DIGITAL DESIGN

LAB10 SYNCHRONOUS SEQUENTIAL CIRCUIT-LATCH-FLIPFLOP

2022 FALL TERM @ CSE . SUSTECH

DIGITAL DESIGN

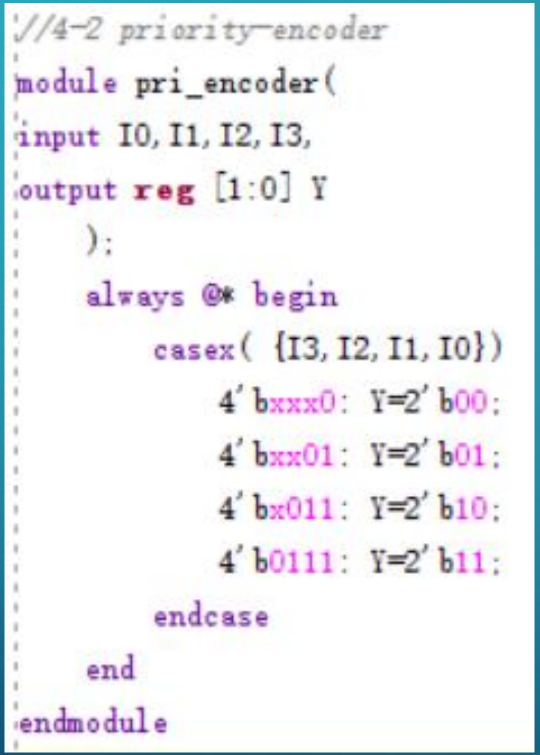
LAB10 SYNCHRONOUS SEQUENTIAL CIRCUIT-LATCH-FLIPFLOP

2022 FALL TERM @ CSE . SUSTECH

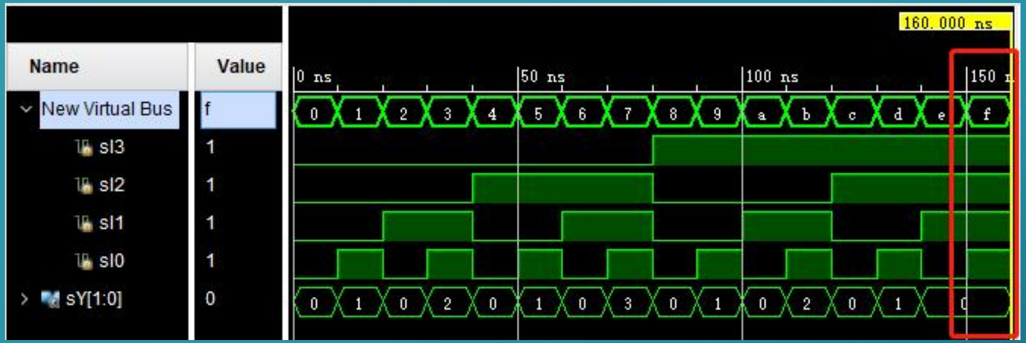
LAB10

- Synchronous sequential circuit
  - Latch
  - Flip Flops (key point of lab10)
- Verilog
  - always@(posedge clk) , always@(negedge clk) vs always@*
  - blocking assignment vs non-blocking assignment
- Practise

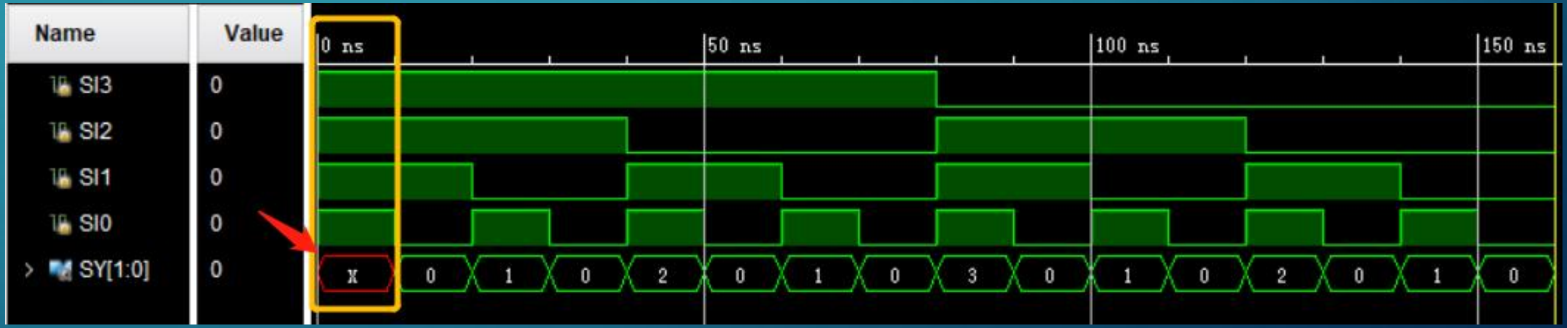
4-2 PRIORITY-ENCODER IN LAB8(1)



```
//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
```



