# 实验九报告
# （加法器、加减法器和ALU基本原理与设计）

姓名：蒋仕彪　学号：3170102587　专业：求是科学班（计算机）1701

课程名称：逻辑与计算机设计基础实验

实验时间：2018-11-12

# 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、实验内容和原理

## 2.1 实验内容：

- 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

## 2.2 实验原理：

### 2.2.1 1位全加器

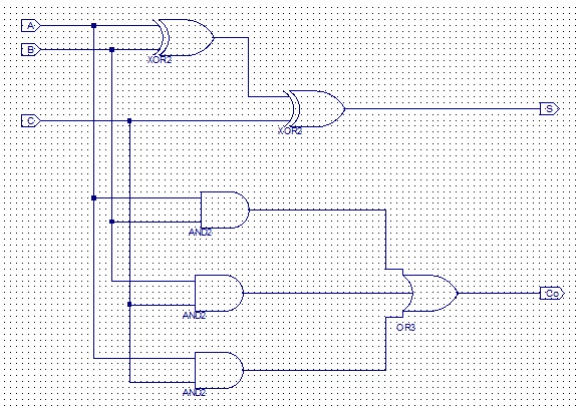
- 三个输入位：数据位 $A_i$ 和 $B_i$，低位进位输入 $C_i$
- 二个输出位：全加和 $S_i$，进位输出 $C_{i+1}$

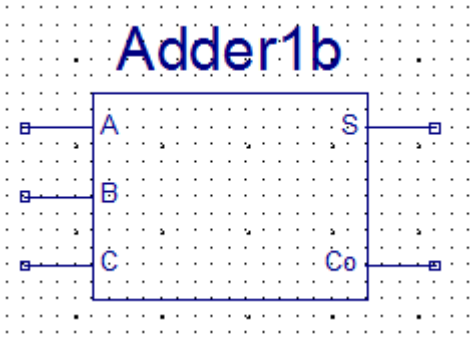
| $A_i$ | $B_i$ | $C_i$ | $S_i$ | $C_{i+1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

$$S_i = A_i \otimes B_i \otimes C_i$$

$$C_{i+1} = A_iB_i + B_iC_i + C_iA_i$$

- 根据一位全加器的输入输出关系，得到电路图

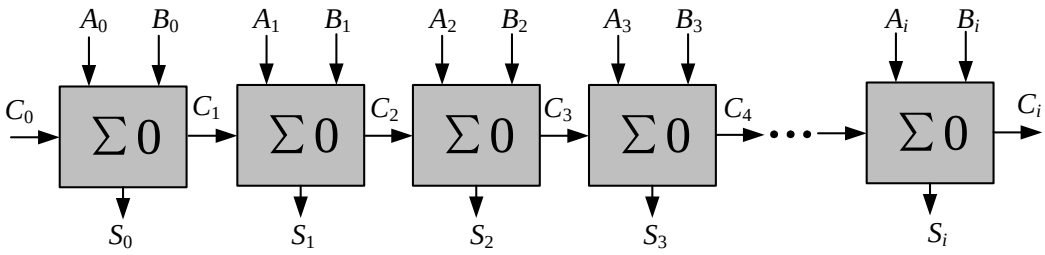



```
module adder_1bit(
    input wire a, b, ci,
    output wire s, co);
    and m0(c1,a,b);
    and m1(c2,b,ci);
    and m2(c3,a,ci);
    xor m3(s1,a,b);
    xor m4(s,s1,ci);
    or  m5(co,c1,c2,c3);
endmodule
```

