实验 8&9--加法器、加减法器 和 ALU 基本原理与设计实验报告

姓名：黄炯睿 学号：3170103455 专业：信息安全

课程名称：逻辑与计算机设计基础实验 同组学生姓名：无

实验时间：2018-11-8 实验地点：紫金港东 4-509 指导老师： 洪奇军

一、实验目的和要求

1.1 掌握一位全加器的工作原理和逻辑功能
1.2 掌握串行进位加法器的工作原理和进位延迟
1.3 掌握减法器的实现原理
1.4 掌握加减法器的设计方法
1.5 掌握 ALU 基本原理及在 CPU 中的作用
1.6 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 位串行进位加法器
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 Ai 和 Bi，低位进位输入 Ci
二个输出位：全加和 Si，进位输出 Ci+1

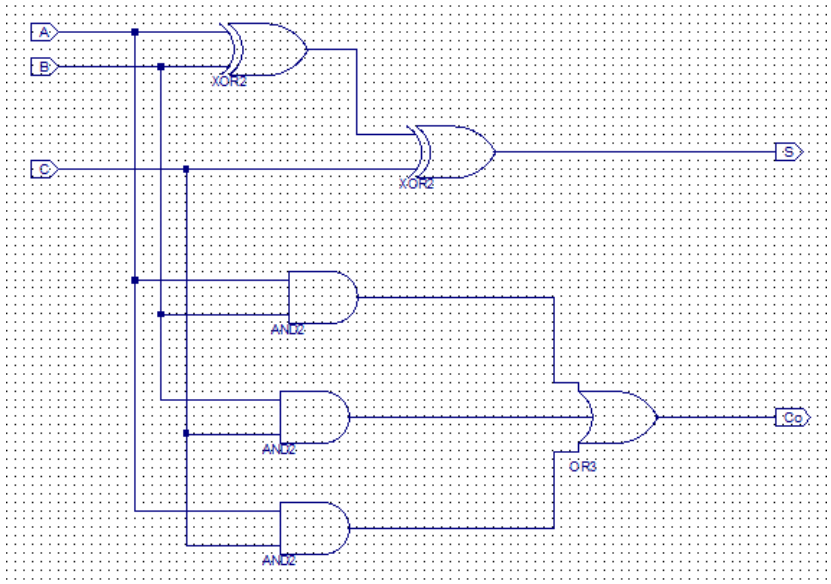
A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

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

$$C_{i+1} = A_i B_i + B_i C_i + C_i A_i$$

图表 1 一位全加器真值表和关系式

根据一位全加器的输入输出关系，得到电路图



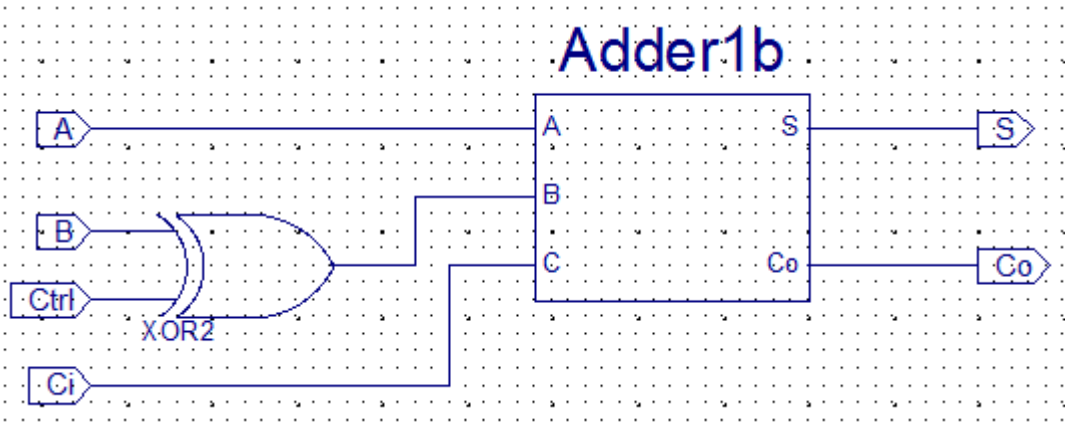
图表 2　一位加法器设计图

2.2.2 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 C0 为 1



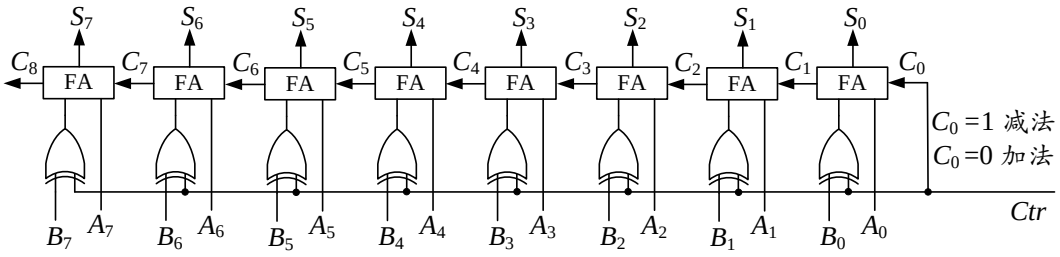
图表 3 一位加减法器设计图

2.2.3 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 C0 为 1



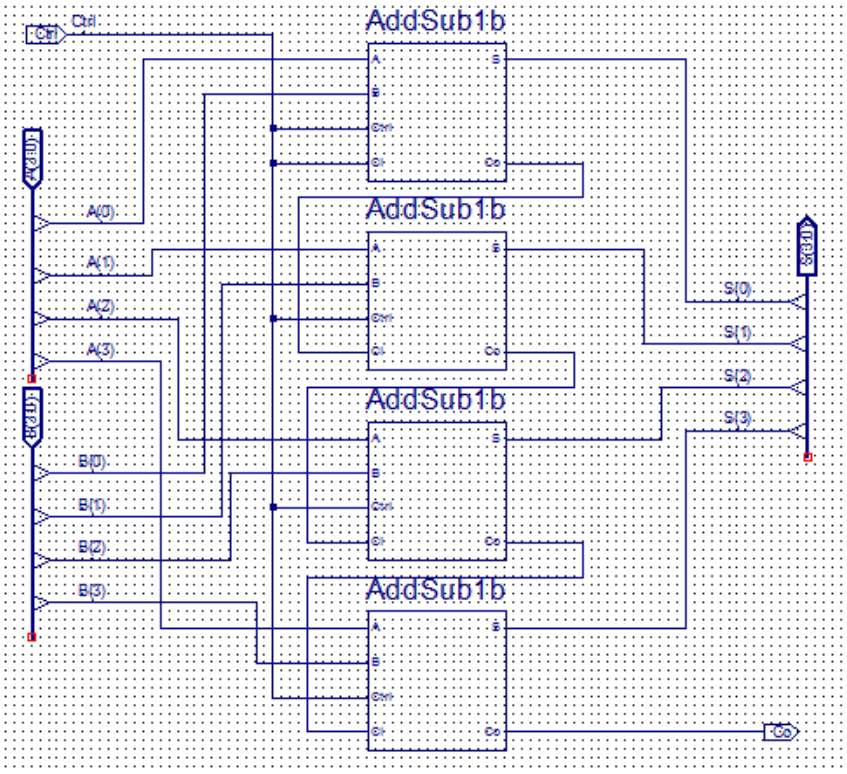
$$Ctr\text{为}0\text{时，}\ S[7:0]=A[7:0]+B[7:0]$$