Name	Value
New Virtual Bus	f
sI3	1
sI2	1
sI1	1
sI0	1
sY[1:0]	0

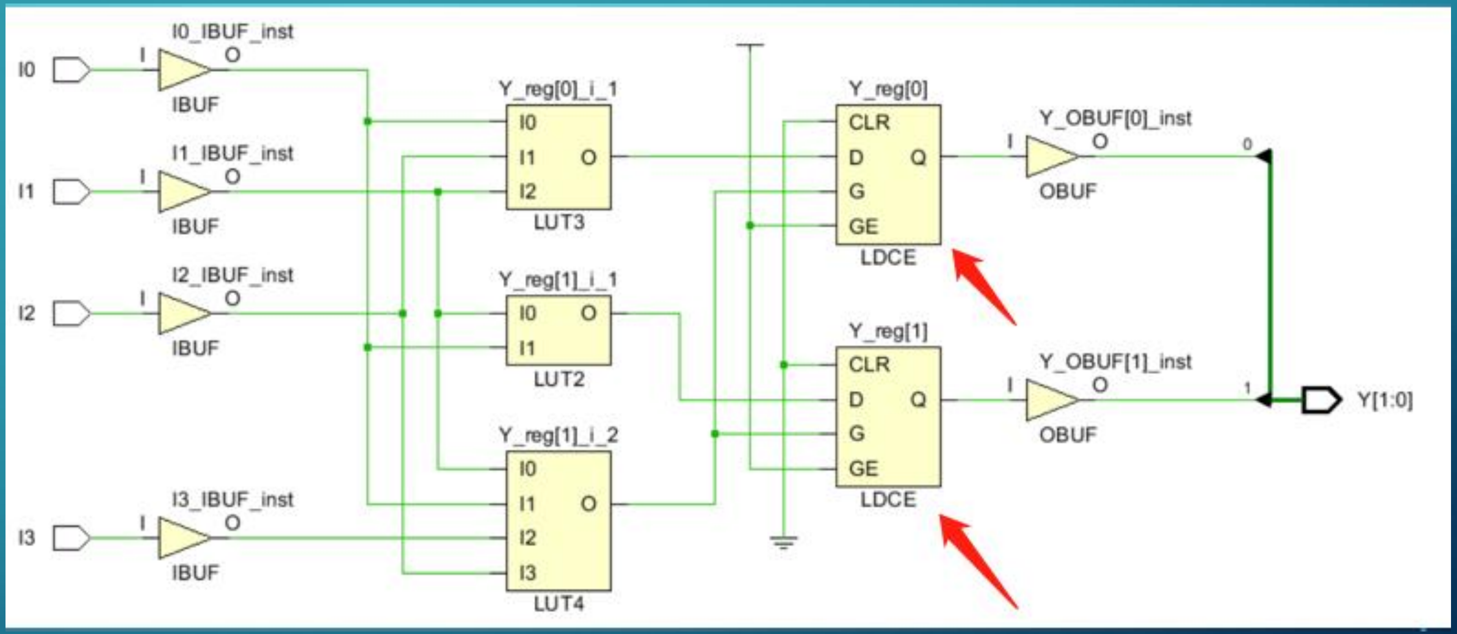


Name	Value
SI3	0
SI2	0
SI1	0
SI0	0
SY[1:0]	0

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

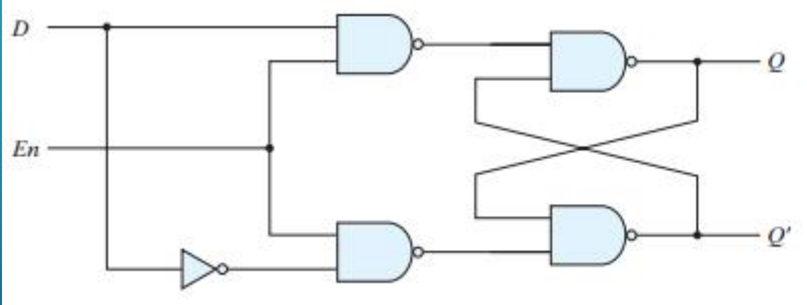
- The schematic on the right hand is the schematic of the synthesized design generated by vivado.
- LDCE: transparent Data **Latch** with Asynchronous Clear and Gate Enable



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

D LATCH

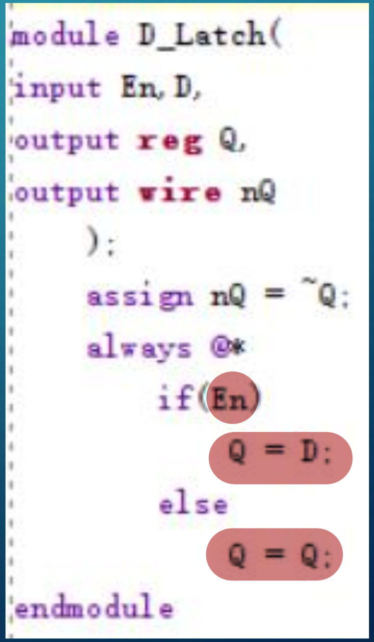
- Latch: the simplest binary memory elements, static device composed of gates. When the input changes, the output changes immediately.



(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

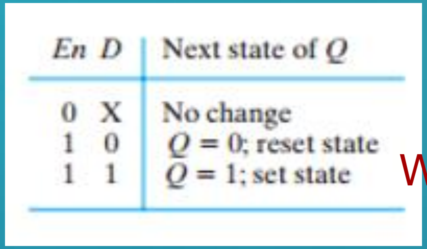


```
module D_Latch(
input En, D,
output reg Q,
output wire nQ
    );
    assign nQ = ~Q;
    always @*
        if(En)
            Q = D;
        else
            Q = Q;
endmodule
```

