//Achutha Aswin Naick
//Roll no. 03
module adder_tb();
    reg [3:0] A;
    reg [3:0] B;
    reg Cin;
    wire [3:0] S;
    wire Cout;

    adder_4_bit DUT(A,B,Cin,S,Cout);
    initial
    begin
        $monitor("A=%b,B=%b,Cin=%b,S=%b,Cout=%b",A,B,Cin,S,Cout);
        #5; A=4'b0000; B=4'b0011; Cin=1'b0;
        #5; A=4'b0001; B=4'b0111; Cin=1'b1;
        #5; A=4'b0011; B=4'b1011; Cin=1'b1;
        #5; A=4'b0010; B=4'b0101; Cin=1'b0;
        #5; A=4'b1010; B=4'b0110; Cin=1'b1;
        #5; A=4'b1100; B=4'b1001; Cin=1'b0;
        #10; $finish;
    end
endmodule
```

SIMULATION

		0.000 ns10.000 ns20.000 ns30.000 ns40.000 ns						
Name	Value							
> A[3:0]	1100	xxxx	0000	0001	0011	0010	1010	1100
> B[3:0]	1001	xxxx	0011	0111	1011	0101	0110	1001
Cin	0							
> S[3:0]	0101	xxxx	0011	1001	0111	0001	0101	
Cout	1							

4 Bit SUBTRACTOR DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module full_subtractor(
    input a,b,bin,
    output d,bout
);
    assign d=a^b^bin;
    assign bout=(~(a)&b) | (~(a^b)&bin) ;
endmodule

module subtractor4_bit(
    input [3:0] A,
    input [3:0] B,
    input Bin,
    output [3:0] D,
    output Bout
);
    full_subtractor fa0 (.a(A[0]),.b(B[0]),.bin(Bin),.d(D[0]),.bout(b1));
    full_subtractor fa1 (.a(A[1]),.b(B[1]),.bin(b1),.d(D[1]),.bout(b2));
    full_subtractor fa2 (.a(A[2]),.b(B[2]),.bin(b2),.d(D[2]),.bout(b3));
    full_subtractor fa3 (.a(A[3]),.b(B[3]),.bin(b3),.d(D[3]),.bout(Bout));
endmodule
```

TESTBENCH FOR 4 Bit SUBTRACTOR

```
//Achutha Aswin Naick
//Roll no. 03
module subtractor_tb();
    reg [3:0] a;
    reg [3:0] b;
    reg bin;
    wire [3:0] d;
    wire bout;

    subtractor4_bit DUT(a,b,bin,d,bout);
    initial
    begin
        $monitor("a=%b,b=%b,bin=%b,d=%b,bout=%b",a,b,bin,d,bout);
        #5 a=4'b0000; b=4'b0011; bin=1'b0;
        #5 a=4'b0001; b=4'b0111; bin=1'b1;
        #5 a=4'b0011; b=4'b1011; bin=1'b1;
        #5 a=4'b0010; b=4'b0101; bin=1'b0;
        #5 a=4'b1010; b=4'b0110; bin=1'b1;
        #5 a=4'b1100; b=4'b1001; bin=1'b0;
        #10 $finish;
    end
endmodule
```

SIMULATION

Name		Value								
>	A[3:0]	1100								
>	B[3:0]	1001								
	Bin	0								
>	D[3:0]	0011								
	Bout	0								

## BCD ADDER DESIGN






```
//Achutha Aswin Naick
//Roll no. 03
module bcd_adder_4bit(
input [3:0] A,
input [3:0] B,
input Cin,
output [3:0] Sum,
output Cout);

wire [4:0] binary_sum;
wire correction_needed;
assign binary_sum = A + B + Cin;
assign correction_needed = (binary_sum > 9);
assign Sum = correction_needed ?
(binary_sum[3:0] + 4'b0110) : binary_sum[3:0];
assign Cout = correction_needed || binary_sum[4];
endmodule
```

## TESTBENCH FOR BCD ADDER

```
//Achutha Aswin Naick
//Roll no. 03
module bcd_adder_4bit_tb;
reg [3:0] A, B;
reg Cin;
wire [3:0] Sum;
wire Cout;
bcd_adder_4bit uut (A,B,Cin,Sum,Cout);
initial begin
A = 4'd5; B = 4'd3; Cin = 0;
#10 A = 4'd9; B = 4'd4; Cin = 0;
#10 A = 4'd6; B = 4'd6; Cin = 0;
#10 A = 4'd7; B = 4'd2; Cin = 0;
#10 A = 4'd8; B = 4'd9; Cin = 0;
#10 A = 4'd9; B = 4'd9; Cin = 0;
#10 $finish;
end
endmodule
```

