DIGITAL DESIGN

LAB9 SYNCHRONOUS SEQUENTIAL CIRCUIT

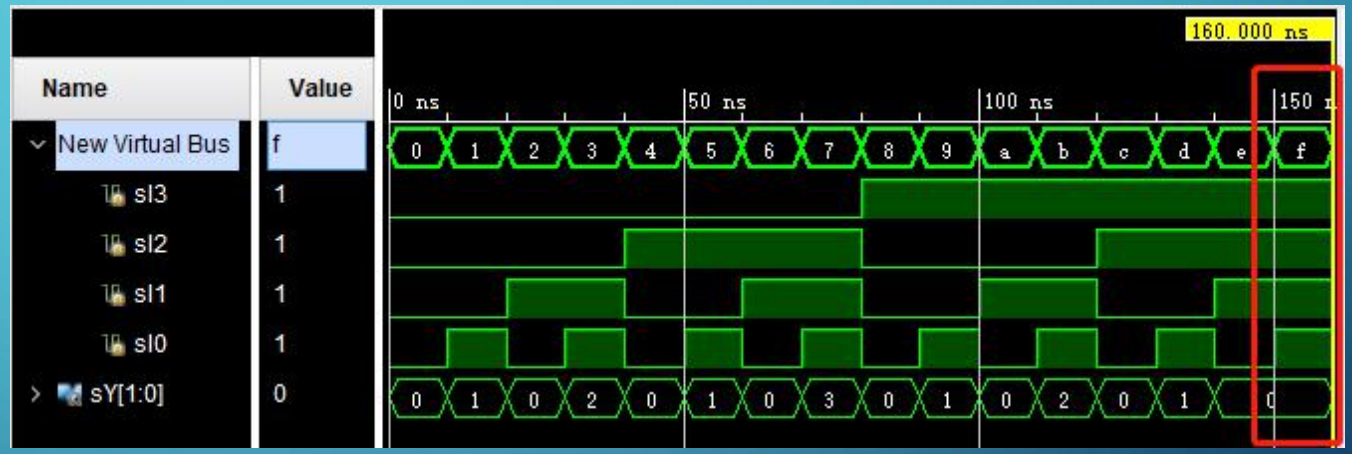
2021 SUMMER TERM

LAB9

- Synchronous sequential circuit
  - Latch
  - Flip Flop (key point of lab10)
  - The conversion between each other
- Practice

4-2 PRIORITY-ENCODER IN LAB8

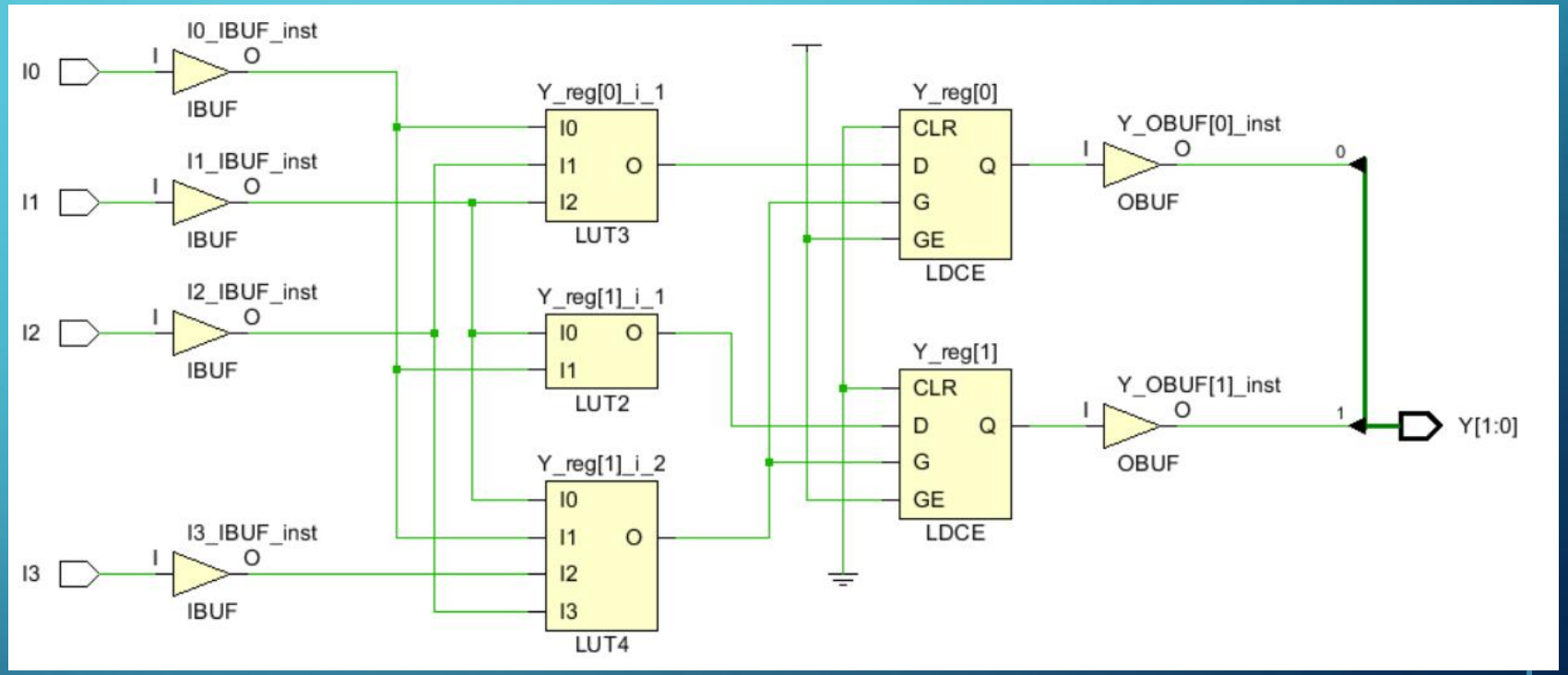
```
//4-2 priencoder
module encoder(
    input I0,
    input I1,
    input I2,
    input I3,
    output reg [1:0] Y
    );
    always @*
     begin
        casex ({I3, I2, I1, I0})
            4'bxxx0: Y=2'b00;
            4'bxx01: Y=2'b01;
            4'bx011: Y=2'b10;
            4'b0111: Y=2'b11;
        endcase
    end
endmodule
```