$$Ctr\text{为}1\text{时，}\ S[7:0]=A[7:0]-B[7:0]$$

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

图表 4 多位串行进位全减器原理图

2.2.4 4 位加减法器



图表 5 4 位加减法器设计图

2.2.5 设计按键数据输入模块

```
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
```

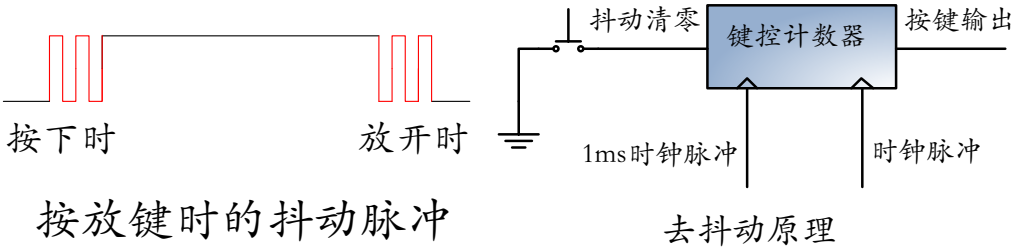
图表 6 CreateNumber 模块代码图

2.2.6 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



图表 6 按键去抖动原理图

三、主要仪器设备

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

四、操作方法与实验步骤

4.1 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

4.2 实验步骤：

4.2.1 AddSub1b 设计

1.新建工程，工程名称用 MyALU。

2.Top Level Source Type 用 HDL

3.新建源文件，类型是 Schematic，文件名称用 AddSub1b。

4.原理图方式进行设计

4.2.2 AddSub4b 设计

1.新建源文件

2.类型是 Schematic

3.文件名称用 AddSub4b

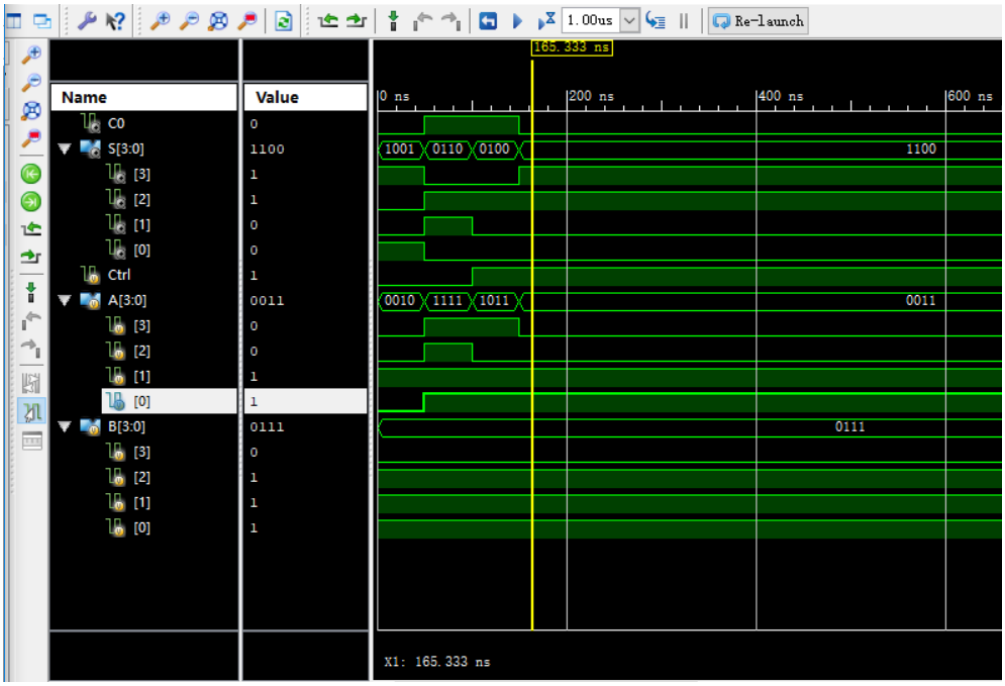
4.原理图方式（图表 5）进行设计，调用前面设计的 AddSub1b

5.进行波形仿真，激励输入至少 4 组

仿真代码如下：

```
initial begin
        Ctrl = 0;
        A = 4'b0010;
        B = 4'b0111;
        #50;
        Ctrl = 0;
        A = 4'b1111;
        B = 4'b0111;
        #50;
        Ctrl = 1;
        A = 4'b1011;
        B = 4'b0111;
        #50;
        Ctrl = 1;
        A = 4'b0011;
        B = 4'b0111;
        #50;
end
```

仿真波形图如下：



图表 7 AddSub4b 模块波形仿真图

仿真结果分析：

仿真输入用了四组数据

第一组数据由于 Ctrl 输入为 0，即为 0011+0111，其结果为 S=1001，而且无进位，即 C0 为 0.

第二组数据由于 Ctrl 输入为 0，即为 1111+0111，其结果为 S=0110，而且有进位，即 C0 为 1.

第三组数据由于 Ctrl 输入为 1，即为 1011-0111，其结果为 S=0100，用补码表示则为 1011+1001，所以它有进位，即 C0 为 1.

第四组数据由于 Ctrl 输入为 1，即为 0011-0111，其结果为 S=1100，用补码表示则为 0011+1001，所以它无进位，即 C0 为 1.

符合模块设计本身要求的逻辑。

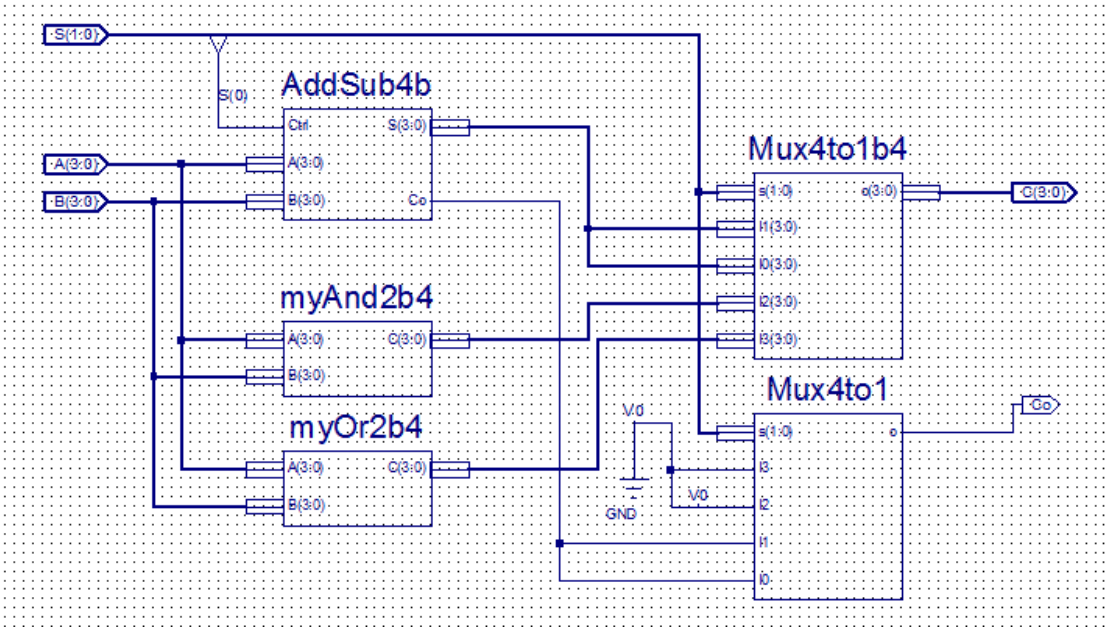
4.2.3 ALU 设计

1.新建源文件

2.类型是 Verilog 或 Schematic

3.文件名称用 ALU

4.原理图方式进行设计，原理图如下：



图表 8 4 位 ALU 原理图

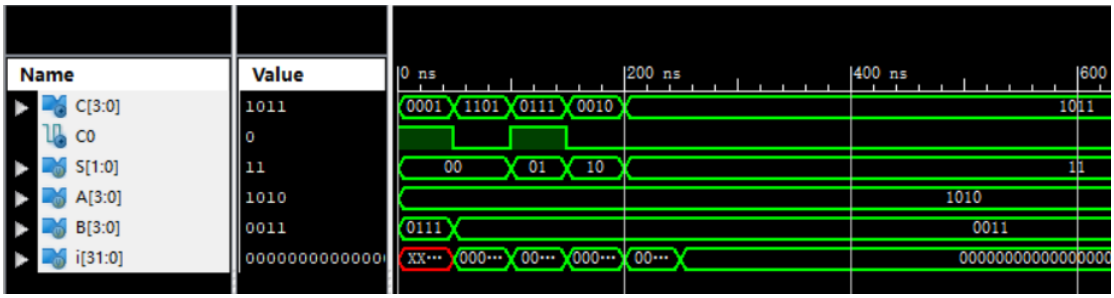
5. 进行波形仿真，激励输入至少 4 组，覆盖 4 种操作

仿真代码如下：

```
integer i;
    initial begin

    S = 2'b00;
    A = 4'b1010;
    B = 4'b0111;
    #50;
    B = 4'b0011;
    for(i = 0; i <= 3; i=i+1)begin
        S = i;
```

```
        #50;
    end
end
```

仿真波形图如下：



图表 8 ALU 模块波形仿真图

仿真结果分析：

第一组数据：由于 S 输入为 00，即两个位数之间进行加法运算，1010+0111，所以输出 C 为 1 0001，含有进位，所以 C0 为 1.

第二组数据：由于 S 输入为 00，即两个位数之间进行加法运算，1010+0011，所以输出 C 为 1101，不含有进位，所以 C0 为 0.

第三组数据：由于 S 输入为 01，即两个位数之间进行减法运算，1010-0011，换算为补码运算即为 1010+1101，所以输出 C 为 1 0111，含有进位，所以 C0 为 1.

第四组数据：由于 S 输入为 10，即两个位数之间进行按位与运算，1010&0011，所以输出 C 为 0010，不含有进位，所以 C0 为 0.

第五组数据：由于 S 输入为 11，即两个位数之间进行按位或运算，1010|0011，所以输出 C 为 1011，不含有进位，所以 C0 为 0.

符合模块设计本身要求的逻辑。

4.2.4 top 设计

1.新建源文件，类型是 Verilog，文件名 Top。 ，右键设为“Set as Top Module”

2.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

top 代码：

```
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 [15:0] num;
    wire [1:0] btn_out;
    wire [3:0] C;
    wire C0;
    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,1'b0,clk_div);

    CreateNumber m3({2'b0,btn_out},{2'b0,SW[3:2]},num);

    ALU m5(.A(num[3:0]),.B(num[7:4]),.S(SW2),.C(C),.C0(C0));
    disp_num                                                m6(clk,
{num[7:0] ,C,3'b0,C0 },4'b0,4'b0,1'b0,AN,SEGMENT);
endmodule
```

pbdebounce 代码：

```
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
```

clkdiv 代码

```
module clkdiv(input clk,
                input rst,
                output reg[31:0]clkdiv
    );
    always@(posedge clk or posedge rst)begin
        if(rst) clkdiv <= 0;
        else clkdiv <= clkdiv + 1'b1;
        end

endmodule
```