## SIMULATION

Name	Value						
>  A[3:0]	1001						
>  B[3:0]	1001						
 Cin	0						
>  Sum[3:0]	1000						
 Cout	1						
		0.000 ns.....20.000 ns.....40.000 ns.....60.000 ns.....					
		0101	1001	0110	0111	1000	1001
		0011	0100	0110	0010	1001	
		1000	0011	0010	1001	0111	1000

# Realization of 4x1 Multiplexer and 1x4 Demultiplexer

## 4X1 MULTIPLEXER DESIGN

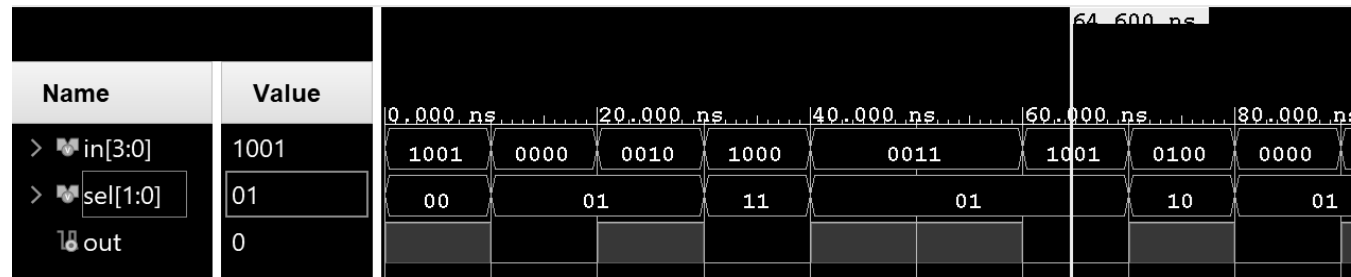
```
//Achutha Aswin Naick
//Roll.no. 03
module mux_4to1(
    input [3:0] in,
    input [1:0] sel,
    output out,
    wire t0,t1,t2,t3
);
    and G1(t0,~sel[1],~sel[0],in[3]);
    and G2(t1,sel[1],~sel[0],in[2]);
    and G3(t2,~sel[1],sel[0],in[1]);
    and G4(t3,sel[1],sel[0],in[0]);
    or G5(out,t0,t1,t2,t3);
endmodule
```

## TESTBENCH FOR 4X1 MULTIPLEXER

```
//Achutha Aswin Naick
//Roll.no. 03
module mux4to1_tb();
    reg [3:0] in;
    reg [1:0] sel;
    wire out;

    mux_4to1 DUT(in,sel,out);
    initial
    begin
        repeat(10)
        begin
            {in,sel}=$random;
            #10 $display("%t,in=%b,sel=%b,out=%b",$time,in,sel,out);
        end
    end
endmodule
```

## SIMULATION



# 1X4 DEMULTIPLEXER DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module demux_1to4(
    input wire in,
    input wire [1:0] sel,
    output reg [3:0] out
);
    always @(*)
    begin
        out = 4'b0000;
        case (sel)
            2'b00: out[0] = in;
            2'b01: out[1] = in;
            2'b10: out[2] = in;
            2'b11: out[3] = in;
            default: out = 4'b0000;
        endcase
    end
endmodule
```

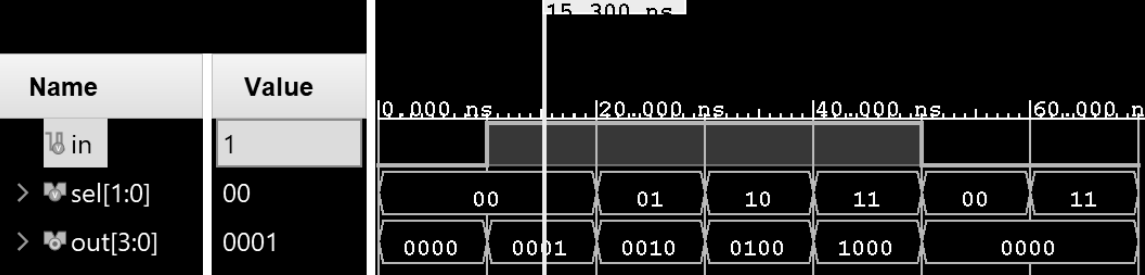
# TESTBENCH FOR 1X4 DEMULTIPLEXER