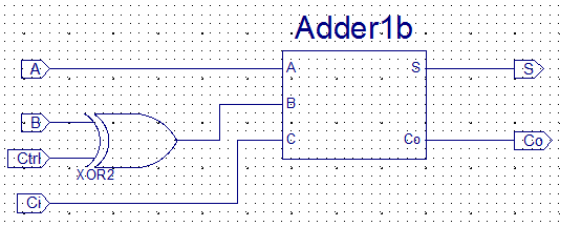
## 2.2.2 多位串行进位加法器

- 由一位全加器将进位串接构成。
- 低位进位$C_0$为0，$C_i$为高位进位输出。



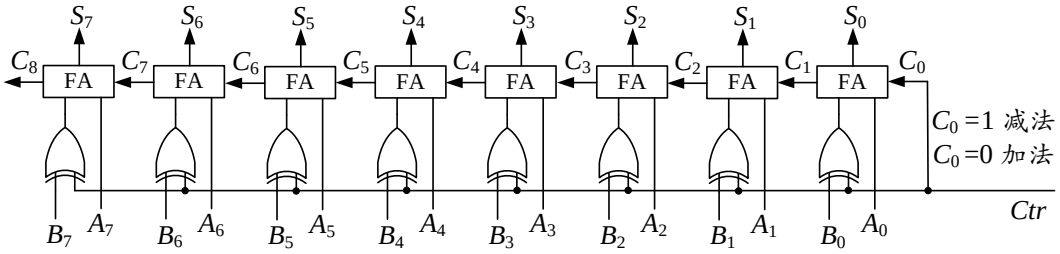

## 2.2.3 1位加减法器

- 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$C_0$为1

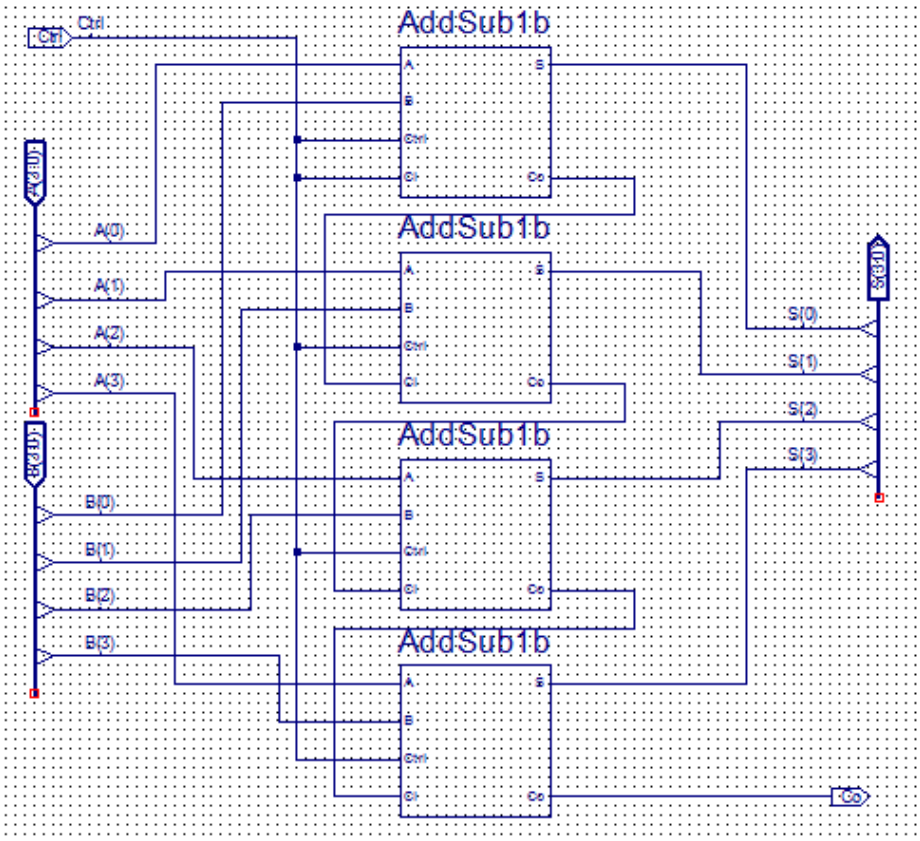


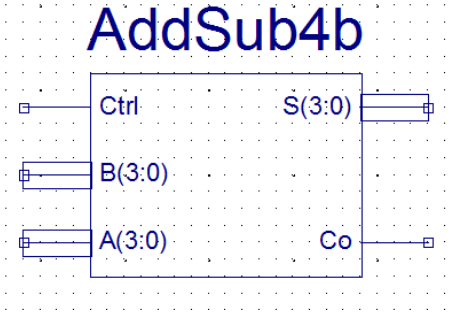
## 2.2.4 多位串行进位全减器

- 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$C_0$为1




## 2.2.5 4位加减法器



## 2.2.6 8位加减法器代码实现

### 8位加减器硬件描述

- Ctr控制加法输入
  - Ctr=0，做加法
  - Ctr=1，做减法，同时控制C0=1

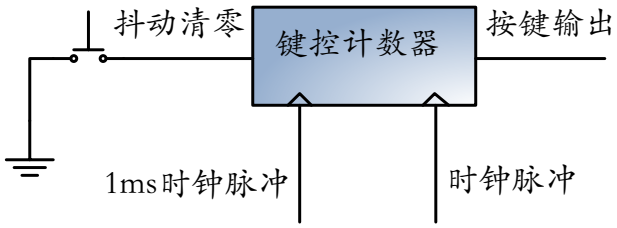
```
module add_sub_8bits(A, B, Ctr, S, Co);
.....                          //端口及变量定义
  assign Bo={Ctr,Ctr,Ctr,Ctr,Ctr,Ctr,Ctr,Ctr}^B;          // = {8{Ctr}} ^ B
  assign Coo=Co^Ctr;
  adder_1bit        A1(.a(A[1]),.b(Bo[1]),.ci(Ctr),.sum(S[1]),.co(Ctemp[1])),
                    A2(A[2],Bo[2],Ctemp[1],S[2],Ctemp[2]),
                    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
```

浙江大学 Zhejiang University 计算机学院 系统结构与系统软件实验室

## 2.2.7 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

## 2.2.8 辅助模块：时钟计数分频器

- 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

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

# 三、主要仪器设备

1. 装有 Xilinx ISE 14.7 的计算机　　1 台
2. SWORD 开发板　　1 套

# 四、操作方法与实验步骤

## 4.1 4位加减法器设计

- 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。
- 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

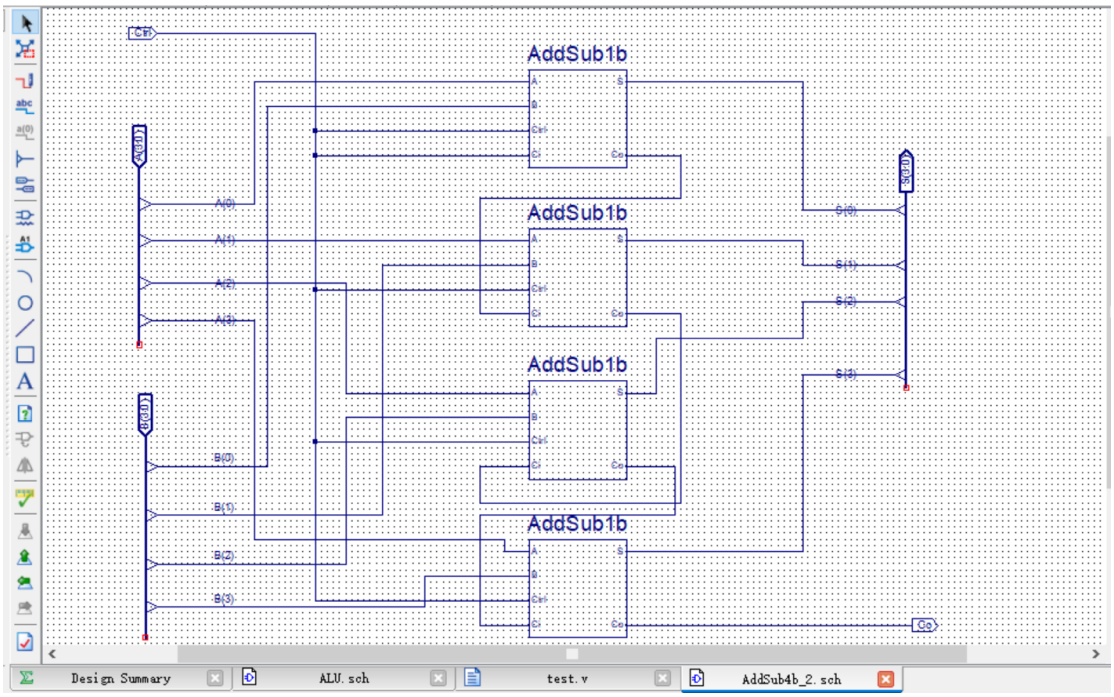
## 4.2 ALU 设计

- 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
- 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
- UCF 引脚定义
  - 输入
    - 时钟：clk
    - 按键控制输入：sw[0]控制 num[3:0]，sw[1]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