CreateNumber 代码：

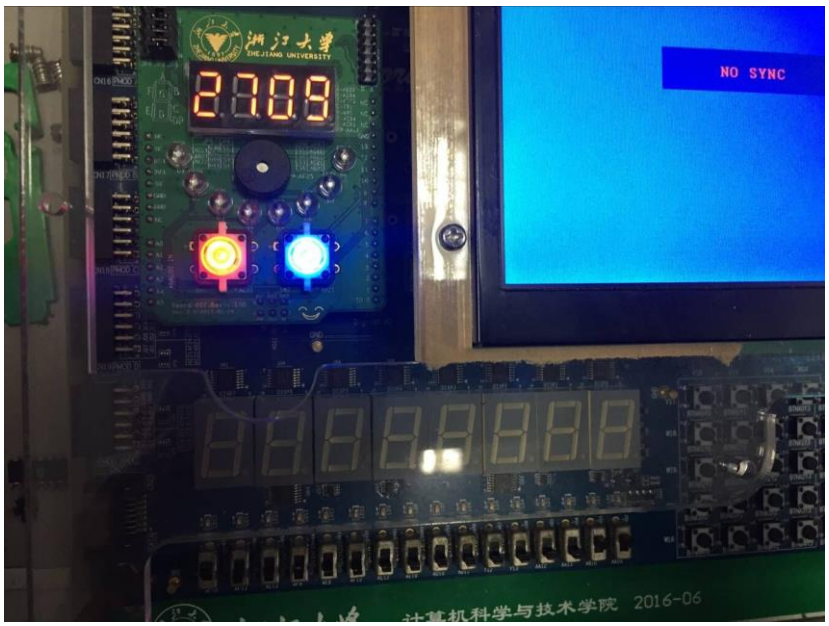
```
module CreateNumber(
    input wire[3:0]btn,
    input wire [3:0] sw,
    output reg [15:0] num
    );
    wire[3:0] A1,B1,C1,D1;

    initial num<=16'b1010_1011_1100_1101;

    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
```

DispNum 模块使用实验 7 中的写过的模块

五、实验结果与分析

5.1.加法：

5.1.1 无进位加法：



图表 9 实验结果图-加法运算结果 1

由图，图上表现的是 7+2=9 的 16 进制运算，本质是二进制 0111+0010 = 1001.

显然，因为没有进位，所以显示管上第三个位是 0。

5.1.2 有进位加法：

图表 10 实验结果图-加法运算结果 2

由图，图上表现的是 E+8=16 的 16 进制运算，本质是二进制 1110+1000 = 0001 0110.

显然，因为有进位，所以显示管上第三个位是 0001，即显示为 1。

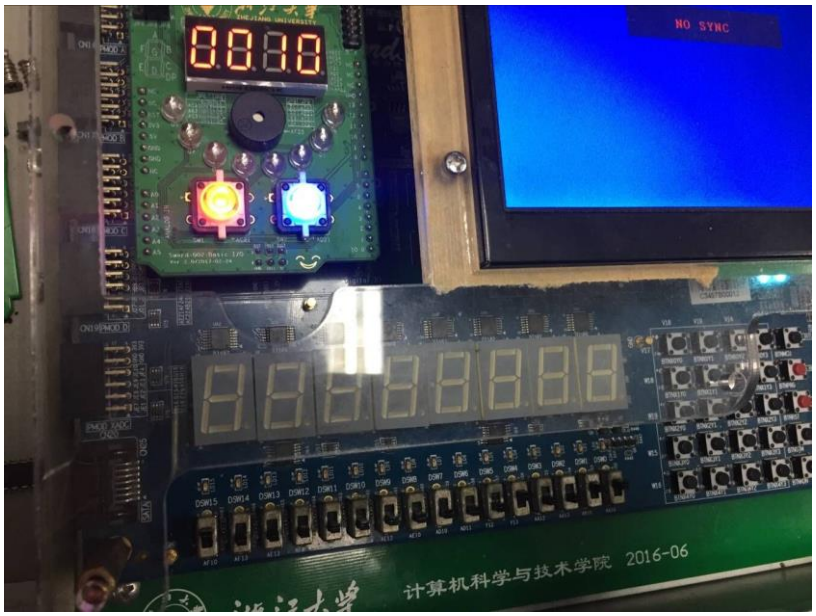
5.2 减法：

5.2.1 大数减小数

图表 11 实验结果图-减法运算结果 1

由图，图上表现的是 7-2 = 5 的 16 进制运算，它等效于 7+（-2）的运算. 二进制的本质是 0111 + 1110 = 0001 0101. 其中 1110 是 0010（2）的补码，表示-2.
显然，因为有进位，所以显示管上第三个位是 0001，即显示为 1。

5.2.2 相同两数相减



图表 12 实验结果图-减法运算结果 2

由图，图上表现的是 0-0 = 0 的 16 进制运算，它等效于 0+（-0）的运算. 二进制的本质是 0000 + 0001 0000 = 0001 0000. 其中 0001 0000 是 0000 的补码，表示-0.
显然，因为有进位，所以显示管上第三个位是 0001，即显示为 1。

5.2.3 小数减大数

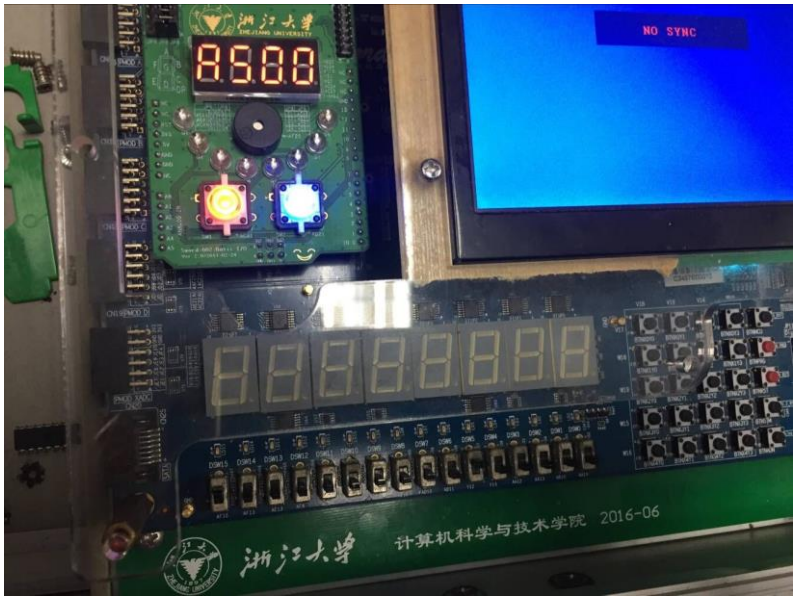
图表 13 实验结果图-减法运算结果 3

由图，图上表现的是 7-9=-2 的 16 进制运算，它等效于 7+（-9）的运算. 二进制的本质是 0111 + 0111 = 1110. 其中 0111 是 1001（9）的补码，表示-9.

所得结果 1110 是补码，其原码恰是 2，即等于-2

显然，因为没有进位，所以显示管上第三个位是 0000，即显示为 0。

5.3 与：



图表 14 实验结果图-与法运算结果

由图，图上表现的是 A&5=0 的 16 进制运算。

二进制的本质是 1010 & 0101 = 0000. 第四位显示 0000，即 0.

显然，因为没有进位，所以显示管上第三个位是 0000，即显示为 0。

5.4 或：

图表 15 实验结果图-或法运算结果

由图，图上表现的是 A|5=F 的 16 进制运算。

二进制的本质是 1010 | 0101 = 1111. 第四位显示 1111，即 F.

显然，因为没有进位，所以显示管上第三个位是 0000，即显示为 0。

六、讨论、心得

这次实验是第一次真正意义上的自己写代码，虽然只是填写了代码中的参数，但这也给我带来了很大的麻烦。它促使我去理解代码，并且对于 ALU 的了解更加深入。在设计 ALU 的过程中，我也对于以往学过的汇编课程里的一些加减处理有了更加底层的认识。尤其是在对于减法的处理，我开始总是不能理解减法的进位，现在看到了它的本质后，我终于明白了减法的进位实际上是与补码相加后的进位。

总得来说，这次实验花费了我很多的时间，但也确实让我明白了很多原理。

实验 10--锁存器与触发器基本原理实验报告

姓名：黄炯睿　　学号：3170103455　　专业：信息安全

课程名称：逻辑与计算机设计基础实验　　同组学生姓名：无

实验时间：2018-11-22　　实验地点：紫金港东 4-509　　指导老师：　洪奇军

一、实验目的和要求

1.掌握锁存器与触发器构成的条件和工作原理
2.掌握锁存器与触发器的区别
3.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
4.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

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

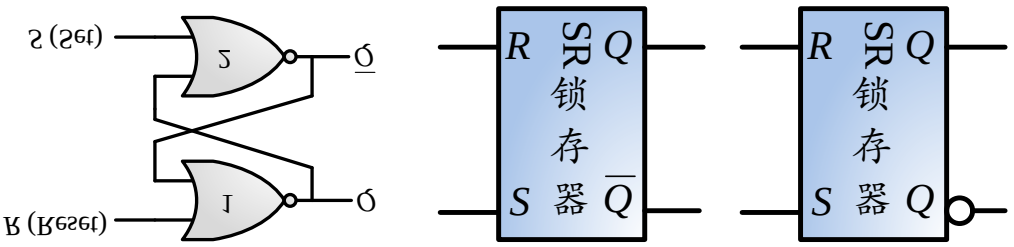
2.2 实验原理

2.2.1 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（1）

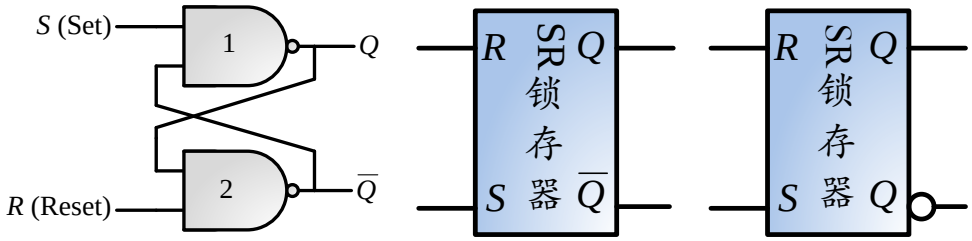
将两个具有 2 输入端的反向逻辑器件的输出与输入端交叉连起来，另一个输入端作为外部信息输出端，就构成最简单的 SR 锁存器