D LATCH

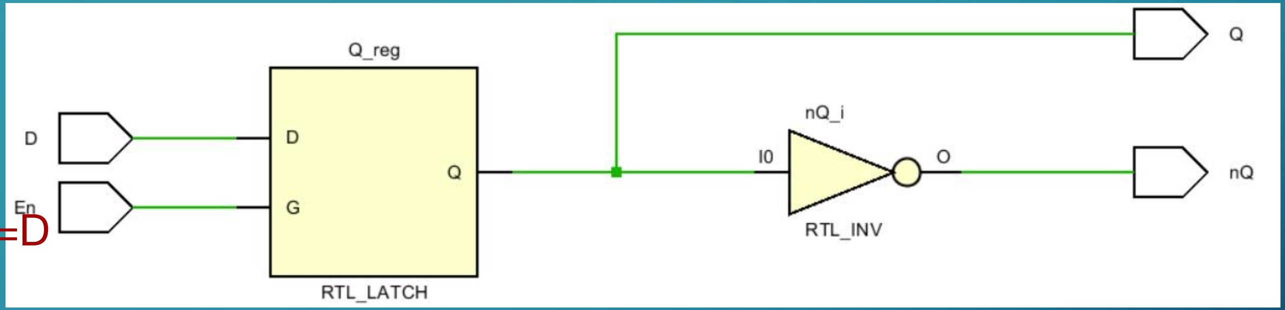
In digital circuit design, register-transfer level (RTL) is a design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals.

- Schematic of RTL analysis:

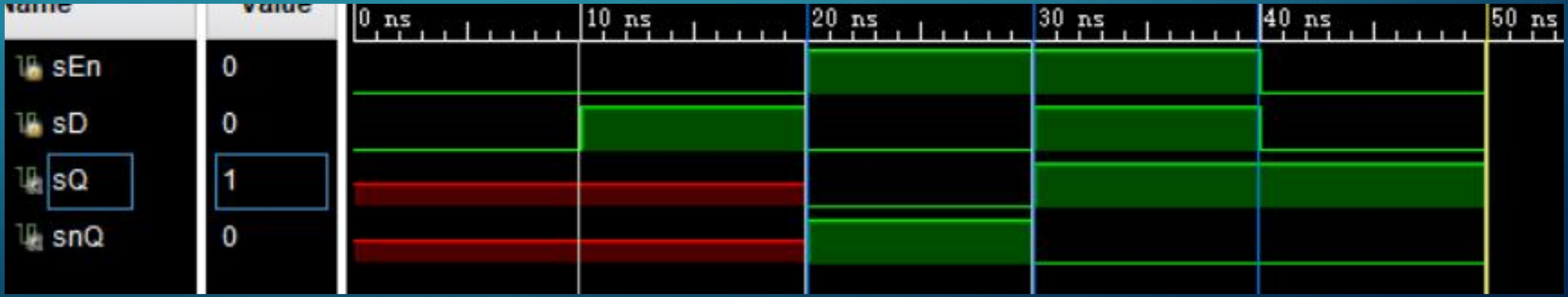


En	*D*	Next state of *Q*
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

When en=1, Q=D



- Simulation result of D Latch



Name	Value
sEn	0
sD	0
sQ	1
snQ	0

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.

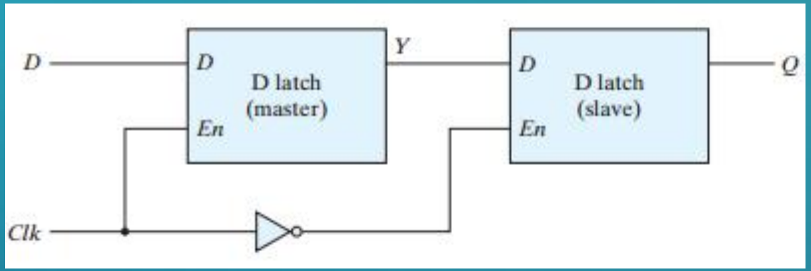
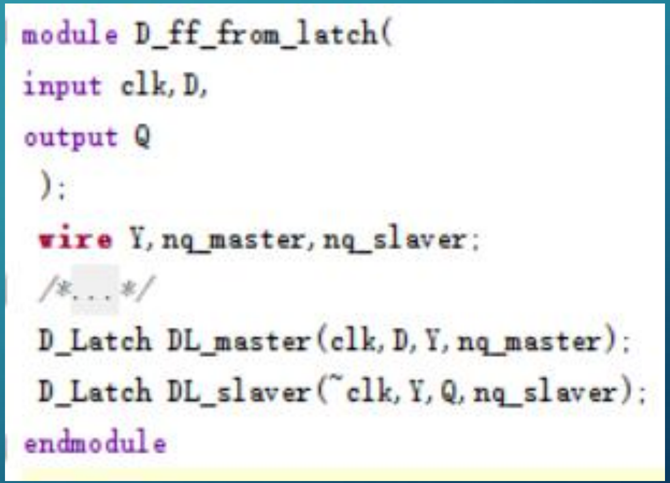
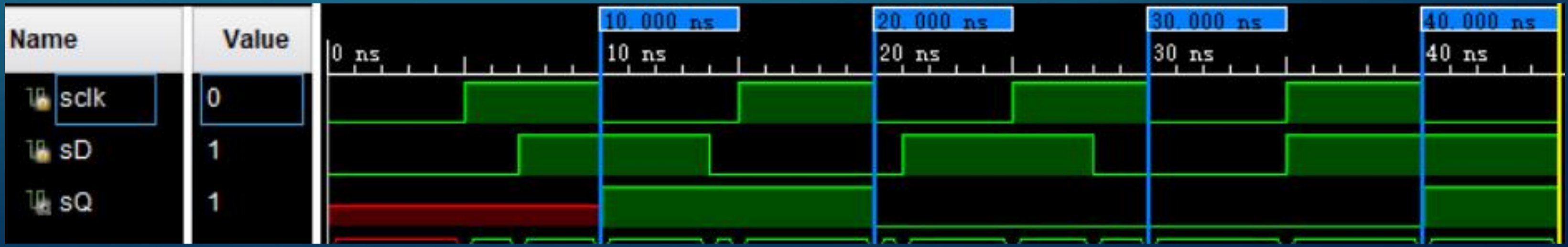


