Lab9--Latch & Trigger

姓名：潘子曰 学号：3180105354 专业：计算机科学与技术
课程名称：逻辑与计算机设计基础实验 同组学生姓名：张佳文
试验时间：2019-11-21 实验地点：紫金港东4-509 指导老师：洪奇军

1. Objectives & Requirements

1. Master the structure and working principle of latches and triggers.
2. Know the difference between latches and triggers.
3. Know the structure of **Static Memory** and **SRAM Storage Unit**s.
4. Master the basic functionality and usage of RS Latches.
5. Master the basic functionality and usage of RS Trigger and D Trigger.
6. Master the function of integrated trigger and asynchronous reset.
7. Know how to implement clock division circuit with D Trigger.
8. Know how to implement anti-jitter circuit with D Trigger.

2. Contents & Principles

2.1 Tasks

1. Implement **RS-Latch** with schematic and do simulation test.
2. Implement **gated RS-Latch & D-Latch** and do simulation test.
3. Implement **RS Master-Slave Flip-Flop** with RS-Latch.
4. Implement **D Master-Slave Flip-Flop** with RS-Latch and D-Latch.
5. Implement **Maintain Block D Flip-Flop** with schematic.
6. Do physical test for all triggers.

2.2 Principles

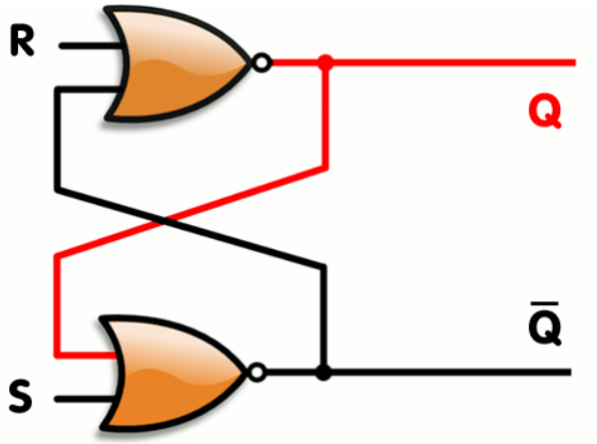
Latches and Flip-Flops

In electronics, a **flip-flop** or **latch** is a circuit that has two stable states and can be used to store state information – a biostable multivibrator. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. It is the basic storage element in sequential logic. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems.

Basic Latches

SR NOR Latch

SR latch operation[3]							
Characteristic table				Excitation table			
S	R	Q_{next}	Action	Q	Q_{next}	S	R
0	0	Q	hold state	0	0	0	X
0	1	0	reset	0	1	1	0
1	0	1	set	1	0	0	1
1	1	X	not allowed	1	1	X	0

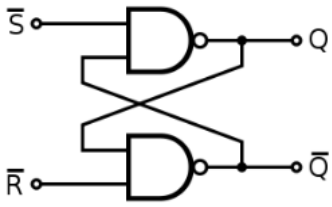


SR NOR Latch can be constructed from a pair of cross-coupled NOR logic gates. The stored bit is present on the output marked Q.

While the R and S inputs are both low, feedback maintains the Q and Q outputs in a constant state, with Q the complement of Q. If S (*Set*) is pulsed high while R (*Reset*) is held low, then the Q output is forced high, and stays high when S returns to low; similarly, if R is pulsed high while S is held low, then the Q output is forced low, and stays low when R returns to low.

Gate Control

$\bar{S}\bar{R}$ NAND Latch

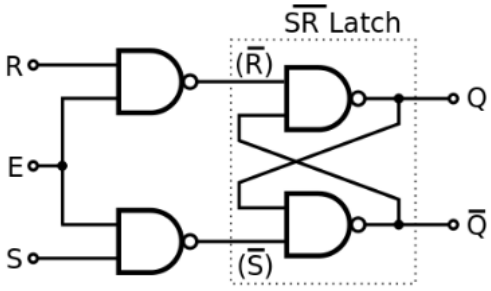


$\overline{SR}$ latch operation		
$\bar{S}$	$\bar{R}$	Action
0	0	Q=1, not(Q)=1, Not allowed
0	1	Q = 1
1	0	Q = 0
1	1	No change & random start

Gated RS Latch

With E high (*enable* true), the signals can pass through the input gates to the encapsulated latch; all signal combinations except for (0,0) = *hold* then immediately reproduce on the (Q,$\bar{Q}$) output, i.e. the latch is *transparent*.

With E low (*enable* false) the latch is *closed (opaque)* and remains in the state it was left the last time E was high.



Gated SR latch operation

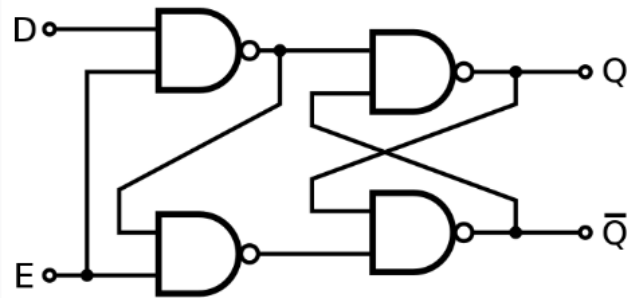
E/C	Action
0	No action (keep state)
1	The same as non-clocked SR latch

Gated D Latch

The truth table and circuit schematic below show that when *enable/clock* input is 0, the D input has no effect on the output. When E/C is high, the output equals D.