If there is no default and the case branch is incomplete, when the case branch is not listed in the case branch, the circuit state will remain unchanged, and then a latch will be generated to save the state.

LATCH

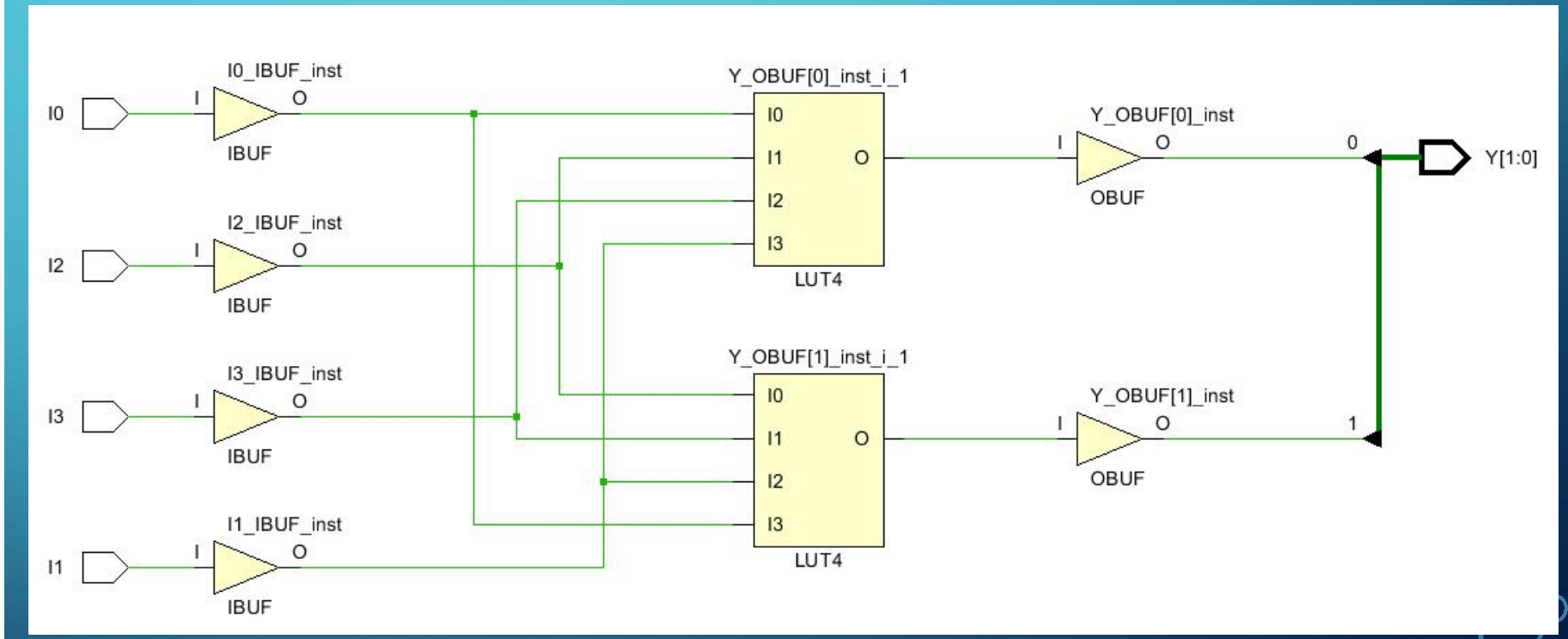
- The schematic of the synthesized design.
- LDCE: transparent Data Latch with Asynchronous Clear and Gate Enable