Figure for master-slave negative-edge triggered D flip-flop

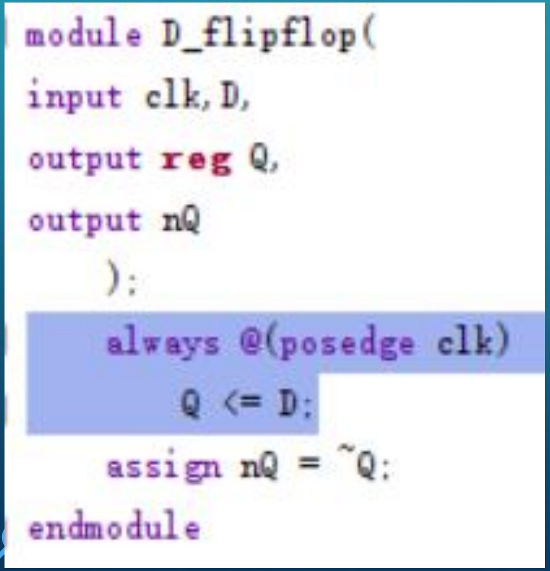


```
module D_ff_from_latch(
input clk, D,
output Q
 );
 wire Y, nq_master, nq_slaver;
 /*...*/
 D_Latch DL_master(clk, D, Y, nq_master);
 D_Latch DL_slaver(~clk, Y, Q, nq_slaver);
endmodule
```

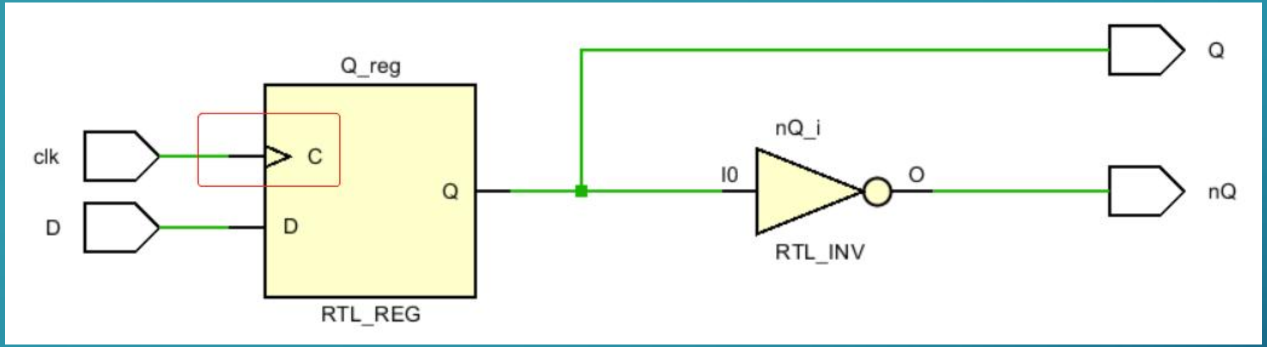


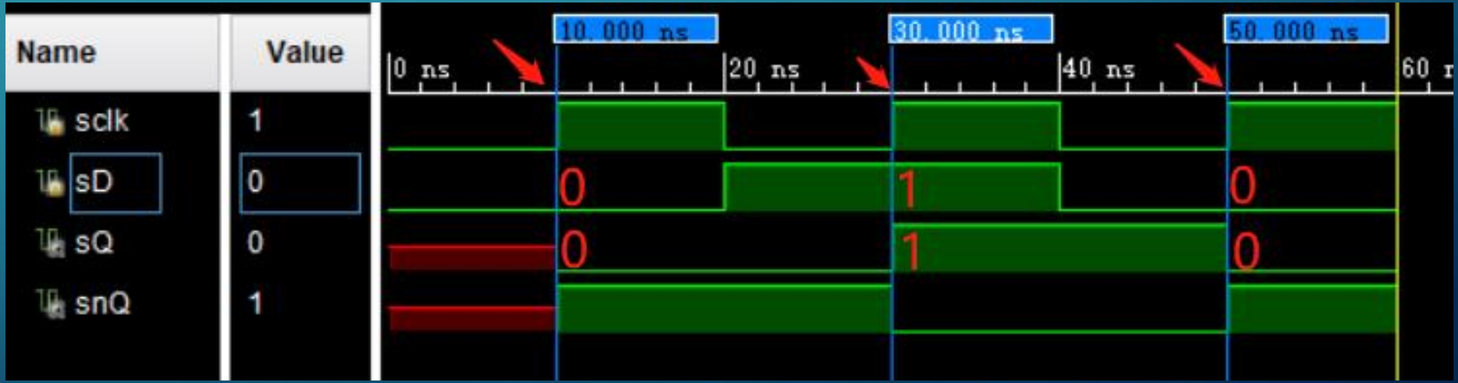
D FLIP FLOP

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



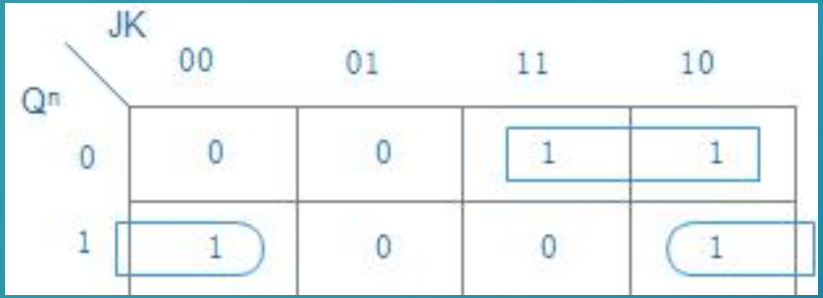
```
module D_flipflop(
input clk, D,
output reg Q,
output nQ
    );
    always @(posedge clk)
        Q <= D;
    assign nQ = ~Q;
endmodule
```



