30 Days of RTL Coding

-By T Tushar Shenoy

Day 3

Problem Statement: Implementing Sequential Circuits i.e. SR Flip Flop, D Flip-flop, T Flip-flop and JK Flip-flop.

Theory:

Sequential circuits are digital circuits that store and use the previous state information to determine their next state. Unlike combinational circuits, which only depend on the current input values to produce outputs, sequential circuits depend on both the current inputs and the previous state stored in memory elements.

1. Sequential circuits are commonly used in digital systems to implement state machines, timers, counters, and memory elements. The memory elements in sequential circuits can be implemented using flip-flops, which are circuits that store binary values and maintain their state even when the inputs change.
2. There are two types of sequential circuits: finite state machines (FSMs) and synchronous sequential circuits. FSMs are designed to have a limited number of states and are typically used to implement state machines and control systems. Synchronous sequential circuits, on the other hand, are designed to have an infinite number of states and are typically used to implement timers, counters, and memory elements.

SR Flip-Flop

SR flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, SR latch operates with enable signal. The **circuit diagram** of SR flip-flop is shown in the following figure.

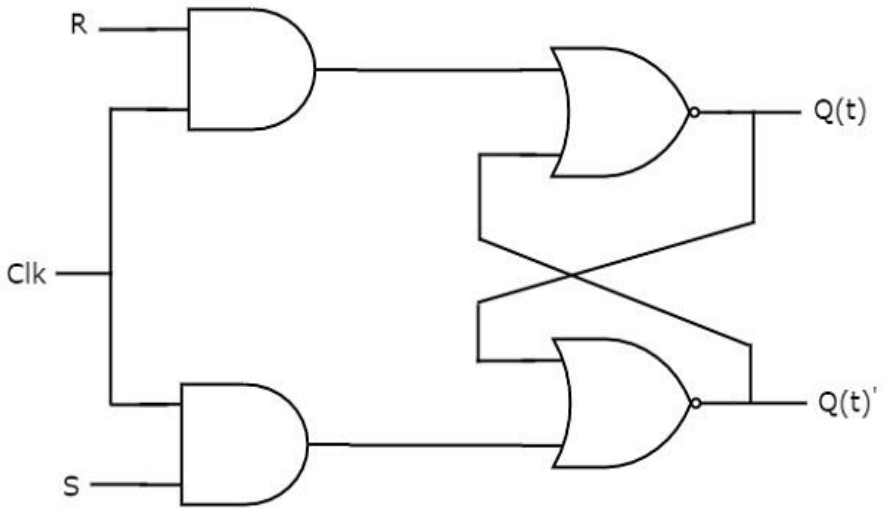


Fig: SR Flip-Flop

This circuit has two inputs S & R and two outputs Qt & Qt' (Q and Qb in the Code). The operation of SR flip-flop is similar to SR Latch. But, this flip-flop affects the outputs only when positive transition of the clock signal is applied instead of active enable.

S	R	$Qt+1$
0	0	Qt
0	1	0
1	0	1
1	1	-

Table: SR Flip-Flop Truth Table

D Flip-Flop

D flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, D latch operates with enable signal. That means, the output of D flip-flop is insensitive to the changes in the input, D except for active transition of the clock signal. The **circuit diagram** of D flip-flop is shown in the following figure.

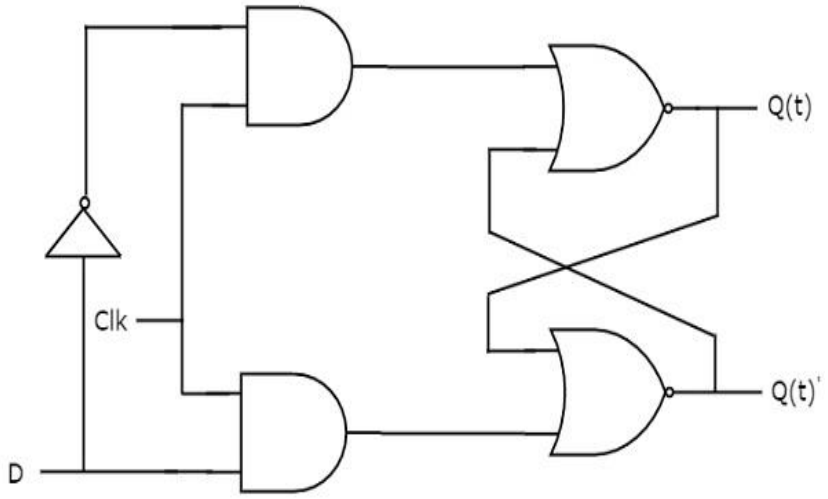


Fig: D Flip-Flop

This circuit has single input D and two outputs Qt & Qt' (Q and Qb in the Code). The operation of D flip-flop is similar to D Latch. But, this flip-flop affects the outputs only when positive transition of the clock signal is applied instead of active enable.

