SEQUENTIAL CIRCUITS

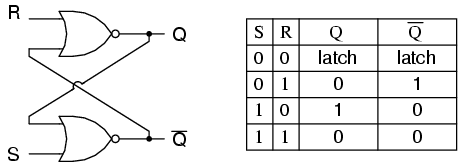
A sequential circuit is a digital logic circuit that includes memory elements to store the history of past operations. Its output depends not only on the present inputs but also on the past outputs stored in its memory. This enables sequential circuits to perform time-dependent operations.

**LATCHES**

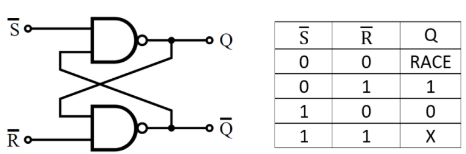
Latches are level-sensitive, meaning they are transparent (allow data to pass through) when the control signal (e.g., enable) is active, and they hold their state when the control signal is inactive.

**SR Latch (Set-Reset)**

SR latch is a circuit with two cross-coupled NOR gates or two cross coupled NAND gates, and two inputs labeled S for set, and R for reset.







Hold

Reset

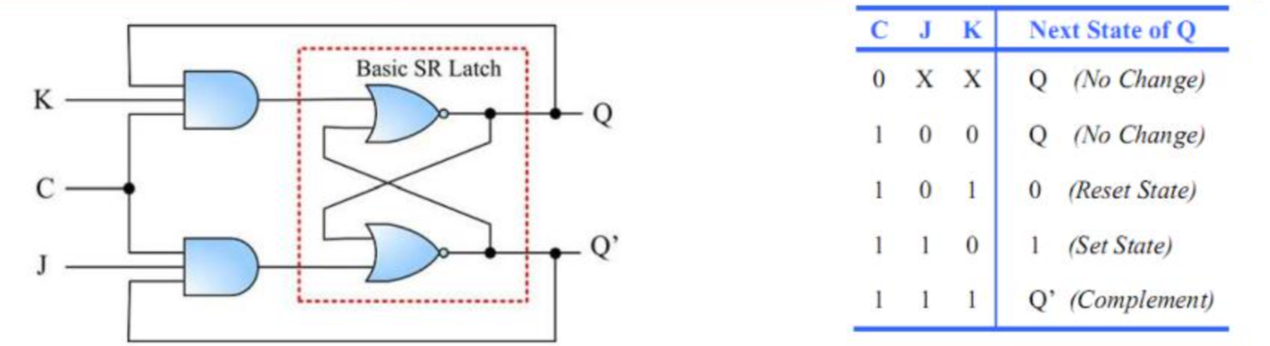
Set

Invalid

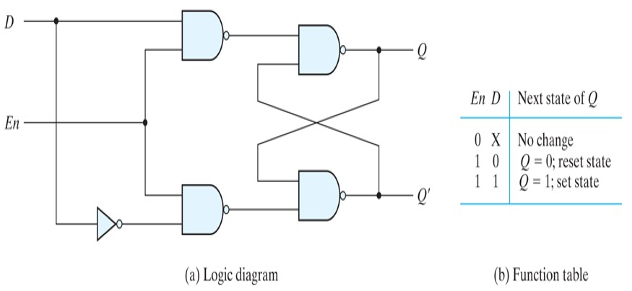


\*\* Invalid or race condition arises then both Q and Q’ both are equal at a time, action produces an undefined next state, because the state that results from the input transitions depends on the order in which they return to 0. So both input should not be same at a time which is handle by using D LATCH OR JK LATCH OR T LATCH.

**JK LATCH**



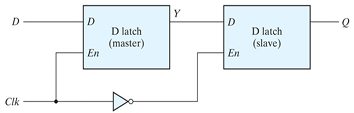
**D Latch (Transparent)**

****

\*\***Race Around Condition In JK latchs –** For J-K latches, if J=K=1, and if clock is enable for a long period of time, then Q output will toggle as long as CLK is high, which makes the output of the flip-flop unstable or uncertain. This problem is called race around condition in J-K flip-flop.

Solutions-

MASTER SLAVE OR EDGE TRIGGER FLIP FLOP



**FLIP FLOP**

MASTER SLAVE

module MasterSlave\_FlipFlop ( input clk, input reset, input D, output reg Q );

reg master;

always @(posedge clk or posedge reset)

begin if (reset) master <= 1'b0; // Reset master to 0

else master <= D; // Capture input into master

end

// Slave flip-flop (negative edge of the clock)

always @(negedge clk or posedge reset)

begin if (reset) Q <= 1'b0; // Reset slave to 0

else

Q <= master; // Transfer master data to slave

end

endmodule

Common Issues with Latches-

* Timing Hazards: Latches can introduce glitches in asynchronous designs.
* Simulation vs. Synthesis Mismatch: Improperly inferred latches may behave differently in hardware.
* Unintentional Latch Inference: Occurs when all possible conditions are not specified combinational logic.