# 五、实验结果与分析

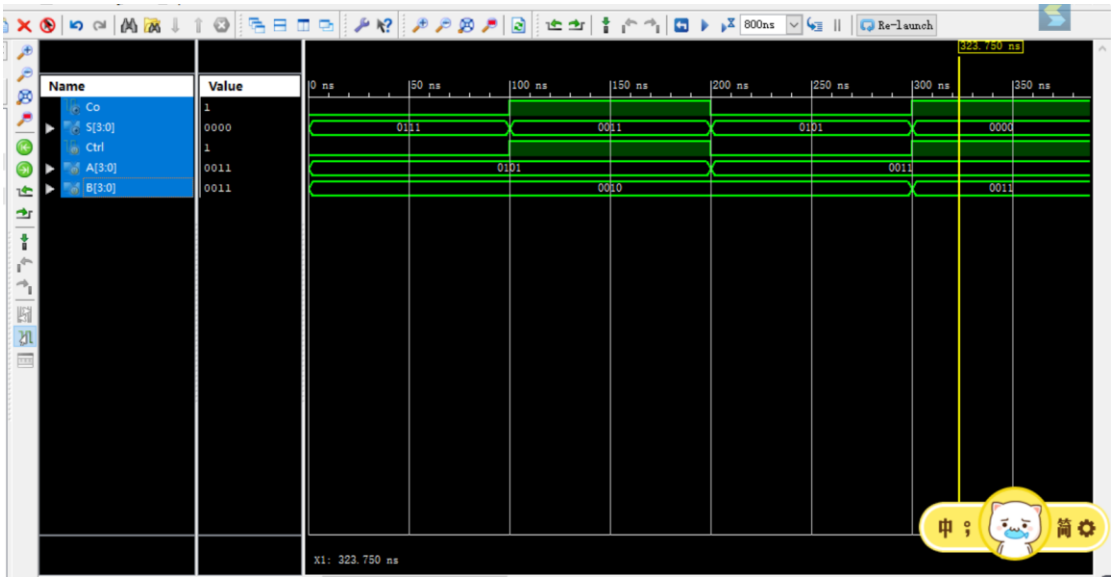
## 5.1 4位加减法器设计




- 以上为AddSub4b原理图的设计

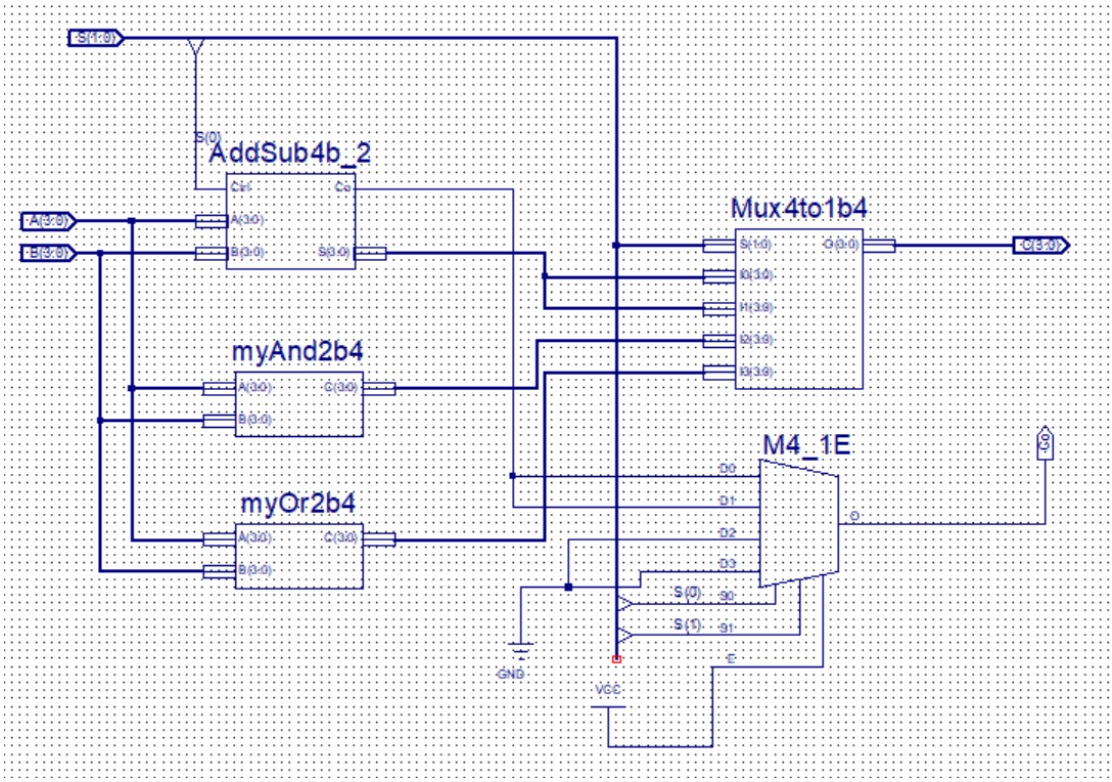
```
13      wire Co;
14      wire [3:0] S;
15
16  // Bidirs
17
18  // Instantiate the UUT
19      AddSub4b UUT (
20          .Ctrl(Ctrl),
21          .A(A),
22          .B(B),
23          .Co(Co),
24          .S(S)
25      );
26  // Initialize Inputs
27      initial begin
28          Ctrl = 0;
29          A = 5;
30          B = 2;
31          #100;
32          Ctrl = 1;
33          A = 5;
34          B = 2;
35          #100
36          Ctrl = 0;
37          A = 3;
38          A = 3;
39          #100
40          Ctrl = 1;
41          A = 3;
42          B = 3;
43      end
44  endmodule
45
```

◘ 以上为AddSub4b的仿真代码




◘ 以上为AddSub4b的仿真结果

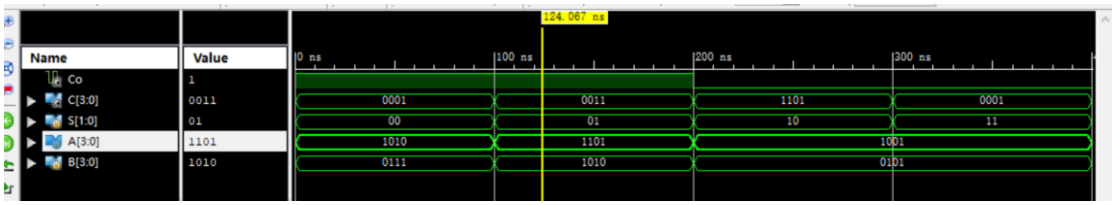
## 5.2 ALU 设计




◘ 以上为ALU原理图设计

```
26  // Initialize Inputs
27     initial begin
28        S=2'b00;
29        A=4'b1010;
30        B=4'b0111;
31
32        #100;
33        S=2'b01;
34        A=4'b1101;
35        B=4'b1010;
36
37        #100;
38        S=2'b10;
39        A=4'b1001;
40        B=4'b0101;
41
42        #100;
43        S=2'b11;
44        A=4'b1001;
45        B=4'b0101;
46
47     end
48  endmodule
49
```

◘ 以上为ALU仿真代码




◘ 以上为ALU仿真结果

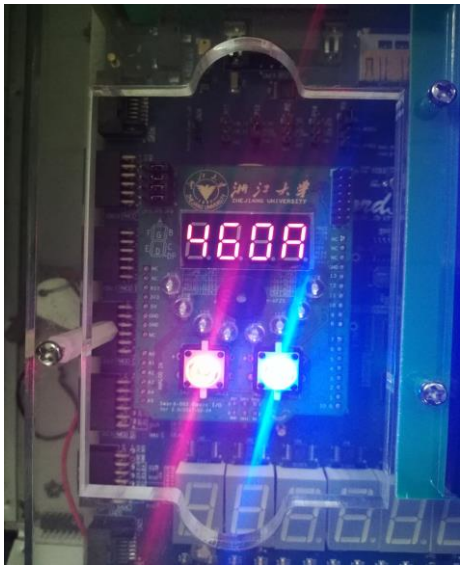
```
module top(
    input wire clk,
    input wire [3:0]SW,
    input wire [1:0]SW2,
    output wire [3:0]AN,
    output wire [7:0]SEGMENT
    );

    wire [7:0] num;
    wire [1:0] btn_out;
    wire [3:0] C;
    wire Co;
    wire [31:0] clk_div;
    pbdebounce m0(clk_div[17],SW[0],btn_out[0]);
    pbdebounce m1(clk_div[17],SW[1],btn_out[1]);
    clkdiv m2(clk,0,clk_div);

    CreateNumber(btn_out, SW[3:2], num[7:0]);

    ALU m5(.A(num[3:0]), .B(num[7:4]), .S(SW2), .C(C), .Co(Co));
    disnum m6( clk,{num[7:0],3'b0,Co,C},4'b0,4'b0,1'b0,AN,SEGMENT );

endmodule
```

◘ 以上为Top.v代码

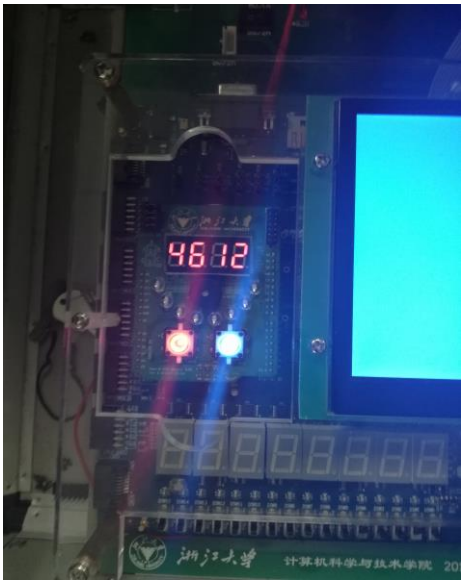
```
NET "SW[0]"      LOC = AA10 | IOSTANDARD = LVCMOS15;
NET "SW[1]"      LOC = AB10 | IOSTANDARD = LVCMOS15;
NET "SW[2]"      LOC = AA13 | IOSTANDARD = LVCMOS15;
NET "SW[3]"      LOC = AA12 | IOSTANDARD = LVCMOS15;
NET "SW2[0]"     LOC = Y13  | IOSTANDARD = LVCMOS15;
NET "SW2[1]"     LOC = Y12  | IOSTANDARD = LVCMOS15;

NET "clk"        LOC = AC18 | 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 = AD21 | IOSTANDARD = LVCMOS33;
NET "AN[1]"      LOC = AC21 | IOSTANDARD = LVCMOS33;
NET "AN[2]"      LOC = AB21 | IOSTANDARD = LVCMOS33;
NET "AN[3]"      LOC = AC22 | IOSTANDARD = LVCMOS33;
```

