# 实验 8&9——加法器、加减法器和 ALU 基本原理与设计

姓名：　　学号：　　专业：

课程名称：逻辑与计算机设计基础实验　　同组学生姓名：

实验时间：2019-11-13　　实验地点：紫金港东 4-509　　指导老师：洪奇军

## 一．实验目的

- 掌握一位全加器的工作原理和逻辑功能
- 掌握串行进位加法器的工作原理和进位延迟
- 掌握减法器的实现原理
- 掌握加减法器的设计方法
- 掌握 ALU 基本原理及在 CPU 中的作用
- 掌握 ALU 的设计方法

## 二．实验设备与材料

### 2.1 实验设备

- 安装有 Xilinx ISE 14.7 的计算机　　1 台
- SWORD 开发板　　1 套

### 2.2 实验材料

无

## 三．实验任务

- 任务 1：原理图方式设计 4 位加减法器

- 任务 2：实现 4 位 ALU 及应用设计

## 四 . 实验原理

- 一位全加器

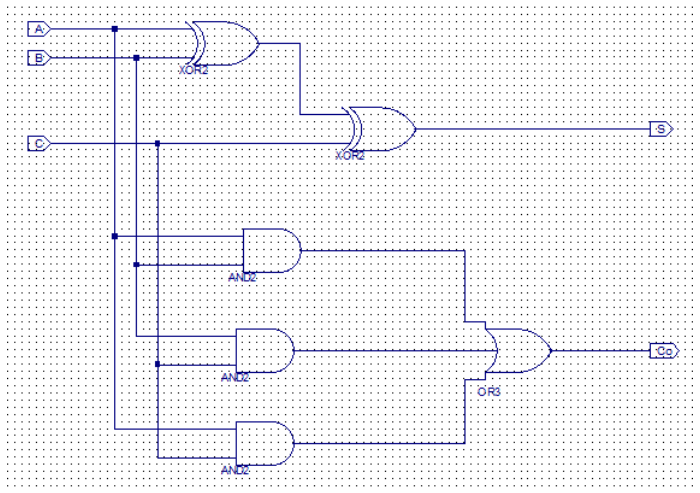
三个输入位：数据位 Ai 和 Bi，低位进位输入 $C_i$；

二个输出位：全加和 Si，进位输出 $C_{i+1}$；

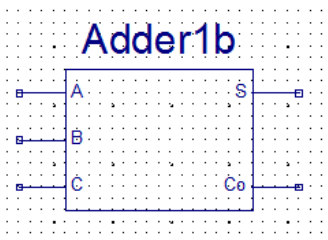
| $A_i$ | $B_i$ | $C_i$ | $S_i$ | $C_{i+1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

$$S_i = A_i \otimes B_i \otimes C_i$$
$$C_{i+1} = A_iB_i + B_iC_i + C_iA_i$$

根据一位全加器的输入输出关系，得到电路图：



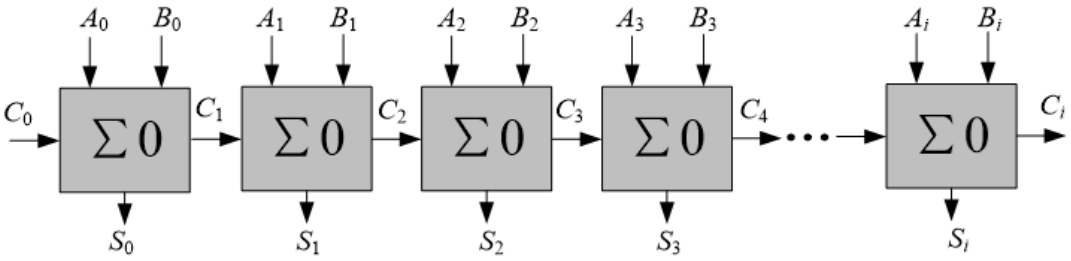

```
module adder_1bit(
    input wire a, b, ci,
    output wire s, co);
    and m0(c1,a,b);
    and m1(c2,b,ci);
    and m2(c3,a,ci);
    xor m3(s1,a,b);
    xor m4(s,s1,ci);
    or  m5(co,c1,c2,c3);
endmodule
```



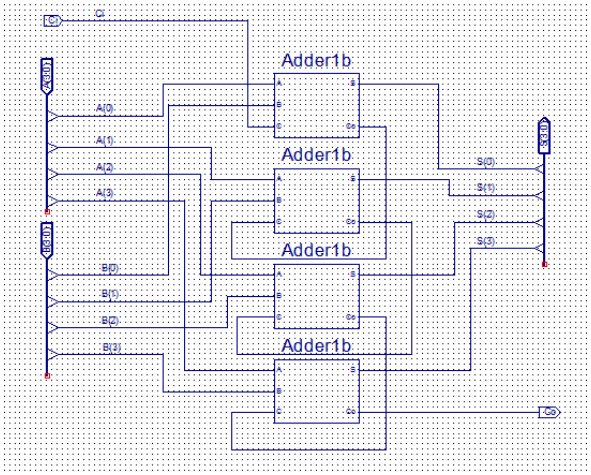
- 多位串行进位加法器

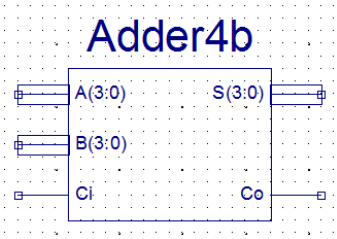
由一位全加器将进位串接构成；

低位进位 C0 为 0，Ci 为高位进位输出；




- 四位全加器



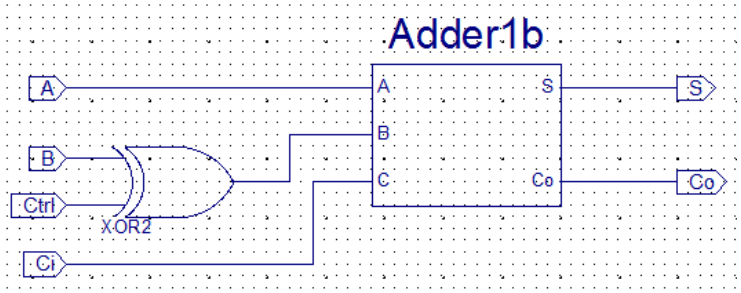



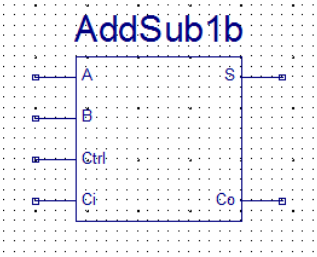

- 一位加减法器

用负数补码加法实现，减数当作负数求补码；

共用加法器；

用“异或”门控制求反，低位进位 Co 为 1；



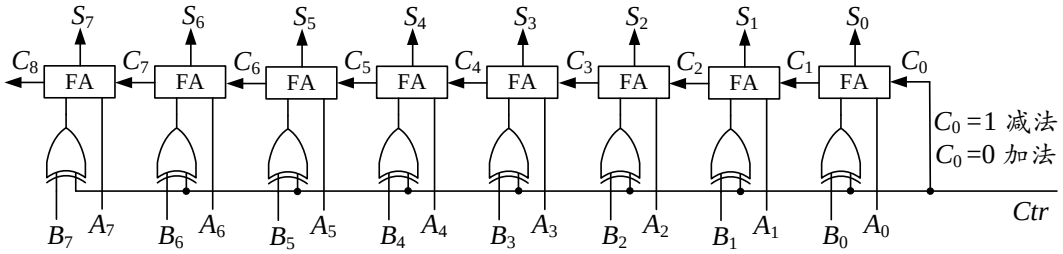



- 多位串行进位全减器

用负数补码加法实现，减数当作负数求补码；

共用加法器；

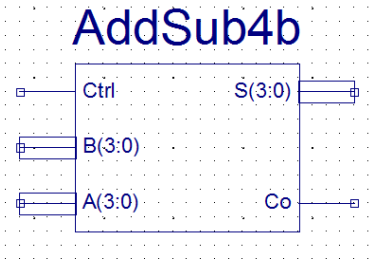
用“异或”门控制求反，低位进位 Co 为 1；

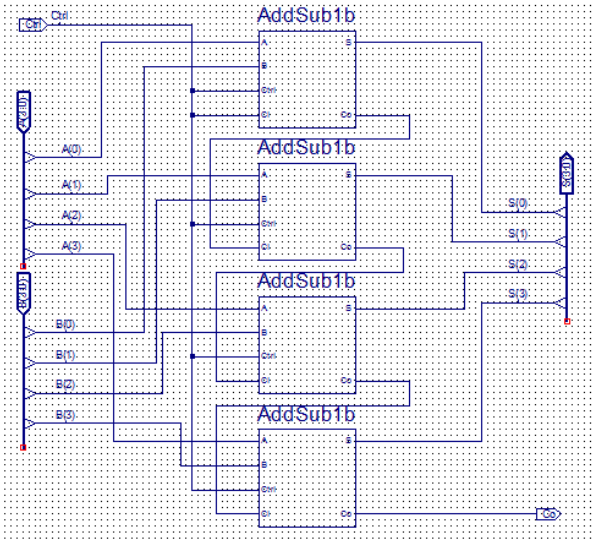



$Ctr$为0时，$S[7:0]=A[7:0]+B[7:0]$

$Ctr$为1时，$S[7:0]=A[7:0]-B[7:0]$

$=A[7:0]+\overline{B[7:0]}+1$




- 8 位加减法器代码实现

Ctr 控制加减法：

Ctr=0 做加法；

Ctr=1 做减法，同时控制 Co=1；

```
module add_sub_8bits(A, B, Ctr, S, Co);
.....                          //端口及变量定义
  assign Bo={Ctr,Ctr,Ctr,Ctr,Ctr,Ctr,Ctr,Ctr}^B;                    // = {8{Ctr}} ^ B
  assign Coo=Co^Ctr;
  adder_1bit        A1(.a(A[1]),.b(Bo[1]),.ci(Ctr),.sum(S[1]),.co(Ctemp[1])),
                    A2(A[2],Bo[2],Ctemp[1],S[2],Ctemp[2]),
                    A3(A[3],Bo[3],Ctemp[2],S[3],Ctemp[3]),
                    A4(A[4],Bo[4],Ctemp[3],S[4],Ctemp[4]),
                    A5(A[5],Bo[5],Ctemp[4],S[5],Ctemp[5]),
                    A6(A[6],Bo[6],Ctemp[5],S[6],Ctemp[6]),
                    A7(A[7],Bo[7],Ctemp[6],S[7],Ctemp[7]),
                    A8(A[8],Bo[8],Ctemp[7],S[8],Co);
endmodule
```

- 设计按键输入模块

使用行为描述设计；

在实验 7 基础上，更新 Adder4b 为 AddSub4b 模块；

```
module CreateNumber(
   input wire [3:0] btn,
   input wire [3:0] sw,
   output reg [15:0] num
    );
   wire [3:0] A1,B1,C1,D1;

   initial num <= 16'b1010_1011_1100_1101;  // display "AbCd"

   AddSub4b a1(.A(num[ 3: 0]),.B(4'b0001),.Ctrl(sw[0]),.S(A1));
   AddSub4b a2(.A(num[ 7: 4]),.B(4'b0001),.Ctrl(sw[1]),.S(B1));
   AddSub4b a3(.A(num[11: 8]),.B(4'b0001),.Ctrl(sw[2]),.S(C1));
   AddSub4b a4(.A(num[15:12]),.B(4'b0001),.Ctrl(sw[3]),.S(D1));

   always@(posedge btn[0]) num[ 3: 0]<= A1;
   always@(posedge btn[1]) num[ 7: 4]<= B1;
   always@(posedge btn[2]) num[11: 8]<= C1;
   always@(posedge btn[3]) num[15:12]<= D1;

endmodule
```

