DIGITAL DESIGN

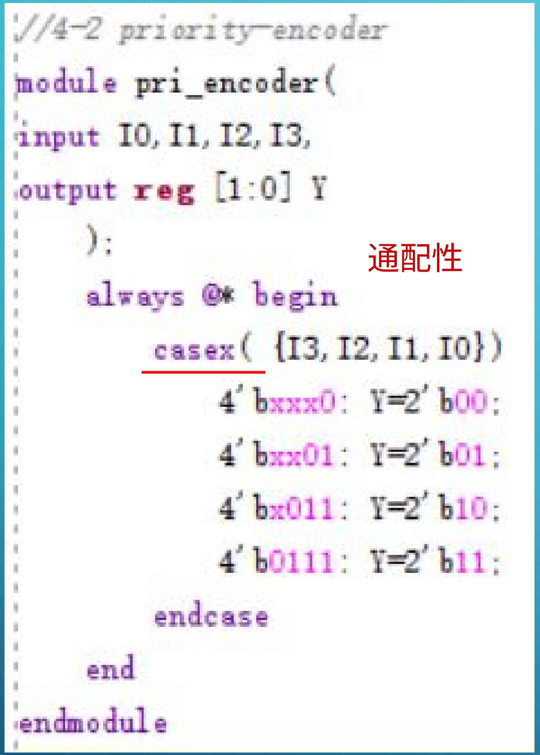
LAB9 SYNCHRONOUS SEQUENTIAL CIRCUIT-LATCH-FLIPFLOP

WANGW6@SUSTECH.EDU.CN

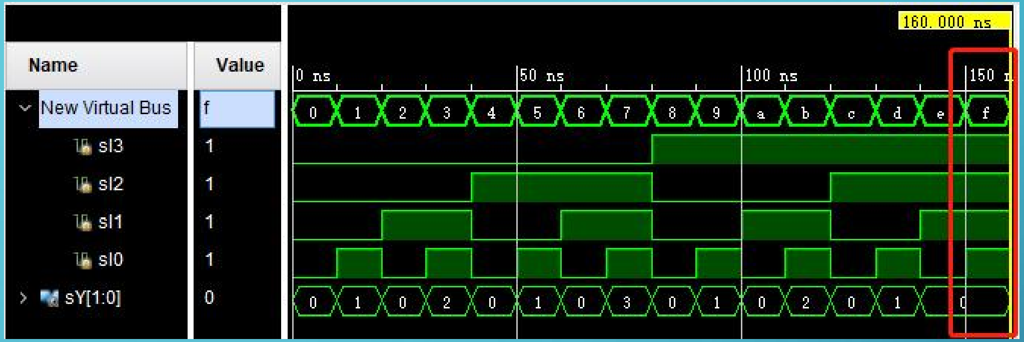
LAB9

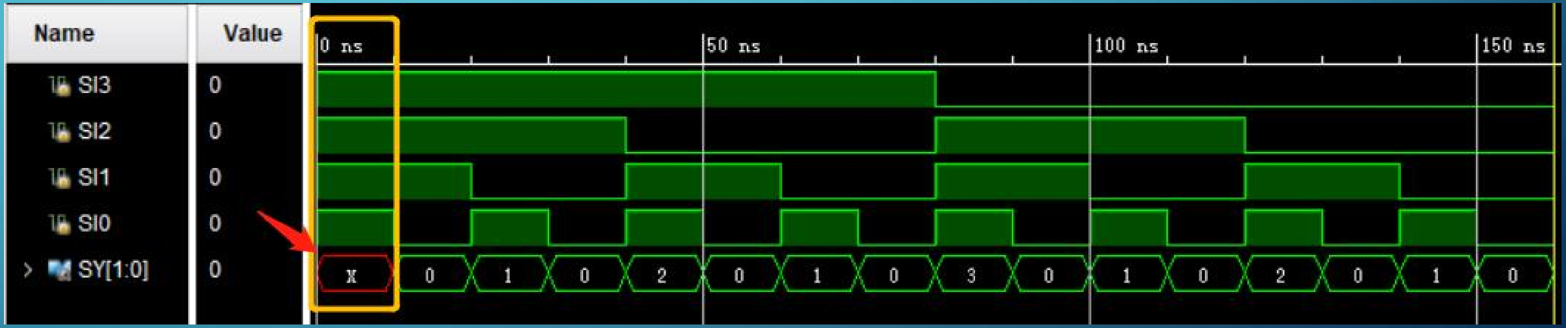
- Synchronous sequential circuit
  - Latch
  - Flip Flops (key point of lab9)
- Verilog
  - always@(posedge clk) , always@(negedge clk) vs always@*
  - blocking assignment vs non-blocking assignment
- Practice

RECALL: 4-2 PRIORITY-EN-CODER



```
//4-2 priority-encoder
module pri_encoder(
input I0, I1, I2, 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
```