For more information on LDCE, please refer to ***Xilinx 7 Series FPGA Libraries Guide for Schematic Design***

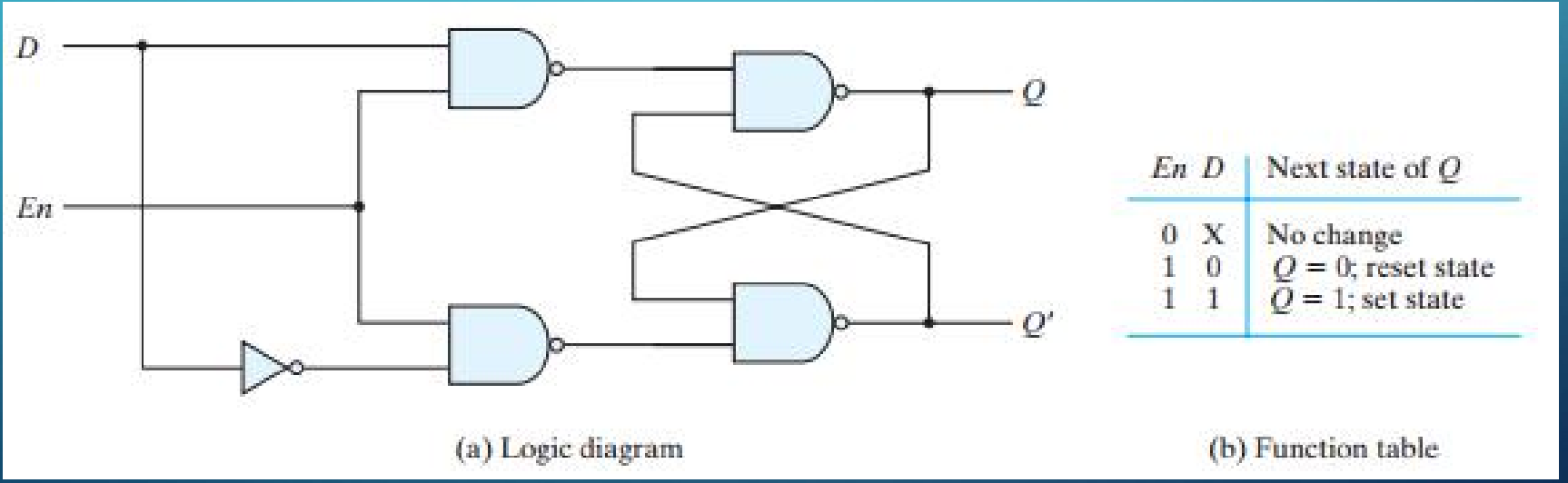
```
module encoder(
    input I0,
    input I1,
    input I2,
    input I3,
    output reg [1:0] Y
    );

    always @*
    begin
        casex({I3, I2, I1, I0})
            4'bxxx0: Y = 2'b00;
            4'bxx01: Y = 2'b01;
            4'bx011: Y = 2'b10;
            4'b0111: Y = 2'b11;
            default: Y = 2'b00;
        endcase
    end
endmodule
```



D LATCH

- Latch: the simplest binary memory elements, static device composed of gates. When the input changes, the output changes immediately.



(a) Logic diagram

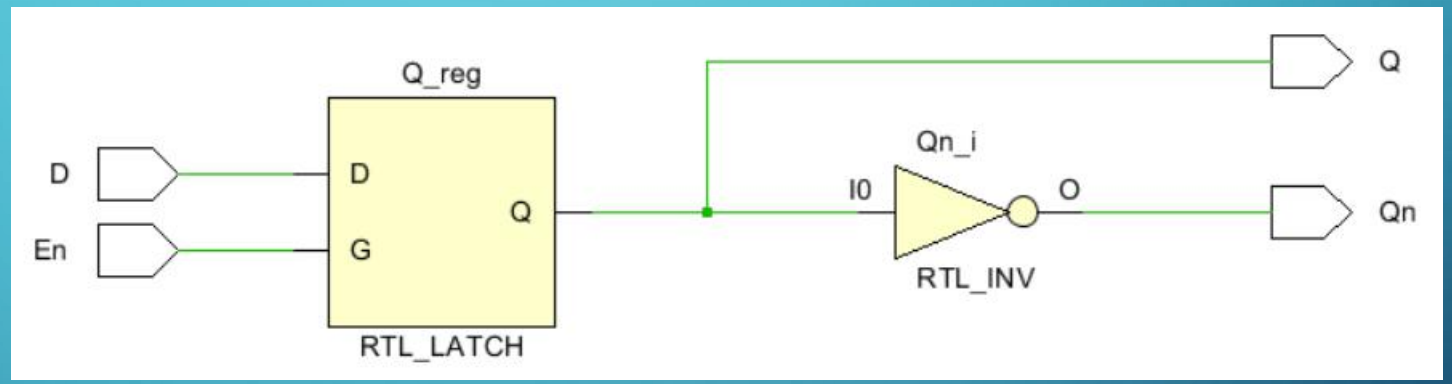
En	D	Next state of Q
0	X	No change
1	0	Q = 0; reset state
1	1	Q = 1; set state