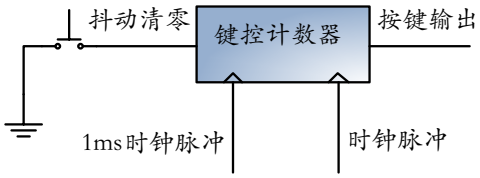
- 按键去抖动原理

抖动原因：按键按下或放开时，存在机械震动

抖动时间一般在 10~20ms

按键去抖动方法：延时，以避开机械抖动


按放键时的抖动脉冲




去抖动原理

```
module pbdebounce(
    input wire clk_1ms,
    input wire button,
    output reg pbreg
    );

    reg [7:0] pbshift;

    always@(posedge clk_1ms) begin
        pbshift=pbshift<<1;
        pbshift[0]=button;
        if (pbshift==8'b0)
            pbreg=0;
        if (pbshift==8'hFF)
            pbreg=1;
    end
endmodule //button要连续按8ms才有效输出一个pbreg.
```

Pbshift 寄存及变化情况

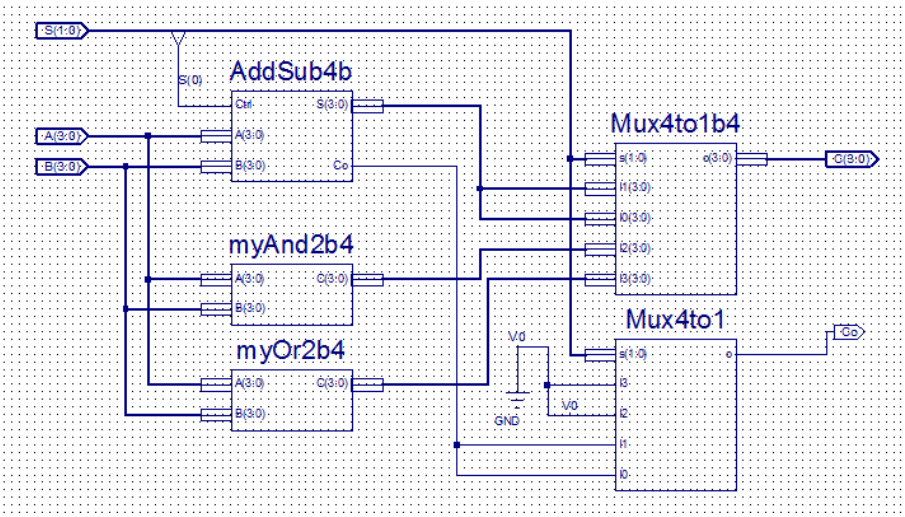
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |


```
(1/100mhz) *X=1ms
    --> x=100MHz*1ms
        =100*10^6*1*10^(-3)
        =10^5s
    2^17=131072,
        =1.3*10^5
clkdiv[16]=2^17
```

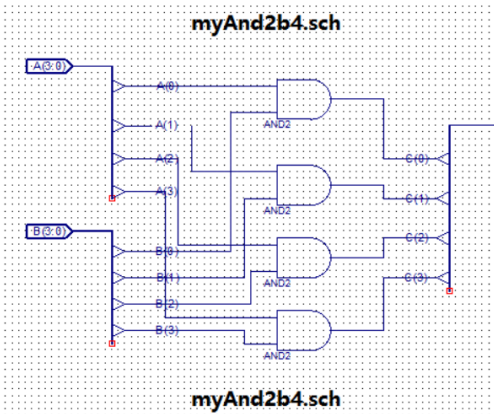
- 辅助模块：时钟分频计数器

```
module clkdiv(input wire clk,output reg[31:0] clkdiv=0);
    //clkdiv[0] 第1个L->h,第2个H->L
    always @(posedge clk) begin
        clkdiv <=clkdiv + 1'b1;
    end
endmodule
```

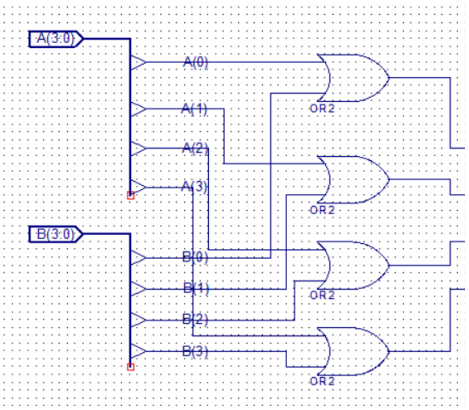
- 四位 ALU 原理图



四位与、或门原理图：




四位与门




四位或门

四位 ALU 仿真：

```
integer i;
initial begin
    A=4'b1010;
    B=4'b0111;#100;
    B=4'b0011;
end
for (i=0; i<=3;i=i+1) begin
    #100;
    S=i;
end
```

# 五 . 实验内容与步骤

## 5.1 实验任务

任务 1：原理图方式设计 4 位加减法器

任务 2：实现 4 位 ALU 及应用设计

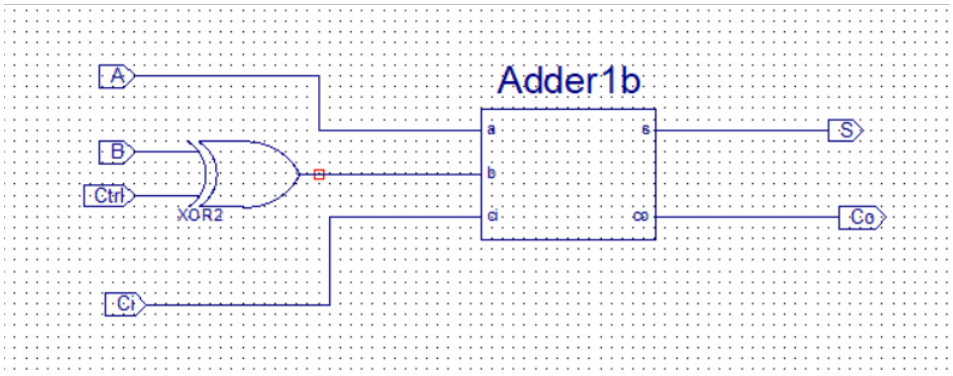
## 5.2 实验内容

新建工程，工程名称用 MyALU；

Top Level Source Type 用 HDL；

新建源文件，类型是 Schematic，文件名称用 AddSub1b；

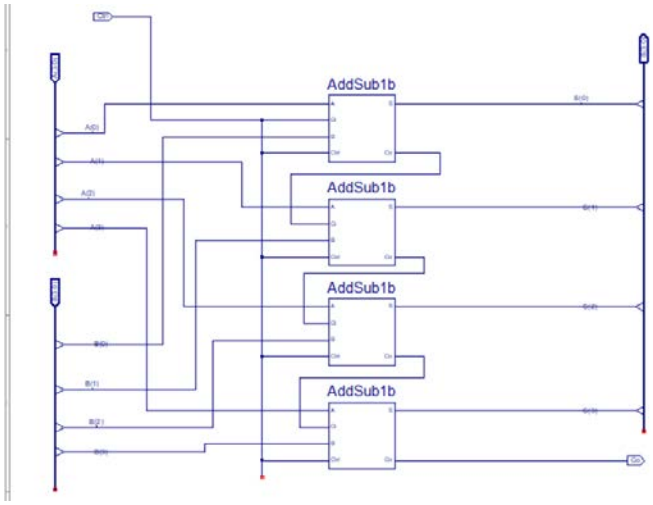
原理图方式进行设计；



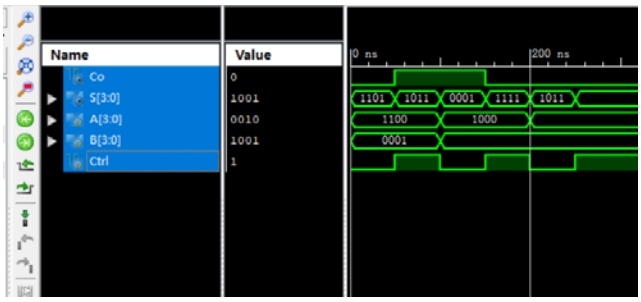

新建源文件，类型是 Schematic，文件名称用 AddSub4b；

原理图方式进行设计，调用前面设计的 AddSub1b；

进行波形仿真，激励输入至少 4 组




```
26  // Initialize Inputs
27      initial begin
28          A = 4'b1100;
29          B = 4'b0001;
30          Ctrl = 0;
31          #50;
32          Ctrl=1;
33          #50;
34          A = 4'b1000;
35          B = 4'b1001;
36          Ctrl=0;
37          #50;
38          Ctrl=1;
39          #50;
40          A = 4'b0010;
41          B = 4'b1001;
42          Ctrl=0;
43          #50;
44          Ctrl=1;
45          #50;
46      end
```

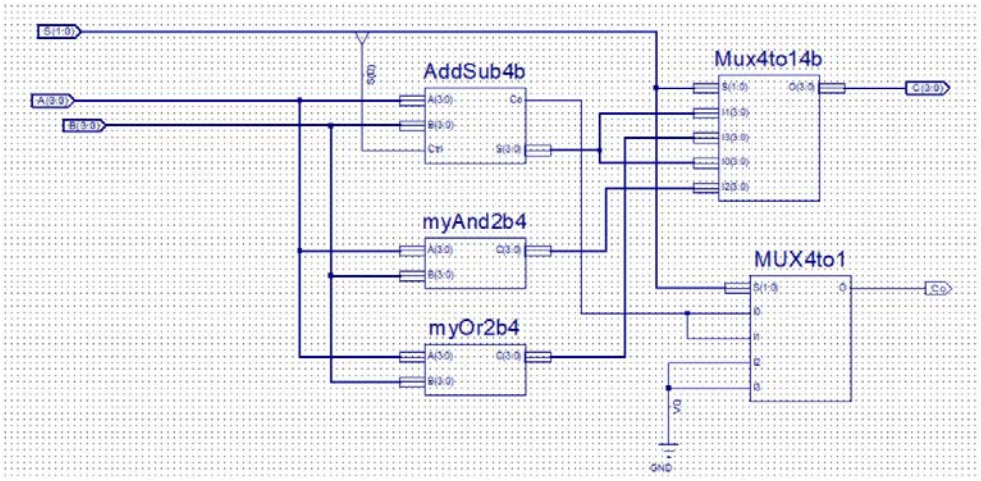


- ALU 设计

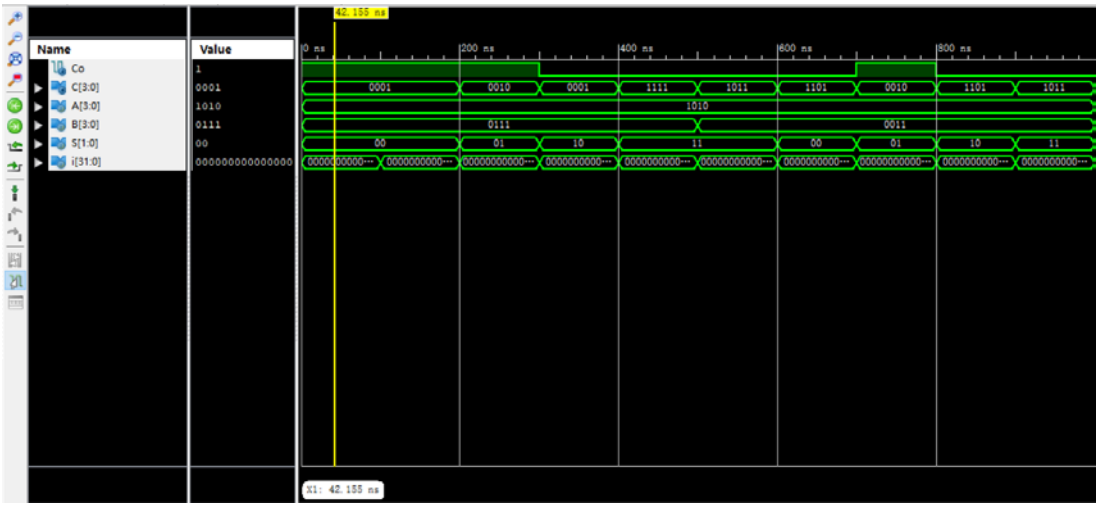
新建源文件，类型是 Verilog 或 Schematic，文件名称用 ALU；