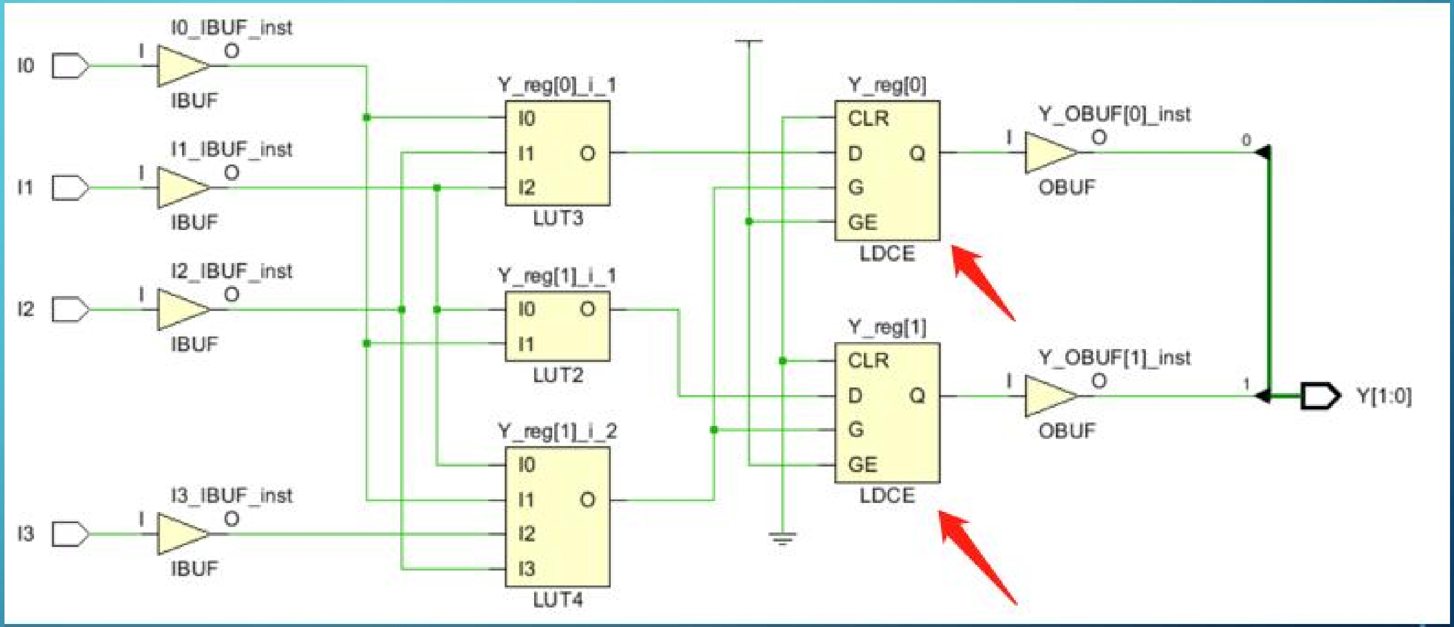


If there is no 'default' branch and the branch(e.g. 4'b1111 in this demo) is not list in the branches of the 'case', the circuit state will remain unchanged, and then a **latch** will be generated to save the state.

LATCH

组合逻辑如果不在设计代码把所有场景罗列出来　否则会自动添加一些时序电路

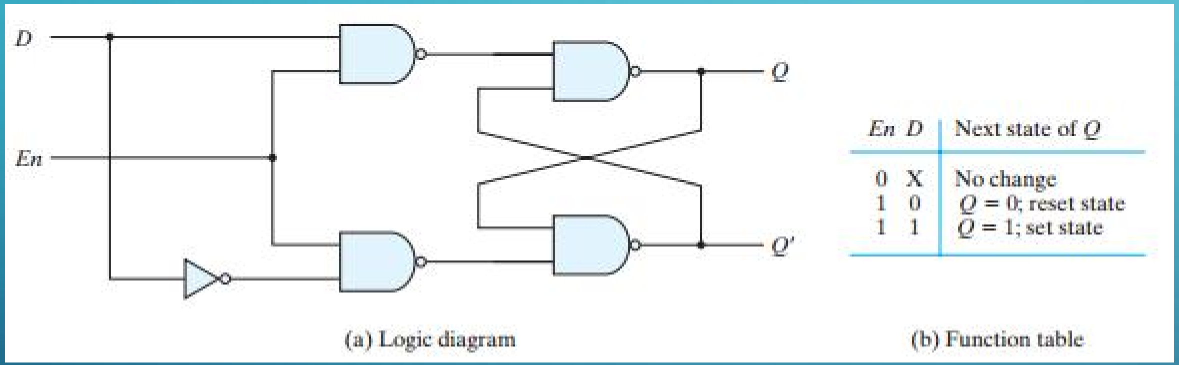
- The schematic on the right hand is the schematic of the synthesized design generated by vivado.
- LDCE: transparent Data **Latch** with Asynchronous Clear and Gate Enable



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

D LATCH

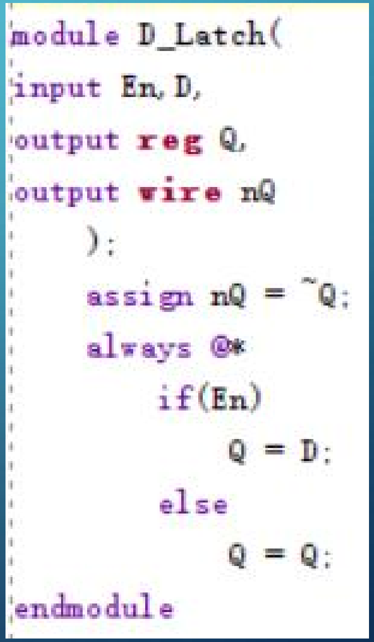
- Latch: the simplest binary memory elements, static device composed of gates. When the input changes, the output changes immediately.



(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

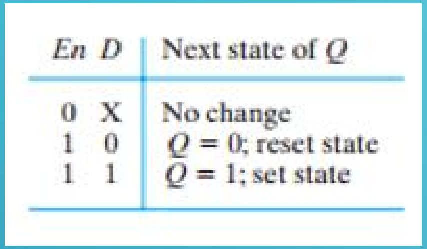
(b) Function table



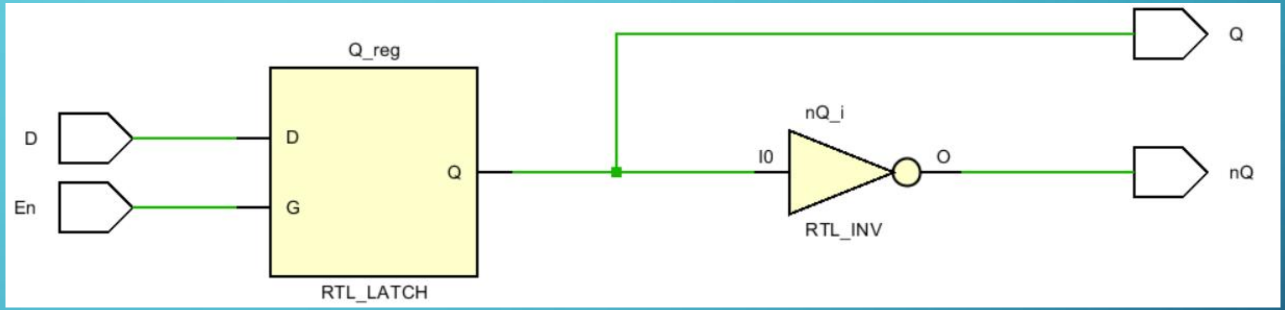
```
module D_Latch(
input En, D,
output reg Q,
output wire nQ
    );
    assign nQ = ~Q;
    always @*
        if(En)
            Q = D;
        else
            Q = Q;
endmodule
```