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

图表 1 SR 锁存器（1）

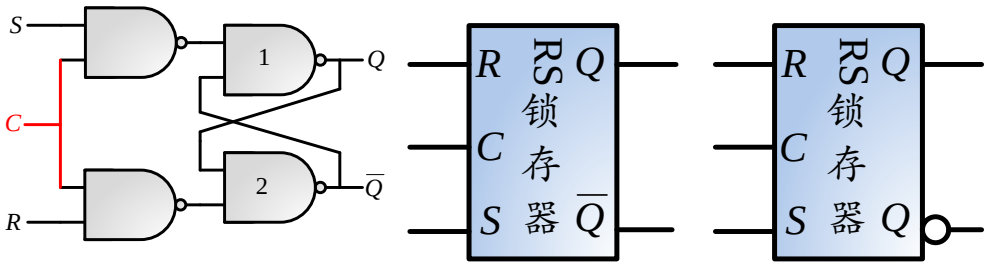
SR 锁存器（2）



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

图表 2 SR 锁存器（2）

2.2.3 门控 SR 锁存器



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

图三 门控 SR 锁存器

2.2.4 D 锁存器

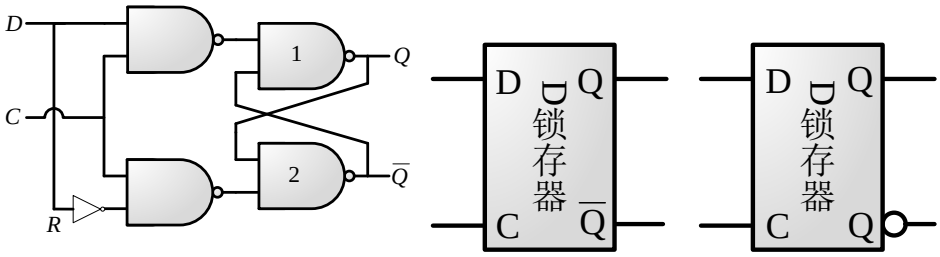
基本 SR 锁存器缺点：存在不确定状态

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



CD	QQ	说明
0×	QQ	保持
10	01	置0
11	10	置1

图四 D 锁存器

2.2.5 触发器

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）

比 D 锁存器更有优势，能够避免空翻现象，使每次触发仅使锁存器内部状态改变一次。

分类：主从触发器，边沿触发器。

常见触发器：主从 SR 触发器、D 触发器、JK 触发器、T 触发器

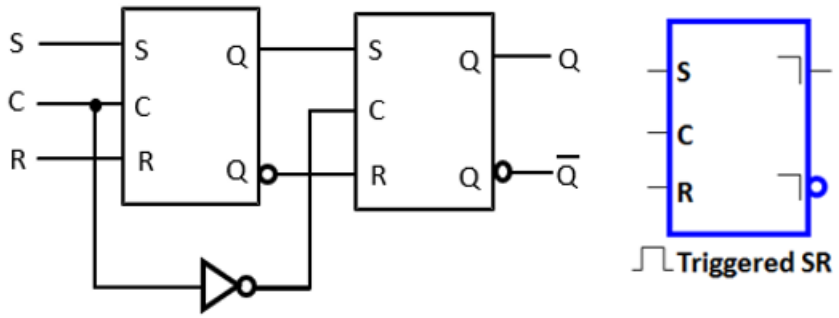
2.2.6 SR 主从触发器

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）

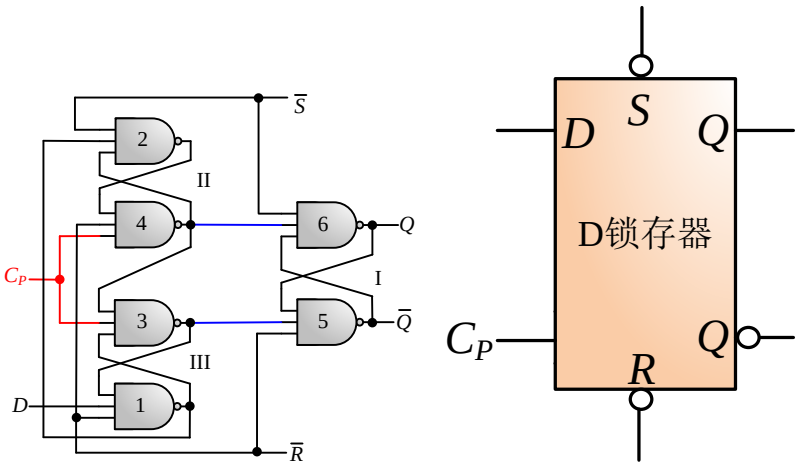
当 C=0 时，第二个锁存器（从锁存器）改变输出

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



图五 SR 主从锁存器

2.2.7 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

图表六 正边沿维持阻塞型 D 触发器

三、主要仪器设备

1 SWORD 开发板 1 套

2 装有 ISE 的 PC 机 1 台

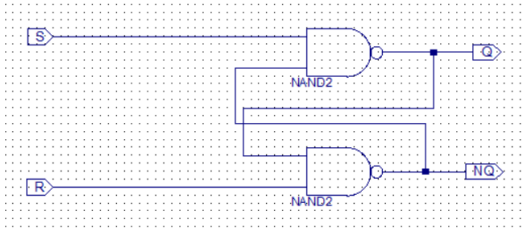
四、操作方法与实验步骤

4.1 基本 SR 锁存器

1.新建工程 MyLATCHS

2.新建源文件 SR_LATCH.sch

3.用原理图方式设计



4.用 NAND2 实现

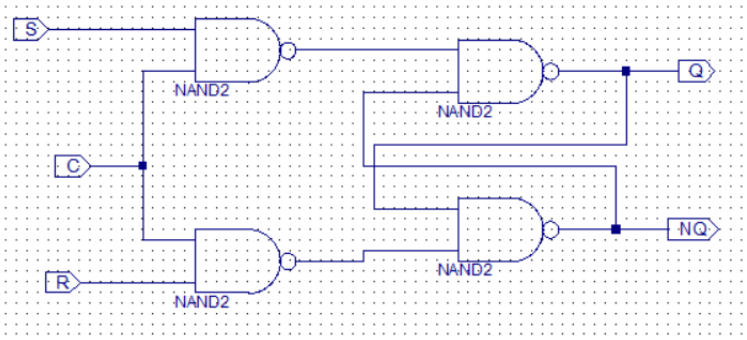
5.仿真

激励代码：

```
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
```

4.2 门控 SR 锁存器

1.新建源文件 CSR_LATCH.sch，用原理图方式设计。



2.用 NAND2 实现

3.仿真（包含空翻）

```
 always begin
      R=1;S=0; #10;
      R=0;S=1; #10;
      R=0;S=0; #10;
end
always begin
      C=0;#50;
      C=1;#50;
```

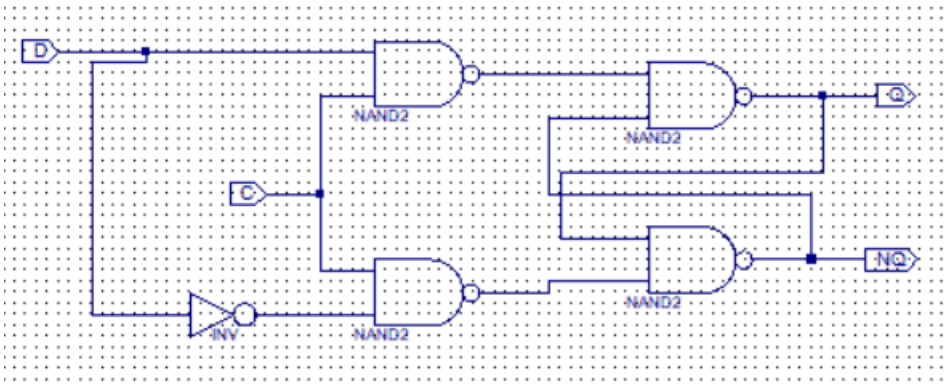
```
end
```

4.生成自定义符号的 CSR_LATCH.sym

4.3 D 锁存器

1.新建源文件 D_LATCH.sch

2.用原理图方式设计



3.用 NAND2 实现

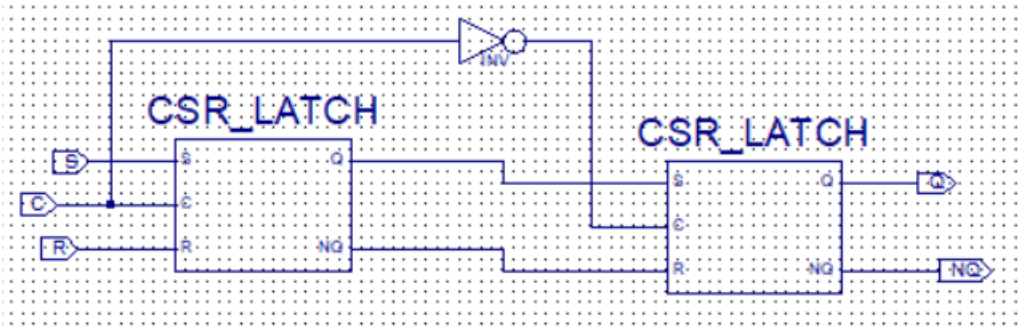
4.仿真（包含空翻）

```
always begin
      D=1; #10;
      D=0; #10;
end
always begin
      C=0;#50;
      C=1;#50;
end
```