原理图方式进行设计；

进行波形仿真，激励输入至少 4 组，覆盖 4 种操作；




```
// Initialize Inputs
    integer i;
        initial begin
        A=4'b1010;
        B=4'b0111;
        S=2'b00;
        for(i=0;i<=3;i=i+1)begin
            #100;
            S=i;
        end
        #100;
        B=4'b0011;
        for(i=0;i<=3;i=i+1)begin
            #100;
            S=i;
        end
    end
```



新建源文件，类型是 Verilog，文件名 Top；

右键设为“Set as Top Module”；

代码输入进行设计：

调用 pbdebounce 模块

调用 AddSub4b 模块

调用 pbdebounce、clkdiv 模块

调用 DispNum 模块

调用 CreateNumber 模块

业务逻辑要求：

两个 4 位操作数 A，B；

可用两个按键进行自增/减；

得到结果 C 和进位 Co；

把 A、B、C 和 Co 动态显示；

```
module Top(
    input wire clk,
    input wire[1:0] btn,
    //input wire[3:0] SW,//enable the led segments
    input wire[1:0] SW1,//control the +/- of each btn
    input wire[1:0] SW2,//control the ALU
    output wire[3:0] AN,
    output wire[7:0] SEGMENT,
    output wire BTNX4);

    wire[15:0] num;
    wire[1:0] btn_out;
    wire[3:0] C;
    wire Co;
    wire[31:0] clk_div;

    assign BTNX4=0;

    clkdiv m2(clk,1'b0,clk_div);
    pbdebounce m0(.clk_1ms(clk_div[17]),.button(btn[0]),.pbreg(btn_out[0]));
    pbdebounce m1(.clk_1ms(clk_div[18]),.button(btn[1]),.pbreg(btn_out[1]));
    CreateNumber m3(.btn({2'b00,btn_out[1:0]}),.sw({2'b00,SW1[1:0]}),.num(num[15:0]));
    ALU m4(.S(SW2),.A(num[3:0]),.B(num[7:4]),.C(C),.Co(Co));
    disp_num m5(.clk(clk),.RST(1'b0),.HEXS({3'b000,Co,C[3:0],num[7:0]}),.LES(4'b0000),.points(4'b1111),
        .AN(AN),.Segment(SEGMENT));

endmodule
```

- 物理验证

UCF 引脚定义：

输入

时钟：clk　　//SWORD 主板的按键控制数据加减

按键控制输入：BTNX4Y1 控制 num[3:0]，BTNX4Y0 控制 num[7:4]

按键加/减 1 控制：sw[2]对应 sw[0]，sw[3]对应 sw[1]

ALU 运算控制：sw2[1:0]，00-加，01-减，10-与，11-或

输出

AN[0]：A - num[3:0]

AN[1]：B - num[7:4]

AN[2]：C - C

AN[3]：Co – Co

用 BTNX4Y3 ~ BTNX4Y0 这 4 个按钮，每个按钮按下一次，对应的数码管的值加 1

按键输入数字：BTNX4Y0~BTNX4Y3 为 btn[3:0]

按键使能控制线：BTNX4

```
net "clk" loc=AC18 | iostandard=LVCMOS18;
net "SW1[1]" loc=AE13 | iostandard=LVCMOS15;
net "SW1[0]" loc=AF8 | iostandard=LVCMOS15;

net "SW2[1]" loc=AE8 | iostandard=LVCMOS15;
net "SW2[0]" loc=AF12 | iostandard=LVCMOS15;

net "btn[1]" loc=V14 | IOSTANDARD=LVCMOS18;
net "btn[1]" CLOCK_DEDICATED_ROUTE=false;
net "btn[0]" loc=W14 | IOSTANDARD=LVCMOS18;
net "btn[0]" CLOCK_DEDICATED_ROUTE=false;
net "BTNX4" loc=W16 | IOSTANDARD=LVCMOS18;

net"SEGMENT[0]"loc=AB22 | IOSTANDARD=LVCMOS33;
net"SEGMENT[1]"loc=AD24 | IOSTANDARD=LVCMOS33;
net"SEGMENT[2]"loc=AD23 | IOSTANDARD=LVCMOS33;
net"SEGMENT[3]"loc=Y21  | IOSTANDARD=LVCMOS33;
net"SEGMENT[4]"loc=W20  | IOSTANDARD=LVCMOS33;
net"SEGMENT[5]"loc=AC24 | IOSTANDARD=LVCMOS33;
net"SEGMENT[6]"loc=AC23 | IOSTANDARD=LVCMOS33;
net"SEGMENT[7]"loc=AA22 | IOSTANDARD=LVCMOS33;

net"AN[0]"loc=AC21 |IOSTANDARD=LVCMOS33;
net"AN[1]"loc=AD21 |IOSTANDARD=LVCMOS33;
net"AN[2]"loc=AB21 |IOSTANDARD=LVCMOS33;
net"AN[3]"loc=AC22 |IOSTANDARD=LVCMOS33;
```

## 5.3 实验结果

当 SW2[1:0]取不同值时，ALU 进行不同运算；

当 SW1[1:0]取不同值时，按钮对应不同的效果；

与预期相符；

## 六．实验心得

经过这一次实验，我对 ALU 的理解深入了许多，知道了它的设计原理，也知道了 ALU 在 CPU 中的作用，在设计 ALU 的时候，采用了分层设计的思想，把 ALU 分成几个模块，再依次完成各个模块，各个模块又可以分成一些更小的模块，这样的设计方式提高了设计的效率，也让整个工程层次分明，思路清晰。

在这次实验中，我对仿真模拟的认识也更深了一些，仿真模拟可以在每个模块完成后用来检验模块的正确性，便于保证整体设计的正确性，也可以用来在整体设计出问题的时候用来定位错误发生的模块。

在这次实验中 Verilog 语言已经用到了很多，但自己还不是很熟练，需要再加强对 Verilog 语言的学习和应用，希望下一次实验可以做的更好。

# 实验 10

# ——锁存器与触发器基本原理

姓名：　　学号：　　专业：

课程名称：逻辑与计算机设计基础实验　　同组学生姓名：

实验时间：2019-11-20　　实验地点：紫金港东 4-509　　指导老师：洪奇军

## 一． 实验目的

- 掌握锁存器与触发器构成的条件和工作原理
- 掌握锁存器与触发器的区别
- 掌握基本 SR 锁存器、门控 SR 锁存器、D 锁存器、SR 锁存器、D 触发器的基本功能
- 掌握基本 SR 锁存器、门控 SR 锁存器、D 锁存器、SR 锁存器存在的时序问题

## 二． 实验设备与材料

### 2.1 实验设备

| | |
|---|---|
| 装有 Xilinx ISE 14.7 的计算机 | 1 台 |
| SWORD 开发板 | 1 套 |

### 2.2 实验材料

无

## 三． 实验任务

- 实现基本 SR 锁存器，验证功能和存在的时序问题
- 实现门控 SR 锁存器，并验证功能和存在的时序问题
- 实现 D 锁存器，并验证功能和存在的时序问题

- 实现 SR 主从触发器，并验证功能和存在的时序问题
- 实现 D 触发器，并验证功能

# 四 . 实验原理

- 锁存器

构成锁存器的充分条件:

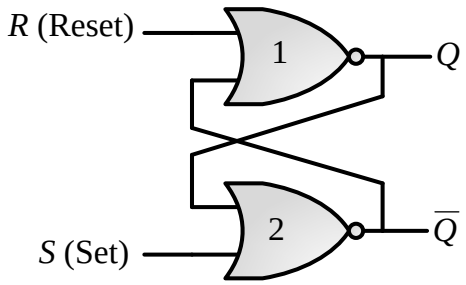
- 能长期保持给定的某个稳定状态
- 有两个稳定状态：0、1
- 在一定条件下能随时改变逻辑状态，即：置 1 或置 0

最基本的锁存器有：SR 锁存器、D 锁存器

锁存器有两个稳定状态，又称双稳态电路

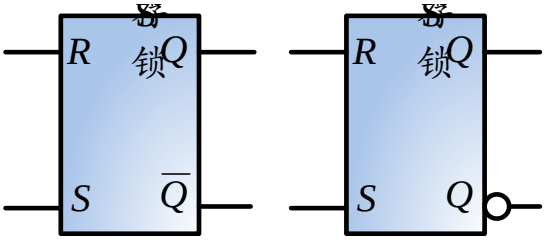
## 4.1 SR 锁存器

将两个具有 2 输入端的反向逻辑器件的输出与输入端交叉连起来，另一个输入端作为外部信息输出端，就构成最简单的 SR 锁存器;

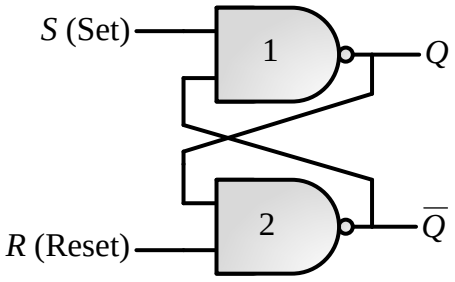



| R S | Q Q | 说明 |
|---|---|---|
| 0 0 | Q Q | 保持 |
| 0 1 | 1 0 | 置1 |
| 1 0 | 0 1 | 置0 |
| 1 1 | 0 0 | 未定义 |

- NOR 结构 SR 锁存器



- NAND 结构 RS 锁存器




| R S | Q Q | 说明 |
|---|---|---|
| 0 0 | 1 1 | 未定义 |
| 0 1 | 0 1 | 置0 |
| 1 0 | 1 1 | 置1 |
| 1 1 | Q Q | 保持 |

仿真主要代码:

R=1;S=1; #50;

R=1;S=0; #50;

R=1;S=1; #50;

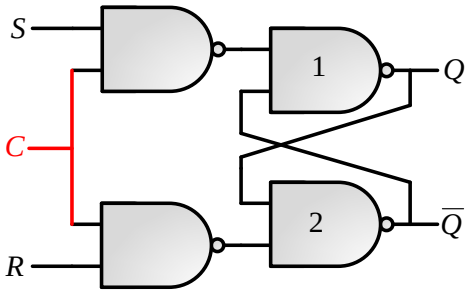
R=0;S=1; #50;

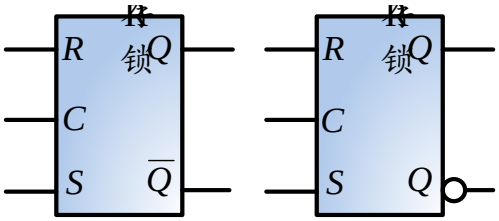
R=1;S=1; #50;

R=0;S=0; #50;

R=1;S=1; #50;

## 4.2 门控 SR 锁存器




| C R S | Q Q | 说明 |
|---|---|---|
| 0 × × | Q Q | 保持 |
| 1 0 0 | Q Q | 保持 |
| 1 0 1 | 1 0 | 置1 |
| 1 1 0 | 0 1 | 置0 |
| 1 1 1 | 1 1 | 未定义 |