1. Missing else Statement

always @\* begin

if (A)

Y = 1; // No `else` -> latch inferred

end

1. Partial Case Statement:

always @\* begin

case (sel)

2'b00: Y = A;

2'b01: Y = B;

// Missing cases -> latch inferred also DEFAULT

endcase

end

1. Missing sensitive elements

always @(A) begin

Y = B;

else if (A)

Y = 0; // B not included in the sensitivity list

end

JK LATCHE

module JK\_Latch (

input J , K,EN,

output reg Q // Output

);

always @(\*) begin

if (En)

begin

case ({J, K})

2'b00: Q = Q; // Hold

2'b01: Q = 0; // Reset

2'b10: Q = 1; // Set

2'b11: Q = ~Q; // Toggle

endcase

end

end

endmodule

**Applications of Latches:**

* TEMPORARY STORAGE.
* MEMORY ELEMENTS.
* SYNCHRONIZA -TION
* GATING SIGNAL

T LATCHE

module T\_Latch (

input T, En,

output reg Q // Output);

always @(\*) begin

if (En) begin

if (T)

Q = ~Q; // Toggle

end

end

endmodule

D LATCHE

module D\_Latch (

input D, // Data

input En, // Enable

output reg Q );

always @(\*) begin

if (En)

Q = D; // Transparent

end

endmodule

SR LATCHE

module SR\_Latch (

input S, // Set

input R, // Reset

output Q, // Output

output Qn // Inverted

);

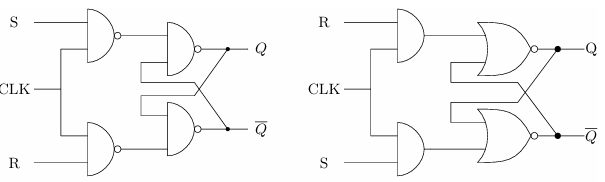
assign Q = ~(R | Qn);

assign Qn = ~(S | Q);

endmodule

A flip-flop is a basic memory element in digital electronics that can store one bit of data. It has two stable states, 0 and 1, and changes its state on the active edge of a triggering clock signal or control input

**SR FLIP FLOP**

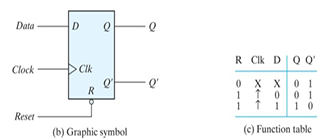


NAND BASED NOR BASED

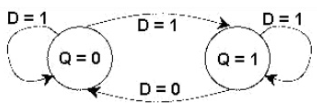
|  |  |  |
| --- | --- | --- |
| S | R | QN+1 |
| 0 | 0 | QN(HOLD) |
| 0 | 1 | 0 (RESET) |
| 1 | 0 | 1(SET) |
| 1 | 1 | X(INVALID) |