```
//Achutha Aswin Naick
//Roll no. 03
module demux_1to4_tb();
    reg in;
    reg [1:0] sel;
    wire [3:0] out;

    demux_1to4 DUT (.in(in),.sel(sel),.out(out));

    initial begin
        $monitor("%t,in = %b,sel = %b,out = %b", $time,in,sel,out);
        in = 0;
        sel = 2'b00;
        #10 in = 1; sel = 2'b00;
        #10 in = 1; sel = 2'b01;
        #10 in = 1; sel = 2'b10;
        #10 in = 1; sel = 2'b11;
        #10 in = 0; sel = 2'b00;
        #10 in = 0; sel = 2'b11;
        #10 $finish;
    end
endmodule
```

# SIMULATION





## Realization of SR, D, JK, T, Master-Slave Flip-Flops

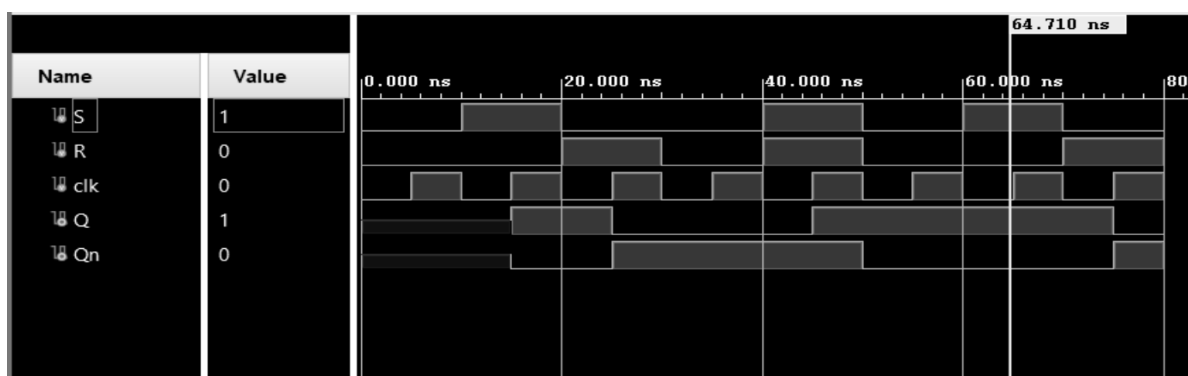
### SR FLIP FLOP DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module sr_flipflop_behavioral (
    input S,
    input R,
    input clk,
    output reg Q,
    output reg Qn
);
always @(posedge clk) begin
    if (S && ~R) begin
        Q <= 1;
        Qn <= 0;
    end else if (~S && R) begin
        Q <= 0;
        Qn <= 1;
    end else if (~S && ~R) begin
        Q <= Q;
        Qn <= Qn;
    end else begin
        Q <= 1'bx;
        Qn <= 1'bx;
    end
end
end
endmodule
```

### TESTBENCH FOR SR FLIP FLOP

```
//Achutha Aswin Naick
//Roll no. 03
module sr_flipflop_tb();
    reg S,R,clk;
    wire Q,Qn;
    sr_flipflop dut(S,R,clk,Q,Qn);
    always #5 clk = ~clk;
    initial begin
        clk = 0; S = 0; R = 0;
        #10 S = 1; R = 0;
        #10 S = 0; R = 1;
        #10 S = 0; R = 0;
        #10 S = 1; R = 1;
        #10 S = 0; R = 0;
        #10 S = 1; R = 0;
        #10 S = 0; R = 1;
        #10 $stop;
    end
endmodule
```

### SIMULATION



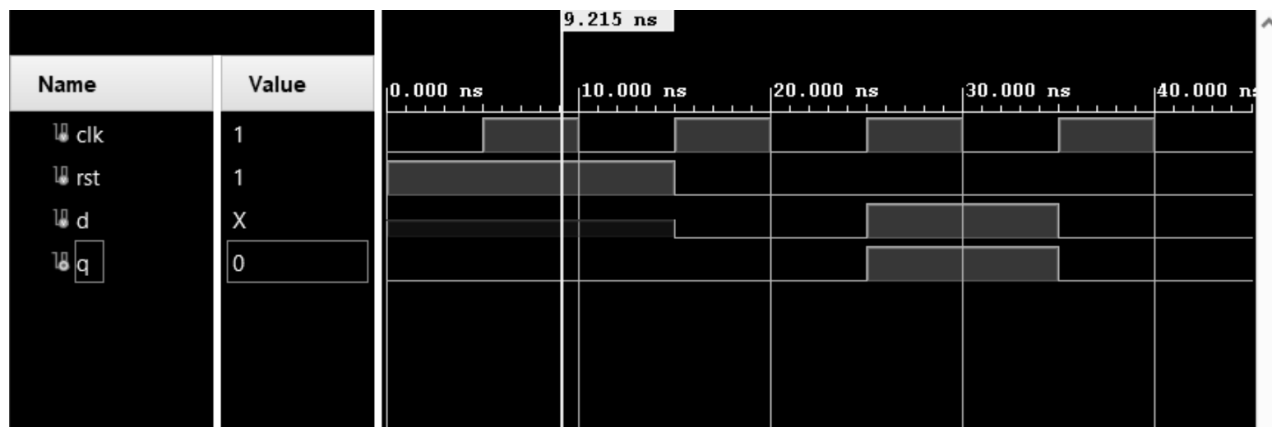
## D FLIP FLOP DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module d_flipflop(
    input clk,
    input rst,
    input d,
    output reg q
);
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            q <= 1'b0;
        end else begin
            q <= d;
        end
    end
endmodule
```

## TESTBENCH FOR D FLIP FLOP