Gated D latch truth table

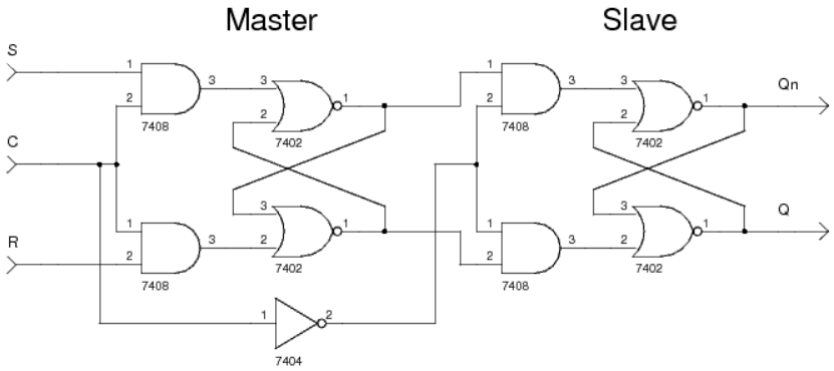
E/C	D		Q	$\overline{Q}$	Comment
0	X		Q_{prev}	$\overline{Q}_{prev}$	No change
1	0		0	1	Reset
1	1		1	0	Set



The low state of the *enable* signal produces the inactive "11" combination. Thus a gated D-latch may be considered as a *one-input synchronous SR latch*. This configuration prevents application of the restricted input combination. It is also known as *transparent latch*, *data latch*, or simply *gated latch*. It has a *data* input and an *enable* signal (sometimes named *clock*, or *control*).

RS Master-Slave Flip-Flop

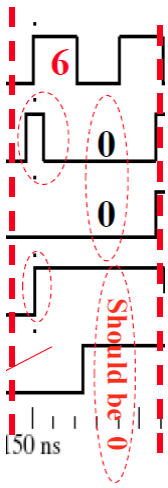
It is built from two gated SR latches: one a master, and the other a slave. The master takes the flip-flops inputs: S (set), R (reset), and C (clock). The clock input is fed to the latch's gate input. The slave takes the master's outputs as inputs (Q to S and Qn to R), and the complement of the flip-flop's clock input. The slave's outputs are the flip-flop's outputs. This difference in clock inputs between the two latches *disconnects* them and eliminates the transparency between the flip-flop's inputs and outputs, which is also a problem in common latches.



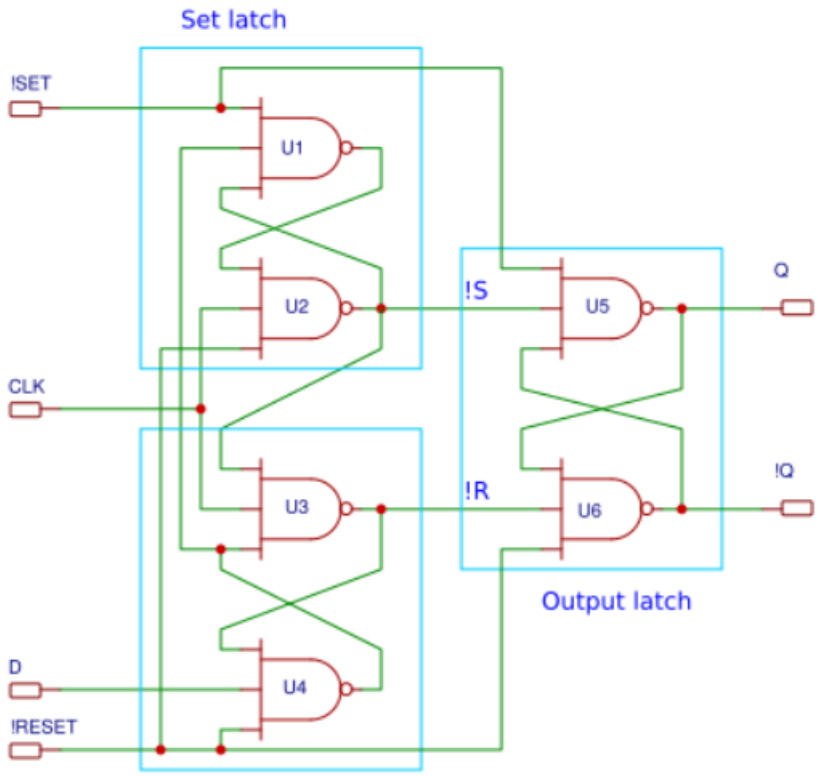
S	R	Q(t+1)	Qn(t+1)	**Meaning**
0	0	Q	Qn	Hold
0	1	0	1	Reset
1	0	1	0	Set
1	1	?	?	Undefined

Disadvantage of RS Master-Slave Flip-Flop

First Catching Problem: take the following simulation result as an example. At the end of the pulse RS=00. But Q finally turned to 1.



Maintain Block Flip-Flop



$\overline{R}$	$\overline{S}$	Cp	D	Q	$\overline{Q}$
0	1	X	X	0	1
1	0	X	X	1	0
1	1	↑	0	0	1
1	1	↑	1	1	0

3. Major Experiment Instruments

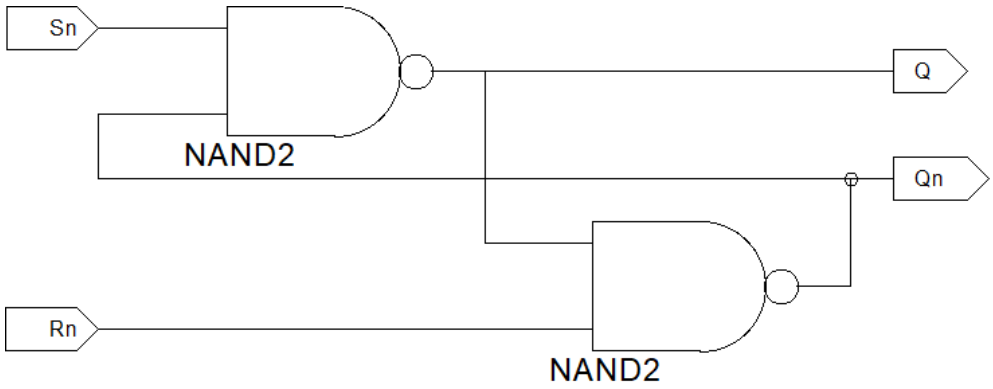
Equipment
Computer (Intel Core i7-9750H, 16GB memory)
Sword circuit design box
Xilinx ISE 14.7