4.4 SR 主从触发器

1.新建源文件 MS_FLIPFLOP.sch

2 用原理图方式设计



3.调用 CSR_LATCH 实现

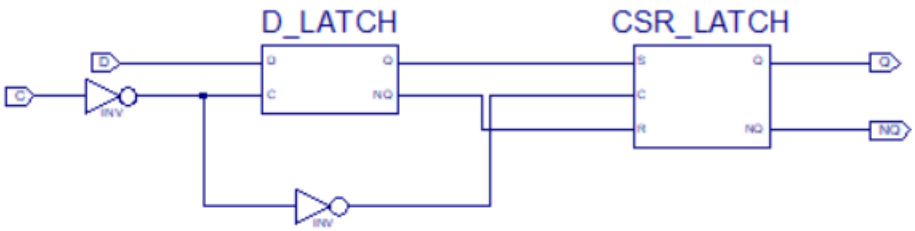
4.仿真（包含一次性采样）

```
initial begin
      R=1;S=0; #50;
      R=0;S=0; #50;
      R=0;S=1; #5;
```

```
        R=0;S=0; #15;
        R=1;S=0; #5;
        R=0;S=0; #15;
        R=0;S=1; #50;
        R=0;S=0; #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 D 触发器

1.新建源文件 D_FLIPFLOP.sch，用原理图方式设计。



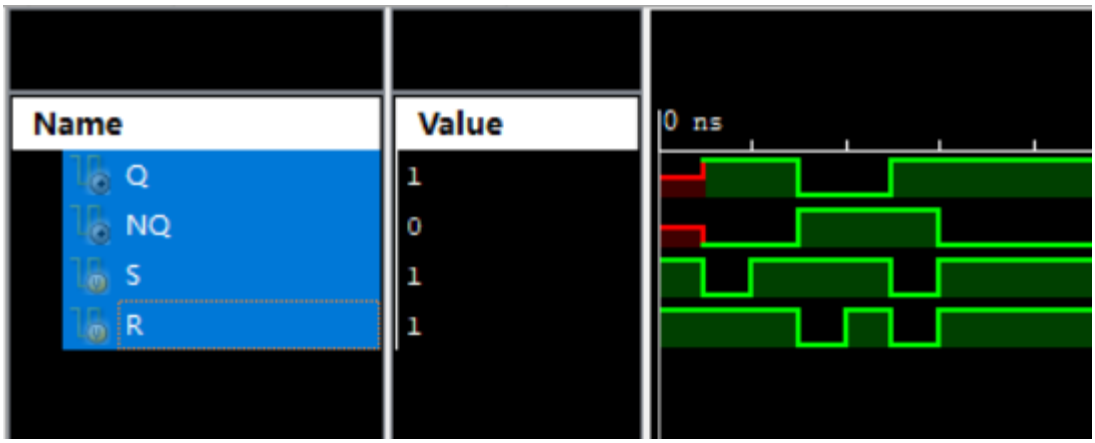
2.调用 NAND3 实现

3.仿真