仿真主要代码：

C=1;R=1;S=1; #50;

R=1;S=0; #50;

R=1;S=1; #50;

R=0;S=1; #50;

R=1;S=1; #50;

R=0;S=0; #50;

R=1;S=1; #50;

C=0;R=1;S=1; #50;

R=1;S=0; #50;

R=1;S=1; #50;

R=0;S=1; #50;

R=1;S=1; #50;

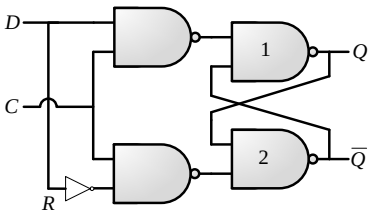
R=0;S=0; #50;

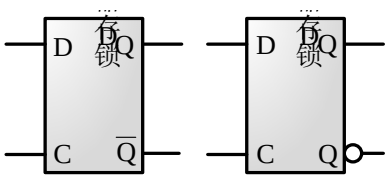
R=1;S=1; #50;

## 4.3 D 锁存器

基本 SR 锁存器缺点：存在不确定状态

解决方法：消除不确定状态




| C D | Q Q | 说明 |
|---|---|---|
| 0 × | Q Q | 保持 |
| 1 0 | 0 1 | 置0 |
| 1 1 | 1 0 | 置1 |

只需 1 个数据输入端 D

输出端 Q 等于输入端 D

采用电平控制 C

仿真主要代码：

C=1;D=1; #50;

D=0; #50;

C=0;D=1; #50;

D=0;

- 触发器

D 锁存器的缺点：存在空翻现象——如果 D 锁存器直接用在时序电路中作为状态存储元件，当使能控制信号有效时，会导致该元件内部的状态值随时多次改变，而不是保持所需的原始状态值

解决方法：消除空翻现象，使每次触发仅使锁存器的内部状态仅改变一次

触发：外部输入使锁存器状态改变的瞬间状态

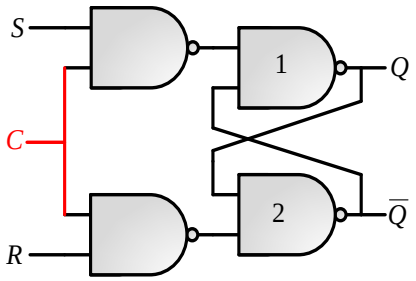
触发器：在锁存器的基础上使每次触发仅使状态改变一次的锁存电路（双稳态）

空翻现象：

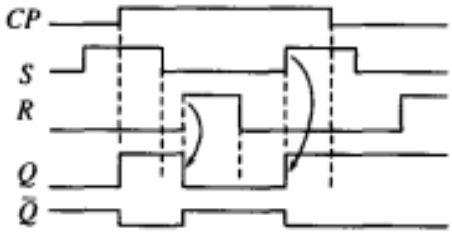
又称为竞态现象，是数字电路中的一个术语，指在同一个时钟脉冲信号作用区间内，由于时钟脉冲的宽度过大，触发器出现在“0”“1”两逻辑信号中多次翻转的现象。它限制了同步 RS 触发器在实际工作中的正常应用。

RS 时钟触发器空翻现象：

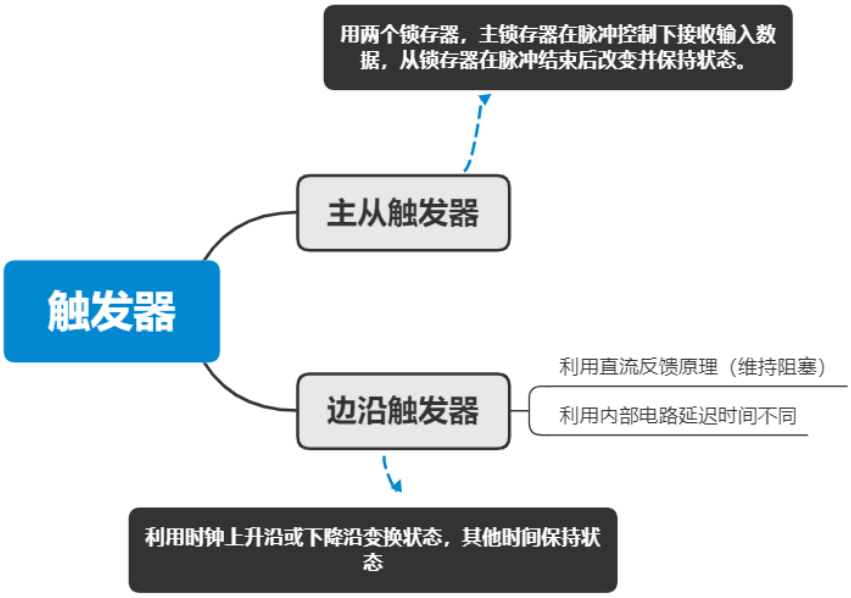
同步 RS 时钟触发器有空翻现象。空翻是在基本 RS 触发器的基础上构造时钟触发器时，因导引电路 C 门和 D 门功能不完善而造成的一种现象，即在一次时钟来到期间，触发器多次翻转的现象称为空翻，如图所示。这违背了构造时钟触发器的初衷，每来一次时钟，最多允许触发器翻转一次，若多次翻转，电路也会发生状态的差错，因而是不允许的。因为在 CP=1 期间，时钟对 C 门和 D 门的封锁作用消失，数据端 R 和 S 端的多次变化就会通过 C 门和 D 门到达基本 RS 触发器的输入端，造成触发器在一次时钟期间的多次翻转。




| $CRS$ | $QQ$ | 说明 |
|---|---|---|
| 0×× | $QQ$ | 保持 |
| 100 | $QQ$ | 保持 |
| 101 | 10 | 置1 |
| 110 | 01 | 置0 |
| 111 | 11 | 未定义 |




常见的触发器有：主从 SR 触发器、D 触发器、JK 触发器、T 触发器



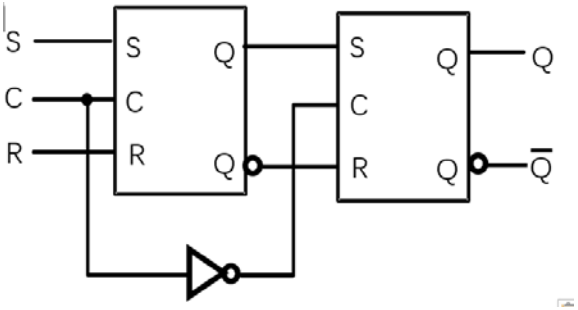

## 4.4 SR 主从触发器

由两个钟控 S-R 锁存器串联构成，第二个锁存器的时钟通过反相器取反

当 C=1 时，输入信号进入第一个锁存器（主锁存器）

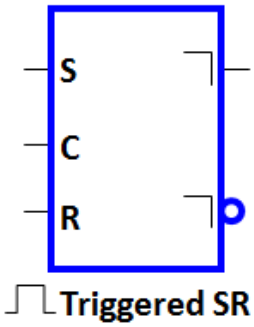
当 C=0 时，第二个锁存器（从锁存器）改变输出

从输入到输出的通路被不同的时钟信号值(C = 1 和 C = 0)所断开




| R S | Q $\overline{Q}$ | S R |
|---|---|---|
| 0 0 | 1 1 | 无定义 |
| 0 1 | 0 1 | 置 0 |
| 1 0 | 1 0 | 置 1 |
| 1 1 | Q $\overline{Q}$ | 保持 |

不确定

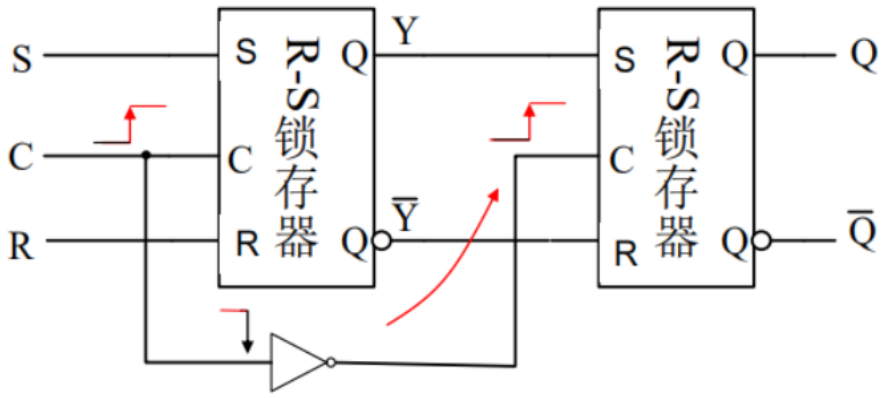



锁存器存在的问题：

- 锁存器的输出端可以直接反应出输入端的数据
- 在时序电路存储状态时出现“空翻”状态无法控制
- 解决思想：切断回路，同步变化一次→触发

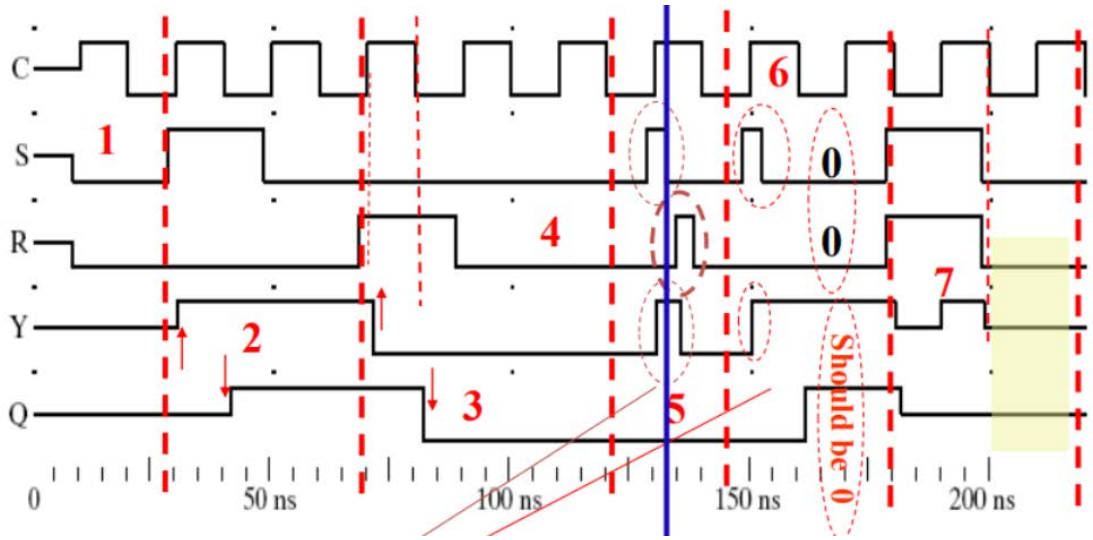
主从触发器：

- 用使能信号控制回路，切断直接通路
- 高电平采样，正脉冲窗口
- 低电平阻断采样，保持



RS 主从触发器存在的问题：

一次性采样：

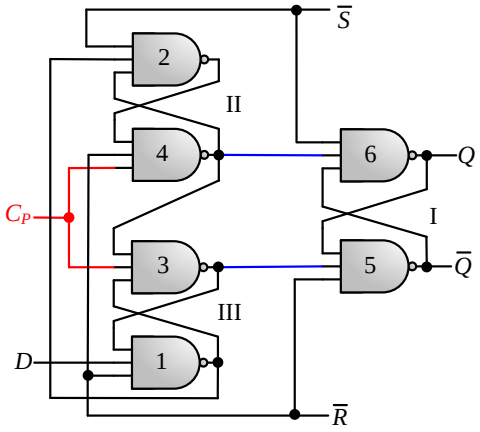


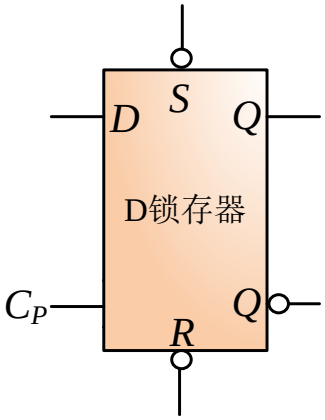

In the pulse arrives before Q=0, at the end of the pulse RS=00, Q should be kept at “0”.