D LATCH

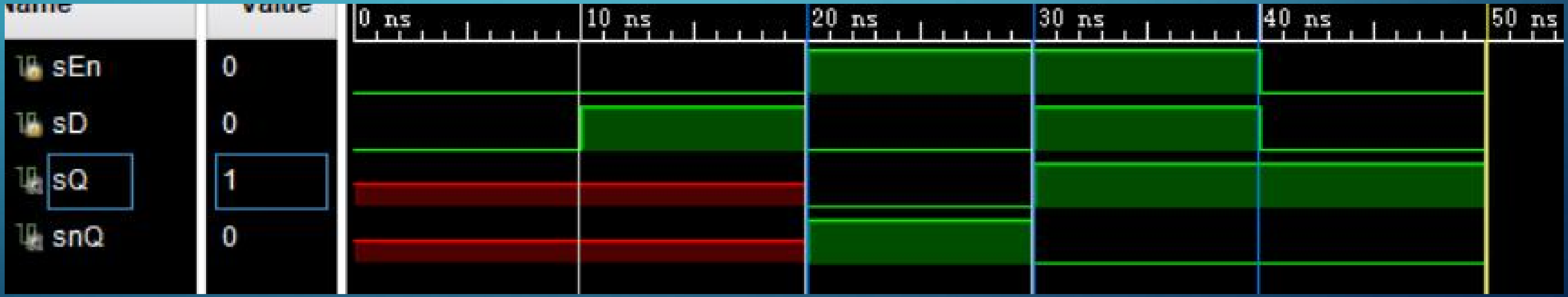
- Schematic of RTL analysis:



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



- Simulation result of D Latch



Name	Value
sEn	0
sD	0
sQ	1
snQ	0

FLIP FLOP (DESIGN)

两个锁存器构成一个触发器

- 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.

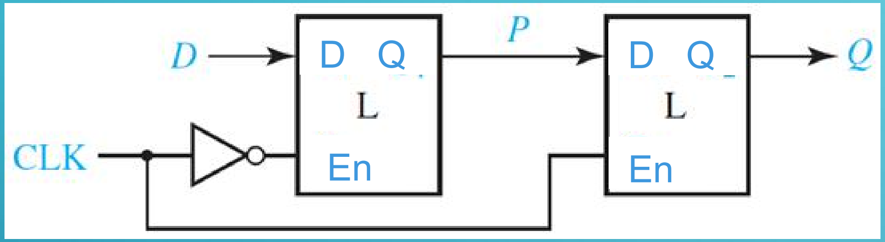
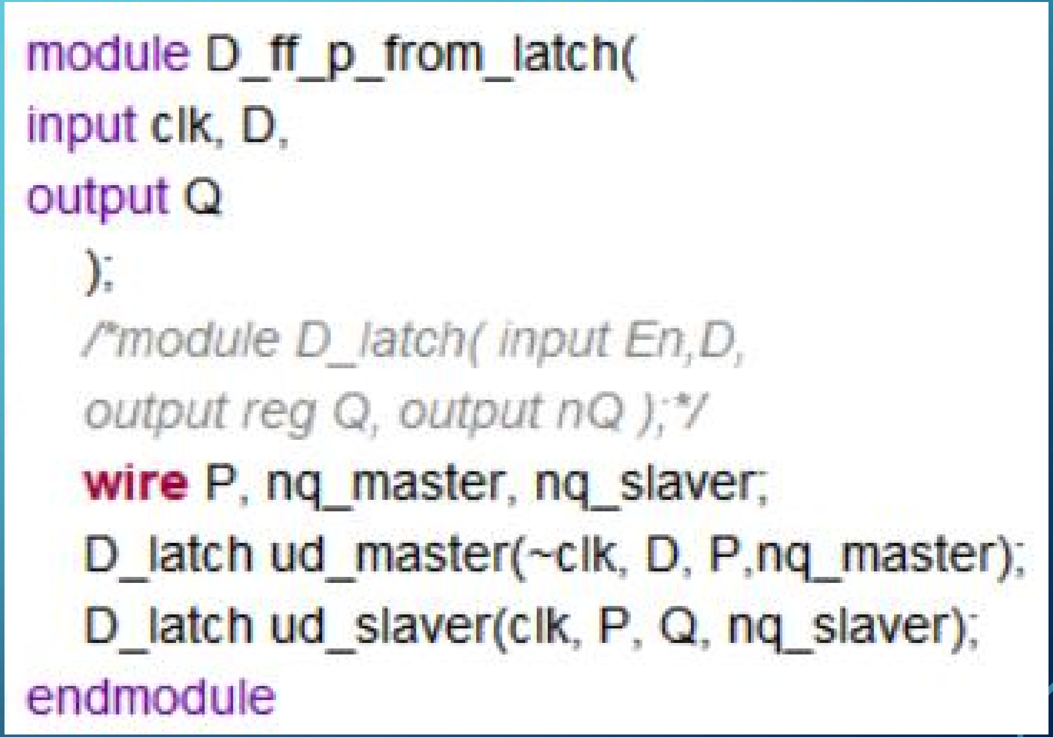


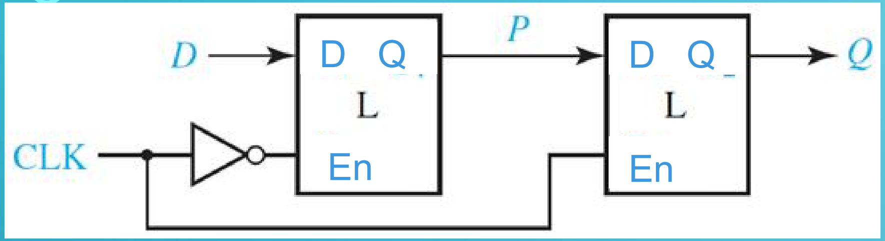
Figure for master-slave positive-edge triggered **D flip-flop**



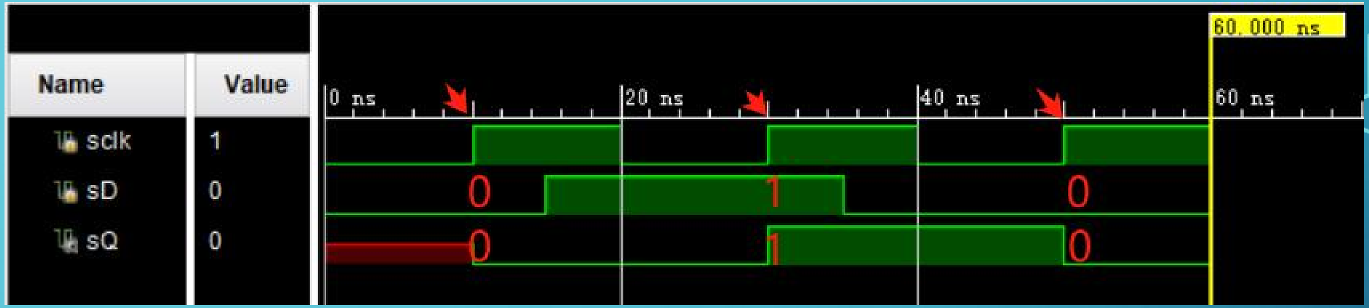
```
module D_ff_p_from_latch(
input clk, D,
output Q
    );
    /*module D_latch( input En,D,
    output reg Q, output nQ );*/
    wire P, nq_master, nq_slaver;
    D_latch ud_master(~clk, D, P,nq_master);
    D_latch ud_slaver(clk, P, Q, nq_slaver);
endmodule
```

