浙江大学

课程名称：逻辑与计算机设计基础实验

姓名：施含容

学号：3180103497

专业：计算机科学与技术

指导老师：洪奇军、施青松

实验 8—全加器的设计与应用实验报告

姓名： 施含容 **学号：** 3180103497 **专业：** 计算机科学与技术

课程名称： 逻辑与计算机设计基础实验 **同组学生姓名：** 刘晓钦

实验时间： 2019-11-14 **实验地点：** 紫金港东 4-509 **指导老师：** 洪奇军、施青松

一、实验目的和要求

1.掌握一位全加器的工作原理和逻辑功能；

2.了解串行进位加法器的工作原理和进位延迟；

3.掌握超前进位的工作原理；

4.掌握加减器的实现原理；

5.掌握 ALU 冗余设计方法；

6.掌握 FPGA 开发平台进行简单的 I/O 数据交互。

二、实验内容和原理

2.1 实验内容

1.设计实现 1 位全加器并仿真验证；

2.设计四位串行进位加法器*

3.设计四超前进位模块(带组进位函数输出)；

3.设计实现四位超前进位加法器并仿真验证；

4.设计实现 32 位混合进位加法器并仿真与物理验证；

5.采用冗余方案设计 ALU 部件并物理验证；

2.2 实验原理

1.全加器原理与电路图

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-1}	S_i	C_i
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

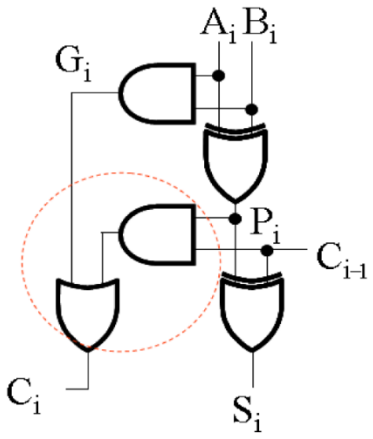


图 1 全加器真值表

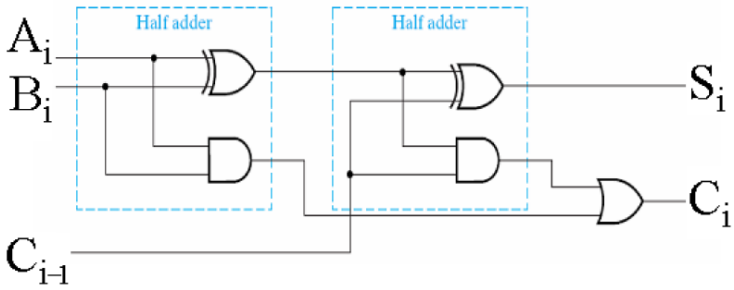


图 2 全加器电路图

2. 一位全加器的硬件描述

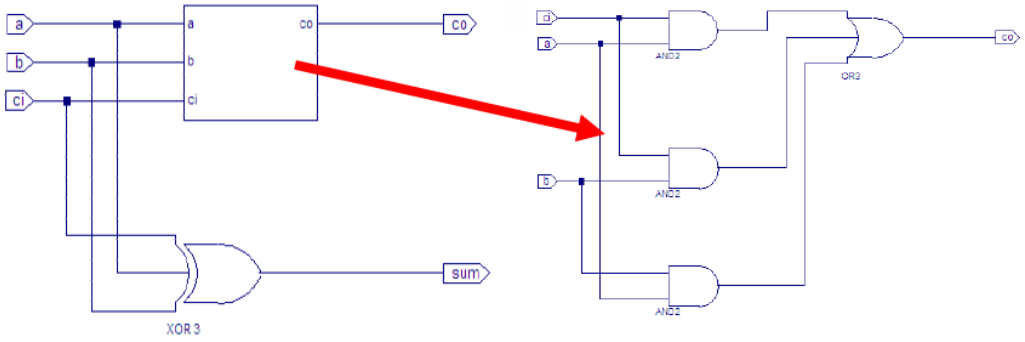


图 3 综合后电路

3. 多位串行进位全加器设计

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 CO 为 0，Ci 为高位进位输出。全加器的进位位首尾相连，但是速度慢。