```
initial begin
        D = 0; #150;
        D = 1; #150;
end

always begin
        C=0; #50;
        C=1; #50;
end
```

五、实验结果与分析

5.1 基本 SR 锁存器



图表 7 SR 锁存器仿真图

在 0-50ns, S = 1　R = 1, Q and NQ 的值未知，所以其线为红色。

在 50-100ns2, S = 0　R = 1, 所以 Q = 1　NQ =0.

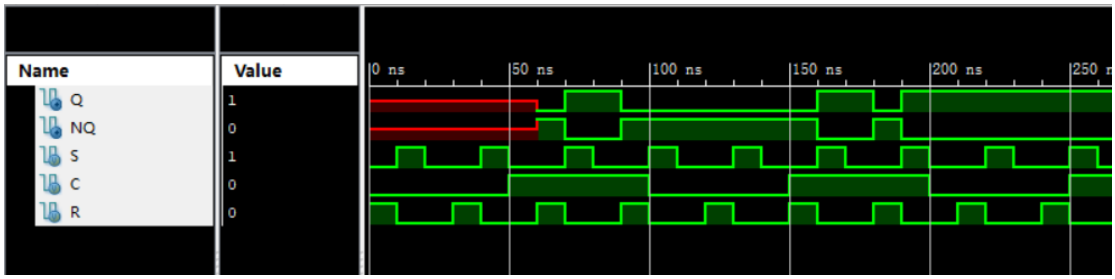
在 100-150ns, S = 1　R = 1, 保持原有状态，所以 Q = 1　NQ = 0.

在 150-200ns, S = 1　R = 0, 重置,所以 Q = 0　NQ = 1.

在 200-250ns, S = 1　R = 1, 保持原有状态, 所以 Q = 0　NQ =1.

在 250ns 之后, S = 0　R = 0, 未定义。

5.2 门控 SR 锁存器



图表 8 门控 SR 锁存器仿真图

在 50-60ns 时，C = 1 表示锁存器可以接收输入数据，S = 0，R = 0 表示保持原有状态，即未定义状态。

在 60-70ns 时，C = 1 表示锁存器可以接收输入数据，S = 0，R = 1 表示复位，所以 Q = 0 和 NQ= 1。

在 70-80ns 时，C = 1 表示锁存器可以接收输入数据，S = 1，R = 0 表示置位，所以 Q = 1 和 NQ= 0。

在 80-90ns 时，C = 1 表示锁存器可以接收输入数据，S = 0，R = 0 表示保持原有状态，即 Q = 1 和 NQ= 0。

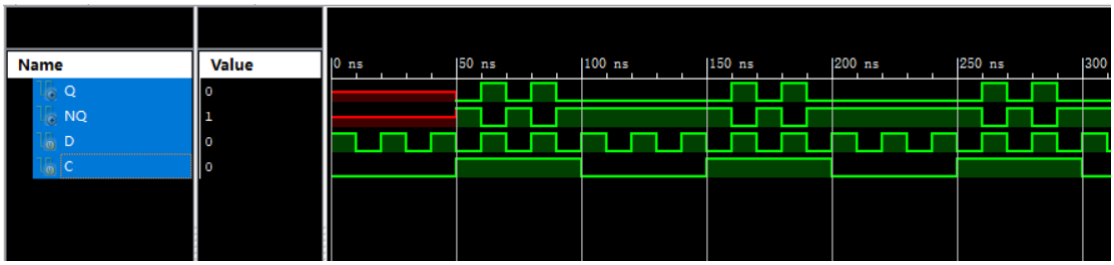
在 90-100ns 时，C = 1 表示锁存器可以接收输入数据，S = 0，R = 1 表示复位，所以 Q = 0 和 NQ= 1。

在 100-150ns 时，C = 0 表示锁存器不可以接收输入数据，表示保持原有状态，所以 Q = 0 和 NQ= 1。

其他情况类似，略去。

可以发现在一次 C=1 的区间里，Q 的值多次变换，这是空翻现象，它限制了同步 RS 触发器在实际工作中的正常应用.

5.3 D 锁存器



图表 9 D 锁存器仿真图

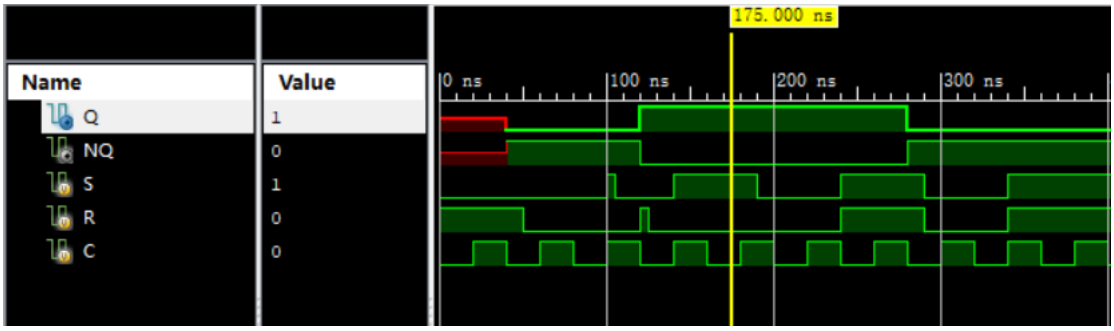
在 50-60ns、70-80ns、90-100ns 时，C = 1 表示锁存器可以接收输入数据，D = 0 因此 Q = 0，NQ= 1。

在 60-70ns、80-90ns 时，C = 1 表示锁存器可以接收输入数据，D = 1 因此 Q = 1，NQ= 0。

在 100-150ns 时，C = 0 表示锁存器不能接收输入数据，只能保持过去的值，所以 Q 和 NQ 不会改变。

可以发现在一次 C=1 的区间里，Q 的值多次变换，这是空翻现象，它限制了同步 RS 触发器在实际工作中的正常应用.

5.4 SR 主从触发器



图表 7 SR 主从触发器仿真图

Q 和 NQ 的值仅在时钟信号为 1 时根据 S 和 R 的值在时钟信号的下降沿发生变化。**因为 c 由 1 变 0 时从锁存器打开，复制主锁存器的值，使输出变化。这就是下降沿的由来。**

设主锁存器的输出为 Y 和 NY

20-40ns 时 主锁存器打开，R 为 0，其输出的 Y 为 0，此时从锁存器关闭，输出的 Q 和 NQ 未定义。

40-60ns 时 主锁存器关闭，此时从锁存器打开，复制主锁存器状态，即 Q=Y=0，NQ=NY=1。

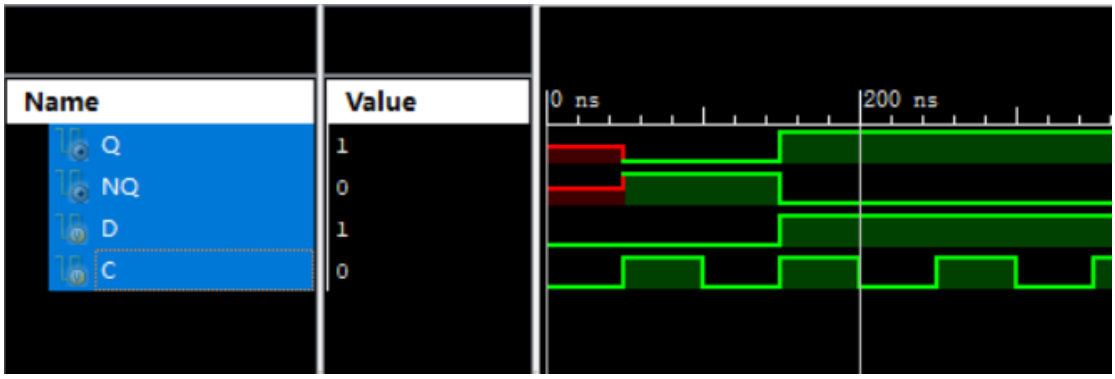
100-105ns 时 主锁存器打开，S 从 0 变为 1，其输出的 Y 变为 1，因为此时从锁存器关闭，保持原有状态，Q=0,NQ=1。

105-120ns 时 主锁存器打开，R=S=0，即 Y 保持不变，为 1，从锁存器关闭，保持原有状态，Q=0,NQ=1。

120-140ns 时 主锁存器关闭， Y=1，此时从锁存器打开，复制主锁存器状态，Q=1，NQ=0. NQ=0.

该部分即为一次性采样，导致了触发器的状态错误。

5.5 D 触发器



图表 7 D 触发器仿真图

根据 D 的值，Q 和 Qbar 的值仅在 CLK 的正边缘处变化。

因为 C 在 0 变 1 时打开了从锁存器，使它复制主锁存器内容，所以是上升沿。

设主锁存器的输出为 Y 和 NY

0-25ns 时 主锁存器打开，D=0,所以 Y=0，此时从锁存器关闭，输出未定义。

25-50ns 时 主锁存器关闭，保持原有状态，此时从锁存器开启，复制主锁存器状态，输出 Q=Y=0。

50ns-75ns 时 主锁存器打开，D 在上升沿变为 1,所以 Y=1，此时从锁存器关闭，输出保持原有状态 Q=Y=0。

75ns-100ns 时 主锁存器关闭，保持原有状态 Y=1，此时从锁存器开启，复制主锁存器状态，输出 Q=Y=1。

100ns 以后 与上述类似，不再赘述。

六、讨论、心得

这一次的实验主要是设计各种锁存器和触发器，并通过仿真加深对各个器件原理的了解。

在此次实验之前，理论课上老师已经讲过了触发器以及锁存器的原理，但是我天资愚笨，还是不曾理解，又在课后复习了很久才大致明白了他们的工作原理，但是对于上升沿和下降沿判断的原理还是很模糊。

而通过此次实验，通过对于仿真波形图的直观分析，再比对课件上的内容，我对于这块内容终于有了比较好的理解，明白了锁存器是怎么一步步加强条件，成为触发器的。这次实验的实验本身并不难，但我在课后确实花费了较长时间来分析它们并加深理解。

实验 11、同步时序电路设计

姓名：黄炯睿 学号：3170103455 专业：信息安全

课程名称：逻辑与计算机设计基础实验 同组学生姓名：无

实验时间：yyyy-mm-dd 实验地点：紫金港东 4-509 指导老师： 洪奇军

一、实验目的和要求

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、实验内容和原理

2.1 实验内容

1. 4 位二进制同步计数器
2. 4 位可逆二进制同步计数器
3. 分频器

2.2 实验原理

2.2.1 4 位二进制同步计数器

状态表：

	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

图表 1：二进制同步计数器的状态表

由状态表可知：

$$D_A = Q_A$$

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

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

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

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

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

2.2.2 可逆二进制同步计数器

1.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

2.S = 1 时，正向计数，各触发器逻辑表达式同前面

3.S = 0 时，反向计数，各触发器逻辑表达式如下式

$$D_A = \overline{Q_A}$$

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

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

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

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

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

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

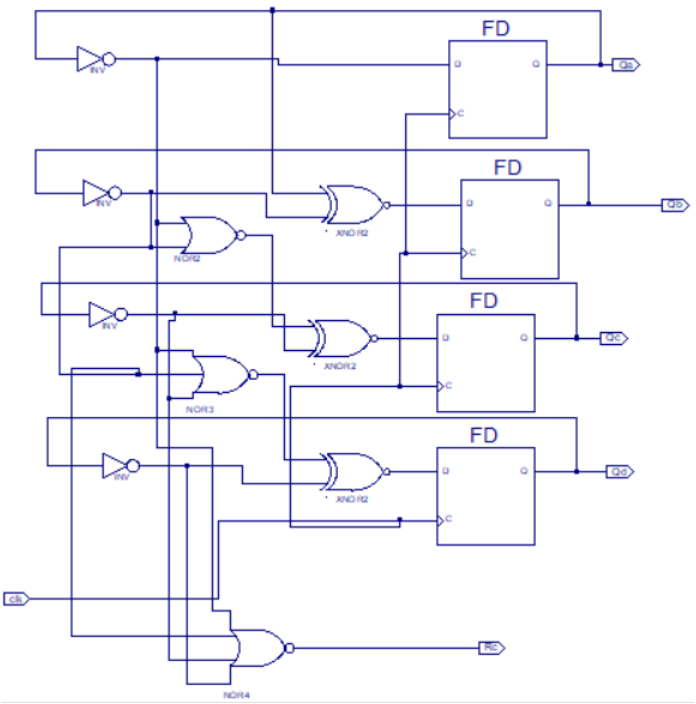
三、主要仪器设备

3. SWORD 开发板 1 套
4. 装有 ISE 的 PC 机 1 台

四、操作方法与实验步骤

4.1 设计 4 位同步二进制计数器

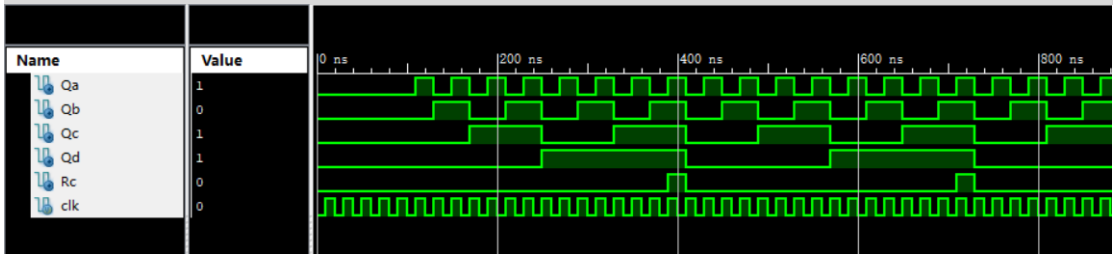
1. 新建工程 MyCounter
2. 新建源文件 Counter4b
3.用原理图进行设计，原理图如下：



图表 2 4 位同步二进制计数器的设计图

4.进行波形仿真，仿真代码如下：

```
initial forever begin
    clk = 1'b0;
    #10;
    clk = 1'b1;
    #10;
    end
```



图表 3 Counter4b 的仿真图

5. 新建源文件 clk_1s，用作时钟

其源代码如下：

```
module 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
```

6. 新建源文件 Top,其源代码如下：

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

    wire k[3:0];
    wire c;
    wire rc;

    clk_1s m1(clk,clk_1s);

    Counter4b m2(clk_1s, k[0],k[1],k[2],k[3],Rc);
```

```
        MyMC14495 m3(k[0],k[1],k[2], k[3],1'b0,1'b0,SEGMENT[0],
SEGMENT[1],SEGMENT[2],SEGMENT[3],SEGMENT[4],SEGMENT[5],SEGMENT[6
] ,SEGMENT[7]);

        assign AN = 4'b1101;
endmodule
```

7. 其引脚约束文件代码如下：

```
    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 = 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 "Rc" LOC = AB26 |IOSTANDARD = LVCMOS33 ;
```

4.2 设计 16 位可逆同步二进制计数器（bonus 并用 LED 灯显示）

1. 新建工程，工程名称用 myRevCounter。
2. 新建 Verilog 源文件，文件名称用 RevCounter。

源代码如下：

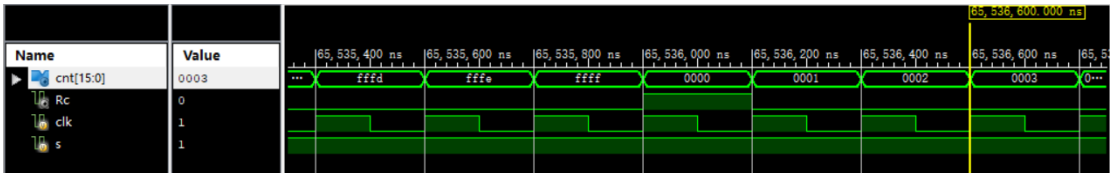
```
module RevCounter(
    input wire clk,
    input wire s,
    output reg [15:0] cnt,//次态的状态
    output reg Rc
    );
initial begin
    cnt = 0;
end
always @ (posedge clk)
begin
     if (s)
     begin
          cnt <= cnt + 1;
```

```
            if(cnt==16'b1111111111111111)
                    Rc <= 1;
                else
                    Rc <= 0;
        end
        else begin
                cnt <= cnt - 1;
                if(cnt==0) begin
                    Rc <= 1;
                end else begin
                    Rc <= 0;
                end
    end
end
endmodule
```