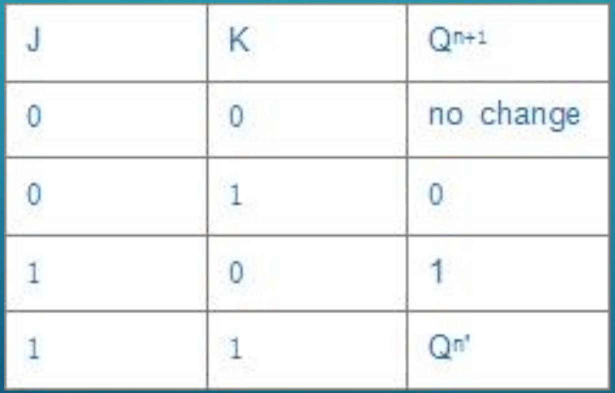


JK FLIP FLOP(1)

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

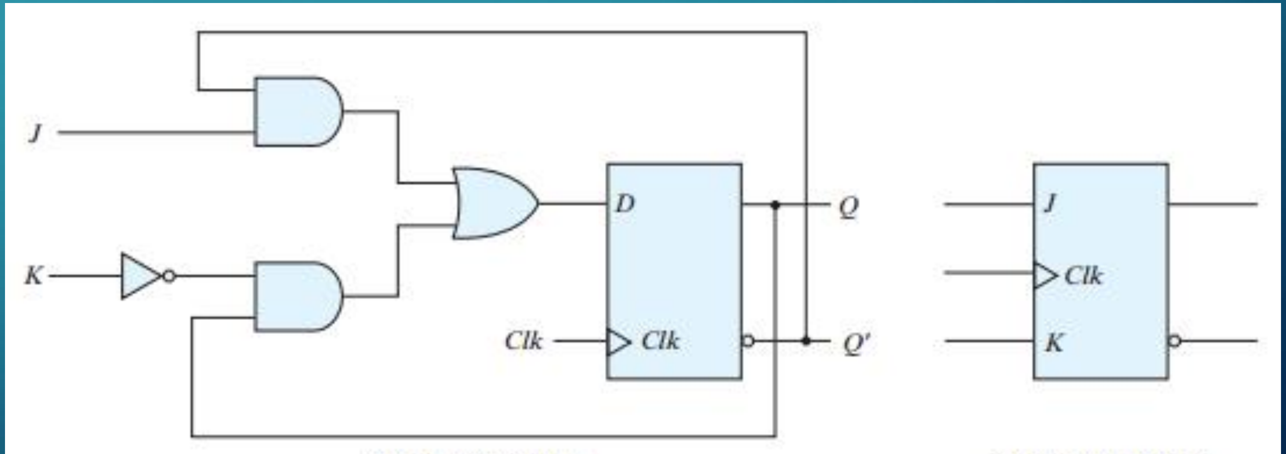


Q^n \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1



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

reverse

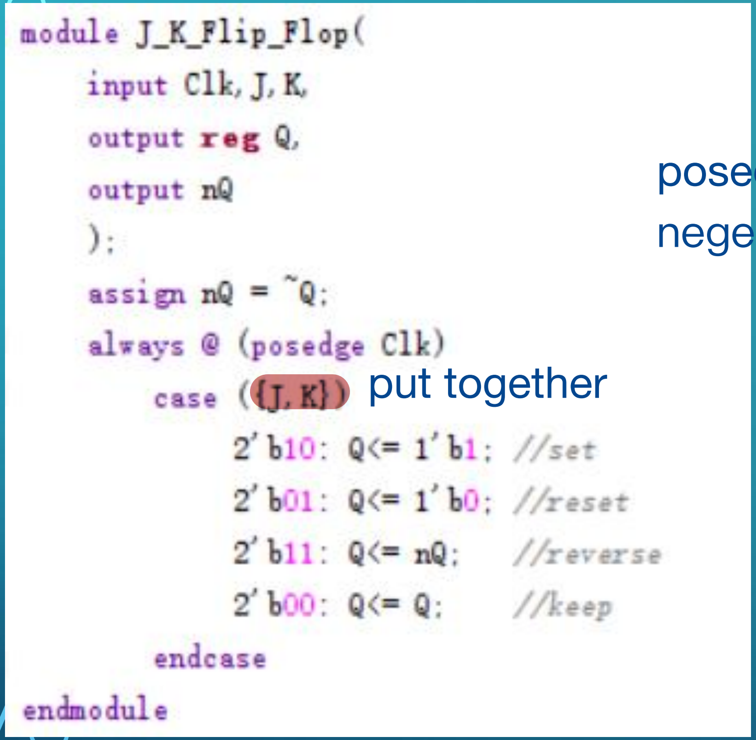


(a) Circuit diagram (b) Graphic symbol

= is blocking statement. In an always block, the line of code will be executed only after it's previous line has executed. Hence, they happens one after the other, just like combinatoral logics in loop.
<= is non-blocking in nature. This means that in an always block, every line will be executed in parallel. Hence leading to implementation of sequential elements.