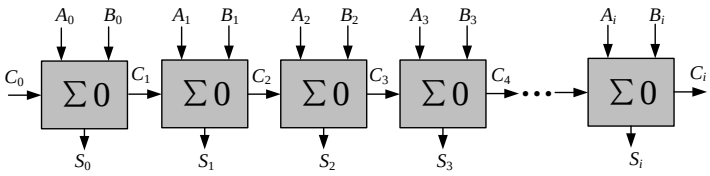


图 4 多位串行进位全加器电路

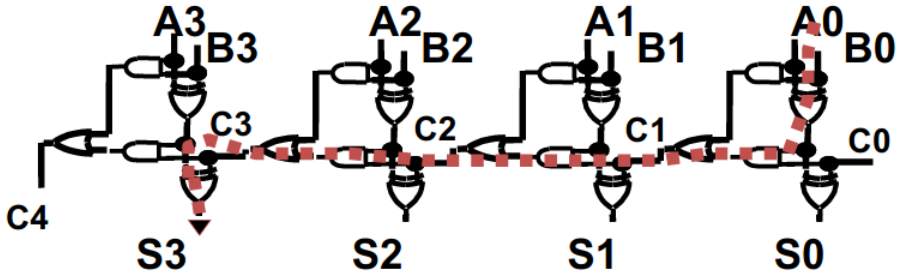


图 5 速度慢

4.32位全加器电路调用层次

顶层：32位输入，32位输出

第二层：4个串联8位全加器

第三层：8个一位串联全加器

基本逻辑块：2个与门2个异或门1个上

5. 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

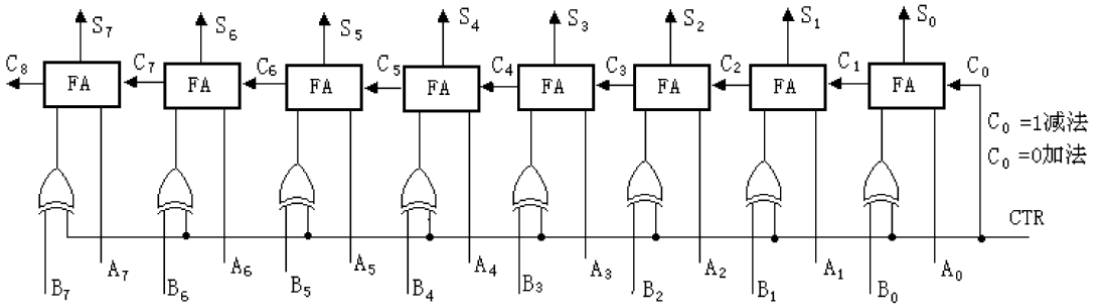


图 6 全减器电路图

6.超前进位加法器

用当前输入直接产生进位输出；输出进位只通过二级电路。

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G0 + P0\ C0) = G_1 + P_1G_0 + P_1P_0 C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2(G_1 + P_1G_0 + P_1P_0 C_0) = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0 C_0$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0 C_0 = G_{0\sim3} + P_{0\sim3}C0$$