```
//Achutha Aswin Naick
//Roll no. 03
module d_flipflop_tb( );
    reg clk; reg rst; reg d; wire q;
    d_flipflop A(clk,rst,d,q);
    always #5 clk = ~clk;
    initial begin
        clk = 0; rst = 1; #15 rst = 0;
        d = 0; #10 d= 1; #10 d = 0; #10 d=1;
        $stop;
    end
endmodule
```

## SIMULATION



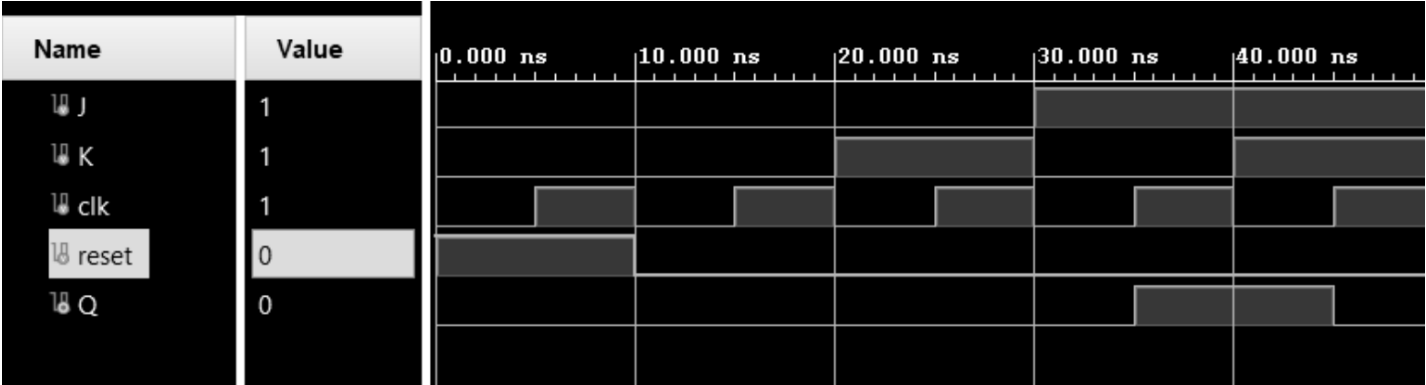
JK FLIP FLOP DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module jk_flipflop(
    input J, K, clk, reset,
    output reg Q
);
always @(posedge clk or posedge reset) begin
    if (reset)
        Q <= 1'b0;
    else begin
        case ({J, K})
            2'b00: Q <= Q;
            2'b01: Q <= 1'b0;
            2'b10: Q <= 1'b1;
            2'b11: Q <= ~Q;
        endcase
    end
end
endmodule
```

TESTBENCH FOR JK FLIP FLOP

```
//Achutha Aswin Naick
//Roll no. 03
module jk_flipflop_tb();
    reg J, K, clk, reset;
    wire Q;
    jk_flipflop uut ( .J(J), .K(K),.clk(clk),.reset(reset), .Q(Q) );
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
    initial begin
        J = 0; K = 0;
        reset = 1;
        #10 reset = 0;
        J = 0; K = 0;
        #10;
        J = 0; K = 1;
        #10;
        J = 1; K = 0;
        #10;
        J = 1; K = 1;
        #10;
        $finish;
    end
end
endmodule
```