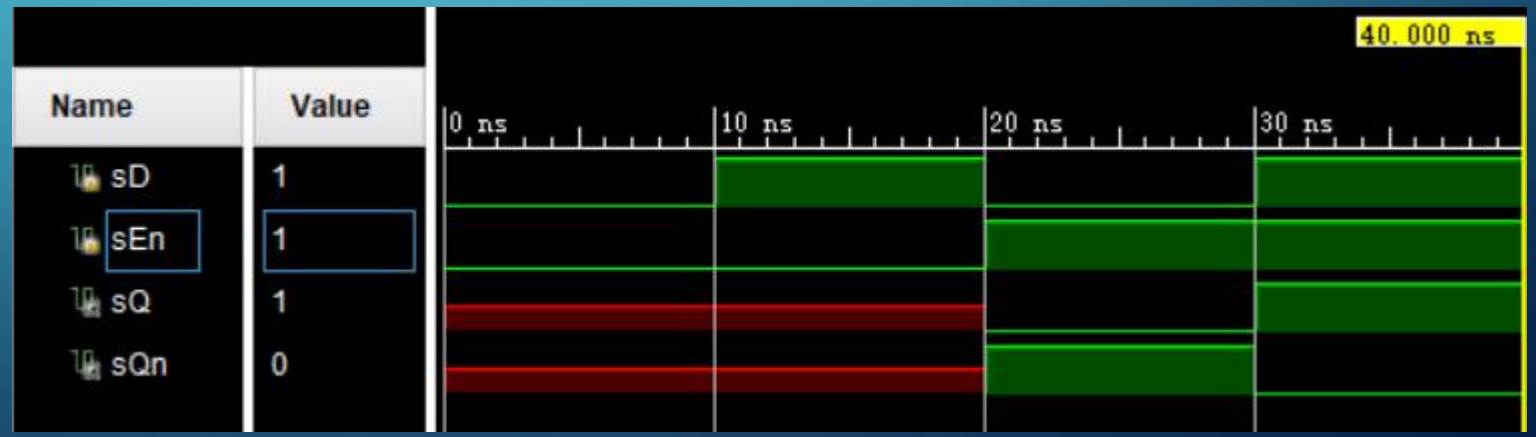
(b) Function table

D LATCH CONTINUED

- Schematic of RTL analysis

```
module D_Latch(
    input En, D,
    output reg Q,
    output Qn
    );
    assign Qn = ~Q;
    always @*
        if (En)
        begin
            Q = D;
        end
        else
            Q = Q;
endmodule
```