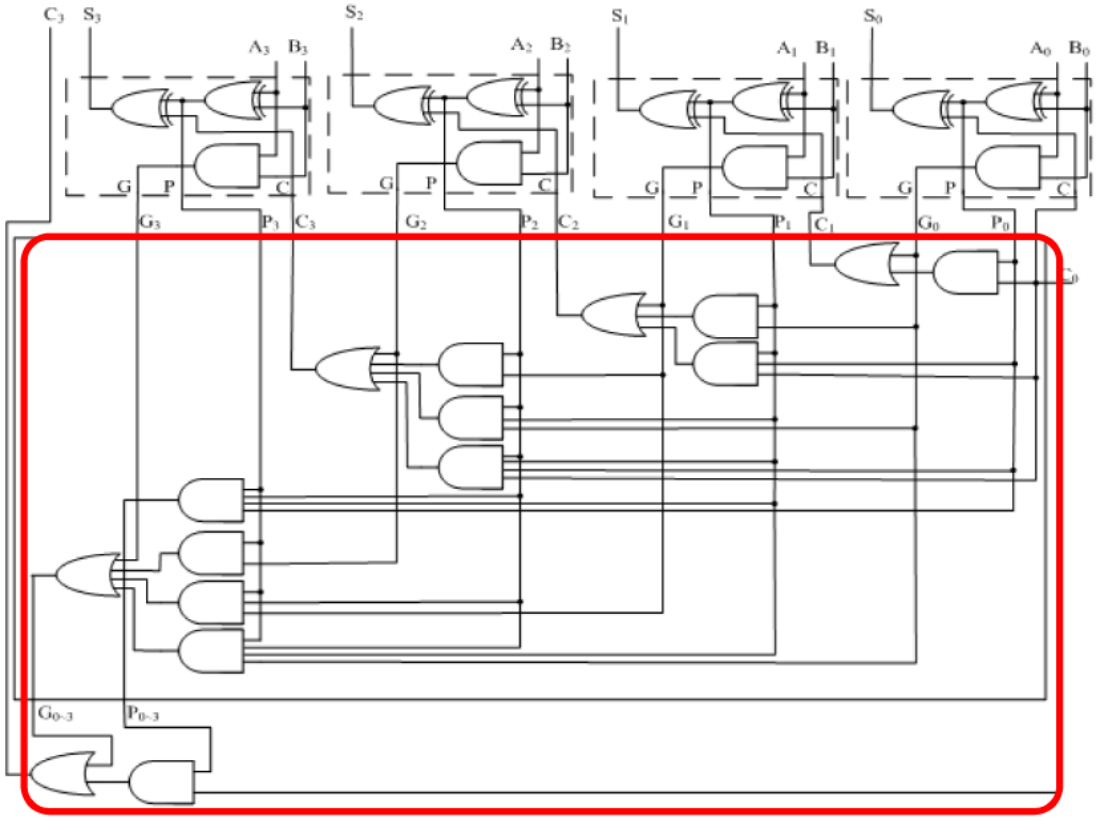


图7 4位可扩展超前进位模块

7.二层芯片实现16位超前进位输出

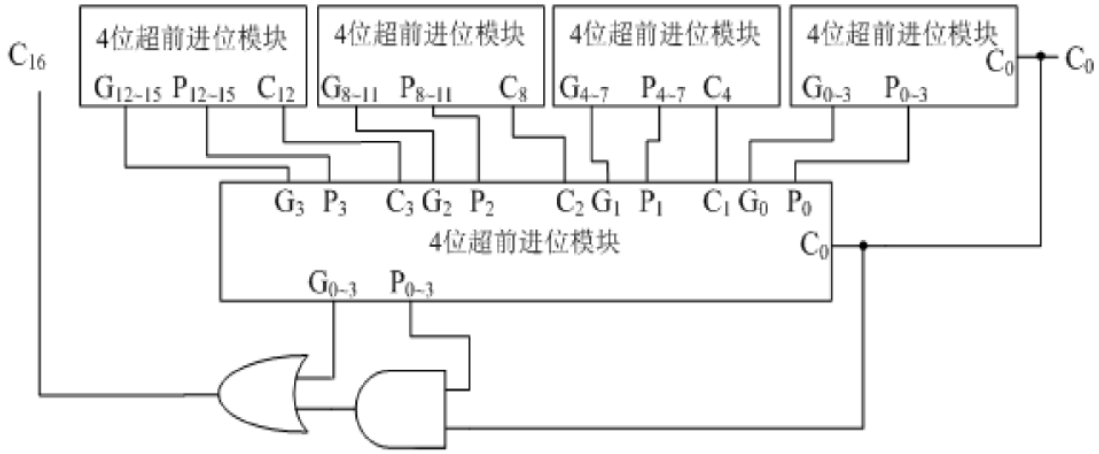


图8 16位超前进位示意图

三、主要仪器设备

1.计算机(Intel Core i5以上，4GB内存以上)系统

2.计算机软硬件课程贯通教学实验系统(Sword)