JK FLIP FLOP(2)

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

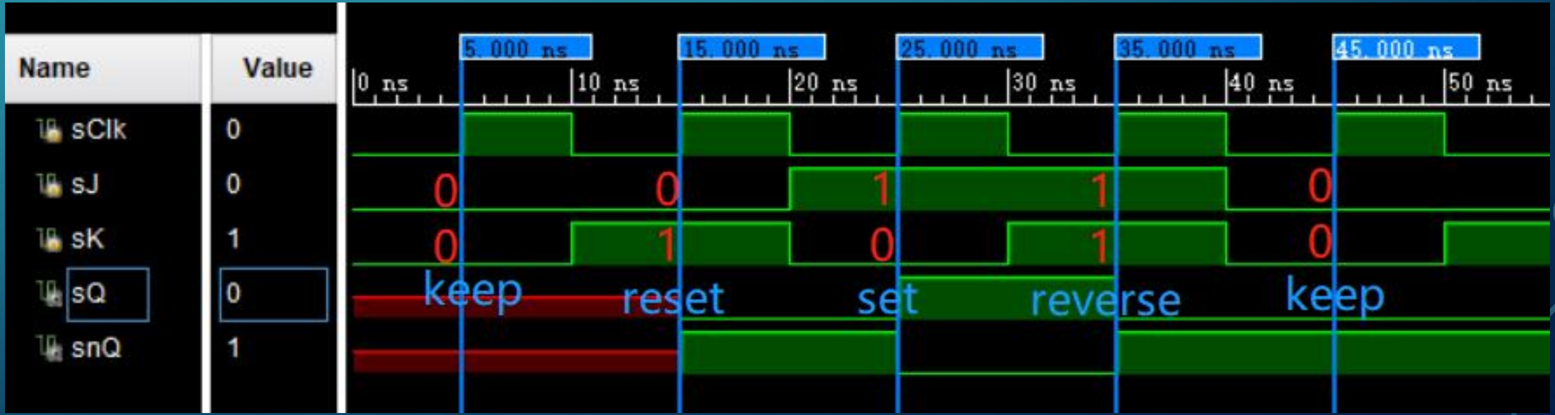


```
module J_K_Flip_Flop(
    input Clk, J, K,
    output reg Q,
    output nQ
    );
    assign nQ = ~Q;
    always @ (posedge Clk)
        case ({J, K})
            2'b10: Q<= 1'b1;  //set
            2'b01: Q<= 1'b0;  //reset
            2'b11: Q<= nQ;    //reverse
            2'b00: Q<= Q;     //keep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

put together

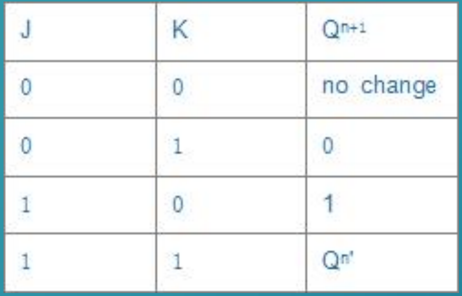
posedge：正向跳变（0–>1）
negedge：负向跳变

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

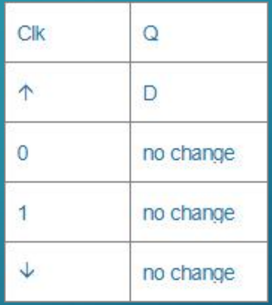


USING JK FLIP FLOP TO IMPLEMENT D FLIP FLOP

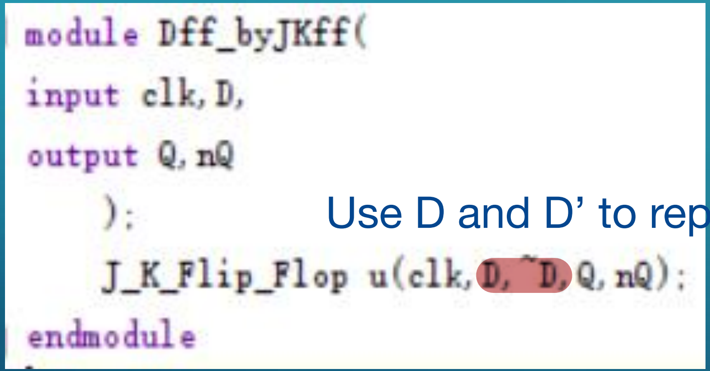
- JK Flip-Flop: $Q^{n+1} = J\overline{Q^n} + \overline{K}Q^n$
- D Flip-Flop: $Q^{n+1} = D = D\overline{Q^n} + DQ^n$



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

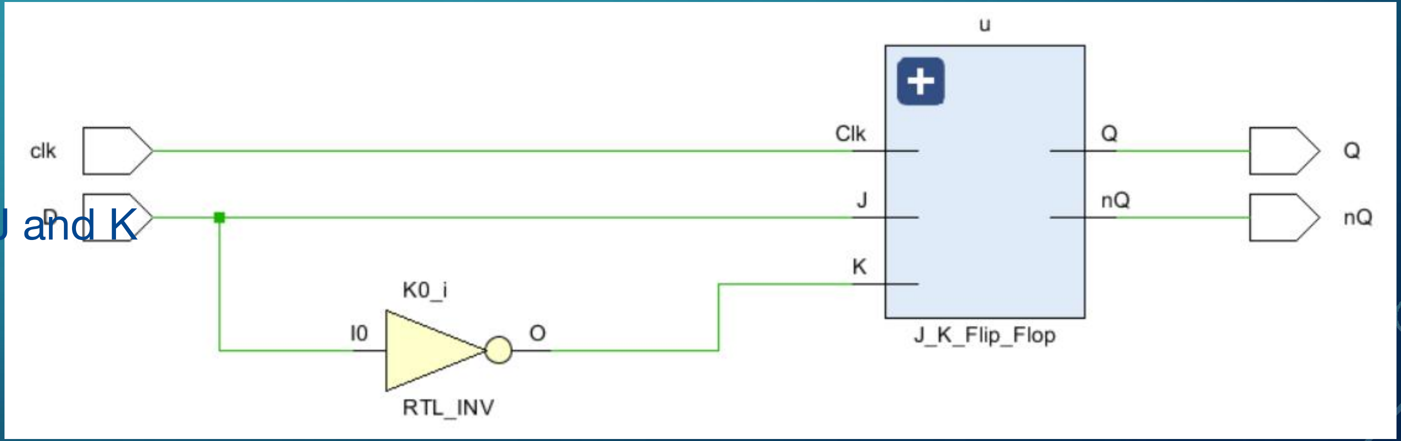


Clk	Q
↑	D
0	no change
1	no change
↓	no change



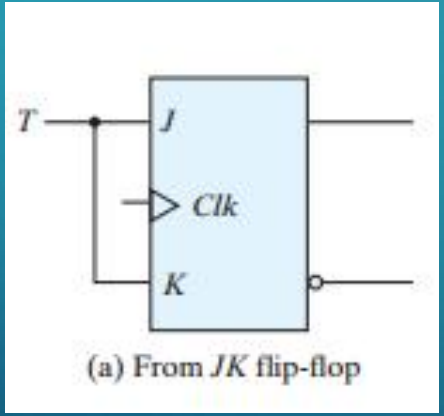
```
module Dff_byJKff(
input clk,D,
output Q,nQ
    );
    J_K_Flip_Flop u(clk,D,~D,Q,nQ);
endmodule
```

Use D and D’ to replace J and K

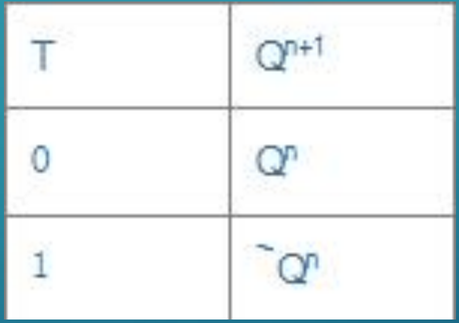


T FLIP FLOP

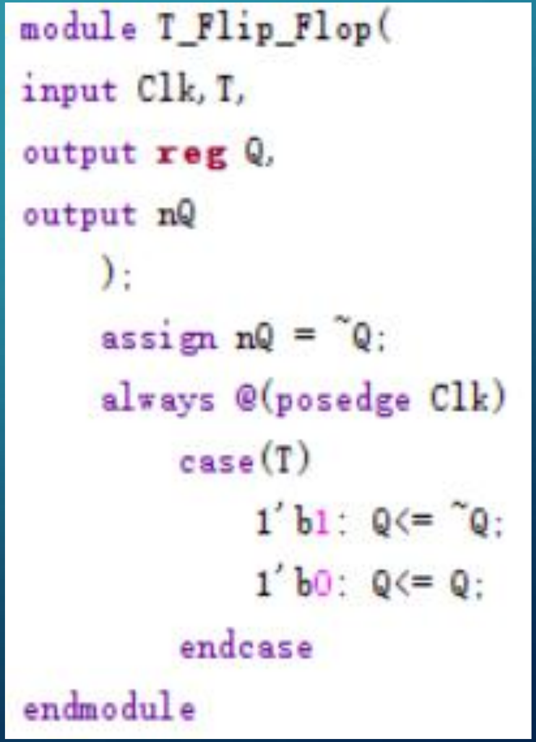
- The *T* (toggle) flip-flop is a complementing flip-flop.



(a) From *JK* flip-flop



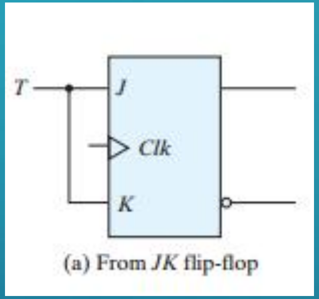
T	Q^{n+1}
0	Q^n
1	~Q^n