FLIP FLOP (TESTBENCH, WAVEFORM)

master-slave positive-edge triggered D flip-flop



```
module D_ff_p_from_latch(
input clk, D,
output Q
    );
    /*module D_latch( input En,D,
    output reg Q, output nQ );*/
    wire P, nq_master, nq_slaver;
    D_latch ud_master(~clk, D, P,nq_master);
    D_latch ud_slaver(clk, P, Q, nq_slaver);
endmodule
```



Name	Value
sclk	1
sD	0
sQ	0

```
module D_ff_p_tb(    );
/*module D_ff_p_from_latch(input clk, D, output Q ); */
reg sclk, sD;
wire sQ;
D_ff_p_from_latch dut1(sclk, sD, sQ);
initial begin
sclk=1'b0;
forever #10  sclk = ~sclk;
end
initial fork
sD = 1'b0;
#15 sD = 1'b1;
#35 sD = 1'b0;
#60 $finish;
join
endmodule
```

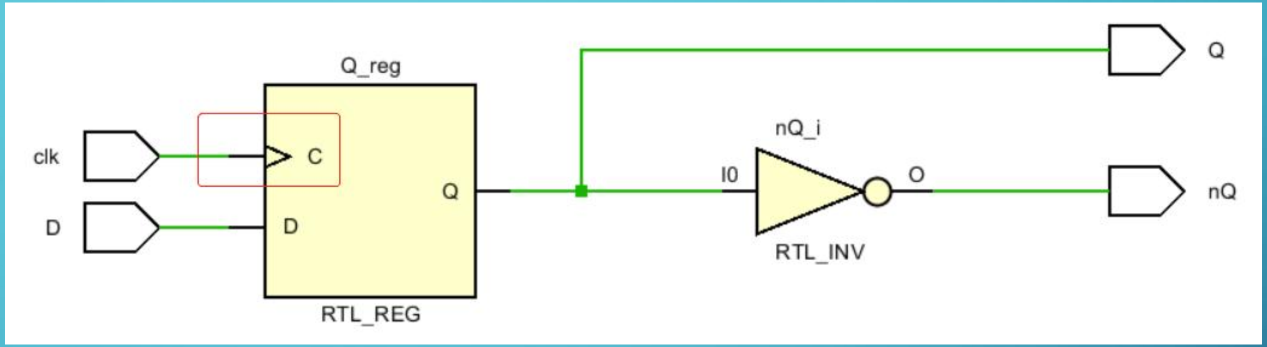
D FLIP FLOP

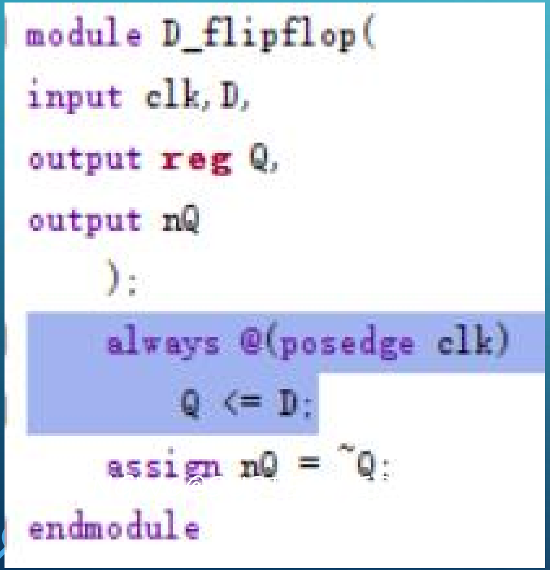
Clk	Q
↑	D
0	no change
1	no change
↓	no change

Here is positive-edge triggered D flip-flop

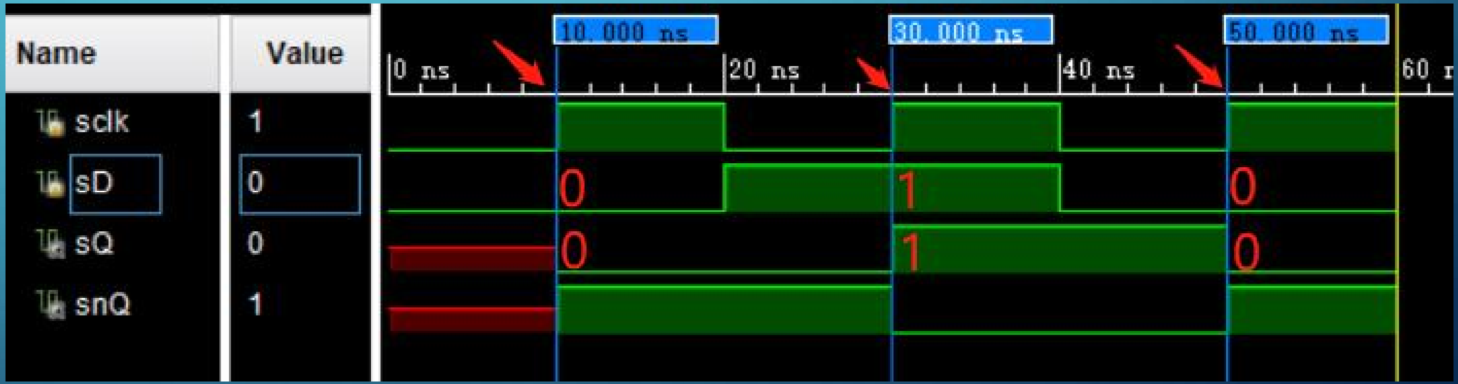
组合逻辑里用阻塞赋值=
时序电路里用非阻塞赋值<=
例：

begin	begin
a=b;	a<=b;
b=a;	b<=a;
end	end
结果：	
a==b	a、b交换b==b



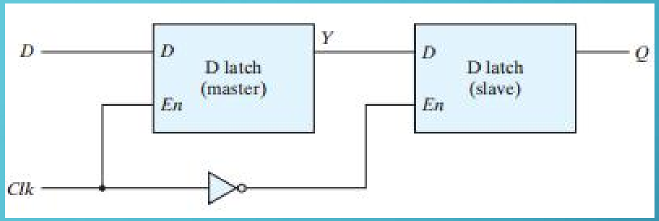


```
module D_flipflop(
input clk,D,
output reg Q,
output nQ
    );
    always @(posedge clk)
        Q <= D;
    assign nQ = ~Q;
endmodule
```