3.Xilinx ISE14.4及以上开发工具

四、操作方法与实验步骤

4.1 工程一：Exp081-ADC32

说明：逻辑图设计实现 32 位全加器

设计 1 位带进位函数的全加器并仿真验证：add.sch；设计超前进位模块：CLA.sch

扩展 4 位全加器并仿真验证：ADD4b.sch；扩展 32 位全加器并仿真验证：ADC32.sch

物理验证 32 位全加器

1.设计 1 位全加器

加法输入信号：ai、bi、ci；加法输出信号：si、co；进位函数输出信号：Gi、Pi

①原理图输入

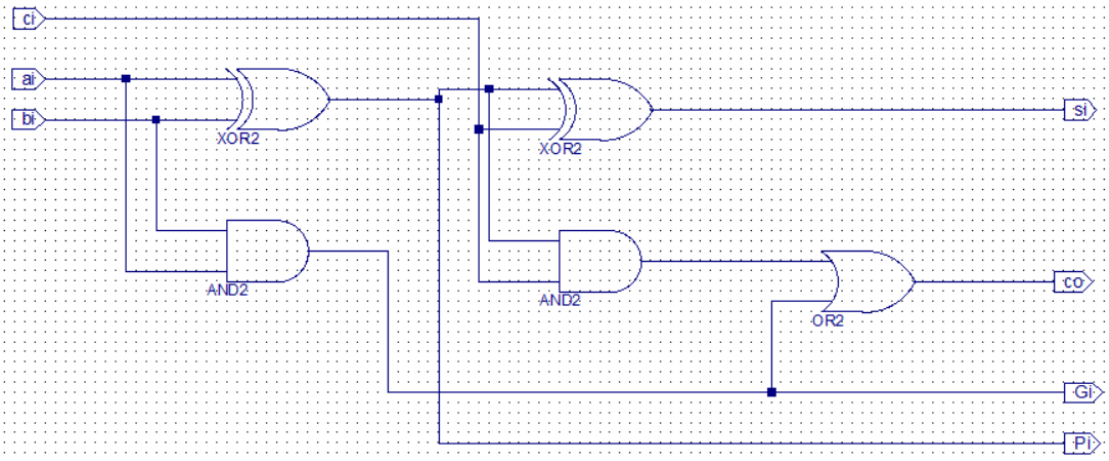


图9 1位全加器原理图输入

②设计仿真激励代码

```
integer i=0;
initial begin
    bi = 0;
    ai = 0;
    ci = 0;

    for(i=0; i<=7; i=i+1)begin
    #50;
    {ci,ai,bi} = i + 1;
```

```
    end
end
```