```
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.

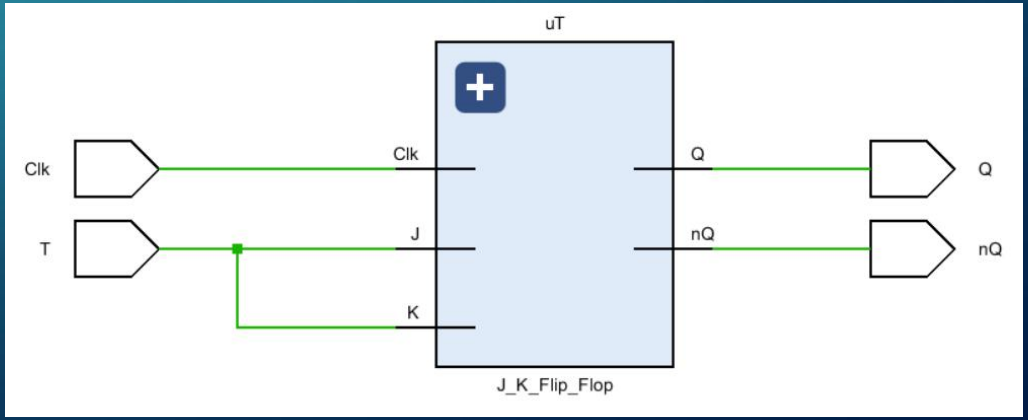


(a) From *JK* flip-flop

T	Q^{n+1}
0	Q^n
1	~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.
  - left hand side
  - 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 **!!!**