仿真主要代码：

```
initial begin
        R=1;S=1; #50;
        R=1;S=0; #50;
        R=1;S=1; #50;
        R=0;S=1; #50;
        R=1;S=1; #50;
        R=0;S=0; #50;
        R=1;S=1; #50;
end
always begin
        C=0;#20;
        C=1;#20;
end
```

## 4.5 正边沿维持阻塞型 D 触发器




| 异步控制 | | 上升沿触发 | | | |
|---|---|---|---|---|---|
| ***R*** | ***S*** | $C_P$ | ***D*** | ***Q*** | ***Q*** |
| **0** | **1** | × | × | **0** | **1** |
| **1** | **0** | × | × | **1** | **0** |
| **1** | **1** | ↑ | **0** | **0** | **1** |
| **1** | **1** | ↑ | **1** | **1** | **0** |

仿真主要代码:

```
initial begin
        D = 0; #150;
        D = 1; #150;
end
always begin
        C=0; #50;
        C=1; #50;
end
```

# 五 . 实验内容与步骤

## 5.1 实验内容

1. 实现基本 SR 锁存器，验证功能和存在的时序问题

2. 实现门控 SR 锁存器，并验证功能和存在的时序问题

3. 实现 D 锁存器，并验证功能和存在的时序问题

4. 实现 SR 主从触发器，并验证功能和存在的时序问题

5. 实现 D 触发器，并验证功能

## 5.2 实验过程

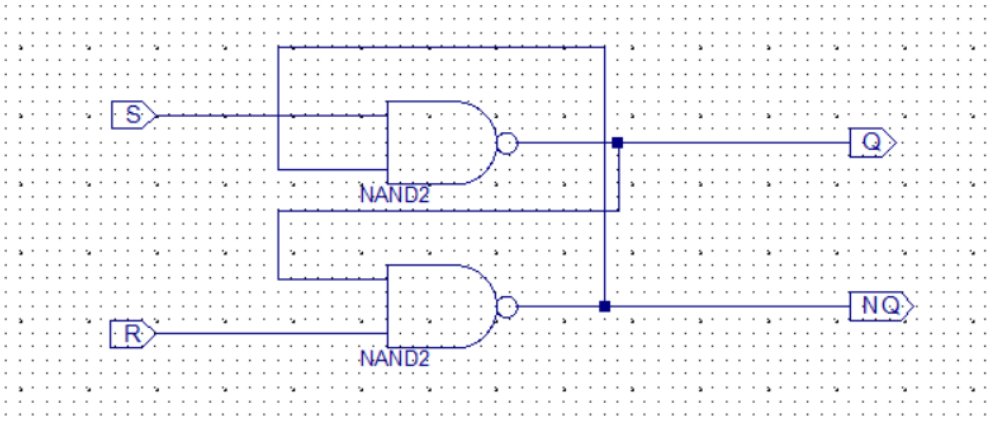
- 基本 SR 锁存器

新建工程 MyLATCHS;

新建源文件 SR_LATCH.sch;

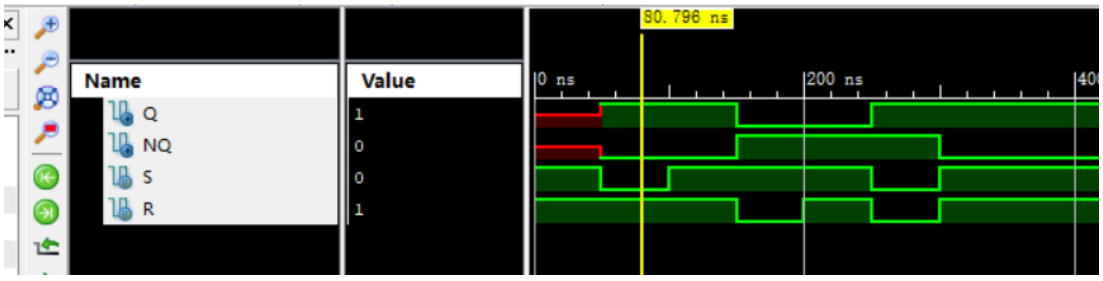
用原理图方式设计;

用 NAND2 实现;




仿真:

```
24  // Initialize Inputs
25      initial begin
26          R=1;S=1; #50;
27          R=1;S=0; #50;
28          R=1;S=1; #50;
29          R=0;S=1; #50;
30          R=1;S=1; #50;
31          R=0;S=0; #50;
32          R=1;S=1; #50;
33      end
```

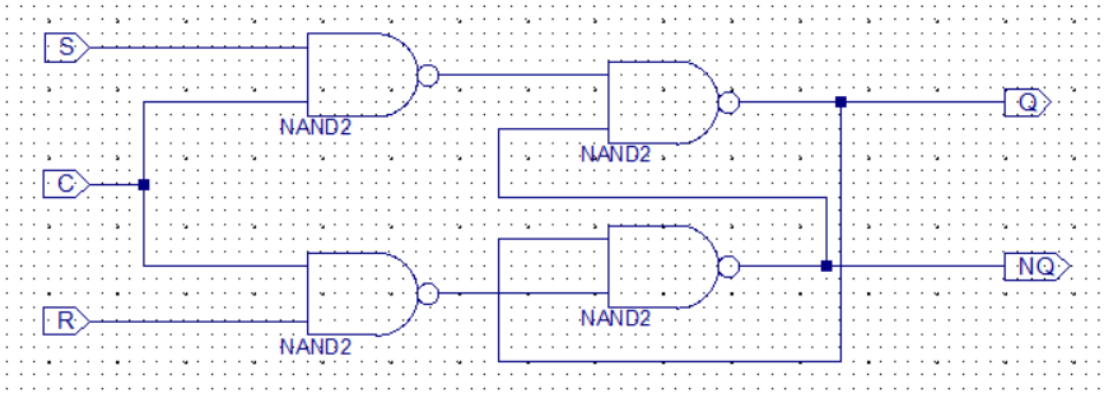


- 门控 SR 锁存器

新建源文件 CSR_LATCH.sch；

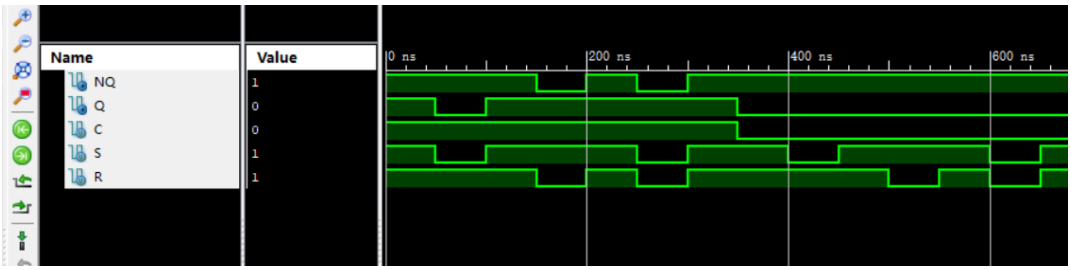
用原理图方式设计；

用 NAND2 实现；




仿真：

```
26  // Initialize Inputs
27      initial begin
28      C=1;R=1;S=1; #50;
29      R=1;S=0; #50;
30      R=1;S=1; #50;
31      R=0;S=1; #50;
32      R=1;S=1; #50;
33      R=0;S=0; #50;
34      R=1;S=1; #50;
35      C=0;R=1;S=1; #50;
36      R=1;S=0; #50;
37      R=1;S=1; #50;
38      R=0;S=1; #50;
39      R=1;S=1; #50;
40      R=0;S=0; #50;
41      R=1;S=1; #50;
42      end
```



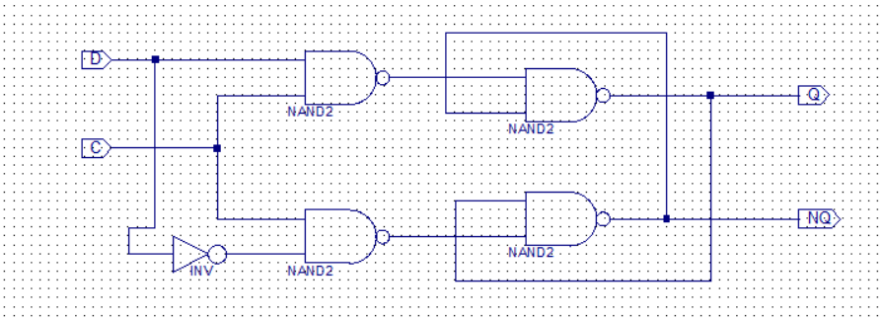

生成自定义符号的 CSR_LATCH.sym；

- D 锁存器

新建源文件 D_LATCH.sch；

用原理图方式设计；

用 NAND2 实现；




仿真：

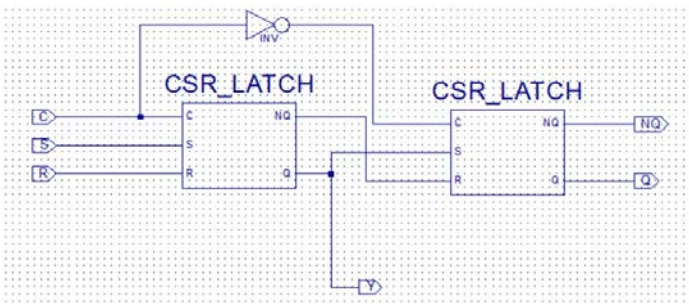
```
24  // Initialize Inputs
25      initial begin
26      C=1;D=1; #50;
27      D=0; #50;
28      C=0;D=1; #50;
29      D=0;#50;
30      C=1;D=1;#50;
31      end
```


- SR 主从触发器

新建源文件 MS_FLIPFLOP.sch；

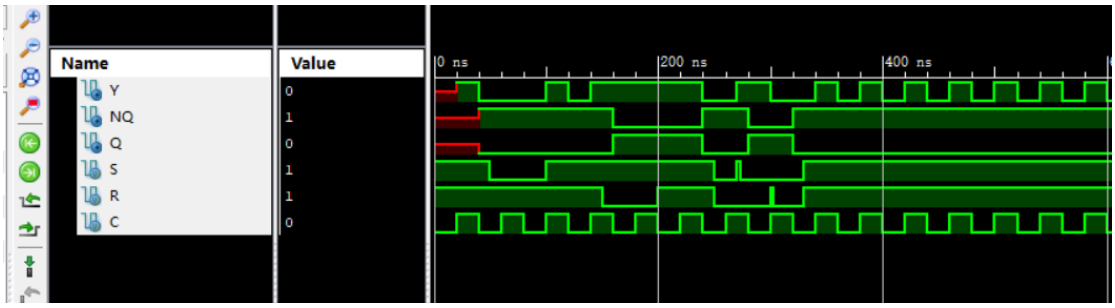
用原理图方式设计；

调用 CSR_LATCH 实现；



仿真（包含空翻）：

```
28  // Initialize Inputs
29     initial begin
30      R=1;S=1; #50;
31      R=1;S=0; #50;
32      R=1;S=1; #50;
33      R=0;S=1; #50;
34      R=1;S=1; #50;
35      R=0;S=0; #20;
36      S=1;#3;S=0;#27;
37      R=1;#3;R=0;#27;
38      R=1;S=1; #50;
39      end
40
41      always begin
42      C=0;#20;
43      C=1;#20;
44      end
```