仿真结果：

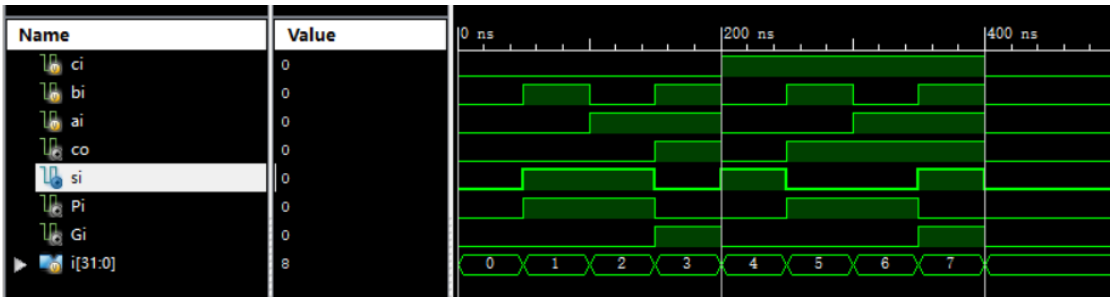


图10 1位全加器仿真结果

2. 设计4位超前进位模块

①采用原理图输入（Schematic）

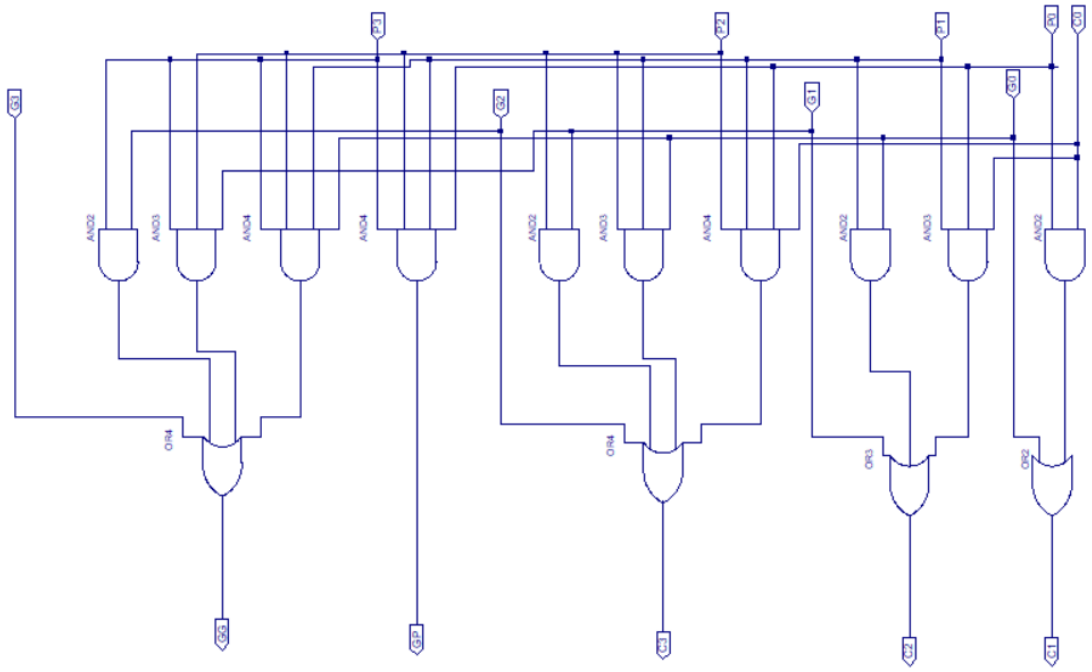


图11 4位超前进位模块原理图输入

②封装成逻辑符号

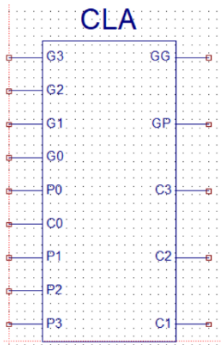


图12 4位超前进位模块逻辑符号图

3.扩展4位超前进位加法器

命名为add4b.sch，调用add和CLA模块实现。

①原理图输入