4. Experiment Procedure

Task1: Locker

Design RS Latch

- Implement with NAND gates in schematic
- Encapsulate symbol after verification



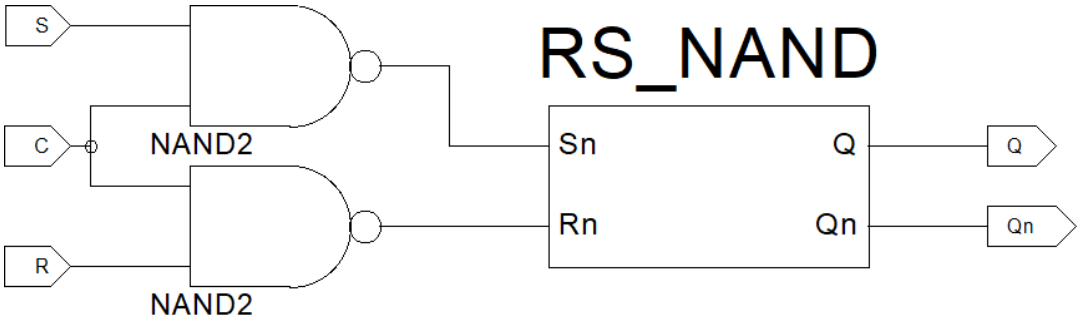
RS Latch Simulation Code

```
`timescale 1ns / 1ps
module RS_NAND_RS_NAND_sch_tb();

// Inputs
reg Rn;
reg Sn;

// Output
wire Q;
wire Qn;

// Bidirs

// Instantiate the UUT
RS_NAND UUT (
        .Q(Q),
        .Qn(Qn),
        .Rn(Rn),
        .Sn(Sn)
);
// Initialize Inputs
    initial begin
        Rn=1;
        Sn=0;
        #50;

        Sn=0;
        Rn=1;
        #50;

        Rn=1;
        Sn=1;
        #50;

        Sn=1;
        Rn=0;
        #50;

        Rn=1;
        Sn=1;
        #50;

        Rn=0;
        Sn=0;
```

```
        #50;

        Rn=1;
        Sn=1;
        #50;

        Rn=0;
        Sn=0;
        #50;

        Sn=0;
        Rn=1;
        #50;

        Rn=0;
        Sn=0;
        #50;

        Sn=1;
        Rn=0;
        #50;

        Rn=1;
        Sn=1;
    end
endmodule
```

Design RS Latch with Enable Signal

- Implement with NAND gates and RS Latch in schematic
- Encapsulate symbol after verification



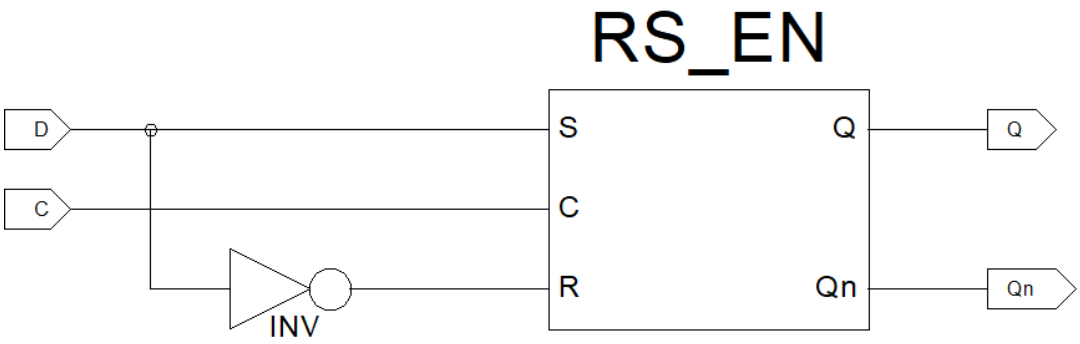
Simulation Code for RS Latch with Enable Signal

```
`timescale 1ns / 1ps
module RS_EN_RS_EN_sch_tb();

// Inputs
reg C;
reg S;
reg R;

// Output
wire Q;
wire Qn;

```

```
// Bidirs

// Instantiate the UUT
RS_EN UUT (
        .C(C),
        .S(S),
        .R(R),
        .Q(Q),
        .Qn(Qn)
);
// Initialize Inputs
    integer i=0;
    initial begin
        C=0;
        S=0;
        R=0;
        #40;

        S=0;    //Hold
        R=0;

        S=1;    //set
        R=0;
        #100;

        S=0;
        R=1;
        #100;

        S=1;    //undefild
        R=1;
        #100;

        S=0;
        R=0;
        #100;

        S=1;    //set
        R=0;
    end

    always@*                        //This is the clock control
module
        for(i=0;i<20;i=i+1) begin
        #50;
        C<=~C;
        end
endmodule
```