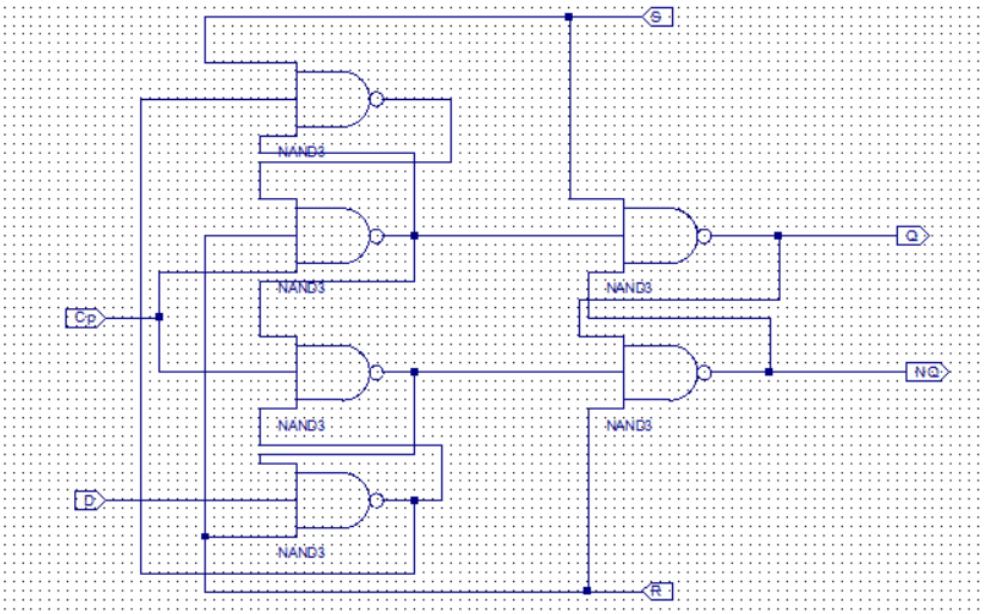



- D 触发器

新建源文件 D_FLIPFLOP.sch；

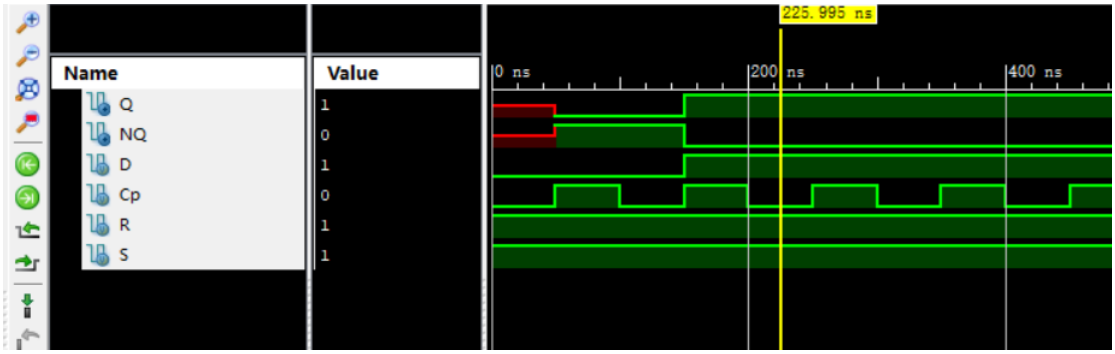
用原理图方式设计；

调用 NAND3 实现；



仿真：

```
28  // Initialize Inputs
29      initial begin
30          S = 1;
31          R = 1;
32          D = 0; #150;
33          D = 1; #150;
34      end
35      always begin
36          Cp=0; #50;
37          Cp=1; #50;
38      end
```




## 六．实验心得与体会

本次实验是数逻第十个实验，与之前的实验不同，这次实验主要的内容是用通过仿真模拟检验设计的正确与否，并加深对元件原理的理解，需要对 ISE 仿真模拟流程以及通过仿真波形判断元件逻辑是否正确比较熟悉，此外，这次试验主要是关于锁存器与触发器，在时序电路中基础的元件，通过实验，我了解了锁存器和触发器一些经典的特性，包括其输入输出的逻辑关系还有其缺点。

实验中的重点是“空翻”现象，刚开始实验时由于对“空翻”的理解不够透彻，不明白“空翻”发生的原因还有情形，所以一直不会让“空翻”体现在仿真波形上，最后在老师的细心讲解下，终于弄懂了，也顺利的做出了带有“空翻”现象的波形，完成了实验。

通过这次实验，我深刻的意识到自己还有很多知识的漏洞以及盲区，需要在实验后加强学习，强化对时序电路这部分内容的理解。

# 实验 11——同步时序电路设计

姓名：　　学号：　　专业：

课程名称：逻辑与计算机设计基础实验　　同组学生姓名：

实验时间：2019-11-20　　实验地点：紫金港东 4-509　　指导老师：洪奇军

## 一．实验目的

1. 掌握典型同步时序电路的工作原理和设计方法
2. 掌握时序电路的激励函数、状态图、状态方程的运用
3. 掌握用 Verilog 进行有限状态机的设计、调试、仿真
4. 掌握用 FPGA 实现时序电路功能

## 二．实验设备与材料

### 2.1 实验设备

| | |
|---|---|
| 装有 Xilinx ISE 14.7 的计算机 | 1 台 |
| SWORD 开发板 | 1 套 |

### 2.2 实验材料

无

## 三．实验任务

任务 1：原理图方式设计 4 位同步二进制计数器

任务 2：以 Verilog 行为描述方式设计 16 位可逆二进制同步计数器

# 四．实验原理

## 4.1 4 位二进制同步计数器

| | 当前状态（现态） | | | | 下一状态（次态） | | | | 触发器激励（输入） | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $Q_A$ | $Q_B$ | $Q_C$ | $Q_D$ | $Q_A^{n+1}$ | $Q_B^{n+1}$ | $Q_C^{n+1}$ | $Q_D^{n+1}$ | $D_A$ | $D_B$ | $D_C$ | $D_D$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 8 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 12 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 13 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

状态变化条件：技术触发，无外部输入

输入激励：满足次态的输入要求，输入方程

状态分配：计数值决定

触发器选择： D 触发器，4 个

$$D_A = Q_A$$

$$D_B = \overline{Q_A}Q_B + Q_A\overline{Q_B} = \overline{Q_A \oplus Q_B}$$

$$D_C = \overline{Q_A}Q_C + \overline{Q_B}Q_C + Q_AQ_B\overline{Q_C}$$

$$= \overline{\overline{(\overline{Q_A} + \overline{Q_B})} \oplus \overline{Q_C}}$$

$$D_D = \overline{Q_A}Q_D + \overline{Q_B}Q_D + \overline{Q_C}Q_D + Q_AQ_BQ_C\overline{Q_D}$$

$$= \overline{\overline{(\overline{Q_A} + \overline{Q_B} + \overline{Q_C})} \oplus \overline{Q_D}}$$

根据 D 触发器原理，在 clk 作用下 Q = D，4 位计数器的 Q 和 D 关系如下表：

| | $Q_A$ | $Q_B$ | $Q_C$ | $Q_D$ | $D_A$ | $D_B$ | $D_C$ | $D_D$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 8 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 12 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 13 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

$D_A$ 卡诺图（$Q_CQ_D$ \ $Q_AQ_B$）

| $Q_CQ_D$ \ $Q_AQ_B$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 1 | | |
| 01 | 1 | 1 | | |
| 11 | 1 | 1 | | |
| 10 | 1 | 1 | | |

$D_B$ 卡诺图（$Q_CQ_D$ \ $Q_AQ_B$）

| $Q_CQ_D$ \ $Q_AQ_B$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | 1 | | 1 |
| 01 | | 1 | | 1 |
| 11 | | 1 | | 1 |
| 10 | | 1 | | 1 |

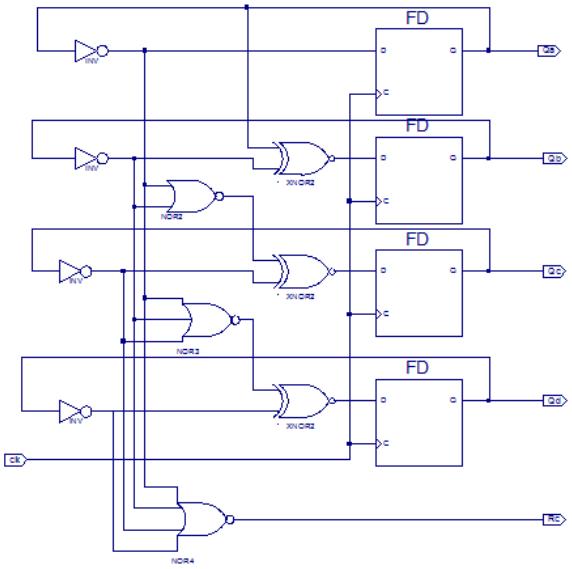
$D_C$ 卡诺图（$Q_CQ_D$ \ $Q_AQ_B$）

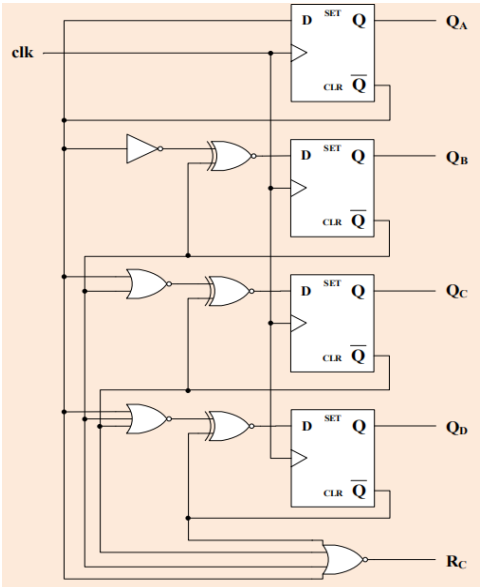
| $Q_CQ_D$ \ $Q_AQ_B$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | 1 | |
| 01 | | | 1 | |
| 11 | 1 | 1 | | 1 |
| 10 | 1 | 1 | | 1 |

$D_D$ 卡诺图（$Q_CQ_D$ \ $Q_AQ_B$）

| $Q_CQ_D$ \ $Q_AQ_B$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | | |
| 01 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 | | 1 |
| 10 | | | 1 | |

$$R_C = \overline{\overline{Q_A} + \overline{Q_B} + \overline{Q_C} + \overline{Q_D}}$$



- 模块结构描述

结构描述：与逻辑电路结构对应的描述

利用模块调用描述系统或部件:

1）调用内置门原语（在门级结构描述）

2）调用开关级原语（在晶体管开关级结构描述）

3）调用用户定义（UDP）的原语（在门级结构描述）

4）模块实例（创建层次结构结构描述）

与原理图对应的描述，加深时序电路的感性认识

实验中需要使用的主要门级单元模块（原语）：

非门:

名称：INV　　接口：I, O

示例：INV　　NotA(.I(a), .O(na));

二输入或非门:

名称：NOR2　　接口：IO, II, O

示例：NOR2　　G1(.I0(a), .I1(b), .O(na-b));