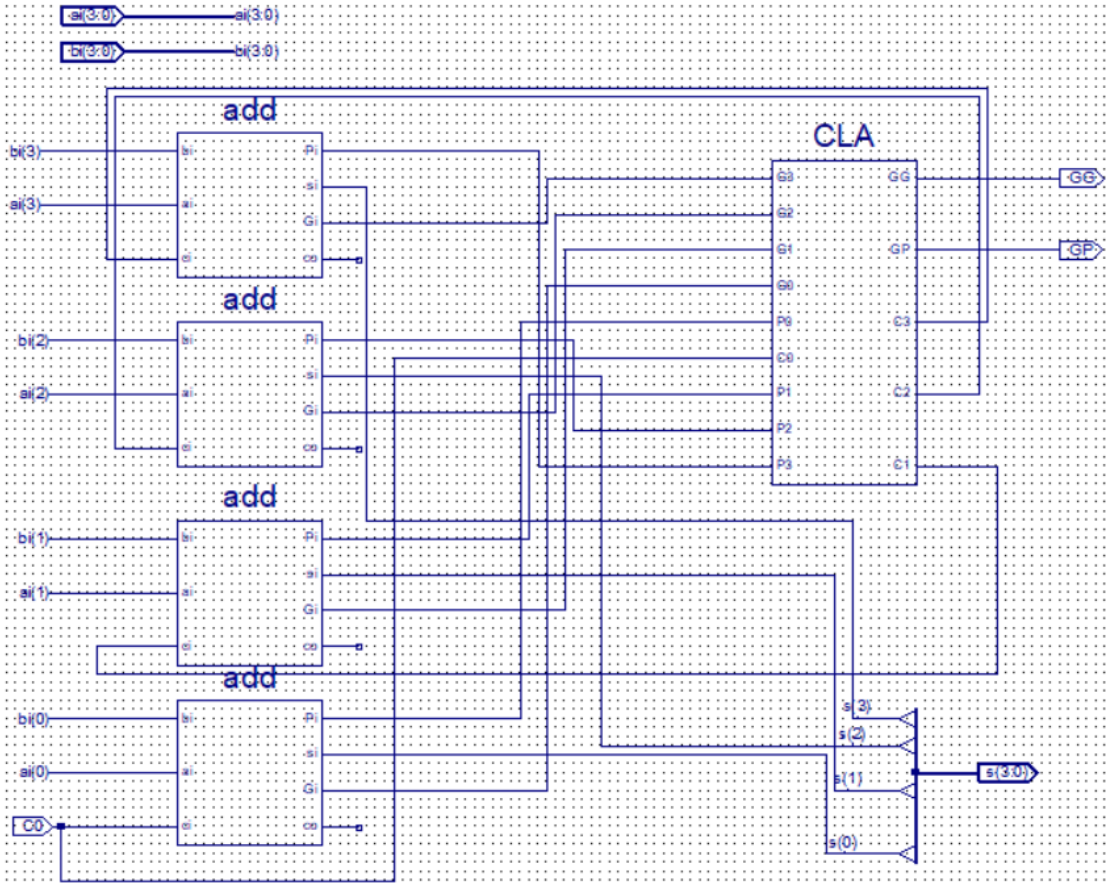


图13 4位超前进位加法器原理图

②设计仿真激励代码

```
integer i=0, j=0;
  initial begin
    ai = 0;
    bi = 0;
```

```
    C0 = 0;

    for(i=0; i<=15; i=i+1)begin
    #50;
    ai = i+1;
      for(j=0; j<=15; j=j+1)begin
      #50;
      bi = j+1;
      end
    end
  end
```