D	Qt+1
0	0
1	1

Table: D Flip-Flop Truth Table

T Flip-Flop

T flip-flop is the simplified version of JK flip-flop. It is obtained by connecting the same input 'T' to both inputs of JK flip-flop. It operates with only positive clock transitions or negative clock transitions. The **circuit diagram** of T flip-flop is shown in the following figure.

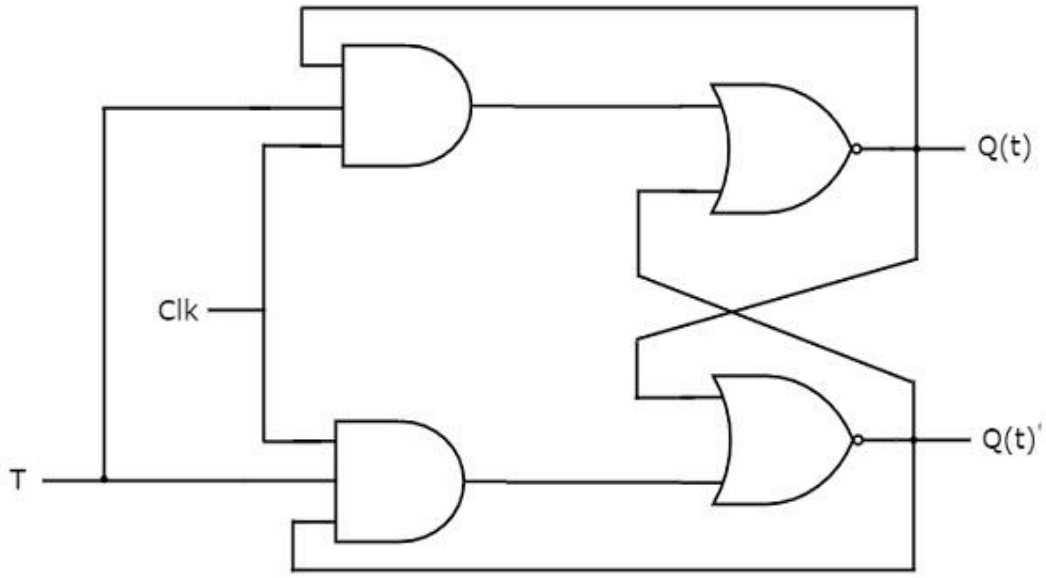


Fig: T Flip-Flop

This circuit has single input T and two outputs Qt & Qt' (Q and Qb in the Code). The operation of T flip-flop is same as that of JK flip-flop. Here, we considered the inputs of JK flip-flop as **J = T** and **K = T** in order to utilize the modified JK flip-flop for 2 combinations of inputs. So, we eliminated the other two combinations of J & K, for which those two values are complement to each other in T flip-flop.

T	**Q$t+1$**
0	Qt
1	Qt'

Table: T Flip-Flop Truth Table

JK Flip-Flop

JK flip-flop is the modified version of SR flip-flop. It operates with only positive clock transitions or negative clock transitions. The **circuit diagram** of JK flip-flop is shown in the following figure.