◘ 以上为引脚定义






◘ 以上为实际成果展示（+，or，and，-）

# 实验十（锁存器与触发器基本原理）

# 实验报告

姓名：蒋仕彪　学号：3170102587　专业：求是科学班（计算机）1701

课程名称：逻辑与计算机设计基础实验

实验时间：2018-11-22

# 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、实验内容和原理

## 2.1 实验内容

- 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

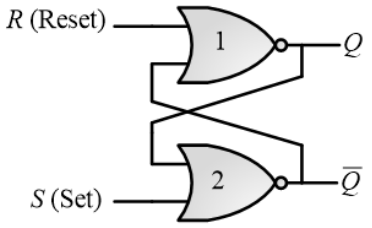
## 2.2 实验原理

### 2.2.1 锁存器

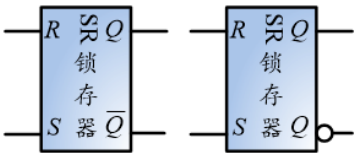
- 构成锁存器的充分条件
  - 能长期保持给定的某个稳定状态
  - 有两个稳定状态：0、1
  - 在一定条件下能随时改变逻辑状态，即：置1或置0
- 最基本的锁存器有：SR锁存器、D锁存器
- 锁存器有两个稳定状态，又称双稳态电路

## 2.2.2 SR锁存器

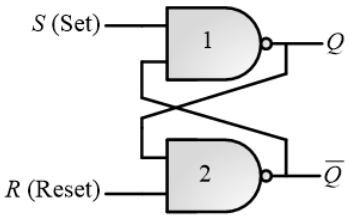
- 将两个具有2输入端的反向逻辑器件的输出与输入端交叉连起来，另一个输入端作为外部信息输出端，就构成最简单的SR锁存器。



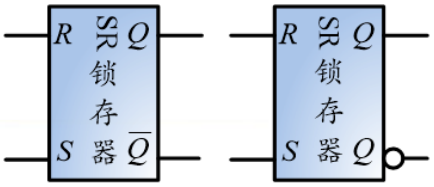

| $RS$ | $Q\,Q$ | 说明 |
|---|---|---|
| 0 0 | $Q\,Q$ | 保持 |
| 0 1 | 1 0 | 置1 |
| 1 0 | 0 1 | 置0 |
| 1 1 | 0 0 | 未定义 |



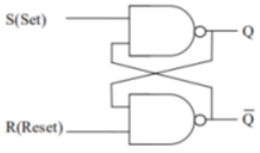

浙江大学




| $RS$ | $Q\,Q$ | 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 |

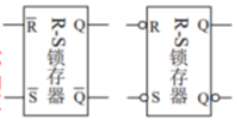



### NAND结构$\overline{R}$-$\overline{S}$锁存器



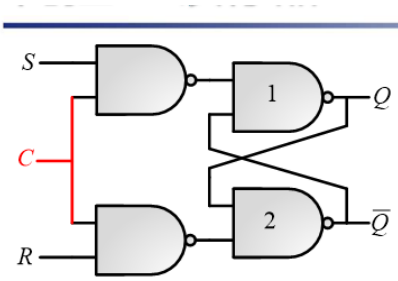

| RS | $Q\overline{Q}$ | SR |
|---|---|---|
| 00 | 11 | 无定义 |
| 01 | 01 | 置0 |
| 10 | 10 | 置1 |
| 11 | $Q\overline{Q}$ | 保持 |

不确定

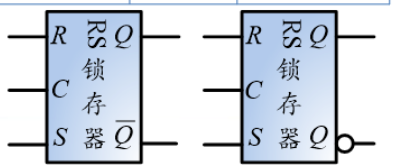



(a) 逻辑图　(b) 功能表　(c) 逻辑符号

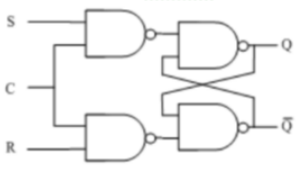
## 2.2.3 门控 SR 锁存器




| $CRS$ | $Q\,Q$ | 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 |

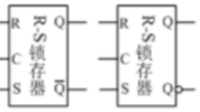



### 门控RS锁存器(电平使能)




| CRS | $Q\overline{Q}$ | 说明 |
|---|---|---|
| 0×× | $Q\overline{Q}$ | 保持 |
| 100 | $Q\overline{Q}$ | 保持 |
| 101 | 10 | 置1 |
| 110 | 01 | 置0 |
| 111 | 11 | 无定义 |

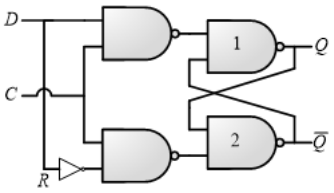
不确定



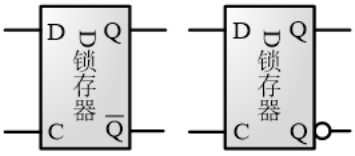

(a) 逻辑图　(b) 功能表　(c) 逻辑符号

## 2.2.4 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
- 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




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

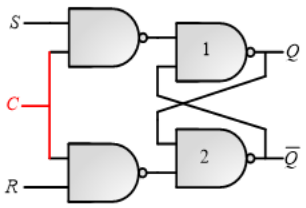



## 2.2.5 触发器

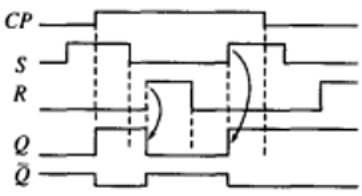
- D锁存器的缺点：存在空翻现象——如果D锁存器直接用在时序电路中作为状态存储元件，当使能控制信号有效时，会导致该元件内部的状态值随时多次改变，而不是保持所需的原始状态值
- 解决方法：消除空翻现象，使每次触发仅使锁存器的内部状态仅改变一次
- 触发：外部输入使锁存器状态改变的瞬间状态
- 触发器：在锁存器的基础上使每次触发仅使状态改变一次的锁存电路（双稳态）

## 2.2.6 空翻现象

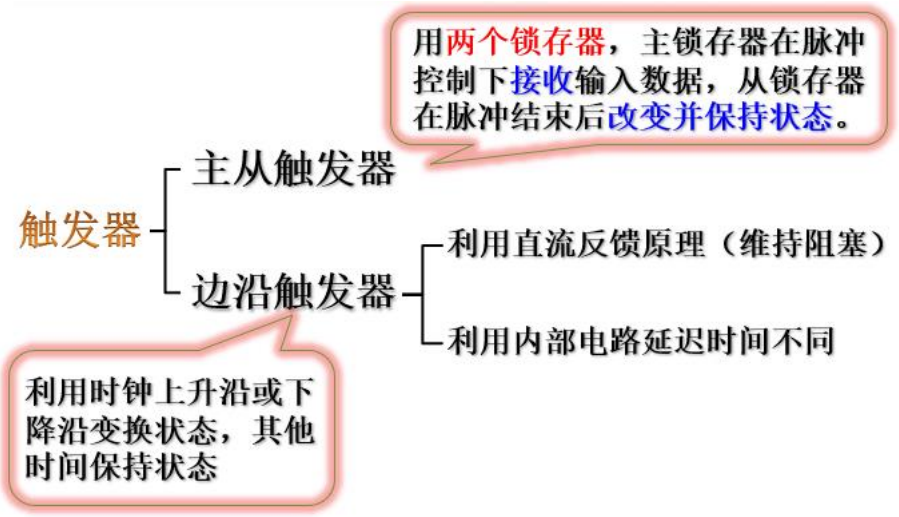
- 空翻现象:又称为竞态现象，是数字电路中的一个术语，指在同一个时钟脉冲信号作用区间内，由于时钟脉冲的宽度过大，触发器出现在“0”“1”两逻辑信号中多次翻转的现象。它限制了同步RS触发器在实际工作中的正常应用。
- 同步RS时钟触发器有空翻现象。空翻是在基本RS触发器的基础上构造时钟触发器时，因导引电路C门和D门功能不完善而造成的一种现象，即在一次时钟来到期间，触发器多次翻转的现象称为空翻，如图所示。这违背了构造时钟触发器的初衷，每来一次时钟，最多允许触发器翻转一次，若多次翻转，电路也会发生状态的差错，因而是不允许的。因为在CP=1期间，时钟对C门和D门的封锁作用消失，数据端R和S端的多次变化就会通过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。