SIMULATION



## T FLIP FLOP DESIGN

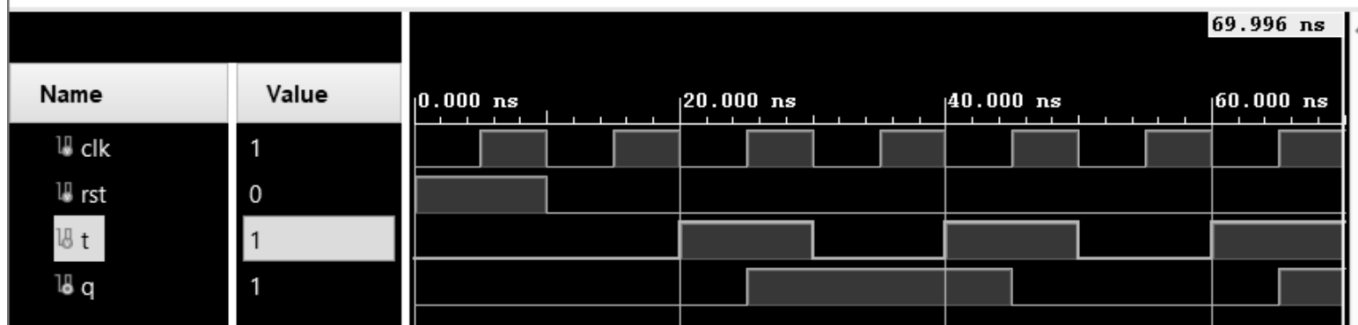
```
//Achutha Aswin Naick
//Roll no. 03
module t_flipflop (
    input wire clk,
    input wire rst,
    input wire t,
    output reg q
);
    always @(posedge clk or posedge rst) begin
        if (rst)
            q <= 0;
        else if (t)
            q <= ~q;
        else
            q <= q;
    end
endmodule
```

## TESTBENCH FOR T FLIP FLOP

```
//Achutha Aswin Naick
//Roll no. 03
module tb_t_flipflop;
    reg clk, rst, t;
    wire q;
    t_flipflop A(.clk(clk),.rst(rst),.t(t), .q(q));
    always #5 clk = ~clk;
    initial begin
        clk = 0;
        rst = 1;
        t = 0;
        #10 rst = 0;

        #10 t = 1;
        #10 t = 0;
        #10 t = 1;
        #10 t = 0;
        #10 t = 1; #10
        $finish;
    end
endmodule
```

## SIMULATION



## Master-Slave FLIP FLOP DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module master_slave_jk_ff(
    input J,
    input K,
    input clk,
    input reset,
    output reg Q
);
reg master, slave;

// Master Flip-Flop
always @(posedge clk or posedge reset) begin
    if (reset)
        master <= 0;
    else begin
        case ({J, K})
            2'b00: master <= master;
            2'b01: master <= 0;
            2'b10: master <= 1;
            2'b11: master <= ~master;
        endcase
    end
end

// Slave Flip-Flop
always @(negedge clk or posedge reset) begin
    if (reset)
        slave <= 0;
    else
        slave <= master; // Slave follows master
end

always @(posedge clk or posedge reset) begin
    if (reset)
        Q <= 0;
    else
        Q <= slave;
end
endmodule
```

## TESTBENCH FOR Master-Slave FLIP FLOP

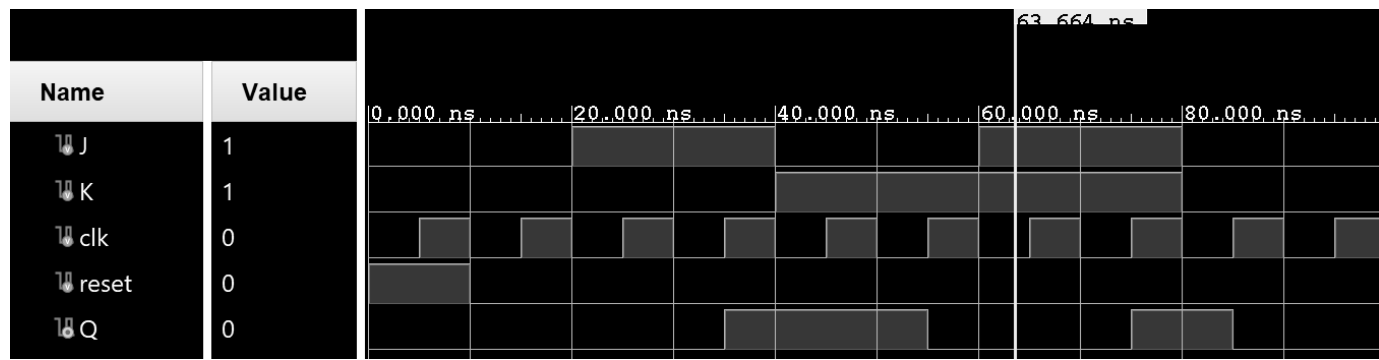
```
//Achutha Aswin Naick
//Roll no. 03
module tb_master_slave_jk_ff();
reg J, K, clk, reset;
wire Q;

master_slave_jk_ff DUT(.J(J),.K(K),.clk(clk),.reset(reset),.Q(Q));