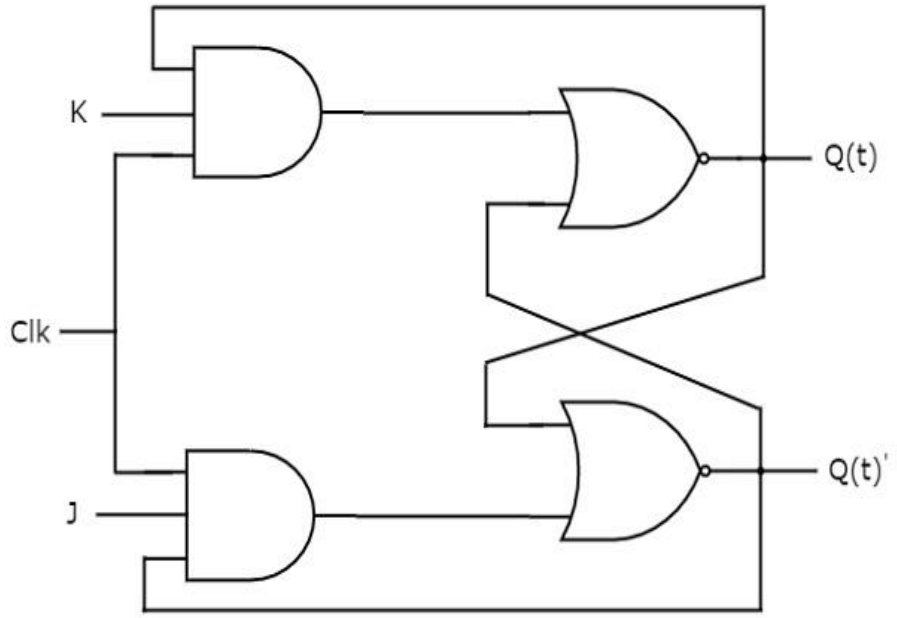


Fig: JK Flip-Flop

This circuit has two inputs J & K and two outputs Qt & Qt' (Q and Qb in the Code). The operation of JK flip-flop is similar to SR flip-flop. Here, we considered the inputs of SR flip-flop as **S =J Q**t**'** and R= **KQ**t in order to utilize the modified SR flip-flop for 4 combinations of inputs.

J	**K**	**Q**$t+1$
0	0	Qt
0	1	0
1	0	1
1	1	Qt'

Table: JK Flip-Flop Truth Table

1. SR Flip-Flop

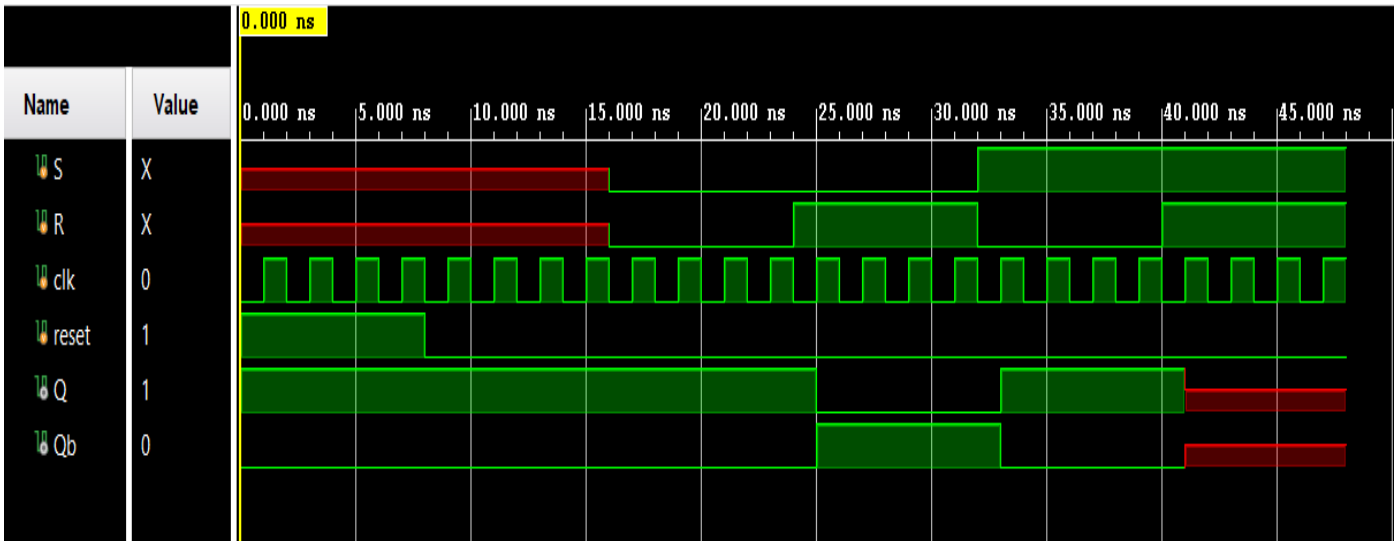
Verilog Code:

```
module SR_Flip_Flop ( S, R, clk, reset,Q, Qb);
 input S, R, clk, reset;
 output reg Q,Qb;
always @(posedge clk or reset) // Posedge clock and asynchronous Reset
begin
    if (reset)
    begin
        Q <= 1'b1;
        Qb <= 1'b0;
    end
    else
    begin
        case ({S, R})
            2'b00: Q= Q;
            2'b01: Q= 1'b0;
            2'b10: Q= 1'b1;
            2'b11: Q= 1'bx;
        endcase
         Qb=~Q;
    end
end
endmodule
```