| 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 |



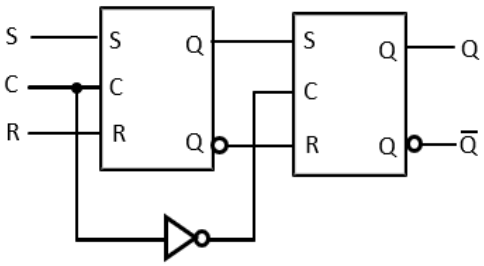
## 2.2.7 触发器

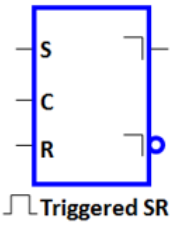



常见的触发器有：主从SR触发器、D触发器、JK触发器、T

## 2.2.8 SR主从触发器

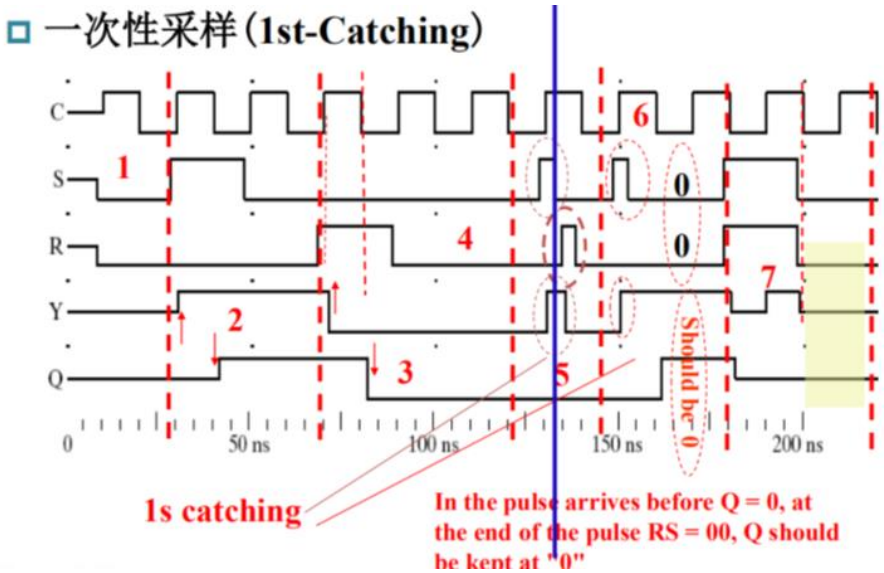
- 由两个钟控S-R锁存器串联构成，第二个锁存器的时钟通过反相器取反
- 当C=1时，输入信号进入第一个锁存器（主锁存器）
- 当C=0时，第二个锁存器（从锁存器）改变输出
- 从输入到输出的通路被不同的时钟信号值(C = 1 和 C = 0)所断开



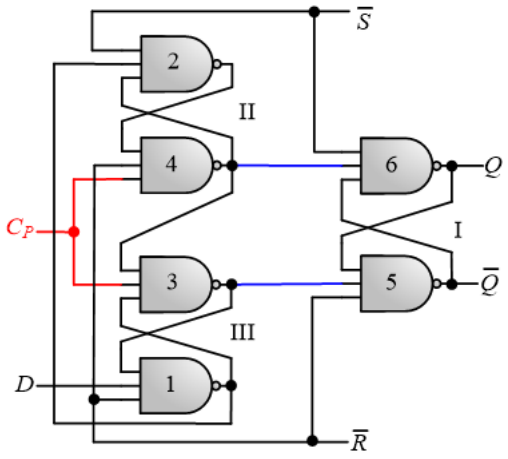




| R S | Q $\overline{Q}$ | S R |
|---|---|---|
| 0 0 | 1 1 | 无定义 |
| 0 1 | 0 1 | 置 0 |
| 1 0 | 1 0 | 置 1 |
| 1 1 | Q $\overline{Q}$ | 保持 |

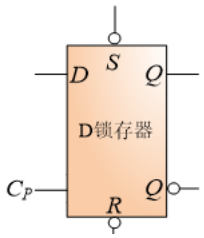
不确定



### 2.2.8 SR主从触发器




| 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** |




# 三、实验设备与材料

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

# 四、操作方法与实验步骤

## 4.1 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

## 4.2 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（包含空翻）
- 生成自定义符号的CSR_LATCH.sym

### 4.3 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（包含空翻）

### 4.4 SR主从触发器

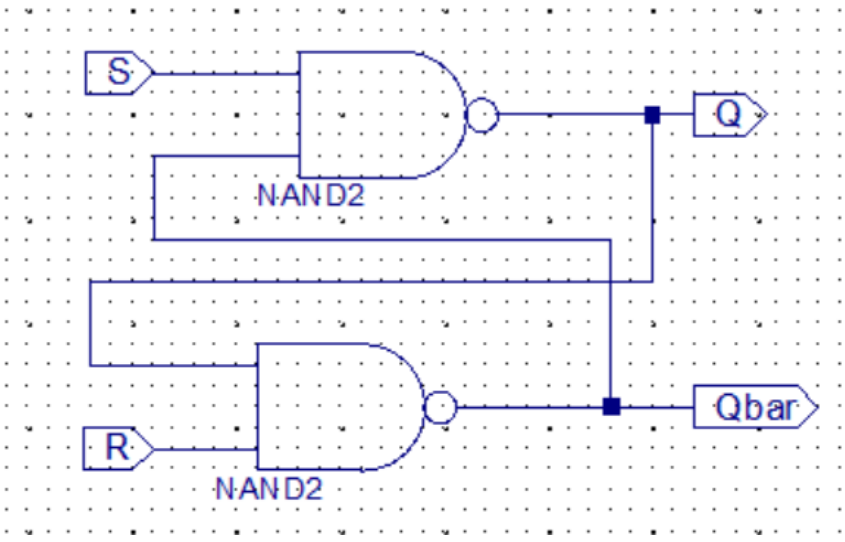
- 新建源文件 MS_FLIPFLOP.sch
- 用原理图方式设计
- 调用 CSR_LATCH 实现
- 仿真（包含一次性采样）