3.进行波形仿真

当计数为递增时的仿真代码如下：

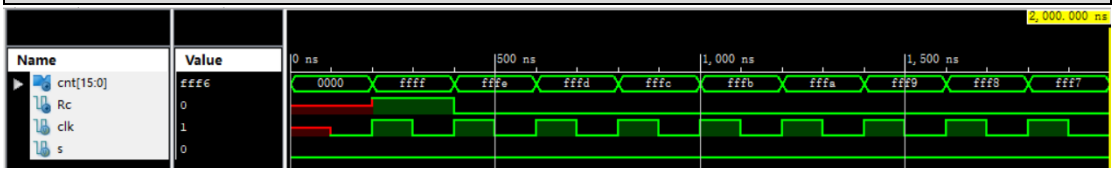
```
    initial    s = 1;
        always begin
            #100 clk = 0;
            #100 clk = 1;
        end
```



图表 4 计数为递增时的仿真图形

当计数为递减时的仿真代码如下：

```
    initial    s = 0;
        always begin
            #100 clk = 0;
            #100 clk = 1;
        end
```



图表 5 计数为递减时的仿真图形

4. 新建源文件，设计 100ms 时钟，源代码如下：

```
module clk_100ms(
    input wire clk,
    output reg clk_100ms
    );
reg [31:0] 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
```

5. 新建源文件 Top（加入了 LED 灯的显示），其代码如下：

```
module Top(input wire clk,
        input s,
        input wire [7:0]SW,
        output [3:0]AN,
        output [7:0]SEGMENT,
        output [7:0]LED,
        output wire led_clk,
        output wire led_sout,
        output wire led_clrn,
        output wire LED_EN,
        output wire seg_clk,
        output wire seg_sout,
        output wire SEG_PEN,
        output wire seg_clrn
);

wire [15:0] cnt;
wire [31:0] div;
assign LED[7:1] = SW[7:1];

clk_100ms m1(clk,clk_100ms);

RevCounter m2(clk_100ms, SW, cnt, LED[0]);

disp_num m3(clk,cnt,4'b0000,4'b0000,1'b0,AN,SEGMENT);

clkdiv m4(.clk(clk),.rst(1'b0),.clkdiv(div[31:0]));
```

```
LEDP2S #(.DATA_BITS(16),.DATA_COUNT_BITS(4),.DIR(0))
U7(.clk(clk),.rst(1'b0),.Start(div[20]),
    .PData({8'hFF,LED}),.sclk(led_clk),.sclrn(led_clrn),.sout(led_sout),.EN(LED_EN));
SSeg7_Dev
m5(.clk(clk),.rst(1'b0),.Start(div[20]),.SW0(SW[0]),.flash(1'b0),.Hexs({16'hFF_FF,cnt}),.point(
8'b0000_1111),
    .LES(8'b1111_0000), .seg_clk(seg_clk),.seg_sout(seg_sout),.SEG_PEN(SEG_PEN),.seg_c
lrn(seg_clrn));
endmodule
```

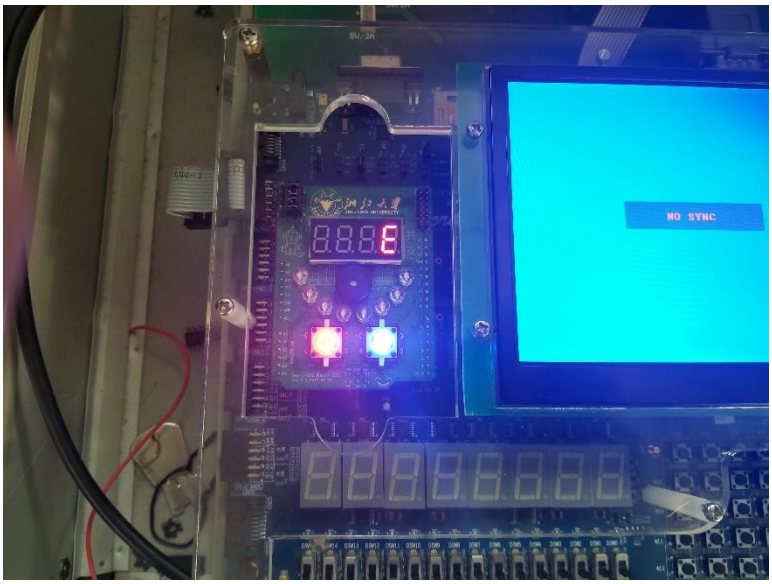
6. 其引脚约束文件如下：

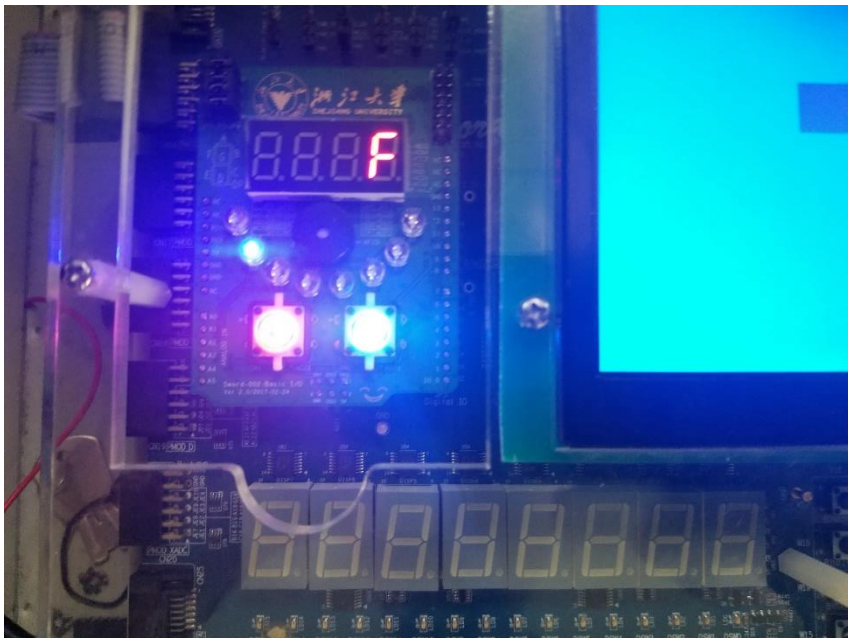
```
NET "clk" LOC = AC18 | IOSTANDARD = LVCMOS18 ;

NET "s" LOC = AF10     | IOSTANDARD = LVCMOS15;

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 "led_clk" LOC = N26 | IOSTANDARD = LVCMOS33;
NET "led_clrn" LOC = N24 | IOSTANDARD = LVCMOS33;
NET "led_sout" LOC = M26 | IOSTANDARD = LVCMOS33;
NET "LED_EN" LOC = P18 | IOSTANDARD = LVCMOS33;

NET "seg_clk" LOC = M24 | IOSTANDARD = LVCMOS33;
NET "seg_clrn" LOC = M20 | IOSTANDARD = LVCMOS33;
NET "seg_sout" LOC = L24 | IOSTANDARD = LVCMOS33;
NET "SEG_PEN" LOC = R18 | IOSTANDARD = LVCMOS33;

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;
```