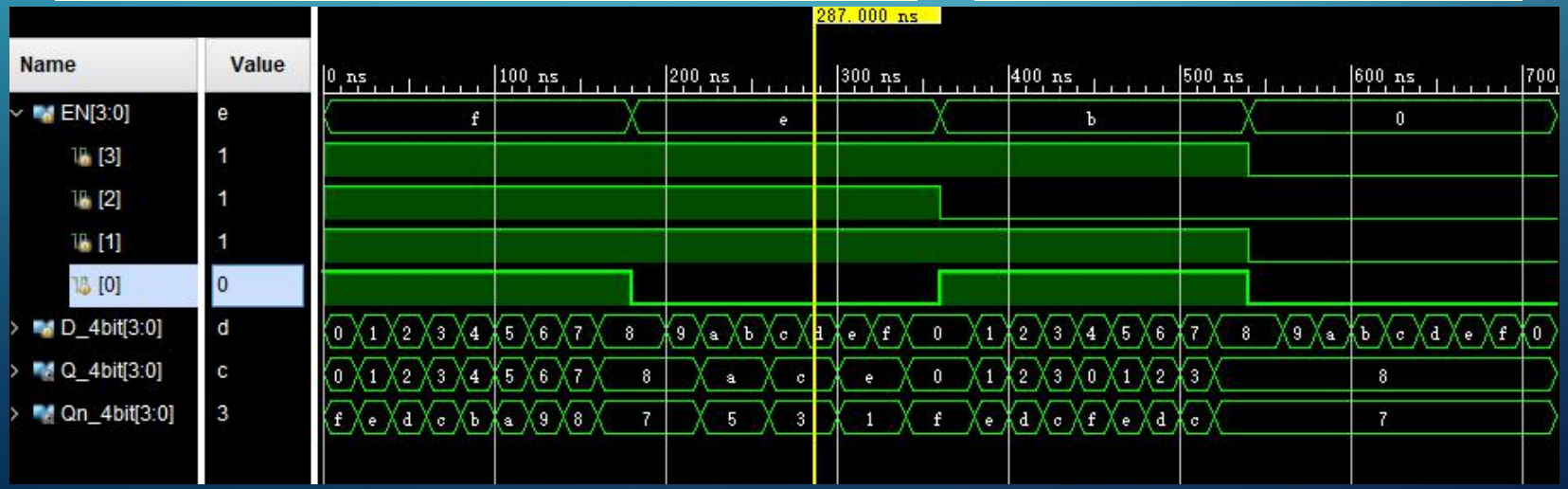
D LATCH CONTINUED

- Generate latch of multi-bit

```
module Latch_4bit(
    input [3:0] EN,
    input [3:0] D_4bit,
    output [3:0] Q_4bit,
    output [3:0] Qn_4bit
    );
    D_Latch UUT3(.EN(EN[3]),.D(D_4bit[3]),.Q(Q_4bit[3]),.Qn(Qn_4bit[3]));
    D_Latch UUT2(.EN(EN[2]),.D(D_4bit[2]),.Q(Q_4bit[2]),.Qn(Qn_4bit[2]));
    D_Latch UUT1(.EN(EN[1]),.D(D_4bit[1]),.Q(Q_4bit[1]),.Qn(Qn_4bit[1]));
    D_Latch UUT0(.EN(EN[0]),.D(D_4bit[0]),.Q(Q_4bit[0]),.Qn(Qn_4bit[0]));
endmodule
```

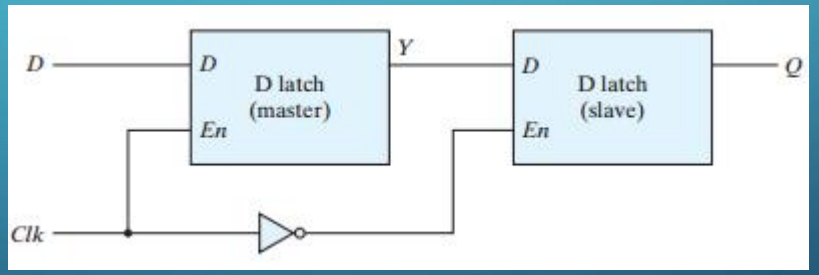
```
Latch_4bit tb(
    .EN(EN),
    .D_4bit(D_4bit),
    .Q_4bit(Q_4bit),
    .Qn_4bit(Qn_4bit)
    );

initial begin
    D_4bit = 4'b0000;
    EN = 4'b1111;
    repeat(8) #20 D_4bit = D_4bit + 1;
    #20 EN = 4'b1110;
    repeat(8) #20 D_4bit = D_4bit + 1;
    #20 EN = 4'b1011;
    repeat(8) #20 D_4bit = D_4bit + 1;
    #20 EN = 4'b0000;
    repeat(8) #20 D_4bit = D_4bit + 1;
    #20 $finish;
end
```



FLIP FLOP

- The storage elements (memory) used in clocked sequential circuits are called *flipflops*.
- The most economical and efficient flip-flop constructed is the edge-triggered *D* flipflop, because it requires the smallest number of gates.



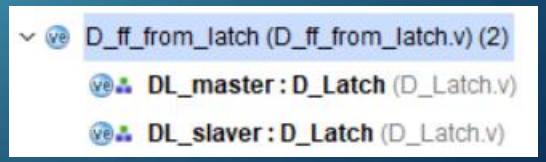


Figure for master-slave negative-edge triggered D flip-flop

FLIP FLOP CONTINUED

```
module DFF_from_Latch(
    input CLK,
    input D,
    output Q
    );
    wire Y;
    D_Latch_2 DL_master(.EN(CLK),.D(D),.Q(Y));
    D_Latch_2 DL_slave(.EN(~CLK),.D(Y),.Q(Q));
endmodule
```

```
DFF_from_Latch tb0(
    .CLK(CLK),
    .D(D),
    .Q(Q)
);
always #100 CLK = ~CLK;
initial begin
    #10 D = 1'b1;
    #20 D = 1'b0;
    #30 D = 1'b1;
    #40 D = 1'b0;
    #50 D = 1'b1;
    #60 D = 1'b0;
    #70 D = 1'b1;
    #80 D = 1'b0;
    #90 D = 1'b1;
end
```