### 4.5 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

# 五、实验结果与分析

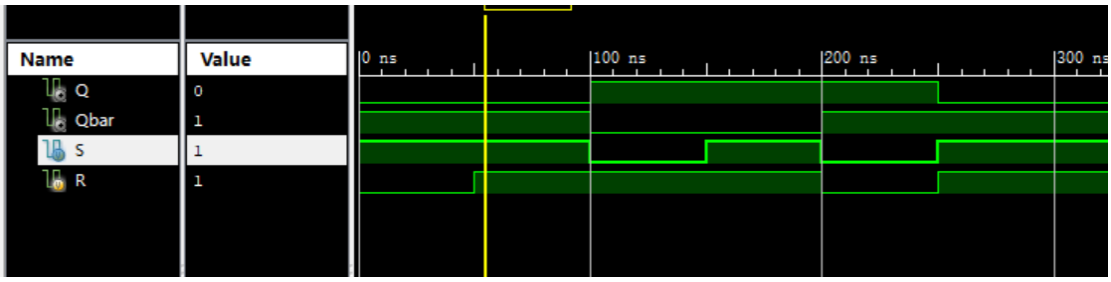
### 5.1 基本SR锁存器




- 以上为原理图设计

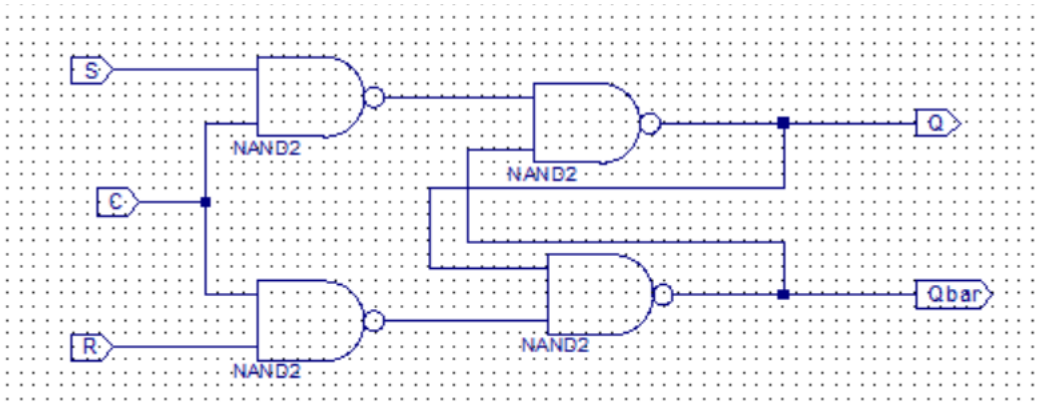
```
module SR_LATCH_SR_LATCH_sch_tb();

// Inputs
   reg S;
   reg R;

// Output
   wire Q;
   wire Qbar;

// Instantiate the UUT
   SR_LATCH UUT (
        .S(S),
        .R(R),
        .Q(Q),
        .Qbar(Qbar)
   );
// Initialize Inputs
      initial begin
            S = 1;  R = 0;  #50;
            S = 1;  R = 1;  #50;
            S = 0;  R = 1;  #50;
            S = 1;  R = 1;  #50;
            S = 0;  R = 0;  #50;
            S = 1;  R = 1;  #50;
            S = 1;  R = 1;  #50;
        end
endmodule
```

◘ 以上为仿真代码




◘ 以上为仿真波形

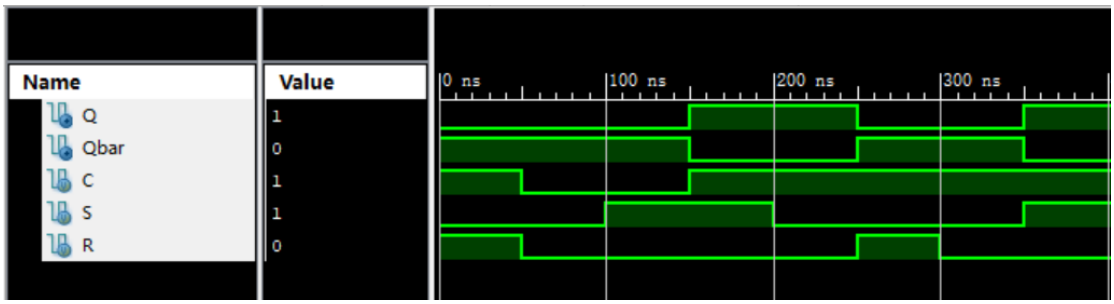
## 5.2 门控SR锁存器




◘ 以上为原理图设计

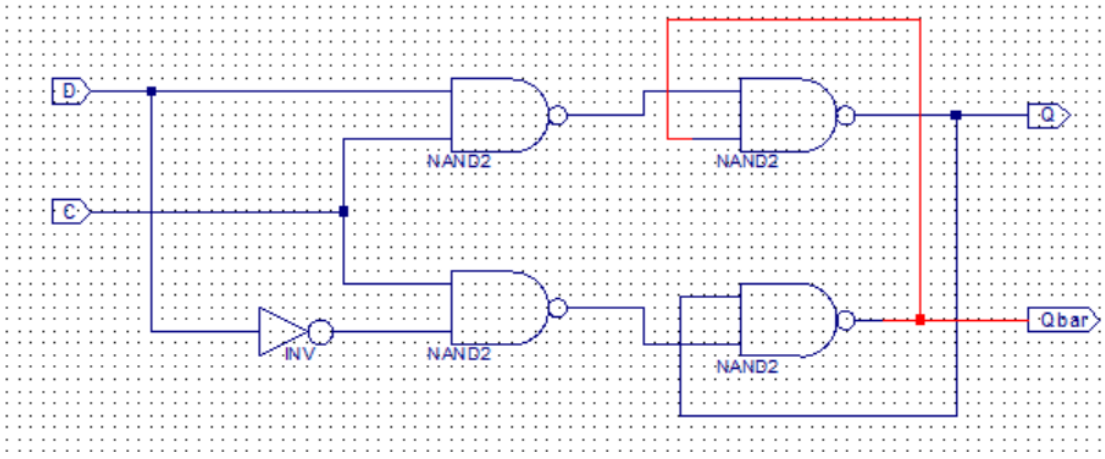
```
    initial begin
          C=1;
          R=1;S=0;#50;
          C=0;
          R=0;S=0;#50;
          R=0;S=1;#50;
          C=1;
          R=0;S=1;#50;
          R=0;S=0;#50;
          R=1;S=0;#50;
          R=0;S=0;#50;
          R=0;S=1;#50;
     end
endmodule
```

◘ 以上为仿真代码




◘ 以上为仿真结果（注意到，C=1后Q进行了多次翻转，发生空翻）

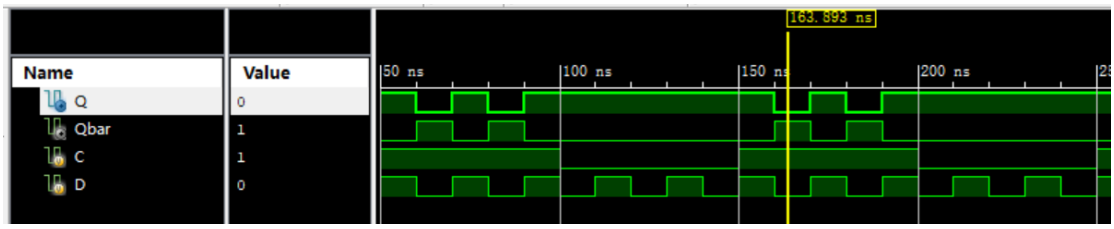
## 5.3 D锁存器




◘ 以上为原理图设计

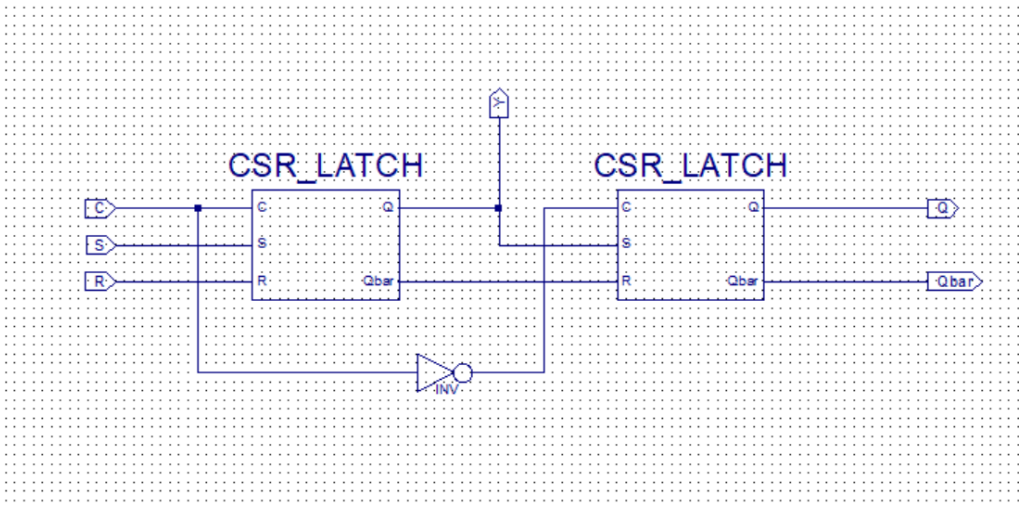
```
   initial forever begin
        C=0;
        #50;
        C=1;
        #50;
    end
    initial forever begin
        D=0;
        #10;
        D=1;
        #10;
    end
endmodule
```