五、实验结果与分析

5.1　4 位同步二进制计数器

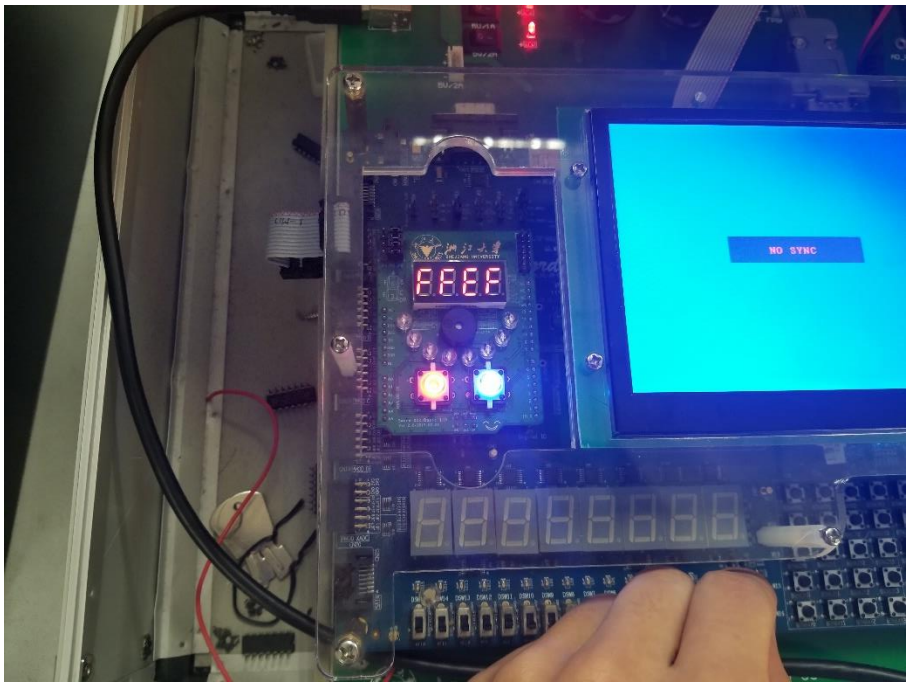


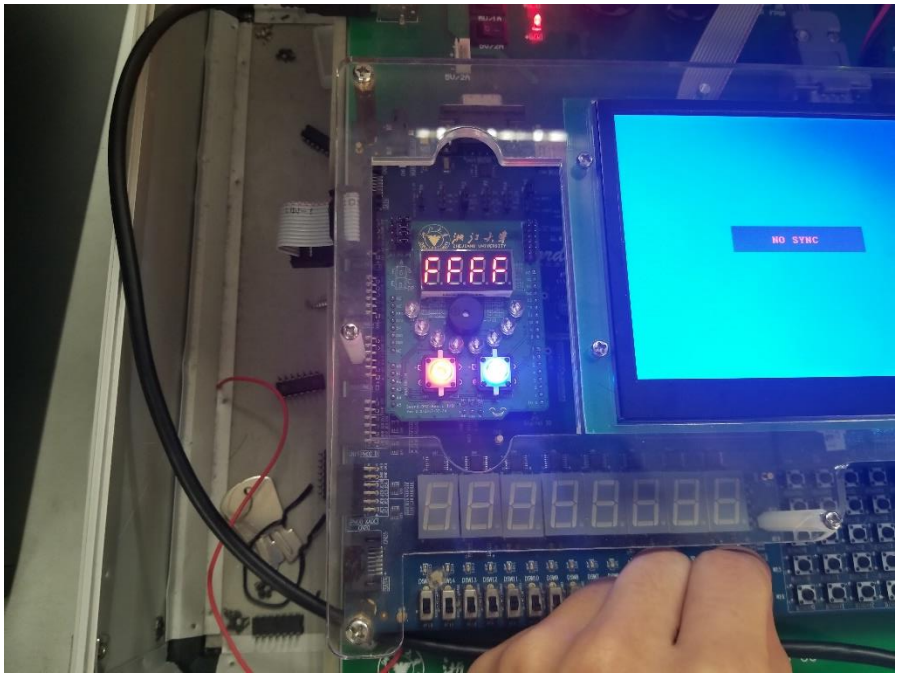


图表 6 4 位同步二进制计数器成果图

图中数字以每秒加 1 的速度递增，并在 F 进位为 0 的瞬间 LED 灯变亮，符合设计需求。

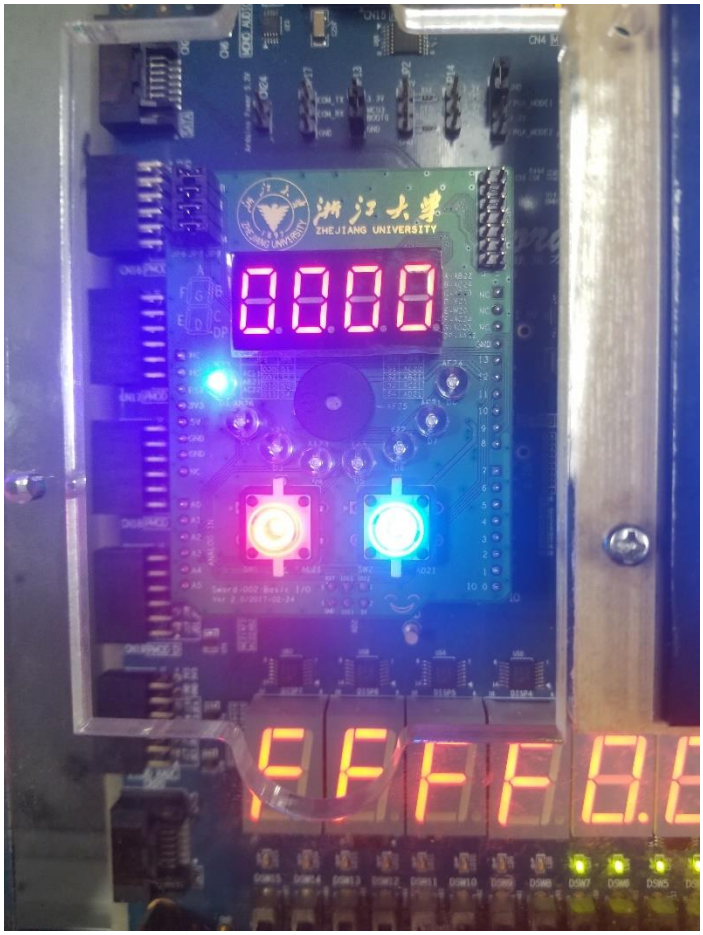
5.2　16 位可逆二进制计数器（变化最快位在第三位）





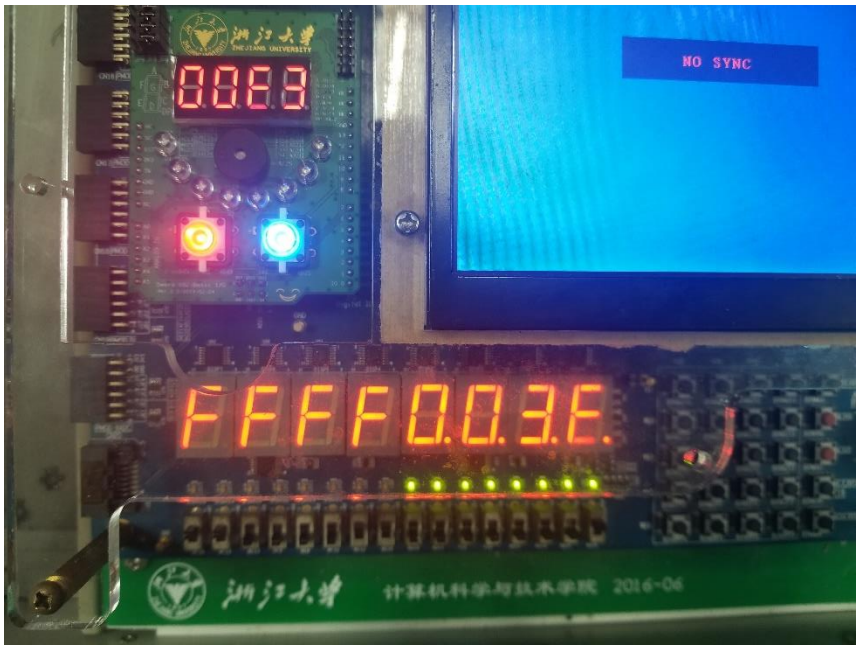
图表 6 4 位同步二进制计数器成果图

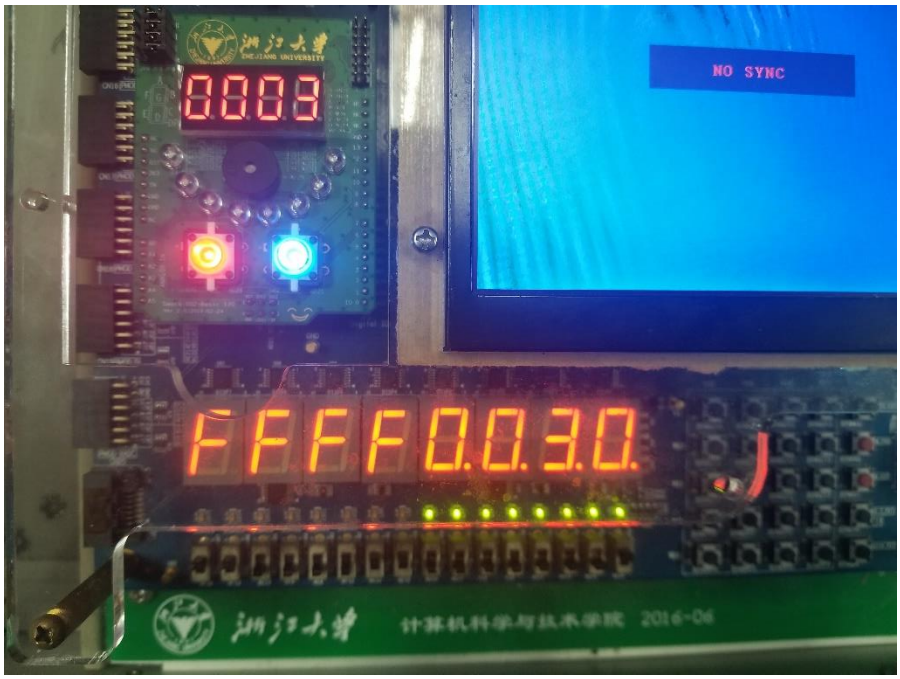
计数器可以通过开关实现正方向递增或者负方向递减，在 FFFF 变为 0000 的一刹那，数码管下的 led 灯会闪烁一次（经过无数次辛苦抓拍），其表现形式如下：



图表 7 在 FFFF 变为 0000 的刹那亮起的 LED 灯

5.3 16 位可逆二进制计数器的 bonus（同时 LED 灯闪亮）





图表 8 LED 灯同时闪亮的计时器

此时该计时器正在做递减的计时操作。

六、讨论、心得

这次实验是我少数几个做的还算顺风顺水的实验。经过这几次的实验，我发现 Verilog 语言中的 top 模块像极了 c 语言中的 main 函数。，这让我感觉，硬件语言的设计思想也是从高到低层的设计模式，先想好要实现哪些功能，在向下设计具体的各个底层的模块。

这次 bonus 代码的书写也很有意思，虽然没有很明白 LED 灯的实现原理，但能看到等的亮起也觉得很快乐。