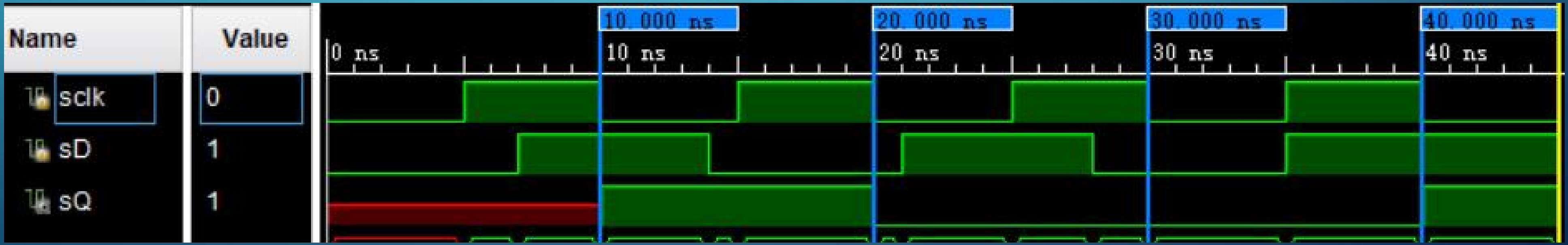


PRACTICE(1)

- Implement a negative-edge triggered D flipflop
  - a) by using two D latch
  - b) using behavior manner



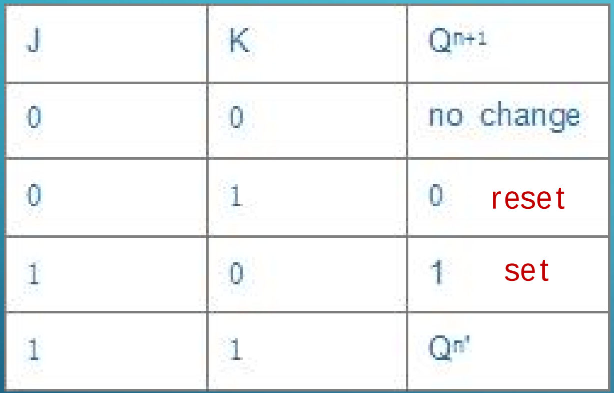
- Build a testbench to test the negative-edge triggered D flipflop, do the simulation.
  - the expected waveform of the negative-edge triggered D flipflop is shown in the following figure.



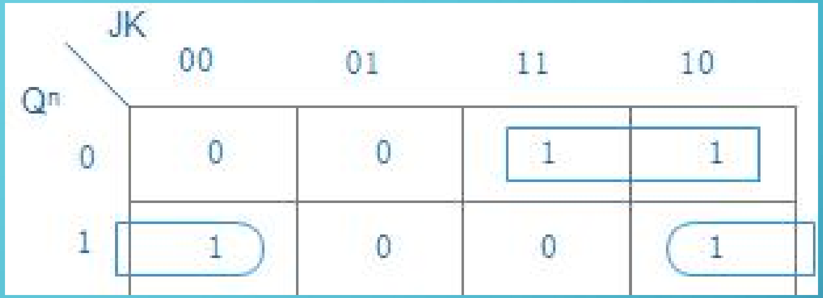
JK FLIP FLOP(1)

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

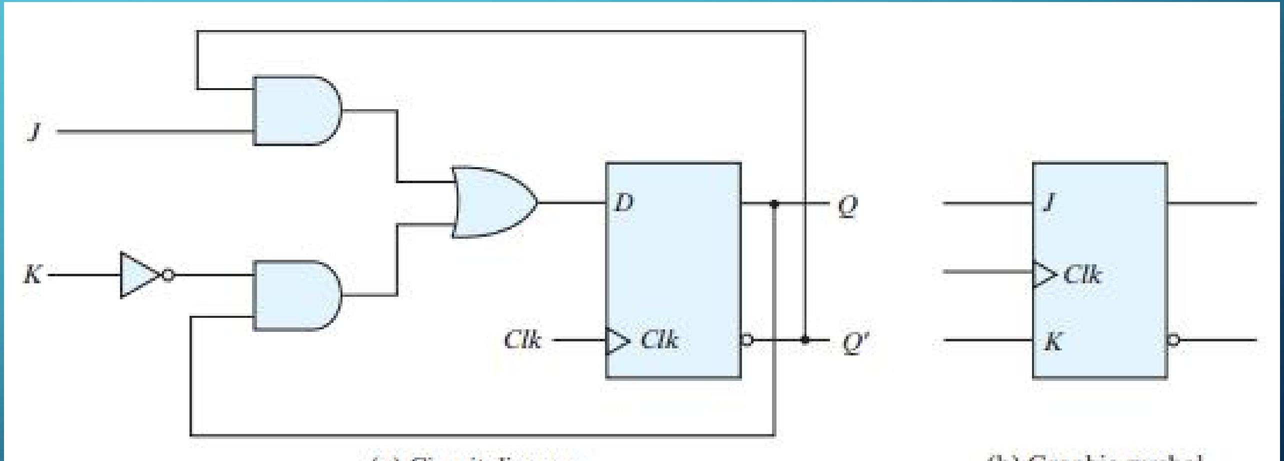
次态 现态



J	K	Q^{n+1}
0	0	no change
0	1	0 reset
1	0	1 set
1	1	$Q^{n\prime}$



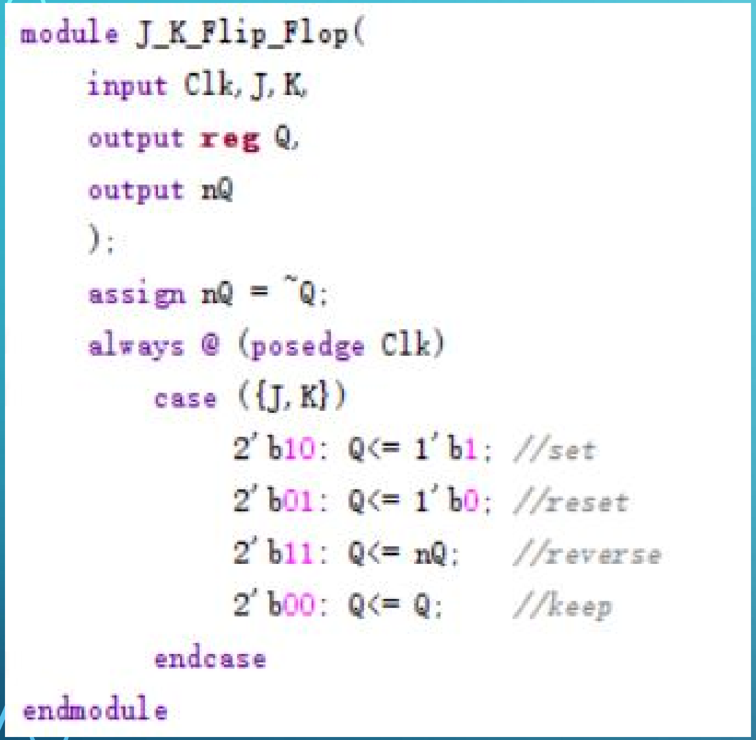
Q^n \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1