本实验用到的原语汇总表:

| 逻辑门 | 名称 | 接口 | 实例 |
|---|---|---|---|
| 非门 | INV | I, O | INV NotA(.I(a), .O(na)); |
| 二输入或非门 | NOR2 | I0, I1, O | NOR2 G1(.I0(a), .I1(b), .O(c)); |
| 三输入或非门 | NOR3 | I0, I1, I2, O | |
| 四输入或非门 | NOR4 | I0, I1, I2, I3, O | |
| 二输入异或非门 | XNOR2 | I0, I1, O | XNOR2 G1(.I0(a), .I1(b), .O(c)); |
| 二输入与门 | AND2 | I0, I1, O | |
| 二输入或门 | OR2 | I0, I1, O | |
| 二输入与非门 | NAND2 | I0, I1, O | |
| D 触发器 | FD | C, clock<br>D, data in<br>Q, data out | FD FFDA(.C(clk), .D(Da), .Q(Qa));<br>defparam FFDA.INIT = 1'b0;<br>//定义 D 触发器的初值为 0 |

四位同步计数器结构化描述：原语描述

```
module counter_4bit(clk, Qa, Qb, Qc, Qd, Rc);
.....                   //端口及变量定义
FD                      FFDA (.C(clk),.D(Da),.Q(Qa)),
                        FFDB (.C(clk),.D(Db),.Q(Qb)),        也可调用实验九设计的D触发器
                        FFDC (.C(clk),.D(Dc),.Q(Qc)),
                        FFDD (.C(clk),.D(Dd),.Q(Qd));
defparam FFDA.INIT = 1'b0;              // define initial value of the D type Flip-Flop
defparam FFDB.INIT = 1'b0;
defparam FFDC.INIT = 1'b0;
defparam FFDD.INIT = 1'b0;
INV       GQa (.I(Qa), .O(nQa)),        //4个非门，  FD实例没有反向输出端
          GQb (.I(Qb), .O(nQb)),
          GQc (.I(Qc), .O(nQc)),
          GQd (.I(Qd), .O(nQd));
assign Da = nQa;                        //赋值描述

XNOR2     ODb (.I0(Qa), .I1(nQb), .O(Db)),                          //2输入异或非
          ODc (.I0(Nor_nQa_nQb), .I1(nQc), .O(Dc)),
          ODd (.I0(Nor_nQa_nQb_nQc), .I1(nQd), .O(Dd));
NOR4      ORc (.I0(nQa), .I1(nQb), .I2(nQc), .I3(nQd), .O(Rc));     //4输入或非门
NOR2      G1 (.I0(nQa), .I1(nQb), .O(Nor_nQa_nQb));                 //2输入或非门
NOR3      G2(.I0(nQa), .I1(nQb), .I2(nQc), .O(Nor_nQa_nQb_nQc));
endmodule
```

四位二进制同步计数器：

```
module counter_4bit(clk, Qa, Qb, Qc, Qd, Rc);
    input wire clk;
    output wire Qa, Qb, Qc, Qd, Rc;
    wire Nor_nQa_nQb, Nor_nQa_nQb_nQc;
    //FD原程序的输入输出变量名：C,D,Q
    FD FD_A(.C(clk), .D(Da), .Q(Qa));
    FD FD_B(.C(clk), .D(Db), .Q(Qb));
    FD FD_C(.C(clk), .D(Dc), .Q(Qc));
    FD FD_D(.C(clk), .D(Dd), .Q(Qd));
    defparam FD_A.INIT = 1'b0, FD_B.INIT = 1'b0;//对应parameter
    defparam FD_C.INIT = 1'b0, FD_D.INIT = 1'b0;

    INV nQa_L(.I(Qa), .O(nQa)), nQb_L(.I(Qb), .O(nQb));
    INV nQc_L(.I(Qc), .O(nQc)), nQd_L(.I(Qd), .O(nQd));
    assign Da = nQa;

    XNOR2 Db_L(.I0(Qa), .I1(nQb), .O(Db));
    XNOR2 Dc_L(.I0(Nor_nQa_nQb), .I1(nQc), .O(Dc));
    XNOR2 Dd_L(.I0(Nor_nQa_nQb_nQc), .I1(nQd), .O(Dd));
    NOR4 Rc_L(.I0(nQa), .I1(nQb), .I2(nQc), .I3(nQd), .O(Rc));
    NOR2 Nor_nQa_nQb_L(.I0(nQa), .I1(nQb), .O(Nor_nQa_nQb));
    NOR3 Nor_nQa_nQb_nQc_L(.I0(nQa), .I1(nQb), .I2(nQc), .O(Nor_nQa_nQb_nQc));
endmodule
```

四位同步计数器结构化描述：门级描述

行为化结构描述

```
module counter_4bit(clk, rst, Qa, Qb, Qc, Qd, Rc);
.....                    //端口定义
wire Da, Db, Dc, Dd, nQa, nQb, nQc, nQc, Rc;
reg  Qa, Qb, Qc, Qd;

    assign Da = nQa;
    assign Db = ~( Qa^nQb);
    assign Dc = ~( (~(nQa | nQb)) ^ nQc);
    assign Dd = ~((~(nQa | nQb | nQc)) ^ nQd);
    assign Rc = ~(nQa | nQb | nQc | nQd));

   always @ (posedge clk)
          if (rst) {Qa,Qb,Qc,Qd} <= 4'b0000;
          else begin
               Qa <= Da;
               Qb <= Db;
               Qc <= Dc;
               Qd <= Dd;
          end
endmodule
```

$$D_A = Q_A$$

$$D_B = \overline{Q}_A Q_B + Q_A \overline{Q}_B = \overline{Q_A \oplus Q_B}$$

$$D_C = \overline{Q}_A Q_C + \overline{Q}_B Q_C + Q_A Q_B \overline{Q}_C$$

$$= \overline{(\overline{Q}_A + \overline{Q}_B) \oplus \overline{Q}_C}$$

$$D_D = \overline{Q}_A Q_D + \overline{Q}_B Q_D + \overline{Q}_C Q_D + Q_A Q_B Q_C \overline{Q}_D$$

$$= \overline{(\overline{Q}_A + \overline{Q}_B + \overline{Q}_C) \oplus \overline{Q}_D}$$

四位同步二进制双向计数器激励方程：

激励函数与结构化行为描述

行为描述简单，结构化有利于学习

S=1 时，正向计数，各触发器逻辑表达式同前

S=0 时，反向计数，各触发器逻辑表达式如下：

$$D_A = \overline{Q}_A$$

$$D_B = \overline{S}(\overline{Q}_A \oplus \overline{Q}_B) + S(\overline{Q}_A \oplus \overline{Q}_B) = \overline{S \oplus Q_A \oplus Q_B}$$

$$D_C = \overline{S}[(Q_A Q_B) \oplus \overline{Q}_C] + S[(\overline{Q}_A + \overline{Q}_B) \oplus \overline{Q}_C]$$

$$= [\overline{S} Q_A Q_B + S(\overline{Q}_A + \overline{Q}_B)] \oplus \overline{Q}_C$$

$$= [\overline{S}(Q_A + Q_B) + S(\overline{Q}_A + \overline{Q}_B)] \oplus \overline{Q}_C$$

$$D_D = \overline{S}[(Q_A Q_B Q_C) \oplus \overline{Q}_D] + S[(\overline{Q}_A + \overline{Q}_B + \overline{Q}_C) \oplus \overline{Q}_D]$$

$$= [\overline{S} Q_A Q_B Q_C + S(\overline{Q}_A + \overline{Q}_B + \overline{Q}_C)] \oplus \overline{Q}_D$$

$$= [\overline{S}(Q_A + Q_B + Q_C) + S(\overline{Q}_A + \overline{Q}_B + \overline{Q}_C)] \oplus \overline{Q}_D$$

$$R = \overline{S Q_A Q_B Q_C Q_D} + S Q_A Q_B Q_C Q_D$$

```
25  module counter_4rev(clk, rst, s, cnt, Rc);
26      ......
27      wire Da, Db, Dc, Dd, nQa, nQb, nQc, Rc;
28      reg Qd, Qc, Qb, Qa;
29
30      assign Da = ?;
31      assign Db = ?;
32      assign Dc = ?;
33      assign Dd = ?;
34      assign Rc = ?;
35
36      assign cnt = {Qd, Qc, Qb, Qa};
37
38      always @(posedge clk) begin
39          if (rst) begin
40              cnt<=4'b0000;
41          end else begin
42              {Qd, Qc, Qb, Qa}<={Dd, Dc, Db, Da};
43          end
44      end
45  endmodule
```

激励代码：

```
initial forever begin
        clk = 1'b0; #100;
        clk = 1'b1; #100;
end
always begin
        #100 clk = 0;
        #100 clk = 1;
end
```

## 4.2 可逆二进制同步计数器

可逆二进制同步计数器通过控制端 S 选择正向或者反向计数

S = 1 时，正向计数，各触发器逻辑表达式同前面

S = 0 时，反向计数，各触发器逻辑表达式如下式

$$D_A = \overline{Q_A}$$

$$D_B = \overline{S}(\overline{Q_A} \oplus \overline{Q_B}) + S(\overline{\overline{Q_A}} \oplus \overline{Q_B}) = \overline{S \oplus \overline{Q_A} \oplus \overline{Q_B}}$$

$$D_C = \overline{S}[\overline{(Q_A Q_B)} \oplus \overline{Q_C}] + S[\overline{(\overline{Q_A} + \overline{Q_B})} \oplus \overline{Q_C}] = \overline{[\overline{S}\,\overline{Q_A Q_B} + S(\overline{Q_A} + \overline{Q_B})] \oplus \overline{Q_C}}$$

$$= \overline{[\overline{S}(Q_A + Q_B) + S(\overline{Q_A} + \overline{Q_B})] \oplus \overline{Q_C}}$$

$$D_D = \overline{S}[\overline{(Q_A Q_B Q_C)} \oplus \overline{Q_D}] + S[\overline{(\overline{Q_A} + \overline{Q_B} + \overline{Q_C})} \oplus \overline{Q_D}] = \overline{[\overline{S}\,\overline{Q_A Q_B Q_C} + S(\overline{Q_A} + \overline{Q_B} + \overline{Q_C})] \oplus \overline{Q_D}}$$

$$= \overline{[\overline{S}(Q_A + Q_B + Q_C) + S(\overline{Q_A} + \overline{Q_B} + \overline{Q_C})] \oplus \overline{Q_D}}$$

$$R = \overline{S}\,\overline{Q_A}\,\overline{Q_B}\,\overline{Q_C}\,\overline{Q_D} + S Q_A Q_B Q_C Q_D \quad \text{（进位、借位输出）}$$

可逆二进制 4 位同步计数器的行为描述：

```
module counter_4bit_rev(clk, s, cnt, Rc);
    input wire clk, s;
    output reg [3:0] cnt;
    output wire Rc;
    Initial begin cnt = 0;
    assign Rc = (~s & (~|cnt)) | (s & (&cnt));
    always @ (posedge clk) begin
        if (s) cnt <= cnt + 1;
        //always中被赋值的变量cnt一定为reg量型。
        else cnt <= cnt - 1;
    end
endmodule
```

32 位同步二进制可逆计数器：行为描述

```
module conter_32bit_rev(
    input wire clk, s,
    output reg[31:0] cnt,//32位计数器
    output wire Rc);

    assign Rc = (~s&(~|cnt))|(s&(&cnt));
    always @(posedge clk) begin
        if(s) cnt<=cnt+1;//正向计数
        else cnt<=cnt-1;//反向计数
    end
endmodule
```

### 4.3 分频器设计

100MHz 信号通过 50,000,000 次分频

得到 1Hz 的秒脉冲方波，作为计数器的脉冲输入

```
module counter_1s(clk, clk_1s);
    input wire clk;
    output reg clk_1s;
    reg [31:0] cnt;
    always @ (posedge clk) begin
        if (cnt < 50_000_000) begin
            cnt <= cnt + 1;
        end else begin
            cnt <= 0;
            clk_1s <= ~clk_1s;
         end
    end
endmodule
```