CHARACTERSTIC EQUATION: Qn+1= S+R’Qn

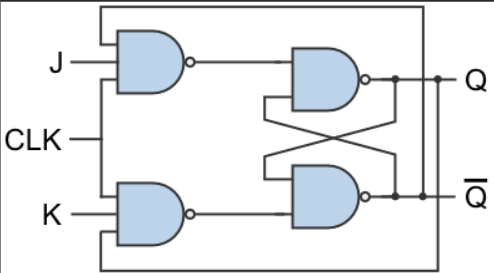
**D-FLIP FLOP**



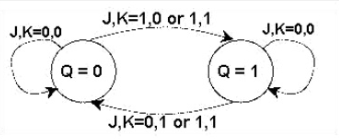
CHARACTERSTIC EQUATION: Qn+1 = D

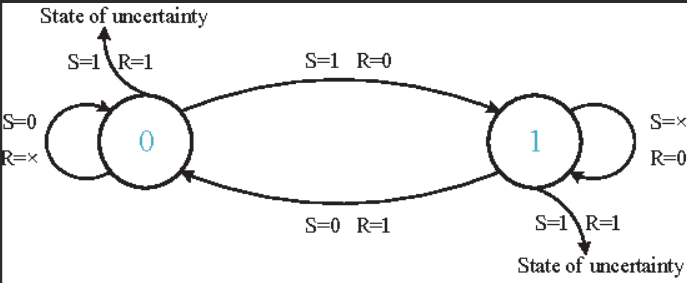


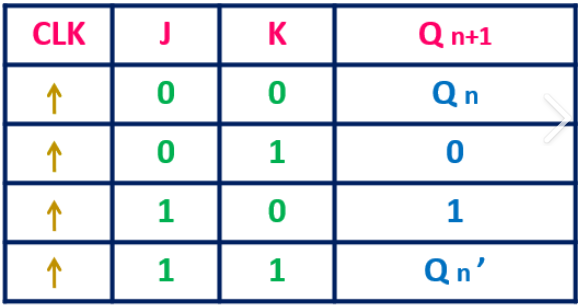
**JK FLIP FLOP**



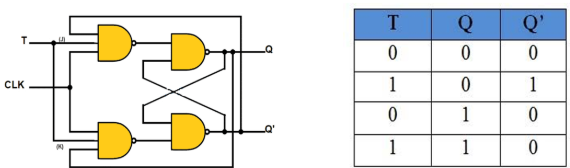
CHARACTERSTIC EQUATION: Qn+1 = JQ’ +K’Q



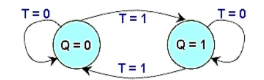




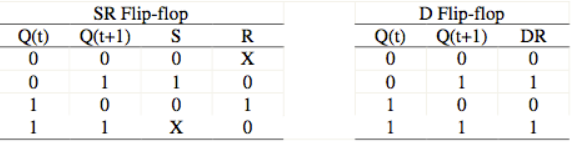
**T FLIP FLOP**

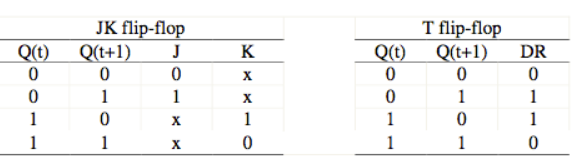


CHARACTERSTIC EQUATION: Qn+1 =



EXITATION TABLE:





**D-FF**

module dff(q,reset ,clk,d);

output reg q; input reset ,d,clk;

initial

begin

q=1’b0;

end

always @ (posedge clk)

if (reset) q <= 1’b0; else q<=d;

endmodule

**SR -FF**

module srff (S,R,clk,reset ,q,qb);

output reg q , qb ;

input S,R,clk,reset;

initial

begin

q=1’b0;

qb =1’b1;

end

always @ (posedge clk)

if (reset)

begin

q <= 0;

qb <= 1;

end

else

begin

if (S!=R)

begin

q <=S;

qb <=R;

end

else

if (S==1 && R==1)

begin

q <= 1’bZ;

qb <= 1’bZ;

end

end

endmodule

**JK-FF**

module jk(q,qb,j ,k,reset ,clk); output reg q,qb;

input j ,k,clk,reset;

initial

begin

q = 1’b0;

qb = 1’b1;

end

always @ (posedge clk)

if(reset)

begin

q = 1’b0;

qb = 1’b1;

end

else

case({j ,k})

{1’b0,1’b0}: begin

q=q; qb=qb;

end

{1’b0,1’b1}: begin

q=1’b0; qb=1’b1;

end

{1’b1,1’b0}: begin

q=1’b1; qb=1’b0;

end

{1’b1,1’b1}: begin

q=~q; qb=~qb;

end

endcase

endmodule

1

**T-FF**

module tff(q,reset ,clk,t);

output reg q; input T,reset ,clk;

initial

begin

q=1’b0;

end

always @ (posedge clk)

if (reset)

q <= 1’b0;

else if (T) q= ∼q;

else q = q;

endmodule

OR

always @ (posedge clk)

if (reset)

q <= 1’b0;

else q <= q ^t;

endmodule

| **Feature** | **Latch** | **Flip-Flop** |
| --- | --- | --- |
| **Sensitivity** | Level-sensitive (triggered by input level or enable signal). | Edge-sensitive (triggered on clock edge, either rising or falling). |
| **Clock Dependency** | Does not depend on the clock directly; uses enable signals. | Depends on clock edges (positive or negative). |
| **Behaviour** | Continuously follows input when enabled. | Changes state only at specific clock edges. |
| **Timing Control** | Transparent while enabled, can cause glitches. | Operates at discrete intervals, ensuring more stable operation. |
| **Speed** | Faster but less reliable due to continuous transparency. | Slower due to clock dependency, but more reliable. |
| **Design Complexity** | Simple design. | More complex design due to clock synchronization. |
| **Usage** | Temporary storage or small data path designs. | Sequential circuits, registers, and synchronous designs. |
| **Power Consumption** | Lower power consumption as they do not require a clock. | Higher power consumption due to the clocking mechanism. |
| **Metastability** | More prone to metastability issues. | Better metastability handling due to clock synchronization. |

* IIT KHARAGPUR **: PROF. INDRANIL SENGUPTA** ( HARDWARE MODELING USING VERILOG)

<https://www.youtube.com/watch?v=NCrlyaXMAn8&list=PLJ5C_6qdAvBELELTSPgzYkQg3HgclQh-5>

* **ANKIT GOYAL SIR**: DIGITAL ELECTRONICS

<https://youtube.com/playlist?list=PLs5_Rtf2P2r41iuDKULDHHnIwfXyTAxBH&si=PhMhOvo8_rT1a53y>

* BOOKS:

DIGITAL ELECTRONICS: **Digital Logic and Computer Design by M. Morris Mano.**

VERILOG : **Verilog HDL - Samir Palnitkar**

Digital Design With an Introduction to the Verilog HDL, VHDL, and SystemVerilog by **M. Morris Mano and Michael D. Ciletti**

R