FLIP FLOP CONTINUED

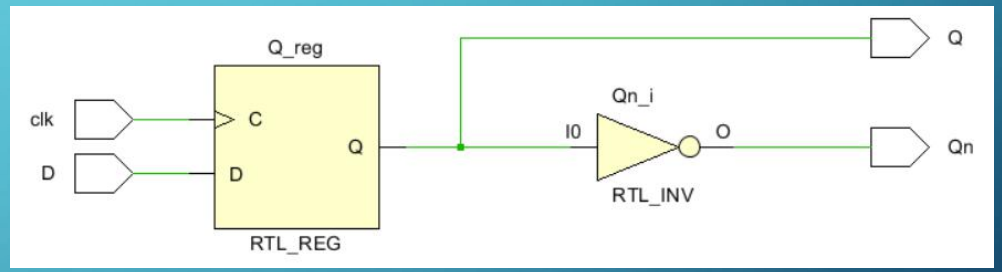
- classification
  - According to different logic functions: RS trigger, D trigger, JK Trigger and T trigger
  - According to different trigger modes: level trigger, edge trigger and pulse trigger
  - According to the different circuit structure: RS trigger and clock controlled trigger
  - According to the principle of storing data: static trigger and dynamic trigger
  - According to the physical components: bipolar flip flops and MOS flip flops

D FLIP FLOP

- $Q^{n+1} = D$

Clk	Q
↑	D
0	no change
1	no change
↓	no change

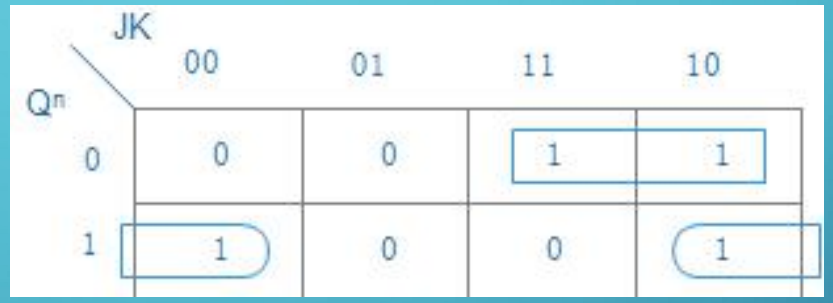
```
module D_flipflop(
input clk,D,
output reg Q,
output Qn
    );
    always @(posedge clk)
    begin
        Q   <= D;
    end
    assign Qn = ~Q;
endmodule
```

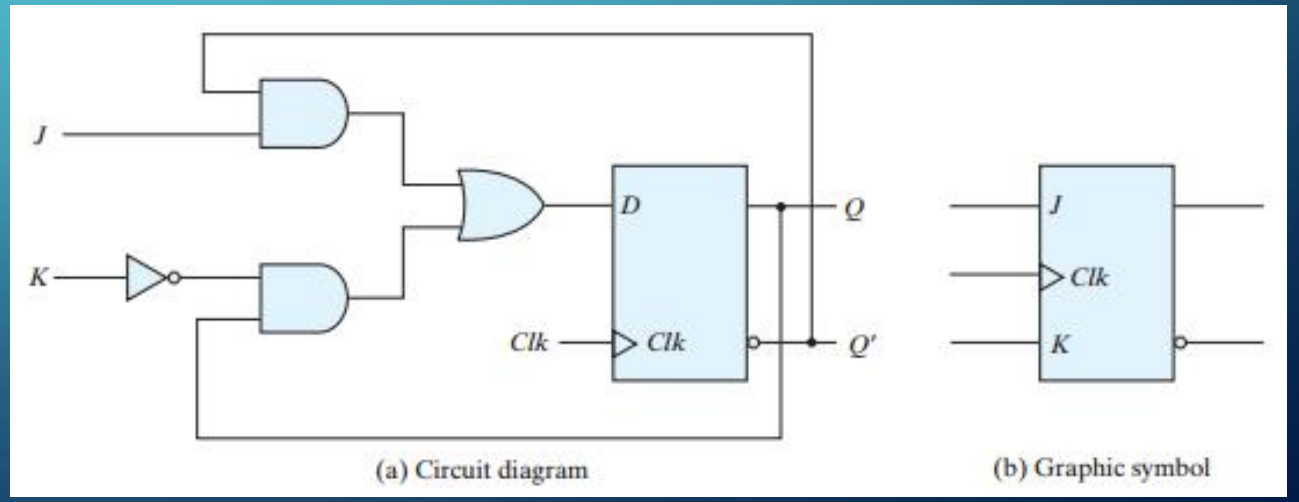


JK FLIP FLOP(1)