always #5 clk = ~clk;
initial begin
    clk = 0;reset = 1;J = 0;K = 0;
    #10 reset = 0; // Release reset
    #10 J = 1; K = 0; // Set Q
    #20 J = 0; K = 1; // Reset Q
    #20 J = 1; K = 1; // Toggle Q
    #20 J = 0; K = 0; // Hold Q
    #20 $finish;
end

initial begin
    $monitor("%t, J = %b, K = %b, Q = %b", $time, J, K, Q);
end
endmodule
```

## SIMULATION



Realization of an Asynchronous 3 Bit Up/Down Counter

ASYNCHRONOUS 3 BIT UP/DOWN DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module asynch_up_down_counter(
    input clk,reset,M,
    output reg [2:0] count
);

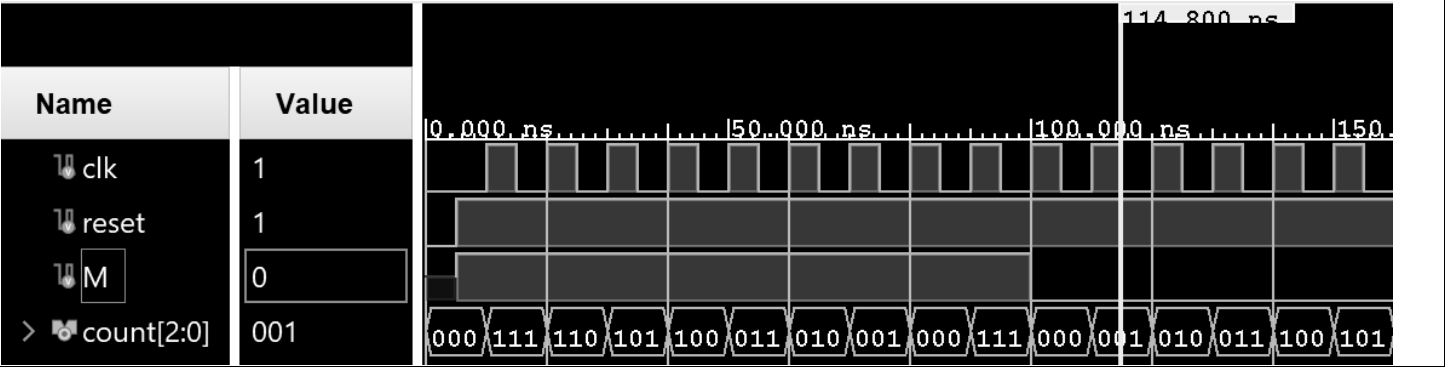
    always @(posedge clk or negedge reset) begin
        if (~reset) count<=3'b000;
        else if (~M) begin
            count[0]<=~count[0];
            count[1]<=count[0] ? ~count[1]:count[1];
            count[2]<=(count[1] & count[0]) ? ~count[2]:count[2];
        end
        else begin
            count[0]<=~count[0];
            count[1]<=~count[0] ? ~count[1]:count[1];
            count[2]<=(~count[1] & ~count[0]) ? ~count[2]:count[2];
        end
    end
endmodule
```

TESTBENCH FOR ASYNCHRONOUS 3 BIT UP/DOWN

```
//Achutha Aswin Naick
//Roll no. 03
module asynch_up_down_counter_tb();
    reg clk,reset,M;
    wire [2:0] count;

    asynch_up_down_counter DUT (clk,reset,M,count);
    initial begin
        $monitor($time,"clk=%b,reset=%b,M=%b,count=%b",clk,reset,M,count);
        reset=0;clk=0;
        #5 reset=1; M=1;
        forever #5 clk=~clk;
    end
    initial #100 $finish;
endmodule
```