③仿真结果：

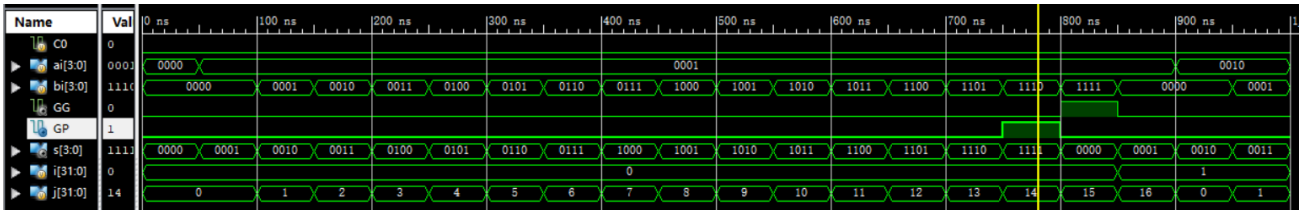


图14 4位超前进位加法器仿真结果

④封装为逻辑符号

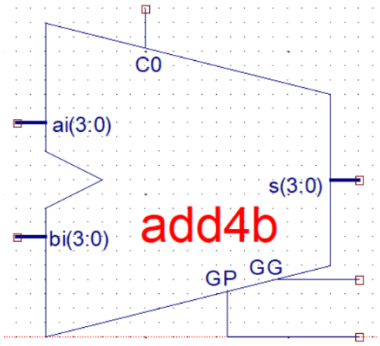


图15 4位超前进位加法器逻辑符号图

4.32位全加器设计

用4位超前进位加法器扩展 命名为：ADC32.sch,调用CLA实现16位，16位用串行

①原理图输入

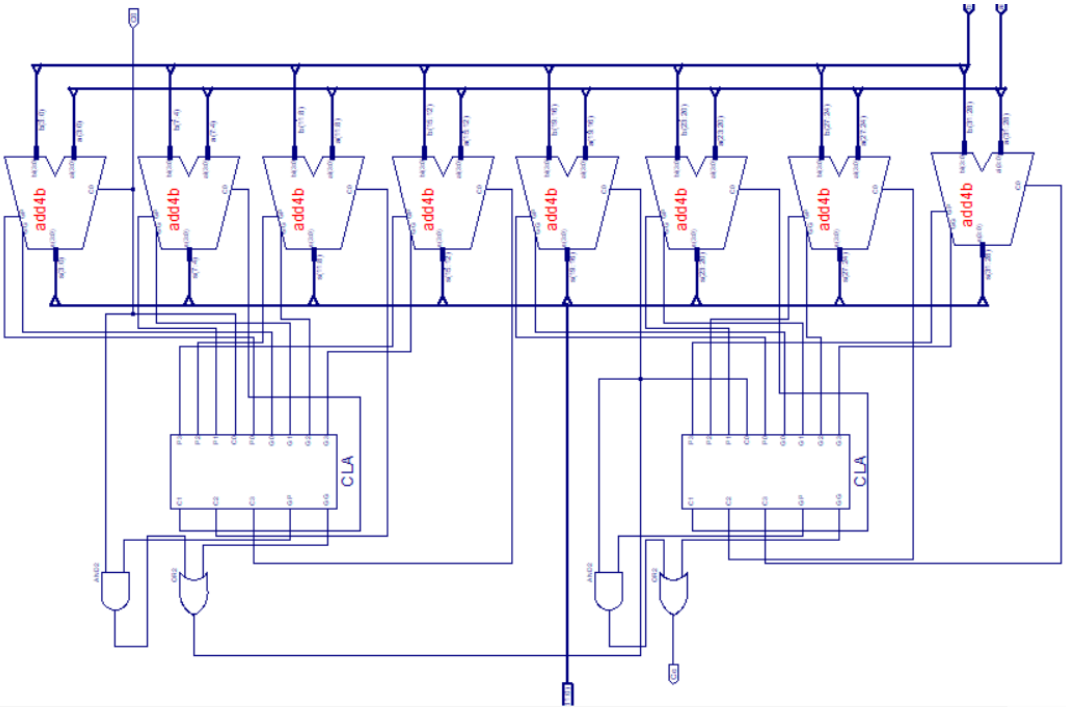


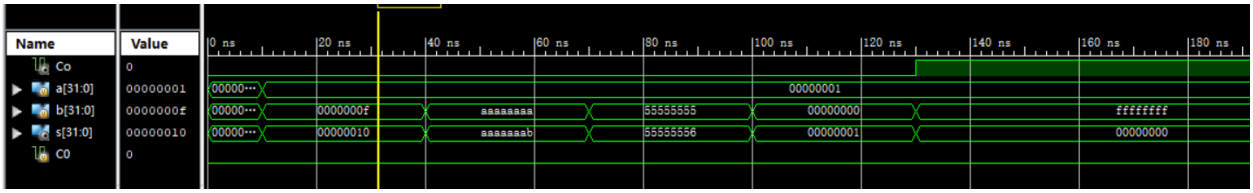
图16 32位全加器原理图

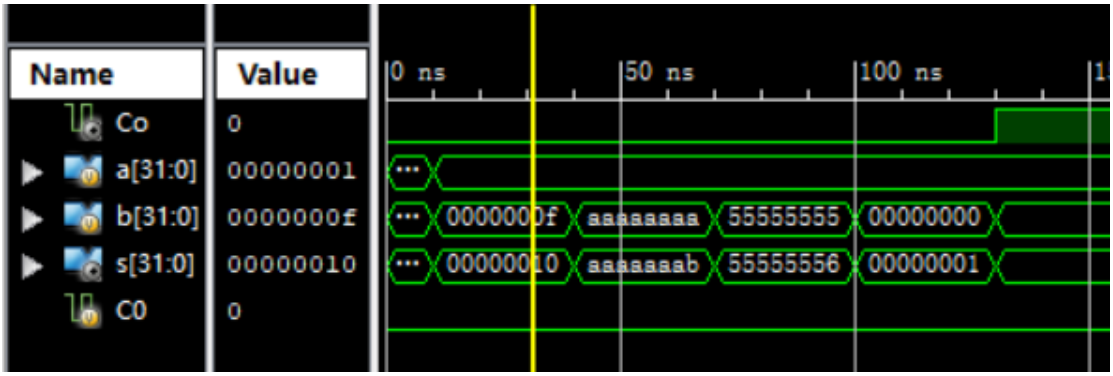
②巨量输入信号设计仿真激励

验证的完备性；数字电路验证常用数值（AAAA,5555,FFFF,0000）；根据事件特征抽样

```
initial begin
    a = 0;
    b = 0;
    C0 = 0;
    #10;
    a = 32'h1;
    b = 32'h0000000F;
    #30;
    b = 32'hAAAAAAAA;
    #30;
    b = 32'h55555555;
    #30;
    b = 32'h00000000;
    #30;
    b = 32'hFFFFFFFF;
    #30;
  end
```