- $Q^{n+1} = J\overline{Q^n} + \overline{K}Q^n$

J	K	Q^{n+1}
0	0	no change
0	1	0
1	0	1
1	1	$Q^{n'}$

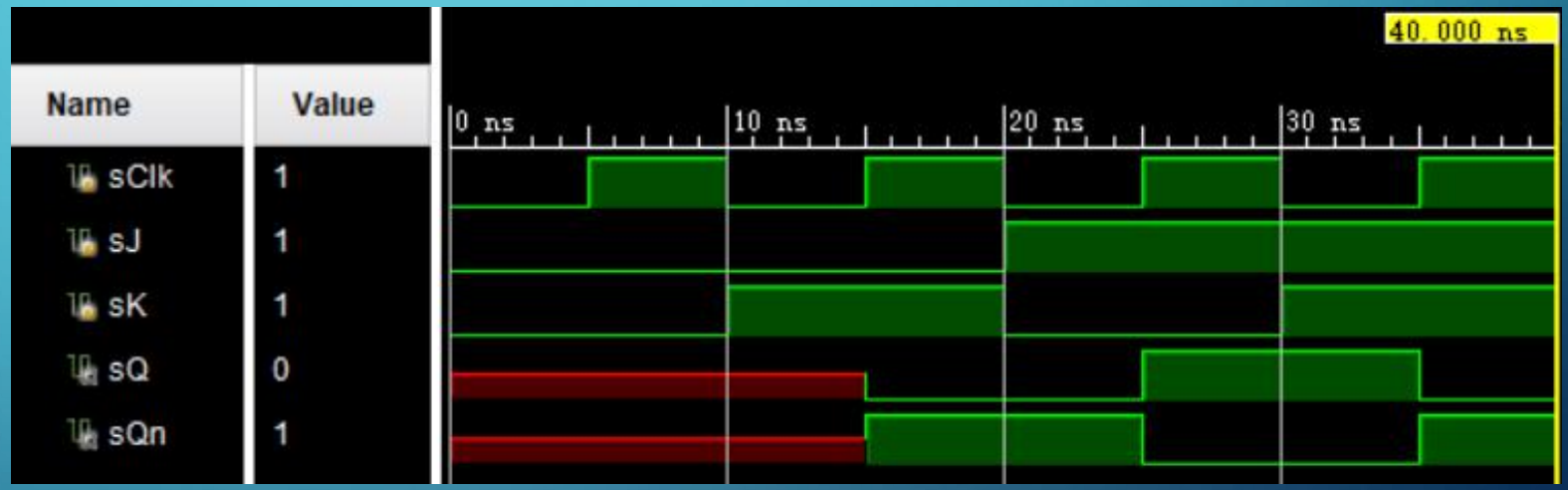




JK FLIP FLOP(2)

$$Q^{n+1} = J\overline{Q^n} + \bar{K}Q^n$$

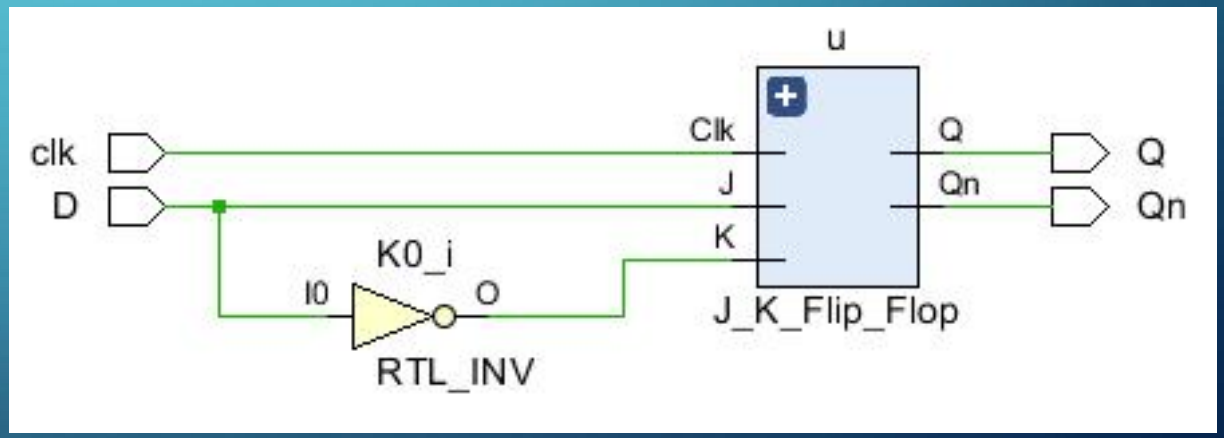
```
module J_K_Flip_Flop(
    input  Clk, J, K,
    output reg Q,
    output Qn
    );
    assign Qn = ~Q;
    always @(posedge Clk)
    if({J, K} == 2'b10 )//set
    begin
        Q <= 1'b1;
    end
    else if ({J,K}==2'b01)//reset
    begin
        Q <= 1'b0;
    end
    else if({J, K} == 2'b11)//reverse
    begin
        Q <= Qn;
    end
endmodule
```