SIMULATION



Realization of an Asynchronous 4 Bit Decade Counter

ASYNCHRONOUS 4 BIT DECADE DESIGN

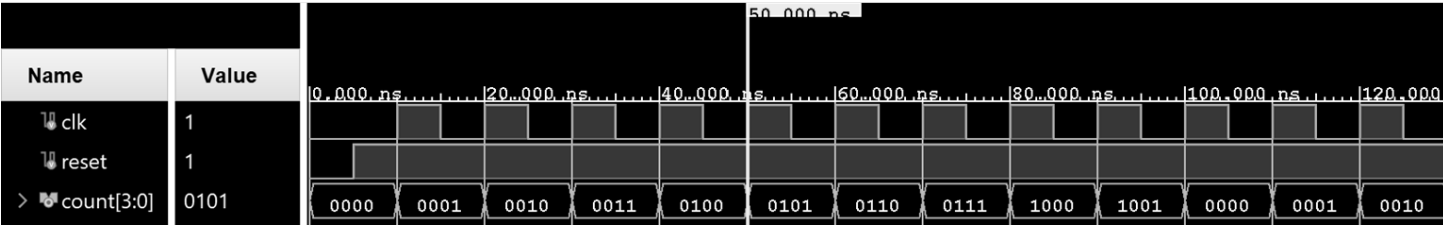
```
//Achutha Aswin Naick
//Rollno. 03
module asynch_decade_counter(
    input clk,reset,
    output reg [3:0] count
);
    always @(posedge clk or negedge reset) begin
        if (~reset)
            count<=4'b0000;
        else begin
            case(count)
                4'b0000: count<=4'b0001;
                4'b0001: count<=4'b0010;
                4'b0010: count<=4'b0011;
                4'b0011: count<=4'b0100;
                4'b0100: count<=4'b0101;
                4'b0101: count<=4'b0110;
                4'b0110: count<=4'b0111;
                4'b0111: count<=4'b1000;
                4'b1000: count<=4'b1001;
                4'b1001: count<=4'b0000;
                default: count<= 4'b0000;
            endcase
        end
    end
endmodule
```

TESTBENCH FOR ASYNCHRONOUS 4 BIT DECADE

```
//Achutha Aswin Naick
//Rollno. 03
module asynch_decade_counter_tb();
    reg clk,reset;
    wire [3:0] count;
    asynch_decade_counter DUT(clk,reset,count);

    initial begin
        $monitor($time,"clk=%b,reset=%b,count=%b",clk,reset,count);
        reset=0; clk=0;
        #5 reset=1;
        forever #5 clk=~clk;
    end
    initial #100 $finish;
endmodule
```

SIMULATION



Realization of a Synchronous 3 Bit Up/Down Counter

SYNCHRONOUS 3 BIT UP/DOWN DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module synch_up_down_counter(
    input clk,reset,M,
    output reg [2:0] count
);

always @(posedge clk or negedge reset)
begin
if (~reset) count<= 3'b000;
else if (!M) count<=count+1;
else count<=count-1;
end
endmodule
```

TESTBENCH FOR SYNCHRONOUS 3 BIT UP/DOWN

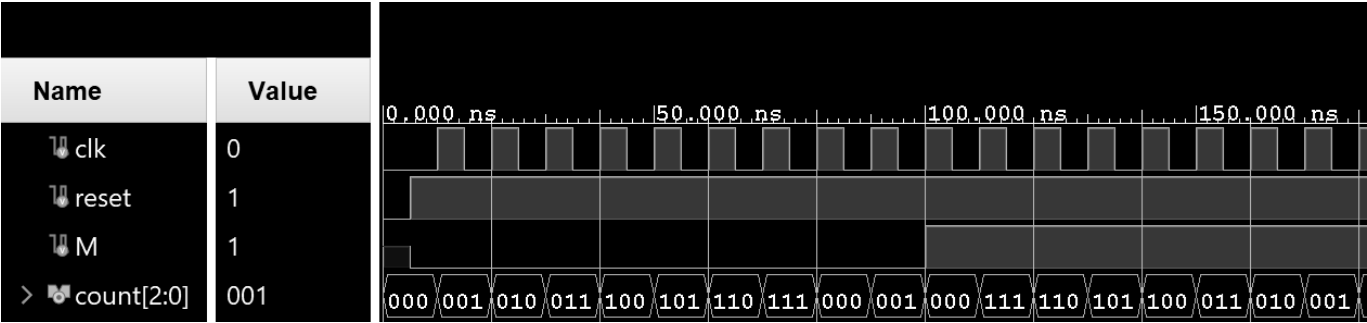
```
//Achutha Aswin Naick
//Roll no. 03
module synch_up_down_counter_tb();
    reg clk,reset,M;
    wire [2:0] count;

    synch_up_down_counter DUT(clk,reset,M,count);

    initial begin
        $monitor($time,"clk=%b,reset=%b,M=%b,count=%b",clk,reset,M,count);
        reset=0;clk=0;
        #5 reset=1; {M}=$random;
        forever #5 clk= ~clk;
    end
    initial
    #100 $finish;