## 五．实验内容和结果

实验任务：

1. 原理图方式设计 4 位同步二进制计数器
2. 以 Verilog 行为描述方式设计 16 位可逆二进制同步计数器

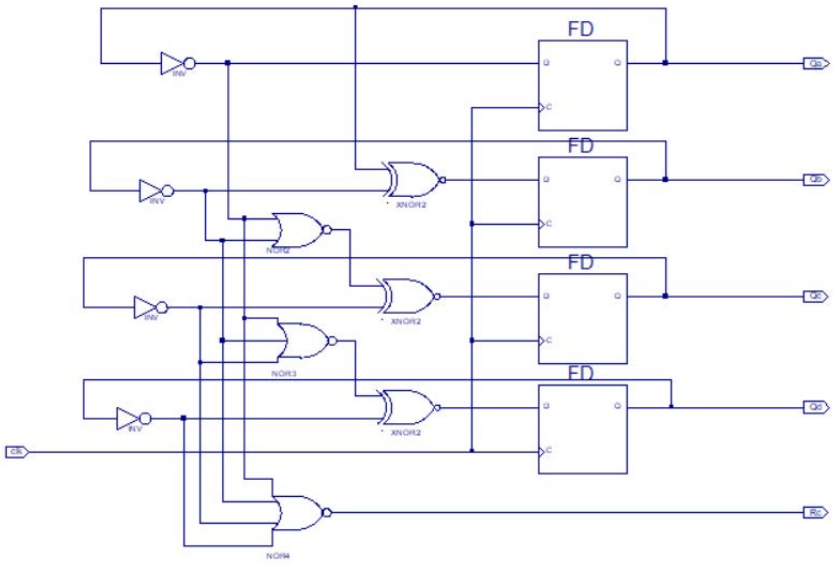
## 5.1 设计 4 位同步二进制计数器

新建工程，工程名称用 MyCounter；

Top Level Source Type 用 HDL；

新建源文件，类型是 Schematic，文件名称用 Counter4b；

原理图方式进行设计；

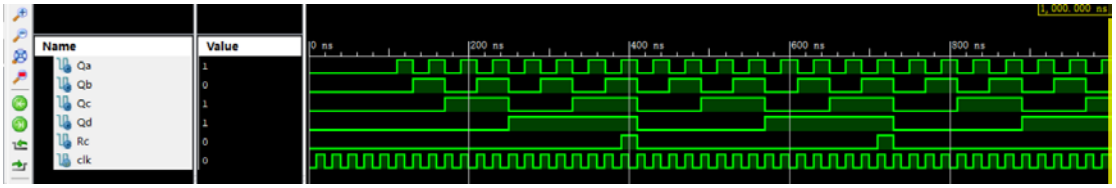



进行波形仿真；

激励代码：

```
28  // Initialize Inputs
29      initial forever begin
30      clk = 1'b0; #10;
31      clk = 1'b1; #10;
32      end
```

仿真波形：




新建源文件，用作时钟；

类型是 Verilog，文件名称用 clk_1s；

Verilog 行为描述；

```
module clk_1s(clk, clk_1s);
    input wire clk;
    output reg clk_1s;
    reg [31:0] cnt;
    always @ (posedge clk) begin
        if (cnt < 50_000_000) begin
            cnt <= cnt + 1;
        end else begin
            cnt <= 0;
            clk_1s <= ~clk_1s;
        end
    end
endmodule
```

新建源文件，类型是 Verilog，文件名称用 Top；

右键设为“Set as Top Module”；

输入为 clk（100MHZ）时钟，每秒自增 1；

显示在 1 位数码管上；

AN0, AN1, AN2, AN3 中选一个；

其他数码管位可以显示 0 或者不显示，最好不显示；

Rc 显示在 LED 灯上；

```
module top(
    input wire clk,
    output wire[3:0] AN,
    output wire[7:0] segments,
    output wire Rc);

    wire clk1s;
    wire[3:0] num;
    clk_1s m2(.clk(clk), .clk_1s(clk1s));
    Counter4b m1(
        .clk(clk),
        .Qa(num[0]),
        .Qb(num[1]),
        .Qc(num[2]),
        .Qd(num[3]),
        .Rc(Rc)
    );
    disp_num m2(
        .clk(clk),
        .HEXS({12'b0,num[3:0]}),
        .LES(4'b0000),
        .points(4'b1111),
        .RST(1'b0),
        .AN(AN),
        .Segment(SEGMENT)
    );
endmodule
```

## 5.2 设计 16 位可逆同步二进制计数器

新建工程，工程名称用 myRevCounter；

Top Level Source Type 用 HDL；

新建源文件，类型是 Verilog，文件名称用 RevCounter；

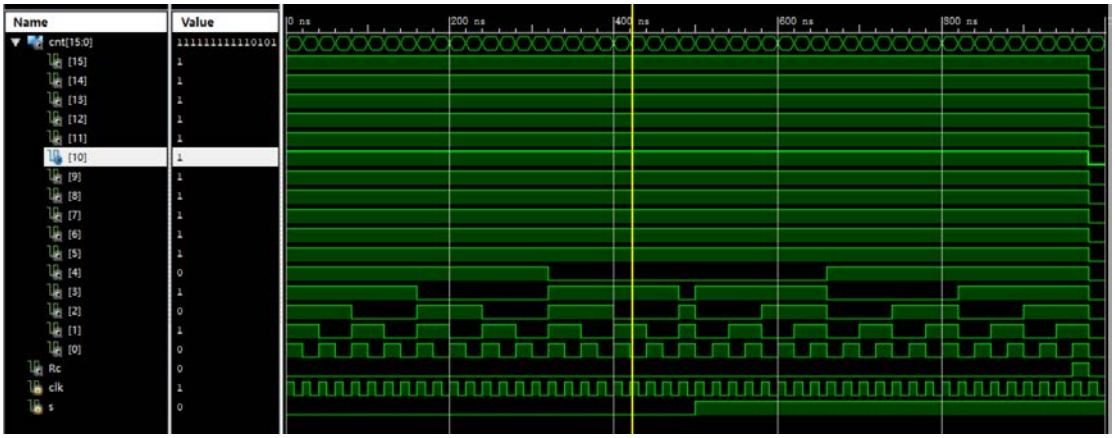
结构化描述方式进行设计；

```
21  module RevCounter(
22      input wire clk,
23      input wire s,
24      output reg[15:0] cnt,
25      output wire Rc);
26
27      initial begin
28          cnt=0;
29      end
30      assign Rc = (~s & (~|cnt)) | (s & (&cnt));
31      always @ (posedge clk) begin
32          if (s)  cnt <= cnt + 1;
33          //always中被赋值的变量cnt一定为reg类型
34          else    cnt <= cnt - 1;
35      end
36  endmodule
```

激励代码：

```
42      // Initialize Inputs
43      initial begin
44          clk = 0;
45          s = 0;
46          #500;
47          s = 1;
48      end
49
50      always begin
51          clk=1;#10;
52          clk=0;#10;
53      end
```

仿真波形：



新建源文件，设计 100ms 时钟；

类型是 Verilog，文件名称用 clk_100ms；

Verilog 行为描述；

```
module clk_100ms(
    input wire clk,
    output wire clk_100ms);

    reg [31:0] cnt;
    always @ (posedge clk) begin
        if (cnt < 5_000_00) begin
            cnt <= cnt + 1;
        end else begin
            cnt <= 0;
            clk_100ms <= ~clk_100ms;
        end
    end
endmodule
```

新建源文件，类型是 Verilog，文件名称用 Top；

右键设为“Set as Top Module”；

输入为 clk（100MHZ）时钟；

用 SW[0]控制自增/自减 1（每 0.1 秒）；

显示在 4 位数码管上，Rc 状态用 LED 灯来显示；

```
module top(input wire clk,
    input wire SW,
    output wire[3:0] AN,
    output wire[7:0] SEGMENT,
    output wire[7:0] LED
    //output ledclk,output ledsout,output ledclrn,output LEDEN//主板LED灯显示需要用到。
    );

    wire [15:0] cnt;

    RevCounter m0(.clk(clk_100ms),.s(SW),.cnt(cnt),.Rc(LED[0]));

    clk_100ms m1(.clk(clk),.clk_100ms(clk_100ms));

    disp_num m2(.clk(clk),.HEXS(cnt),.LES(4'b0000),.points(4'b1111),.RST(1'b0),.AN(AN),.Segment(SEGMENT));

    assign LED[7:1] = 7'b111_1111;

endmodule
```

引脚约束

```
net "clk" loc=AC18 | iostandard=LVCMOS18;
net "SW" loc=AE13 | iostandard=LVCMOS15;

net"SEGMENT[0]"loc=AB22 | IOSTANDARD=LVCMOS33;
net"SEGMENT[1]"loc=AD24 | IOSTANDARD=LVCMOS33;
net"SEGMENT[2]"loc=AD23 | IOSTANDARD=LVCMOS33;
net"SEGMENT[3]"loc=Y21  | IOSTANDARD=LVCMOS33;
net"SEGMENT[4]"loc=W20  | IOSTANDARD=LVCMOS33;
net"SEGMENT[5]"loc=AC24 | IOSTANDARD=LVCMOS33;
net"SEGMENT[6]"loc=AC23 | IOSTANDARD=LVCMOS33;
net"SEGMENT[7]"loc=AA22 | IOSTANDARD=LVCMOS33;
```

```
net"AN[0]"loc=AC21 |IOSTANDARD=LVCMOS33;
net"AN[1]"loc=AD21 |IOSTANDARD=LVCMOS33;
net"AN[2]"loc=AB21 |IOSTANDARD=LVCMOS33;
net"AN[3]"loc=AC22 |IOSTANDARD=LVCMOS33;
net"LED[0]"loc=W23  | IOSTANDARD=LVCMOS33;
net"LED[1]"loc=AB26 | IOSTANDARD=LVCMOS33;
net"LED[2]"loc=Y25  | IOSTANDARD=LVCMOS33;
net"LED[3]"loc=AA23 | IOSTANDARD=LVCMOS33;
net"LED[4]"loc=Y23  | IOSTANDARD=LVCMOS33;
net"LED[5]"loc=Y22  | IOSTANDARD=LVCMOS33;
net"LED[6]"loc=AE21 | IOSTANDARD=LVCMOS33;
net"LED[7]"loc=AF24 | IOSTANDARD=LVCMOS33;
```

### 5.3 实验结果

Synthesis-XST;

Implement Design;

Generate Programming File;

下载到 sword 板;

可以看到数码管上的数字随时间增大,

拨动控制开关后，数码管上的数字随时间减小,

当经过零点时，LED 灯会亮;

## 六．实验心得与体会

本次实验为数逻第十一个实验，主要内容为设计同步时序电路，在任务一中，用原理图方式设计四位同步二进制计数器，在任务二中，就需要用 Verilog 代码设计十六位可逆二进制同步计数器，对 Verilog 代码能力有一定的要求。

实验中体会较深的是，用行为描述方式写 Verilog 代码与用结构化方式写 Verilog 代码的区别：行为描述方式写起来比较简单，但需要对电路的逻辑关系有深刻的理解，而结构化方式直接对电路的结构进行复刻，更为直接的展现了电路的细节，更有利于学习，还有就是原理图方式相对代码方式设计，虽然更为简单直接，更适合初学者，但对于较为复杂的电路，用原理图方式就比较繁琐和低效，而 Verilog 代码的优势就体现了出来，因才还需加强对 Verilog 代码的学习。

本次实验也让我加深了同步时序电路的理解，以及对寄存器在电路中应用的认识，希望下次实验可以做的更好。