Design D Latch with Enable Signal

- Implement with RS Latch in schematic
- Encapsulate symbol after verification



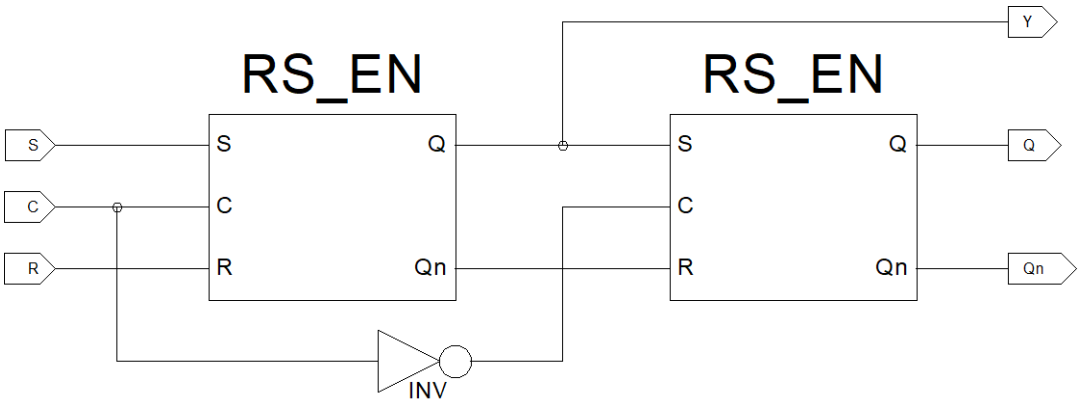
Simulation Code for D Latch with Enable Signal

```
`timescale 1ns / 1ps

module D_EN_D_EN_sch_tb();

// Inputs
reg C;
reg D;

// Output
wire Q;
wire Qn;

// Bidirs

// Instantiate the UUT
D_EN UUT (
        .Q(Q),
        .Qn(Qn),
        .C(C),
        .D(D)
);
// Initialize Inputs
    integer i=0;
    initial begin
        C = 0;
        D = 0;
        #40;
        D=1;
        #50;
        D=0;

        #50;
        D=1;
        #50;
        D=0;
        #50;
        D=1;

    end
    always@*
        for(i=0;i<20;i=i+1) begin
        #50;
        C<=~C;
        end
endmodule
```

Task2: Trig

RS Master-Slave Flip-Flop

- Implement with RS Latch



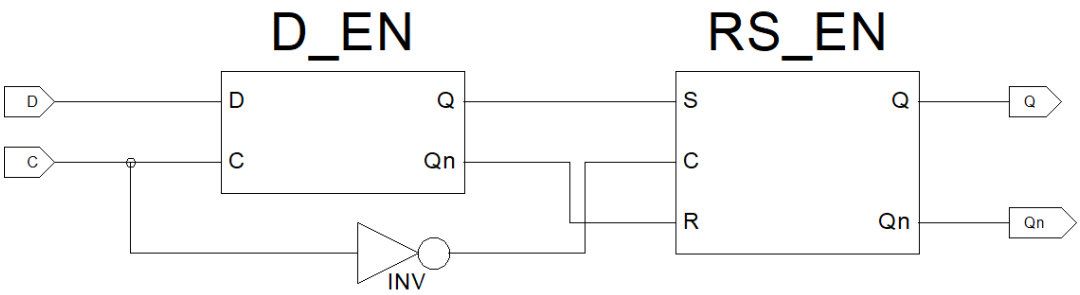
Simulation Code for RS Master-Slave Flip-Flop

```
`timescale 1ns / 1ps

module RS_Trig_RS_Trig_sch_tb();

// Inputs
reg S;
reg C;
reg R;

// Output
wire Y;
wire Q;
wire Qn;

// Bidirs

// Instantiate the UUT
RS_Trig UUT (
        .Y(Y),
        .S(S),
        .C(C),
        .R(R),
        .Q(Q),
        .Qn(Qn)
);
// Initialize Inputs
    integer i=0;
    initial begin
        C=0;
        S=0;
        R=0;
        #50;

        #5;
        S=1;
```

```
36
37          #80;
38          S=0;
39
40          #100;
41          R=1;
42
43          #100;
44          R=0;
45
46          #100;
47          S=1;
48
49          #20;
50          S=0;
51
52          #5;
53          R=1;
54
55          #20;
56          R=0;
57
58          #55;
59          S=1;
60
61          #20;
62          S=0;
63
64          #120;
65          S=1;
66          R=1;
67
68          #100;
69          S=0;
70          R=0;
71
72      end
73      always@*
74          for(i=0;i<30;i=i+1) begin
75          #50;
76          C<=~C;
77          end
78
79  endmodule
```

D Master-Slave Flip-Flop

- Implement with **D Latch** and **RS Latch**.