endmodule
```

SIMULATION

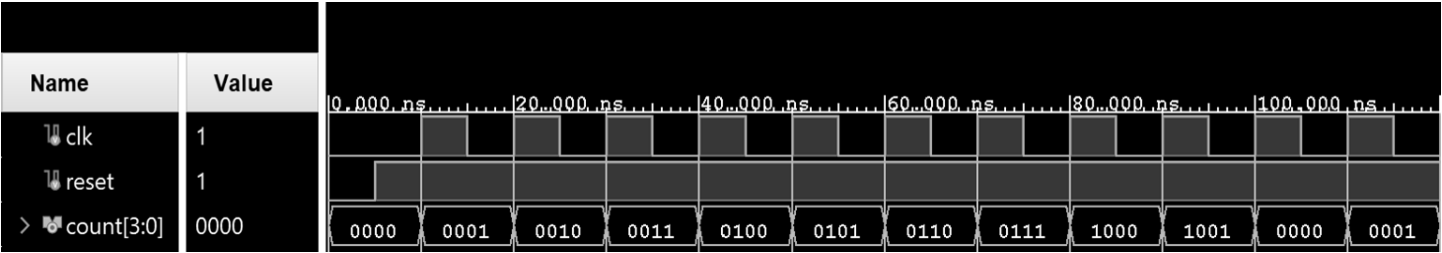




Realization of a Synchronous 4 Bit Decade Counter

SYNCHRONOUS 4 BIT DECADE DESIGN	TESTBENCH FOR SYNCHRONOUS 4 BIT DECADE
<pre>//Achutha Aswin Naick //Roll no. 03 module synch_decade_counter(     input clk,reset,     output reg [3:0] count );     always @(posedge clk or negedge reset) begin         if(~reset    count==4'b1001) begin             count&lt;=4'b0000;         end         else             count&lt;=count+1;         end     end endmodule</pre>	<pre>//Achutha Aswin Naick //Roll no. 03 module synch_decade_counter_tb();     reg clk,reset;     wire [3:0] count;     synch_decade_counter DUT (clk,reset,count);      initial begin         \$monitor (\$time,"clk=%b,reset=%b,count=%b",clk,reset,count);         reset=0;clk=0;         #5 reset=1;         forever #5 clk=~clk;     end     initial         #100 \$finish; endmodule</pre>

SIMULATION



## Realization of Ring and Johnson Counter

### RING COUNTER DESIGN

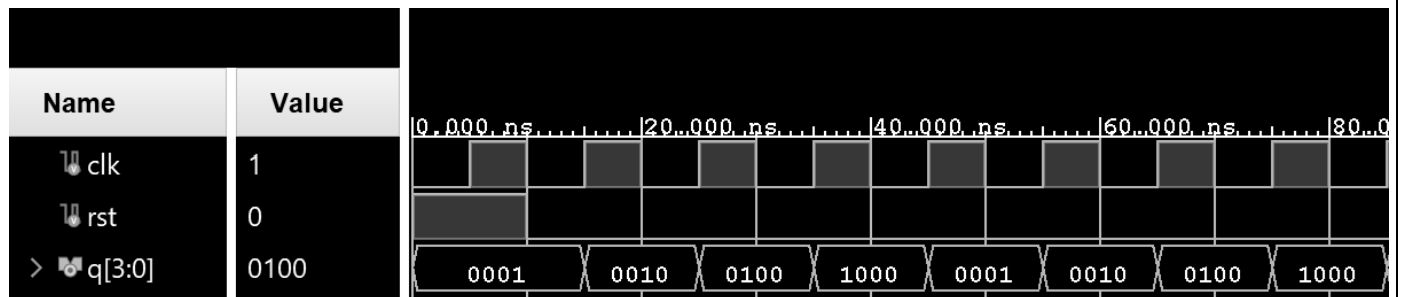
```
//Achutha Aswin Naick
//Roll no. 03
module ring_4bitcounter(
input wire clk,
input wire rst,
output reg [3:0]q);

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

### TESTBENCH FOR RING COUNTER

```
//Achutha Aswin Naick
//Roll no. 03
module ring_4bitcounter_tb;
reg clk;
reg rst;
wire [3:0]q;
ring_4bitcounter uut(clk,rst,q);
initial begin
clk = 0;
forever #5 clk = ~clk;
end
initial begin
rst = 1;
#10 rst = 0;
#100 $finish;
end
endmodule
```

### SIMULATION



JOHNSON COUNTER DESIGN

```
//Achutha Aswin Naick
//Roll no. 03
module johnson_4bitcounter(
input wire clk,
input wire rst,
output reg [3:0] q);

always @(posedge clk or posedge rst) begin
if(rst) begin
q <= 4'b0000;
end else begin
q <= {~q[0], q[3:1]};
end
end
endmodule
```

TESTBENCH FOR JOHNSON COUNTER

```
//Achutha Aswin Naick
//Roll no. 03
module johnson_4bitcounter_tb;
reg clk;
reg rst;
wire [3:0]q;
johnson_4bitcounter uut(clk,rst,q);
initial begin
clk = 0;
forever #5 clk = ~clk;
end
initial begin
rst = 1;
#10;rst = 0;
#100 $finish;
end
endmodule
```

SIMULATION