Testbench Code:

```
module SR_Flip_Flop_tb();
reg S,R,clk,reset;
wire Q,Qb;

SR_Flip_Flop dut(.S(S),.R(R),.clk(clk),.reset(reset),.Q(Q),.Qb(Qb));

initial begin
clk=1'b0;
reset=1'b1;
#8 reset=1'b0;

//Stimulus
#8   S=1'b0;R=1'b0;
#8   S=1'b0;R=1'b1;
#8   S=1'b1;R=1'b0;
#8   S=1'b1;R=1'b1;
#8   $finish;
end

always #1 clk=~clk;
endmodule
```

Simulation Output:



2. D Flip-Flop

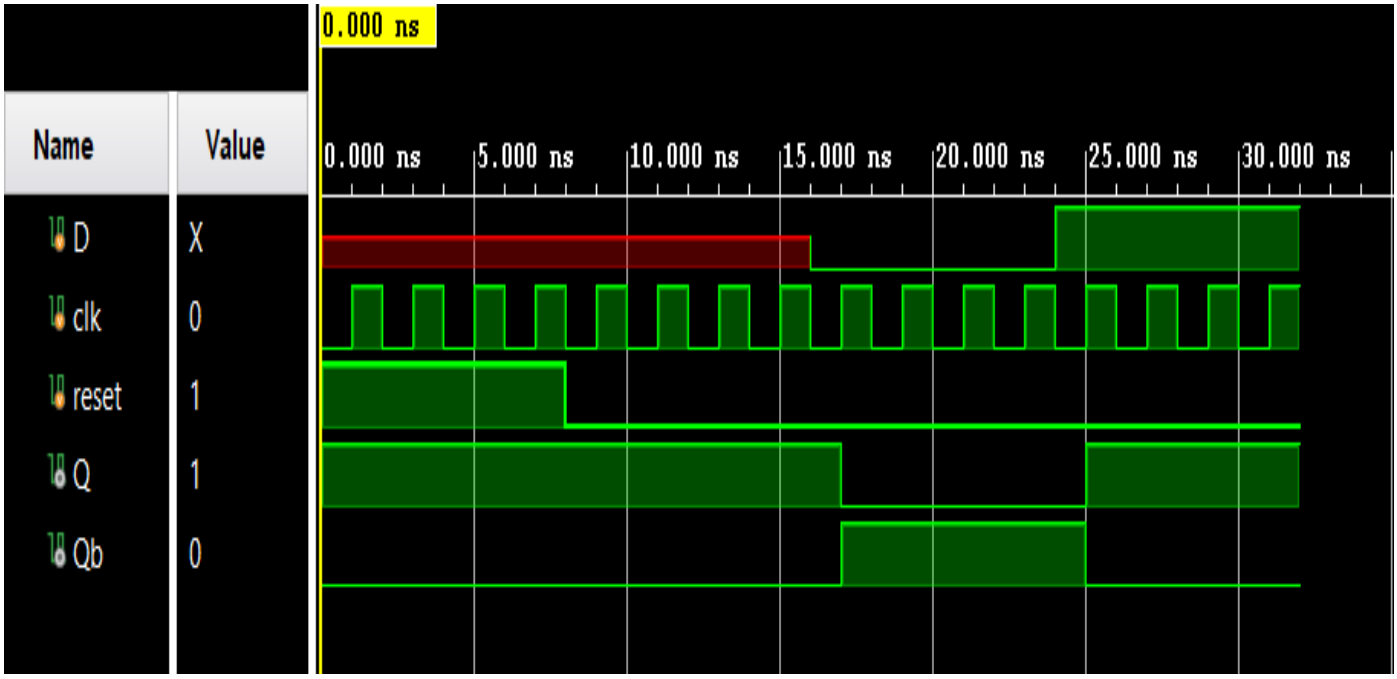
Verilog Code:

```
module D_Flip_Flop(D, clk, reset,Q, Qb);
 input D,clk, reset;
 output reg Q,Qb;


always @(posedge clk or posedge reset) // Posedge clock and asynchronous Reset
begin
    if (reset)
    begin
        Q <= 1'b1;
        Qb <= 1'b0;
    end
    else
    begin
        case(D)
            1'b0: Q= 1'b0;
            1'b1: Q= 1'b1;
        endcase
         Qb=~Q;
    end
end
endmodule
```