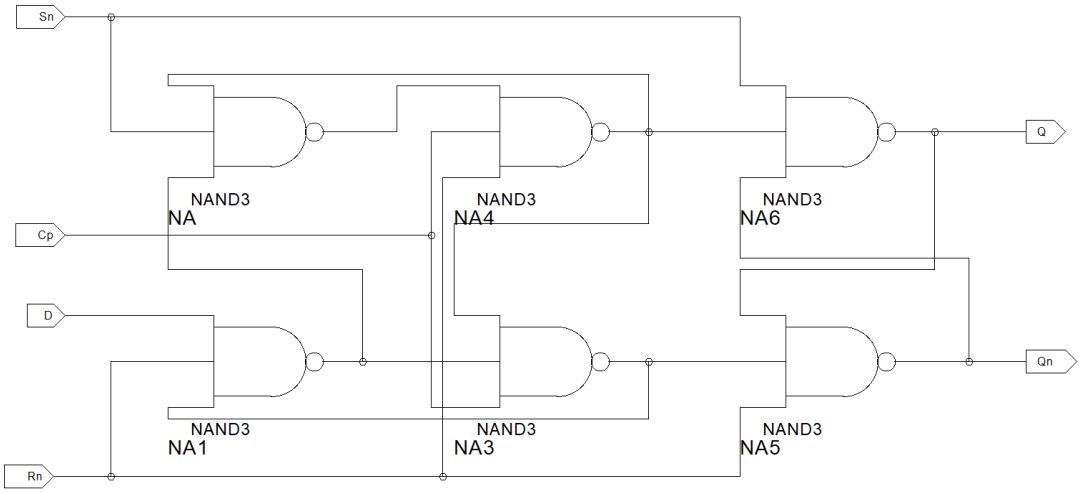
Simulation Code for D Master-Slave Flip-Flop

```
`timescale 1ns / 1ps

module D_Trig_D_Trig_sch_tb();

// Inputs
reg D;
reg C;

// Output
wire Q;
wire Qn;

// Bidirs

// Instantiate the UUT
D_Trig UUT (
        .D(D),
        .C(C),
        .Q(Q),
        .Qn(Qn)
);
// Initialize Inputs
    integer i=0;
    initial begin

        D = 0;
        C = 0;

        #25;
        D=1;

        #50;
        D=0;

        #50;
        D=1;

        #200;
        D=0;

    end
    always@*
        for(i=0;i<30;i=i+1) begin
        #50;
        C<=~C;
        end
```

```
endmodule
```

Maintain Block D Flip-Flop

- Add asynchronous input Rn & Sn



Simulation Code for Maintain Block D Flip-Flop

```
`timescale 1ns / 1ps

module MB_DFF_MB_DFF_sch_tb();

// Inputs
reg Cp;
reg Sn;
reg D;
reg Rn;

// Output
wire Qn;
wire Q;

// Bidirs

// Instantiate the UUT
MB_DFF UUT (
        .Cp(Cp),
        .Qn(Qn),
        .Q(Q),
        .Sn(Sn),
        .D(D),
        .Rn(Rn)
);
// Initialize Inputs
    integer i=0;
    initial begin
        Cp = 0;
        Rn = 0;
        Sn = 1;
        D = 1;

```

```
        #25;
        D=0;
        #50;
        D=1;

        #100;
        Rn=1;

        #50;
        Sn=0;

        #50;
        Sn=1;

        #50;
        D=0;

        #50;
        D=1;


    end
    always@*
        for(i=0;i<30;i=i+1) begin
        #50;
        Cp<=~Cp;
        end
endmodule
```

Physical Test for all Triggers

First Design the clock division module with pulse input.

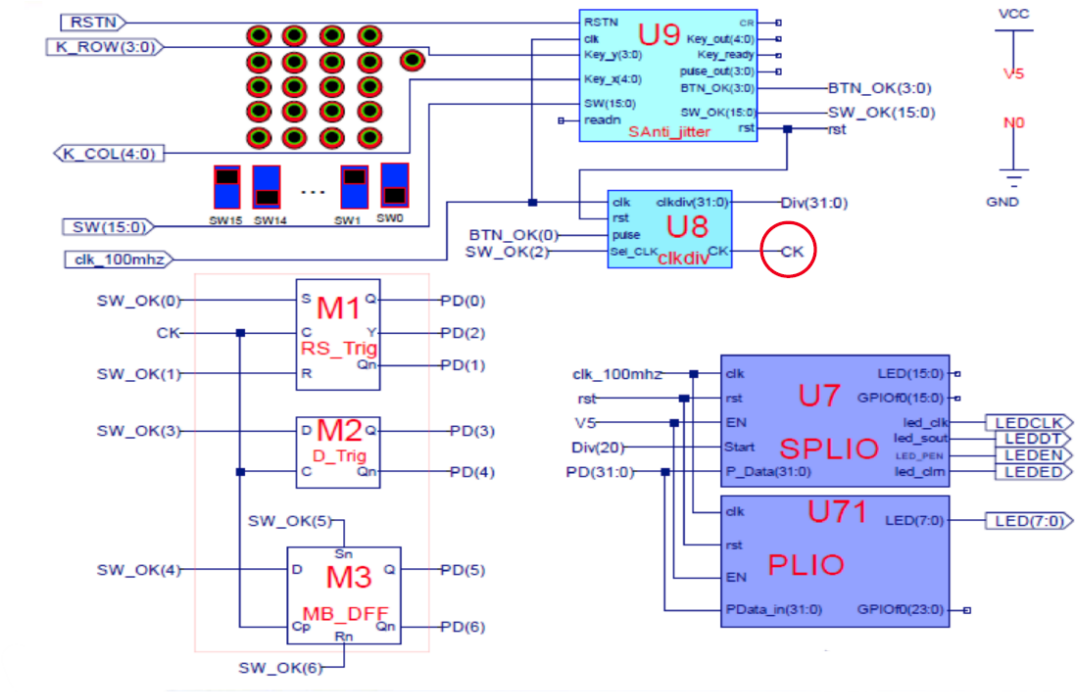
```
`timescale 1ns / 1ps
module clkdiv(
    input clk,
    input rst,
    input pulse,
    input Sel_CLK,
    output reg [31:0] clkdiv,
    output reg CK
    );

    always @ (posedge clk or posedge rst) begin
        if (rst) clkdiv <= 0;
        else clkdiv <= clkdiv + 1'b1;
    end

    always @* begin
        CK <= (Sel_CLK)? pulse : clkdiv[26];
    end

endmodule
```