◘ 以上为仿真代码




◘ 以上为仿真结果（注意到，50~100ns和150ns~200ns发生空翻）

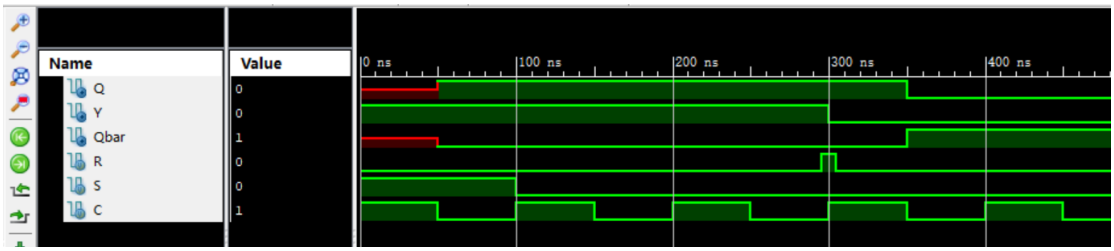
## 5.4 SR 主从触发器




◘ 以上为原理图设计

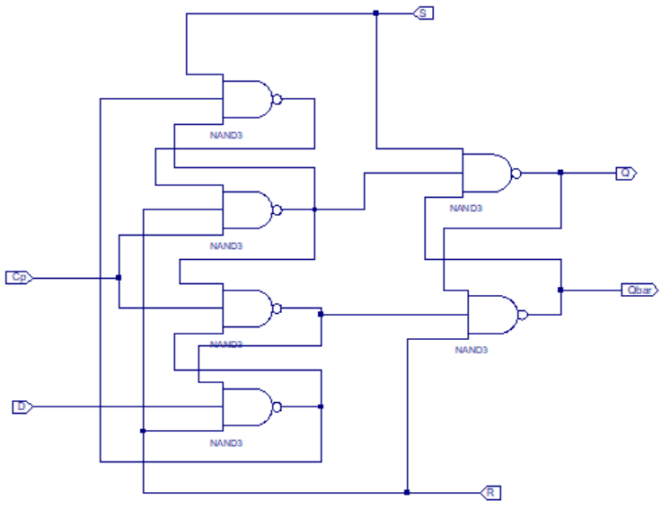
```
initial begin
    S=1;R=0;
    #100;
    S=0;R=0;
    #195;
    S=0;R=1;
    #10;
    S=0;R=0;
    #500;

end
initial forever begin
    C=1; #50;
    C=0; #50;
end
```

◘ 以上为仿真代码




◘ 以上为仿真结果（300ns处发生了一次性采样，理论上Q不能变成0）

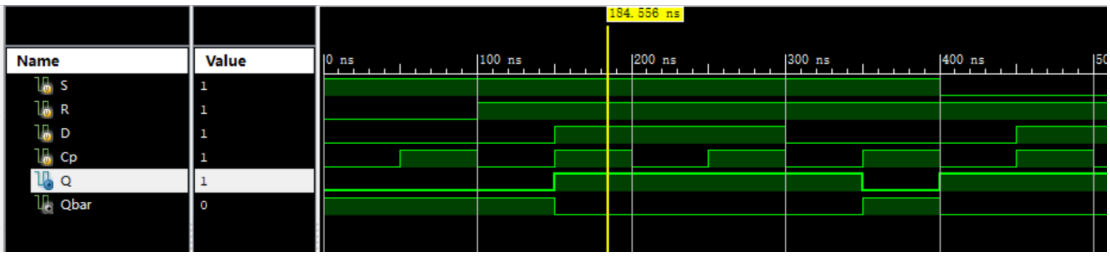
## 4.5 D触发器




◘ 以上为原理图设计

```
   initial begin
        D = 0; S = 1; R = 0; #100;
        S = 1; R = 1; #300;
        S = 0; R = 1;
   end
    initial forever begin
        D = 0;
        Cp = 0; #50;
        Cp = 1; #50;
        Cp = 0; #50;
        D = 1;
        Cp = 1; #50;
        Cp = 0; #50;
        Cp = 1; #50;
    end
endmodule
```

◘ 以上为仿真代码




◘ 以上为仿真结果

# 实验十一（同步时序电路设计）实验报告

姓名：蒋仕彪　学号：3170102587　专业：求是科学班（计算机）1701

课程名称：逻辑与计算机设计基础实验

实验时间：2018-11-29

# 一、实验目的

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

# 二、实验内容和原理

## 2.1 实验内容：

- 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

## 2.2 实验原理：

### 2.2.1 4 位二进制计数器状态表

**4位同步二进制计数器状态表**

◎状态变化条件
- 计数触发
- 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

浙江大学 系统结构与

| | 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 |

## 2.2.2 4位二进制同步计数器

| | $Q_A$ | $Q_B$ | $Q_C$ | $Q_D$ | $D_A$ | $D_B$ | $D_C$ | $D_D$ |
|---|---|---|---|---|---|---|---|---|
| 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 |

**激励函数**

$$D_A = Q_A$$

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

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

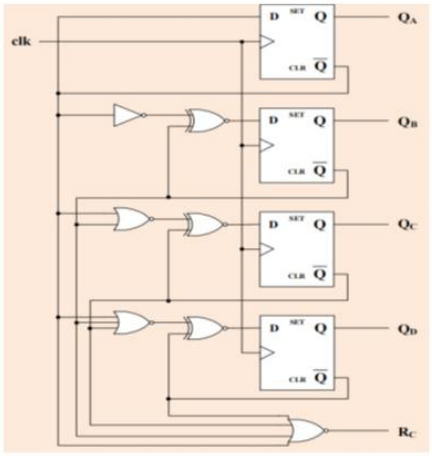
$$= \overline{(\overline{Q_A} + \overline{Q_B}) \oplus Q_C}$$

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

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

**进位RC的输出函数**

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




$$D_A = Q_A$$

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

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

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

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

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

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

## 4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
```

## 4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
```

$$D_A = Q_A$$

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

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

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

## 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面

$S=0$时，反向计数，各触发器逻辑表达式如下式

根据理论课自行完成分析过程

```
module counter_4rev (clk, rst,s, cnt, Rc);
.....                    //端口定义
wire Da,Db,Dc,Dd,nQa,nQb,nQc,nQc, Rc;
reg  Qd , Qc, Qb, Qa;

    assign Da = ? ;
    assign Db = ? ;
    assign Dc = ? ;
    assign Dd = ? ;
    assign Rc = ? ;

    assign cnt = {Qd , Qc, Qb, Qa};

  always @ (posedge clk)
      if (rst) cnt <= 4'b0000; //同步清零
      else {Qd, Qc, Qb, Qa} <=
                        {Dd,Dc,Db,Da};
endmodule
```

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

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

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

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

$$R = \overline{S}\,\overline{Q}_A\overline{Q}_B\overline{Q}_C\overline{Q}_D + SQ_AQ_BQ_CQ_D$$