(a) Circuit diagram (b) Graphic symbol

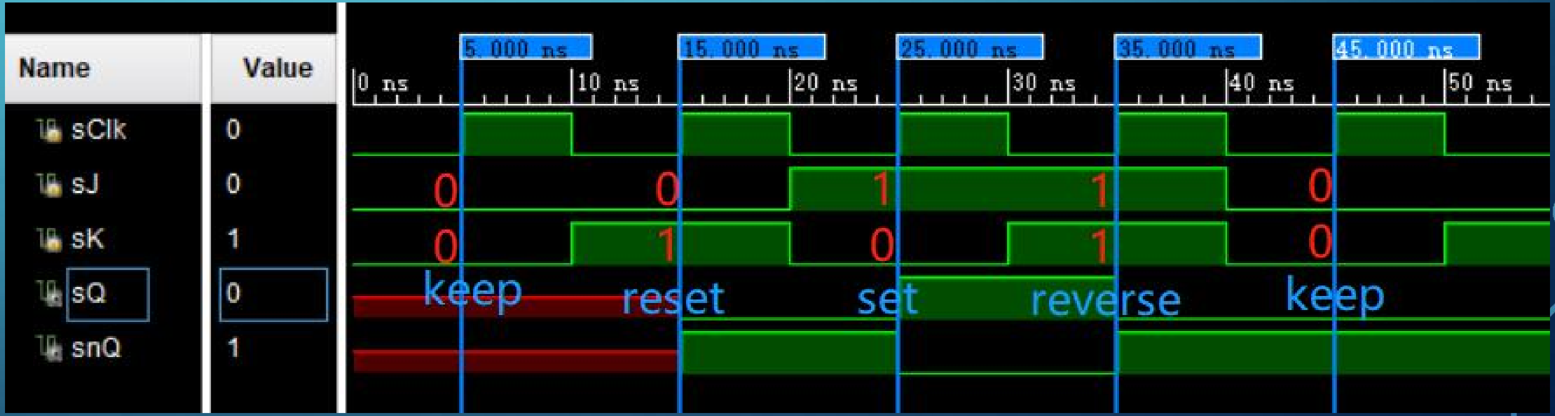
JK FLIP FLOP(2)

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



```
module J_K_Flip_Flop(
    input Clk, J, K,
    output reg Q,
    output nQ
    );
    assign nQ = ~Q;
    always @ (posedge Clk)
        case ({J, K})
            2'b10: Q<= 1'b1; //set
            2'b01: Q<= 1'b0; //reset
            2'b11: Q<= nQ;   //reverse
            2'b00: Q<= Q;    //keep
        endcase
endmodule
```

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



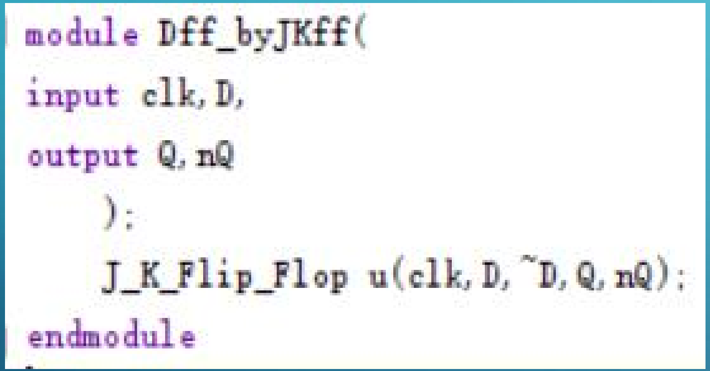
Name	Value
sClk	0
sJ	0
sK	1
sQ	0
snQ	1

USING JK FLIP FLOP TO IMPLEMENT D FLIP FLOP

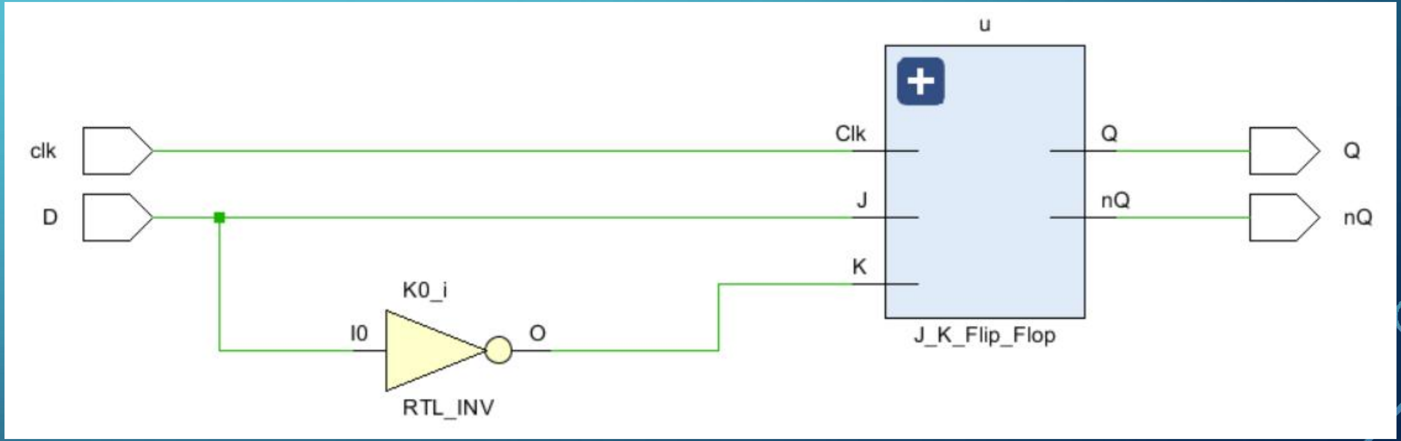
- JK Flip-Flop: $Q^{n+1} = J\overline{Q^n} + \overline{K}Q^n$
- D Flip-Flop: $Q^{n+1} = D = D\overline{Q^n} + DQ^n$

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

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

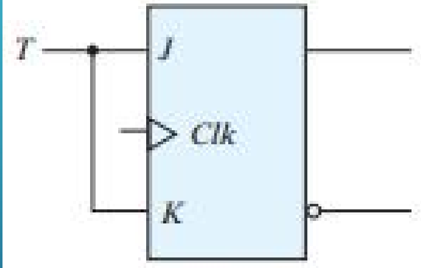


```
module Dff_byJKff(
input clk,D,
output Q,nQ
    );
    J_K_Flip_Flop u(clk,D,~D,Q,nQ);
endmodule
```