USING JK FLIP FLOP TO IMPLEMENT D FLIP FLOP

- JK Flip-Flop: $Q^{n+1} = J\overline{Q^n} + \overline{K}Q^n$
- D Flip-Flop: $Q^{n+1} = D = D\overline{Q^n} + DQ^n$

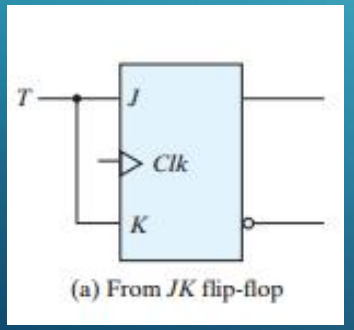
```
module D_Flip_Flop_JK(
    input clk, D,
    output Q,Qn
    );
    J_K_Flip_Flop u(clk, D, ~D, Q, Qn);
endmodule
```



T FLIP FLOP

- The *T* (toggle) flip-flop is a complementing flip-flop, which can be obtained from a *JK* flip-flop when inputs *J* and *K* are tied together.

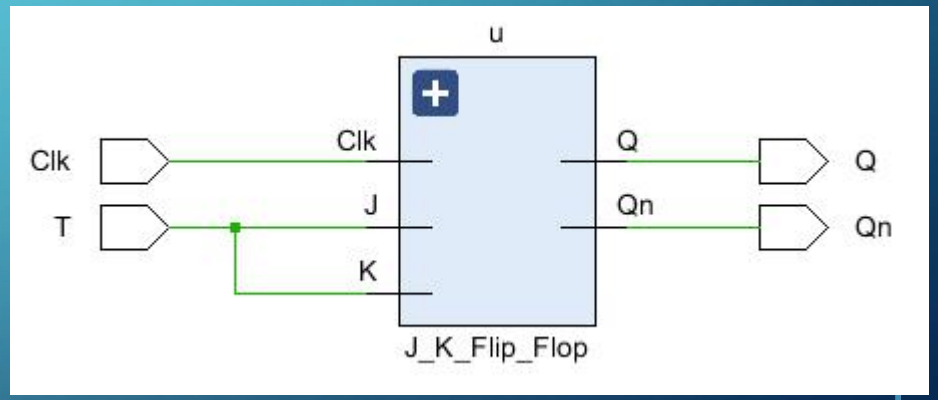
T	Q^{n+1}
0	Q^n
1	$\sim Q^n$



(a) From *JK* flip-flop

```
module T_Flip_Flop(
    input Clk, T,
    output reg Q,
    output Qn
    );
    assign Qn = ~Q;
    always @(posedge Clk)
        case (T)
            1'b1: Q <= Qn;
        endcase
endmodule
```

```
module T_Flip_Flop_JK(
    input T, Clk,
    output Q,
    output Qn
    );
    J_K_Flip_Flop u(Clk, T, T, Q, Qn);
endmodule
```



PRACTICE 1

- Try to construct a T Flip Flop with gates.
  - Do the design and verify the function of your design.
  - Create the constraint file, do the synthetic and implementation, generate the bitstream file and program the device, then test on the minisys develop board.

PRACTICE 2

$$Q^{n+1} = J\overline{Q^n} + \overline{K}Q^n$$

- Can this JK flip-flop work?
- Try to use it to implement a T flip-flop, do the design, create constraint file, generate the bitstream file and program the device.
- Can the T flip-flop work? Explain the reason.

```
module J_K_Flip_Flop(
    input Clk, J, K,
    output reg Q,Qn
    );
    always @(posedge Clk)
        if({J, K} == 2'b10 )//set
        begin
            {Q, Qn} <= 2'b10;
        end
        else if ({J,K}==2'b01)//reset
        begin
            {Q, Qn} = 2'b01;
        end
        else if({J, K} == 2'b11)//reverse
        begin
            Q <= Qn;
            Qn <= Q;
        end
endmodule
```

TIPS:

- Constraints: if you want to use IO pin as clock, you should use the CLOCK_DEDICATED_ROUTE constraint in the .xdc file to demote the Error message to a WARNING.