Then we add some dependencies to the project and construct the top module:

```
`timescale 1ns / 1ps
module Top_Trig(
    input clk_100mhz,
    input RSTN,
    input [3:0] K_COL,
    output [4:0] K_ROW,
    input [15:0] SW,

    output LEDCLK,
    output LEDDT,
    output LEDCLR,
    output LEDEN,
    output [7:0] LED
 );

    wire [31:0] Div, PD;
    wire [15:0] SW_OK;
    wire [3:0] BTN_OK, pulse_out;
    wire rst, CK;

    assign clk = clk_100mhz;
    RS_Trig M1(
        .R(SW_OK[1]),
        .S(SW_OK[0]),
        .Y(PD[2]),
        .Q(PD[0]),
        .Qn(PD[1]),
        .C(CK)
);

    D_Trig M2(
        .D(SW_OK[3]),
        .Q(PD[3]),
        .Qn(PD[4]),
        .C(CK)
);

    MB_DFF M3(
        .Sn(SW_OK[5]),
        .Rn(SW_OK[6]),
        .D(SW_OK[4]),
        .Q(PD[5]),
        .Qn(PD[6]),
        .Cp(CK)
);

    SAnti_jitter U8(
        .clk(clk),
        .RSTN(RSTN),
        .readn(),
        .Key_y(K_COL),
        .Key_x(K_ROW),
        .SW(SW),
        .Key_out(),
        .Key_ready(),
        .pulse_out(),
        .BTN_OK(BTN_OK),
```

```
        .SW_OK(SW_OK),
        .CR(),
        .rst(rst)
    );

    clkdiv U9(
        .clk(clk),
        .rst(rst),
        .Sel_CLK(SW_OK[2]),
        .pulse(BTN_OK[0]),
        .clkdiv(Div),
        .CK(CK)
    );

    SPLIO U7(
        .clk(clk),
        .rst(rst),
        .Start(Div[20]),
        .EN(1'b1),
        .P_Data(PD),
        .LED(),
        .led_clk(LEDCLK),
        .led_sout(LEDDT),
        .led_clrn(LEDCLR),
        .LED_PEN(LEDEN),
        .GPIOf0()
    );

//  PLIO U71(
//      .clk(clk),
//      .rst(rst),
//      .EN(1'b1),
//      .PData_in(PD),
//      .LED(LED),
//      .GPIOf0()
//  );

endmodule
```

The corresponding schematic is presented following:



And also the UCF constraints file:

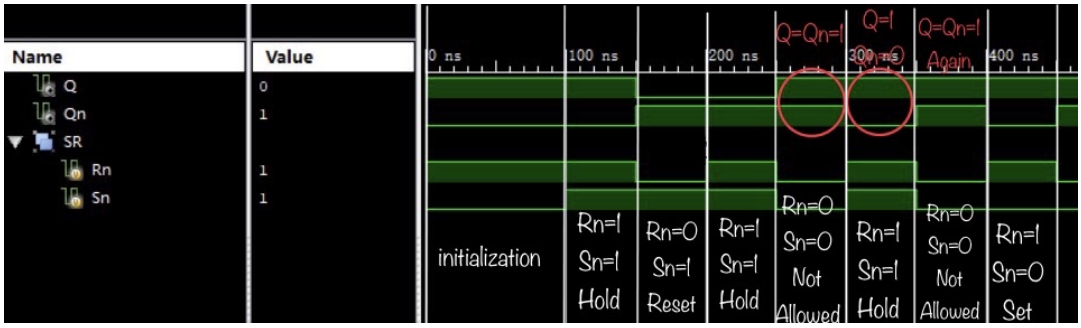
```
NET "clk_100mhz"    LOC=AC18    |   IOSTANDARD=LVCMOS18;
NET "clk_100mhz"    TNM_NET=TM_CLK;
TIMESPEC TS_CLK_100M = PERIOD "TM_CLK" 10 ns HIGH 50%;

NET "RSTN"  LOC=W13 |   IOSTANDARD=LVCMOS18;

NET "K_ROW[0]"  LOC=V17 |   IOSTANDARD=LVCMOS18;
NET "K_ROW[1]"  LOC=W18 |   IOSTANDARD=LVCMOS18;
NET "K_ROW[2]"  LOC=W19 |   IOSTANDARD=LVCMOS18;
NET "K_ROW[3]"  LOC=W15 |   IOSTANDARD=LVCMOS18;
NET "K_ROW[4]"  LOC=W16 |   IOSTANDARD=LVCMOS18;

NET "K_COL[0]"  LOC=V18 |   IOSTANDARD=LVCMOS18;
NET "K_COL[1]"  LOC=V19 |   IOSTANDARD=LVCMOS18;
NET "K_COL[2]"  LOC=V14 |   IOSTANDARD=LVCMOS18;
NET "K_COL[3]"  LOC=W14 |   IOSTANDARD=LVCMOS18;

NET "LEDCLK"    LOC=N26 |   IOSTANDARD=LVCMOS33;
NET "LEDCLR"    LOC=N24 |   IOSTANDARD=LVCMOS33;
NET "LEDDT" LOC=M26 |   IOSTANDARD=LVCMOS33;
NET "LEDEN" LOC=P18 |   IOSTANDARD=LVCMOS33;

NET "SW[0]" LOC=AA10    |   IOSTANDARD=LVCMOS15;
NET "SW[1]" LOC=AB10    |   IOSTANDARD=LVCMOS15;
NET "SW[2]" LOC=AA13    |   IOSTANDARD=LVCMOS15;
NET "SW[3]" LOC=AA12    |   IOSTANDARD=LVCMOS15;
NET "SW[4]" LOC=Y13 |   IOSTANDARD=LVCMOS15;
NET "SW[5]" LOC=Y12 |   IOSTANDARD=LVCMOS15;
NET "SW[6]" LOC=AD11    |   IOSTANDARD=LVCMOS15;
NET "SW[7]" LOC=AD10    |   IOSTANDARD=LVCMOS15;
NET "SW[8]" LOC=AE10    |   IOSTANDARD=LVCMOS15;
NET "SW[9]" LOC=AE12    |   IOSTANDARD=LVCMOS15;
NET "SW[10]"    LOC=AF12    |   IOSTANDARD=LVCMOS15;
NET "SW[11]"    LOC=AE8 |   IOSTANDARD=LVCMOS15;
NET "SW[12]"    LOC=AF8 |   IOSTANDARD=LVCMOS15;
```

```
36 NET "SW[13]"    LOC=AE13    |   IOSTANDARD=LVCMOS15;
37 NET "SW[14]"    LOC=AF13    |   IOSTANDARD=LVCMOS15;
38 NET "SW[15]"    LOC=AF10    |   IOSTANDARD=LVCMOS15;
```