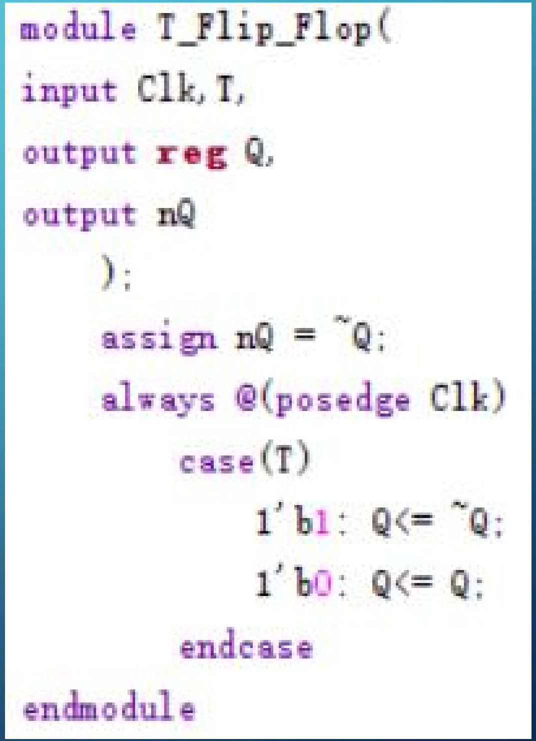
T FLIP FLOP

- The *T* (toggle) flip-flop is a complementing flip-flop.



(a) From *JK* flip-flop

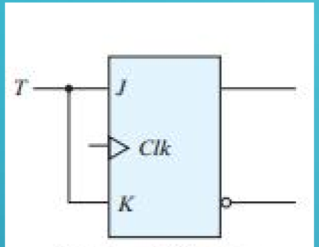
T	Q^{n+1}
0	Q^n
1	$\sim Q^n$



```
module T_Flip_Flop(
input Clk,T,
output reg Q,
output nQ
    );
    assign nQ = ~Q;
    always @(posedge Clk)
        case(T)
            1'b1: Q<= ~Q;
            1'b0: Q<= Q;
        endcase
endmodule
```

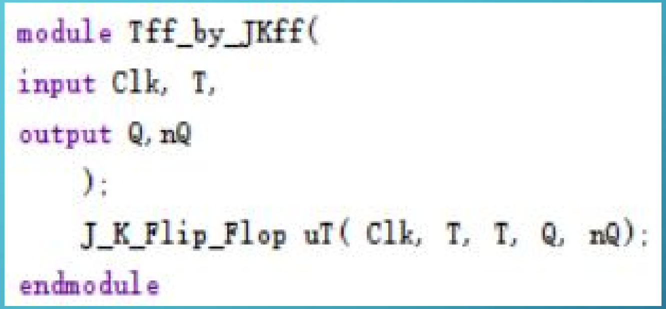
USING JK FLIP FLOP TO IMPLEMENT T FLIP FLOP

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



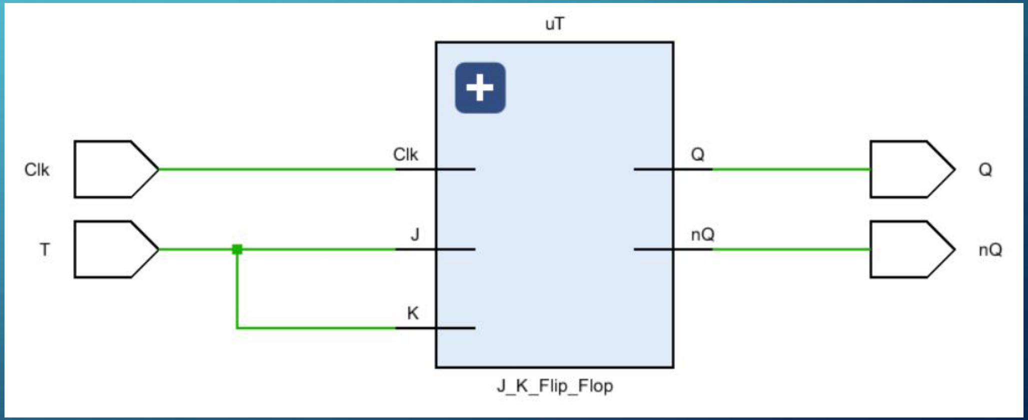
(a) From *JK* flip-flop

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



```
module Tff_by_JKff(
input Clk, T,
output Q, nQ
    );
    J_K_Flip_Flop uT( Clk, T, T, Q, nQ);
endmodule
```

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



WIRE VS REG(1)

- There are two data types in Verilog: wire and register.
  - wire is a kind of net, which is equivalent to physical connection.
  - wire is used to connects two points, and thus does not have any driving strength
  - wire data types can be used for connecting the output port to the actual driver
  - a wire can be assigned a value by a continuous assign statement, which is used for designing **combinational** logic
  - default data type is wire: this means that if you declare a port or variable without specifying reg or wire, it will be a 1-bit wide wire.
  - reg is a kind of register, which is equivalent to memory cell.
  - reg can store value and drive strength. Reg can be used for modeling both combinational and sequential logic.
  - reg data type can be driven from initial and always block.
  - The LHS of a behavioral block(initial , always) should be declared *as* **reg**
  - **input port** could drive by both wire/register, but it could ONLY be declared as wire
  - **output port** can be declared as wire or register, but it can ONLY drive wire
  - bidirectional port can only be declared as wire

http://www.asic-world.com/tidbits/wire_reg.html

NON-BLOCKING ASSIGNMENT VS BLOCKING ASSIGNMENT

- The ‘**=**’ token represents a blocking **procedural assignment**
- A combinational logic **always block should use** Blocking assignments(“=“).
- The ‘**<=**’ token represents a non-blocking **procedural assignment**
- A sequential logic **always block should use** Non-blocking assignments(“**<=**“).