## 2.2.3 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

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

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

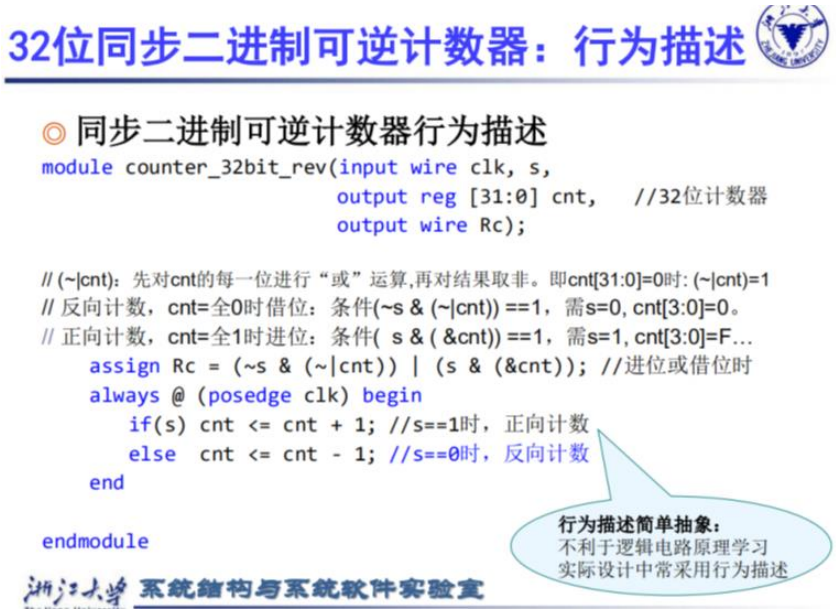
$$D_C = \overline{S}[\overline{(Q_A Q_B)} \oplus \overline{Q}_C] + S[\overline{(Q_A + Q_B)} \oplus \overline{Q}_C] = [\overline{SQ_AQ_B} + S(\overline{Q}_A + \overline{Q}_B)] \oplus \overline{Q}_C = [\overline{S(Q_A + Q_B)} + S(\overline{Q}_A + \overline{Q}_B)] \oplus \overline{Q}_C$$

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

$$R = \overline{S}\,\overline{Q}_A\overline{Q}_B\overline{Q}_C\overline{Q}_D + SQ_AQ_BQ_CQ_D$$ （进位、借位输出）

- 可逆二进制4位同步计数器的行为描

```
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
```




## 2.2.4 分频器设计

- 100MHz 信号通过 50,000,000 次分频后，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
```

# 三、主要仪器设备

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

# 四、操作方法与实验步骤

## 4.1 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
- 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
- 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//根据“分频器设计”程序得到 1s 时钟
- 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

## 4.2 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

- 波形仿真（包含正向计数和反向计数）
- 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
- 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

# 五、实验结果与分析

## 5.2 设计16位可逆同步二进制计数器

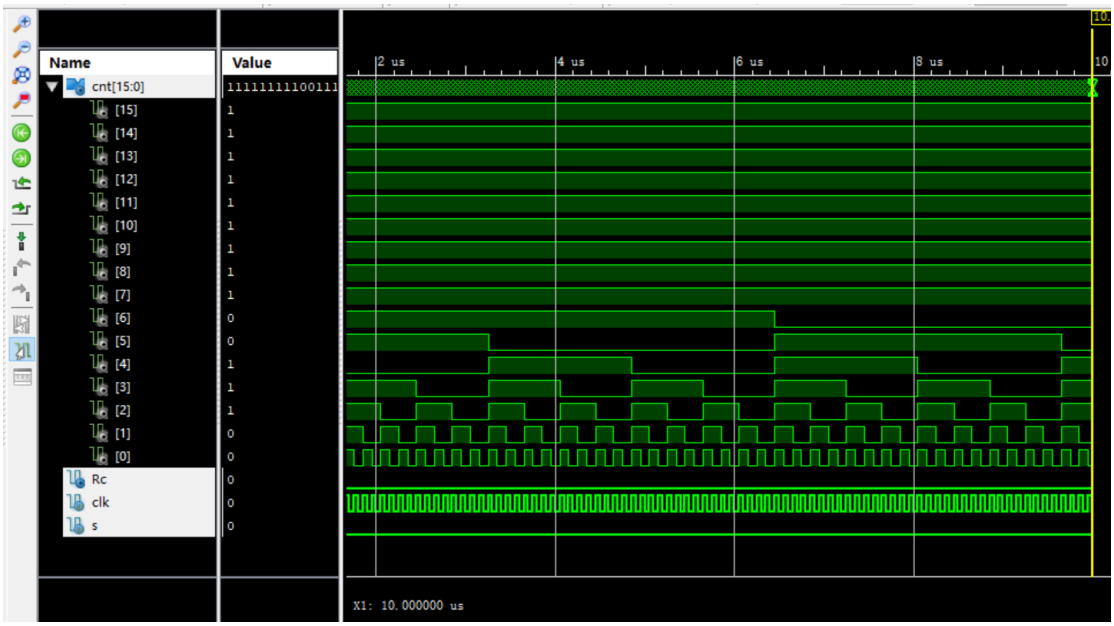
```
module RevCounter(
    input wire clk, s,
    output reg[15:0] cnt,
    output wire Rc
    );
    assign Rc=(~s & (~|cnt)) | (s & (&cnt));
    initial begin
        cnt = 0;
    end
    always @ (posedge clk) begin
        if (s) cnt <= cnt + 1;
        else cnt <= cnt - 1;
    end
endmodule
```

- 以上为 Revcounter 的 verlog 代码

```
module test_counter;

    // Inputs
    reg clk;
    reg s;
    // Outputs
    wire [15:0] cnt;
```

```
    wire Rc;
    // Instantiate the Unit Under Test (UUT)
    RevCounter uut (
        .clk(clk),
        .s(s),
        .cnt(cnt),
        .Rc(Rc)
    );
    initial begin
        // Initialize Inputs
        clk = 0;
        s = 0;
    end
        // Wait 100 ns for global reset to finish
        initial forever begin
            clk = 0;
            #50;
            clk = 1;
            #50;
        end
        // Add stimulus here

endmodule
```

◘ 以上为仿真代码



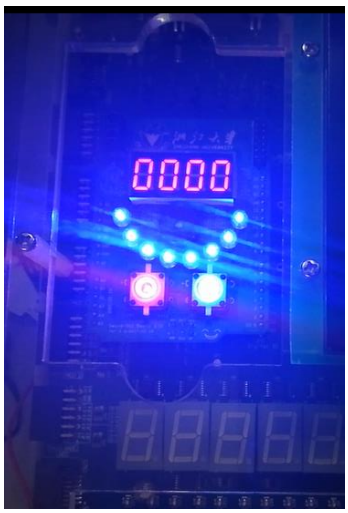

◘ 以上为仿真结果

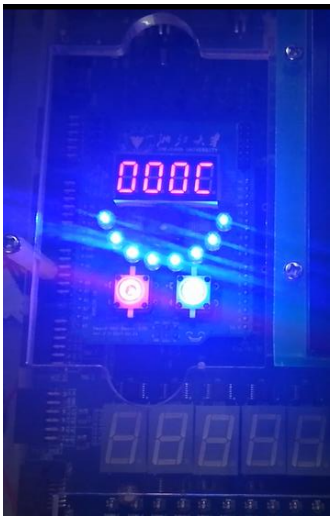
```
module clk_100ms(clk,clk_100ms);
input wire clk;
output reg clk_100ms;
reg [15:0] cnt;
always @(posedge clk)begin
    if(cnt<5_000_000)begin
        cnt <= cnt + 1;
    end else begin
        cnt <= 0;
        clk_100ms <= ~clk_100ms;
    end
end


endmodule
```

◘ 以上为 clk_100ms.v 的代码

```
module top(input wire clk,
        input wire s,
        output wire [3:0]AN,
        output wire [7:0]SEGMENT,
        output wire [7:0]LED

    );

    wire [15:0] cnt;
    wire clk_100ms;

    RevCounter m0(clk_100ms, s, cnt, LED[0]);

    clk_100ms m1(clk,clk_100ms);

    disnum m2(clk, cnt , 4'b0 , 4'b0, 1'b0, AN, SEGMENT);

    assign LED[7:1] = 7'b1111111;

endmodule
```

◘ 以上为 top.v 的代码






◘ 以上为成果展示