5. Experiment Results and Analyses

Task1: Locker

RS Latch



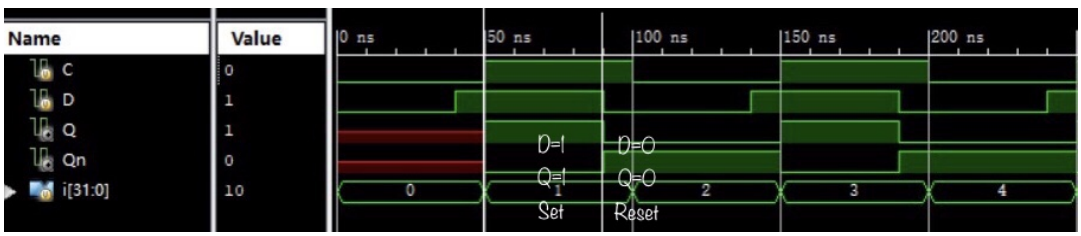
It's much interesting that when we set **both Rn and Sn to 0** which is not allowed, the output is always **Q=Qn=1**.

I think it is somehow a special phenomenon of **ISE14.7**, because my friend from SJTU told me that his simulation test is **XX** on his **vivado** software. Perhaps the simulation software just ignored the signal conflicts and output certain value.

RS Latch with Enable Signal

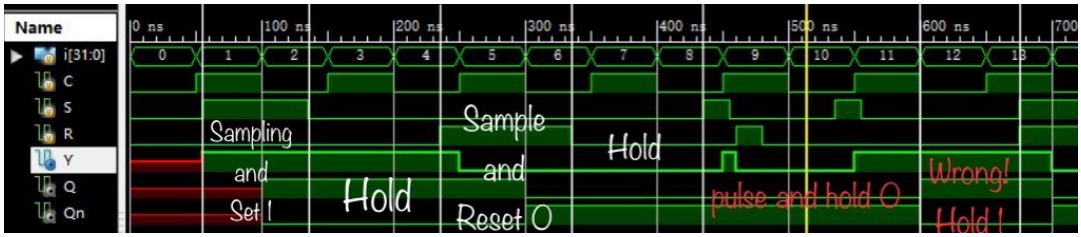
This time I met with the same situation with RS Latch simulation. That Q=Qn=1 when R=S=1 which is not allowed.

D Latch



Task2: Trigger

RS Master-Slave Flip-Flop

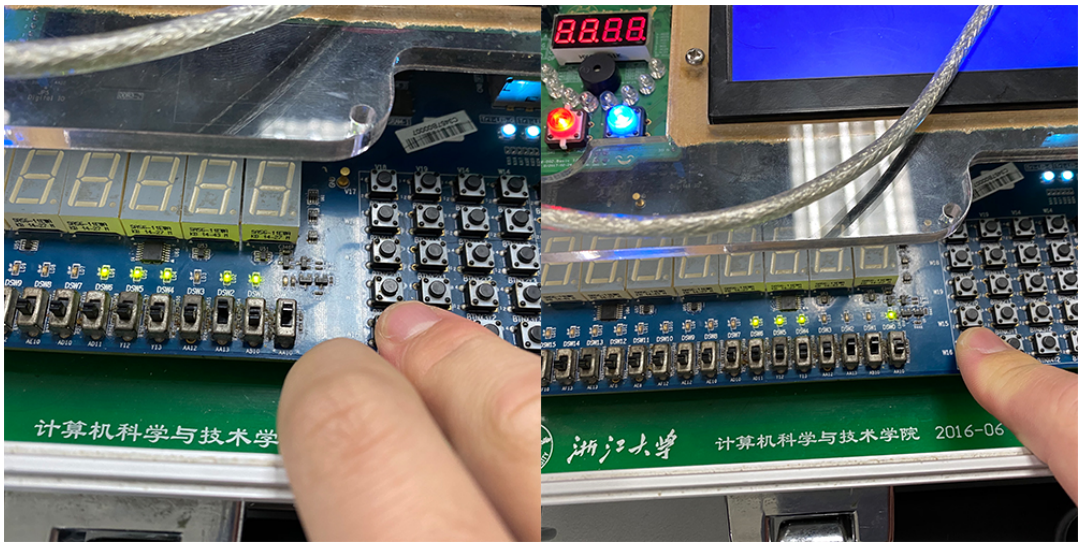


In this simulation I examines the flip phenomenon, which has been marked by red color in the picture.

It happens when S is set 1 at sampling time and then 0 during the sampling intervals, which causes Y be 1 during the output intervals. That's exactly what drives us to more advanced triggers.