NOTE: **DO NOT** mixing blocking assignment and non-blocking assignment in the same always block **!!!**

PRACTICE(2)

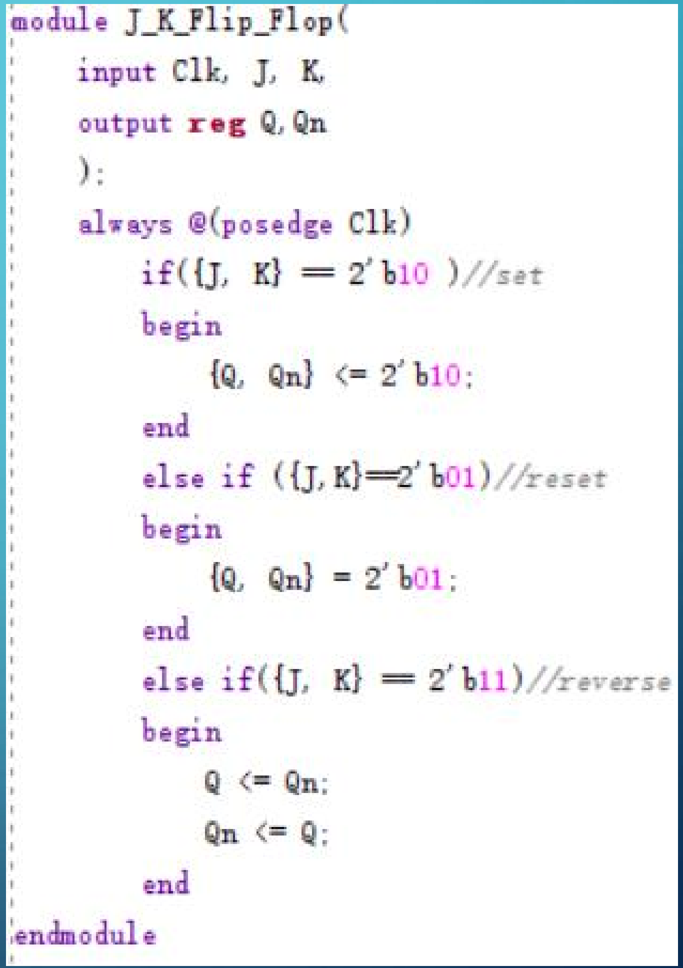
- Try to construct a T flipflop with a reset input.
  - There would be a reset input port apart from other inputs(clk, T), while it enable the output of T flipflop is 1'b0, while it disable the circuit works as T flipflop
  - Do the design and verify the function of your design.
  - Create the constraint file, do the synthetic and implementation, generate the bitstream file and program the device, then test on the minisys develop board.

PRACTICES(3)

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

- Can this JK flip-flop work?
- Try to use it 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.

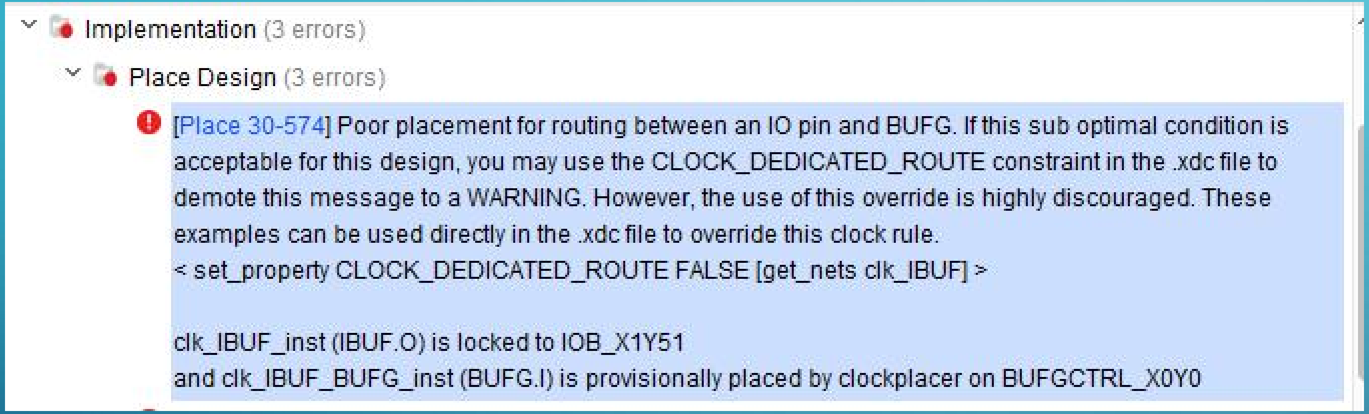
缺少else分支



```
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
```

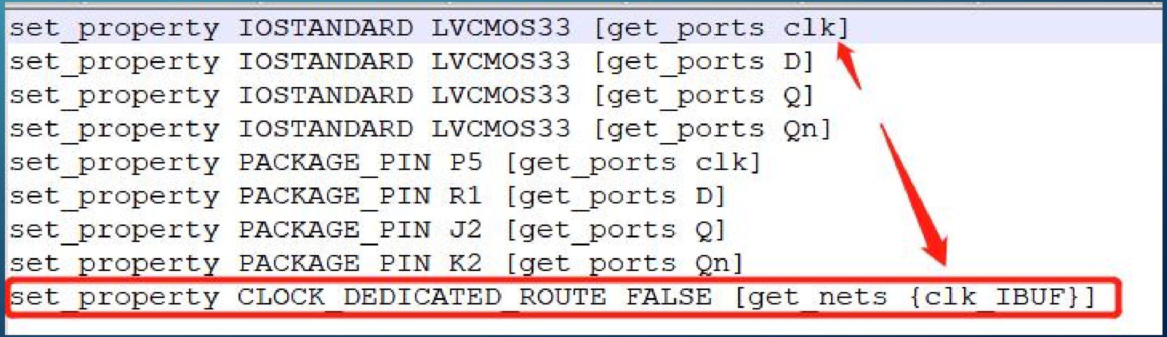
TIPS:

- Constraints: if you want to use IO pin as clock, you should use the CLOCK_DEDICATED_ROUTE FALSE constraint in the .xdc file to demote the Error message to a (Critical) WARNING. NOTES： This (Critical) WARNING HERE could be ignored.



```
Implementation (3 errors)
  Place Design (3 errors)
    [Place 30-574] Poor placement for routing between an IO pin and BUFG. If this sub optimal condition is
    acceptable for this design, you may use the CLOCK_DEDICATED_ROUTE constraint in the .xdc file to
    demote this message to a WARNING. However, the use of this override is highly discouraged. These
    examples can be used directly in the .xdc file to override this clock rule.
    < set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF] >

    clk_IBUF_inst (IBUF.O) is locked to IOB_X1Y51
    and clk_IBUF_BUFG_inst (BUFG.I) is provisionally placed by clockplacer on BUFGCTRL_X0Y0
```



```
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports D]
set_property IOSTANDARD LVCMOS33 [get_ports Q]
set_property IOSTANDARD LVCMOS33 [get_ports Qn]
set_property PACKAGE_PIN P5 [get_ports clk]
set_property PACKAGE_PIN R1 [get_ports D]
set_property PACKAGE_PIN J2 [get_ports Q]
set_property PACKAGE_PIN K2 [get_ports Qn]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}]
```