Testbench Code:

```
module D_Flip_Flop_tb();
reg D,clk,reset;
wire Q,Qb;

D_Flip_Flop dut(.D(D),.clk(clk),.reset(reset),.Q(Q),.Qb(Qb));

initial begin
clk=1'b0;
reset=1'b1;
#8 reset=1'b0;

//Stimulus
#8  D=1'b0;
#8  D=1'b1;
#8  $finish;
end

always #1 clk=~clk;
endmodule
```

Simulation Output:



3. T Flip-Flop

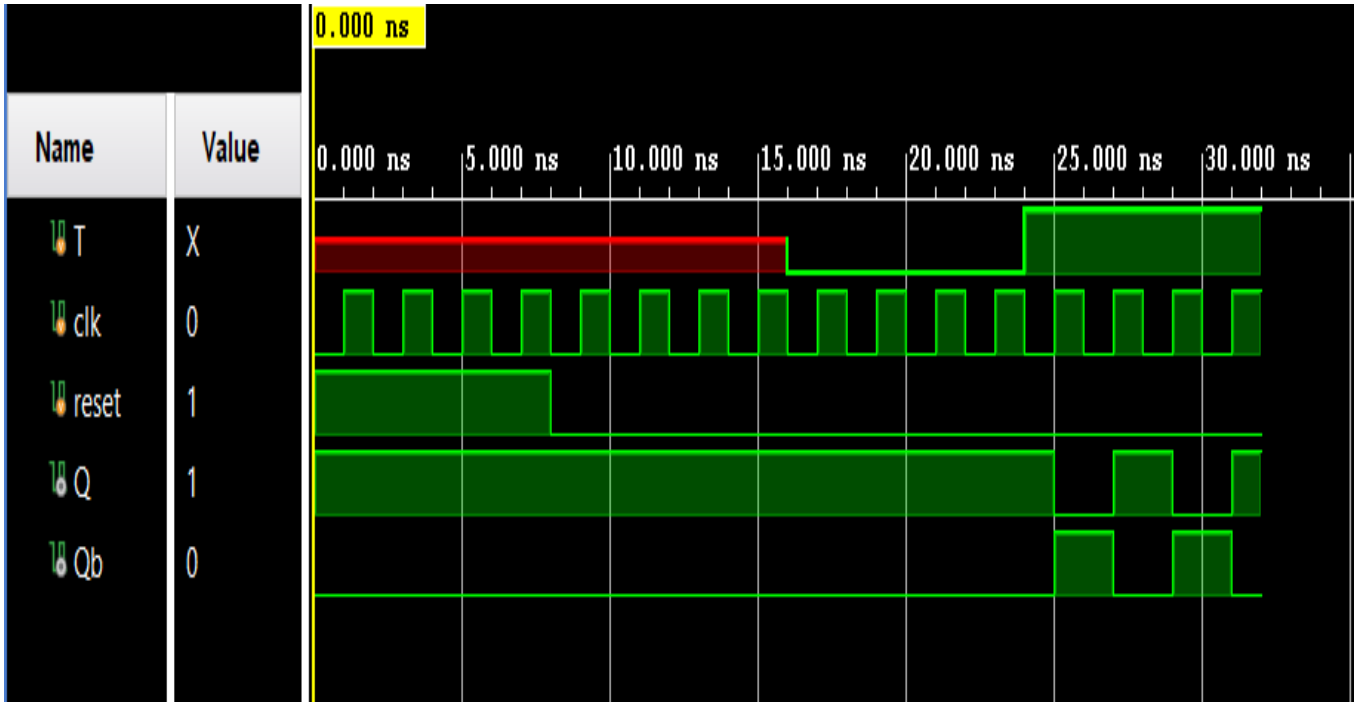
Verilog Code:

```
module T_Flip_Flop(T, clk, reset,Q, Qb);
 input T,clk, reset;
 output reg Q,Qb;


always @(posedge clk or posedge reset) // Posedge clock and asynchronous Reset
begin
    if (reset)
    begin
        Q <= 1'b1;
        Qb <= 1'b0;
    end
    else
    begin
        case(T)
            1'b0: Q= Q;
            1'b1: Q= ~Q;
        endcase
         Qb=~Q;
    end
end
endmodule
```

Testbench Code:

```
module T_Flip_Flop_tb();
reg T,clk,reset;
wire Q,Qb;

T_Flip_Flop dut(.T(T),.clk(clk),.reset(reset),.Q(Q),.Qb(Qb));

initial begin
clk=1'b0;
reset=1'b1;
#8 reset=1'b0;

//Stimulus
#8  T=1'b0;
#8  T=1'b1;
#8  $finish;
end

always #1 clk=~clk;
endmodule
```