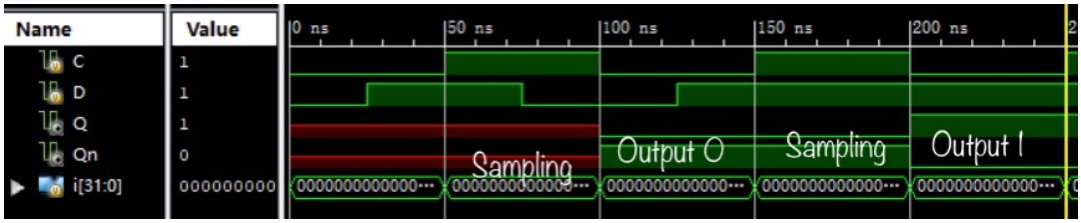
R-S Slave Flip-Flop						
Functionality	S	R	C single step	Y	Q	Qn	Note
Set 1	1	0	0	1	0	1	
			1→0	0	1	0	
Hold	0	0	0	0	1	0	
			1→0	0	1	0	
Reset 0	0	1	0	0	1	0	
			1→0	1	0	1	
Hold	0	0	0	1	0	1	
			1→0	1	0	1	
Undefined	1	1	0	1	0	1	
			1→0	1	0	1	No change
Hold	0	0	0	1	0	1	
			1→0	1	0	1	

R-S Slave Flip-Flop							
Functionality		S	R	C single step	Y	Q	Qn	Note
Single sampling	Set 1	1	0	1→0	0	1	0	
	Sampling RS	0	0	1	0	1	0	
	R=pulse	0	1	1	1	1	0	
	Hold	0	0	1→0	1	0	1	
Single sampling	Reset 0	0	1	1→0	1	0	1	
	Sampling RS	0	0	1	1	0	1	
	S=pulse	1	0	1	0	0	1	
	Hold	0	0	1→0	0	1	0	



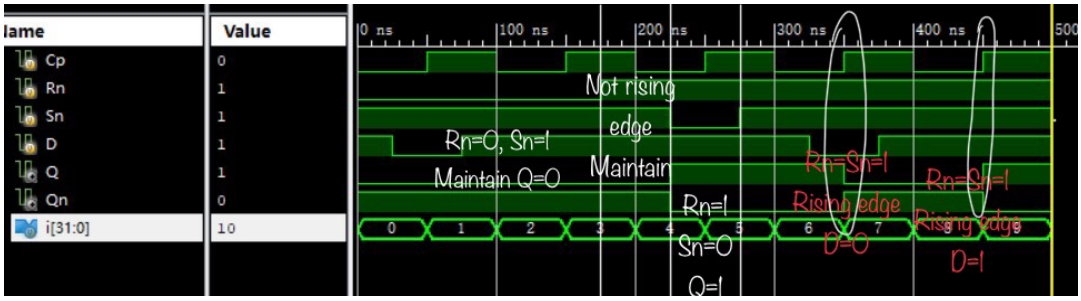
This is the the phenomenon where pulse occurs during the sampling.

D Master-Slave Flip-Flop



D Slave Flip-Flop			
D	C	Q	Qn
0	0	Maintain Previous Value	
1			
0	1→0	1	0
1	1→0	0	1

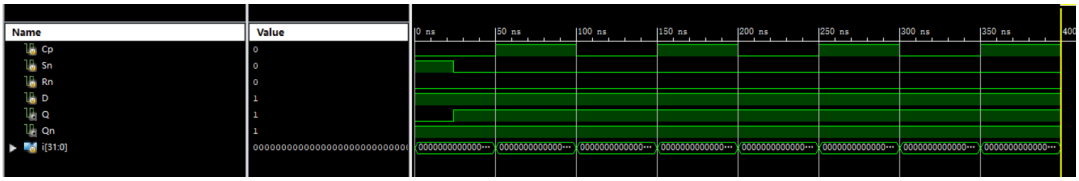
Maintain Block D Flip-Flop



When Rn !=1 and Sn!=0, D input is maintained block. When Rn=Sn=1 and C is 0->1, then D input is accepted.

Maintain Block D Trigger					
Sn	Rn	D	C	Q	Qn
0	1	X	X	1	0
1	0	X	X	0	1
1	1	↑	0	1	0
1	1	↑	1	0	1

However, the following situation is also not allowed by standard that Rn=Sn=0, where Q=Qn=1.



6. Discussion and Conclusion

Thinking Questions

- *Can we simulate on unknown state of trigger? Why?*
  - No we can't. The output state should be unable to predict. Yet the simulation result seems so perfect that each time the result is Q=Qn=1, which is kind of weird.
- *Can we do physical test on unknown state of trigger? Why?*

    - Yes we can. Yet the results are unable to predict. And the result is that nothing changes, which is also reasonable by theory.
- *Describe what caused single sampling and its influence.*
    - When a sample pulse has finished its first change and then changes again within single sampling interval, master trigger will accepted the first sample and thus rejecting the following signal. This will cause problem when using triggers.
- *What is the difference when the state is changed between **D Master-Slave Flip-Flop** and **Maintain Block D Flip-Flop**?*
    - The output signal will be changed anytime within C=1 in D Master-Slave Flip-Flop. Yet that in Maintain Block D Flip-Flop can only be changed when meeting a **rising edge** of C signal
- *Describe how the above difference will influence the real usage of the triggers.*
    - There will be higher probability that receive single sample pulse when using common RS Master-Slave Flip-Flop. If we use this kind of trigger in high frequency circuit, some problems will be thus caused.

Feeling

> This experiment consumes us little more than just finishing each module sequentially. But I met with much complication when doing analyses. Even I have to consult to my classmate in SJTU. Much confusion caused by the weird phenomenon when input prohibited data.