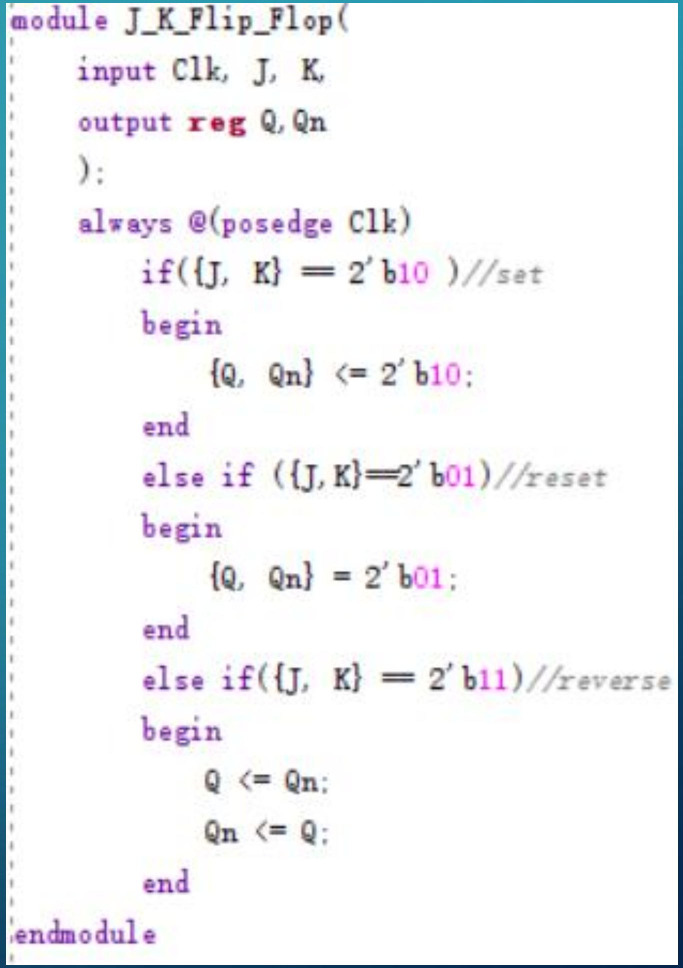
PRACTICE(1)

- 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(2)

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

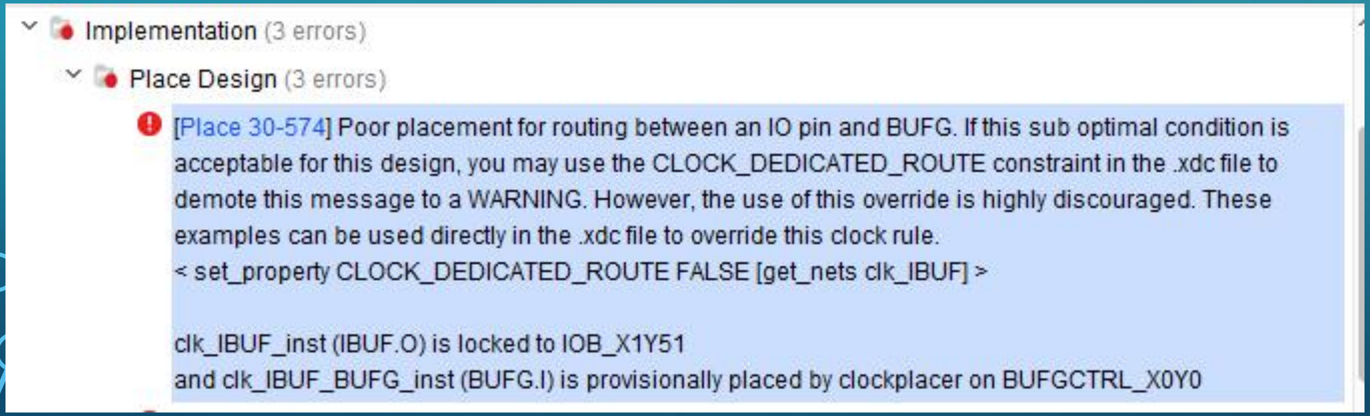
- Can this JK flip-flop work?
- Try to use it implement a T flip-flop, do the design, create constraint file, generate the bitstream file and program the device.
- Can the T flip-flop work? Explain the reason.



```
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
```

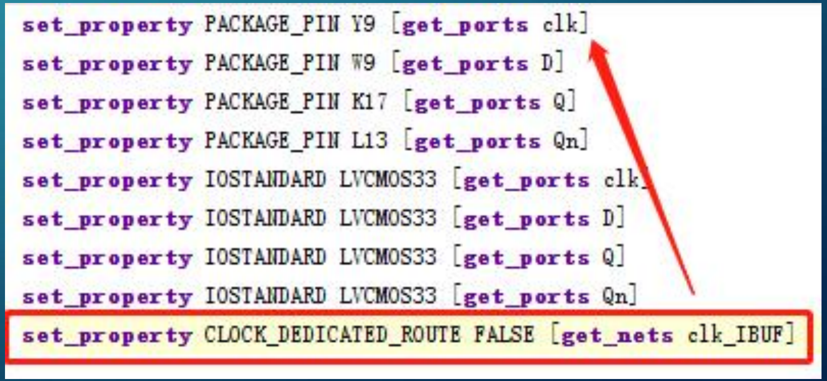
TIPS:

- 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.



```
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 PACKAGE_PIN Y9 [get_ports clk]
set_property PACKAGE_PIN W9 [get_ports D]
set_property PACKAGE_PIN K17 [get_ports Q]
set_property PACKAGE_PIN L13 [get_ports Qn]
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 CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
```