Simulation Output:



4. JK Flip-Flop

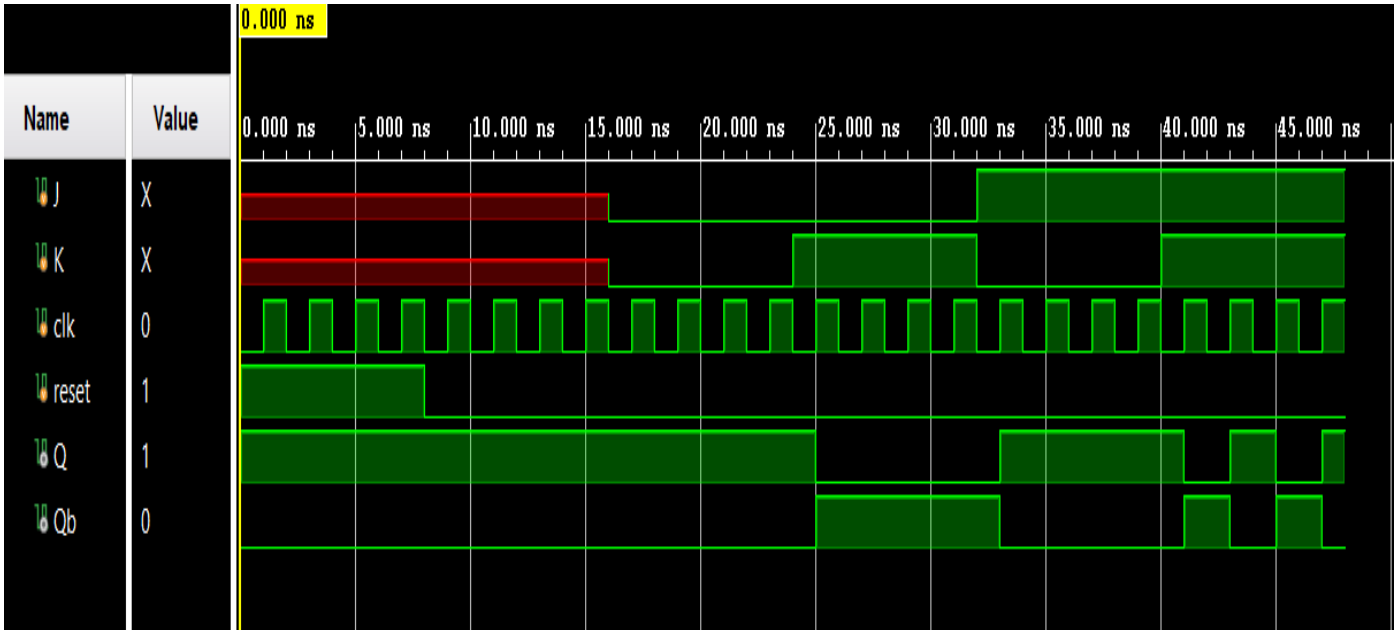
Verilog Code:

```
module JK_Flip_Flop ( J, K, clk, reset,Q, Qb);
 input J, K, clk, reset;
 output reg Q,Qb;
always @(posedge clk or posedge reset) // Posedge clock and asynchronous Reset
begin
    if (reset)
    begin
        Q <= 1'b1;
        Qb <= 1'b0;
    end
    else
    begin
        case ({J, K})
            2'b00: Q= Q;
            2'b01: Q= 1'b0;
            2'b10: Q= 1'b1;
            2'b11: Q= ~Q;
        endcase
         Qb=~Q;
    end
end
endmodule
```

Testbench Code:

```
module JK_Flip_Flop_tb();
reg J,K,clk,reset;
wire Q,Qb;

JK_Flip_Flop dut(.J(J),.K(K),.clk(clk),.reset(reset),.Q(Q),.Qb(Qb));

initial begin
clk=1'b0;
reset=1'b1;
#8 reset=1'b0;

//Stimulus
#8   J=1'b0;K=1'b0;
#8   J=1'b0;K=1'b1;
#8   J=1'b1;K=1'b0;
#8   J=1'b1;K=1'b1;
#8   $finish;
end

always #1 clk=~clk;
endmodule
```

Simulation Output:



GitHub Repository URL: - https://github.com/tusharshenoy/RTL-Day-3-Sequential-Circuits