③仿真结果





④封装

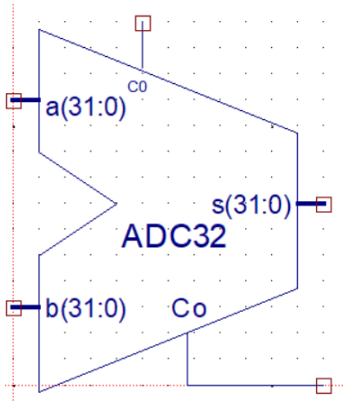


图18 32位全加器符号图

5.约束与实现

顶层采用实验7-工程4设计的测试环境模块

输入：SW[7:5]= 通道选择：输入 Ai, Bi 对应通道0/1显示和通道2

输出：LED[7]=进位C32

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

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

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

```
NET "K_ROW[4]"  LOC=W16 | IOSTANDARD=LVCMOS18;

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

NET "readn" LOC=U21 | IOSTANDARD=LVCMOS33;
NET "RDY" LOC=U22 | IOSTANDARD=LVCMOS33;
NET "CR"  LOC=V22 | IOSTANDARD=LVCMOS33;

NET "SEGCLK"  LOC=M24 | IOSTANDARD=LVCMOS33;
NET "SEGCLR"  LOC=M20 | IOSTANDARD=LVCMOS33;
NET "SEGDT" LOC=L24 | IOSTANDARD=LVCMOS33;
NET "SEGEN" LOC=R18 | IOSTANDARD=LVCMOS33;

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

NET "SW[0]" LOC=AA10  | IOSTANDARD=LVCMOS15;
NET "SW[1]" LOC=AB10  | IOSTANDARD=LVCMOS15;
NET "SW[2]" LOC=AA13  | IOSTANDARD=LVCMOS15;
NET "SW[3]" LOC=AA12  | IOSTANDARD=LVCMOS15;
NET "SW[4]" LOC=Y13 | IOSTANDARD=LVCMOS15;
NET "SW[5]" LOC=Y12 | IOSTANDARD=LVCMOS15;
NET "SW[6]" LOC=AD11  | IOSTANDARD=LVCMOS15;
NET "SW[7]" LOC=AD10  | IOSTANDARD=LVCMOS15;
NET "SW[8]" LOC=AE10  | IOSTANDARD=LVCMOS15;
NET "SW[9]" LOC=AE12  | IOSTANDARD=LVCMOS15;
NET "SW[10]"  LOC=AF12  | IOSTANDARD=LVCMOS15;
NET "SW[11]"  LOC=AE8 | IOSTANDARD=LVCMOS15;
NET "SW[12]"  LOC=AF8 | IOSTANDARD=LVCMOS15;
NET "SW[13]"  LOC=AE13  | IOSTANDARD=LVCMOS15;
NET "SW[14]"  LOC=AF13  | IOSTANDARD=LVCMOS15;
NET "SW[15]"  LOC=AF10  | 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=AD21  | IOSTANDARD=LVCMOS33;
NET "AN[1]" LOC=AC21  | IOSTANDARD=LVCMOS33;
NET "AN[2]" LOC=AB21  | IOSTANDARD=LVCMOS33;
NET "AN[3]" LOC=AC22  | IOSTANDARD=LVCMOS33;

NET "LED[0]"  LOC=AB26  | IOSTANDARD=LVCMOS33;
NET "LED[1]"  LOC=W24 | IOSTANDARD=LVCMOS33;
NET "LED[2]"  LOC=W23 | IOSTANDARD=LVCMOS33;
NET "LED[3]"  LOC=AB25  | IOSTANDARD=LVCMOS33;
NET "LED[4]"  LOC=AA25  | IOSTANDARD=LVCMOS33;
NET "LED[5]"  LOC=W21 | IOSTANDARD=LVCMOS33;
NET "LED[6]"  LOC=V21 | IOSTANDARD=LVCMOS33;
NET "LED[7]"  LOC=W26 | IOSTANDARD=LVCMOS33;
```

6.下载 .bit 文件

①流代码生成

生成能下载到硬件中的二进制比特流文件。双击过程管理区的Generate Programming File，ISE 生成相应的二进制比特文件HCT138_sch.bit。

②下载FPGA编程流代码

7.加入实验7的测试环境模块物理验证

4.2 工程二：Exp082-ALU

采用冗余结构设计ALU

在全加器基础融合设计无符号数减法；用并行冗余法扩展32逻辑运算与加减法器合成ALU功能：加、减、与、或和比较；自由扩展功能3个

调用实验七的MUX8T1_8选择器

ALU实验可以用代码描述或逻辑图描述；物理验证32位ALU

1.定制32位逻辑运算模块符号

①复制HCT138和Decoder_38逻辑符号到当前工程根目录 HCT138.sym、Decoder_38.sym

②复制Decoder_38.sch和HTC138_sch.sch代码存放目录

2.代码设计ALU模块

```
module ALU( input [31:0] A,
        input [31:0] B,
        input [2:0] ALU_Ctr,
        output overflow,
        output [31:0] res,
        output Co,
        output zero
    );
wire [31:0] Sum, Bo, And, Or, Xor, Nor, Slt, Srl;
  wire sub = ALU_Ctr[2];
  assign Bo = B ^ {32{sub}};

  ADC32 ADD_32( .a(A),
            .b(Bo),
            .C0(sub),
            .s(Sum),
            .Co(Co)
            );
  and32 u1  ( .A(A),
            .B(B),
            .rst(And)
            );
  or32  u2  ( .A(A),
            .B(B),
            .rst( Or)
            );
  xor32 u3  (.A(A),
            .B(B),
            .rst(Xor)
            );
  nor32 u4  ( .A(A),
            .B(B),
            .rst(Nor)
            );
  srl32 u5  (.A(A),
            .B(B),
            .rst(Srl)
            );
  slt32 u6  ( .A(A),
            .B(B),
            .rst(Slt)//if A<B return 1, else return 0
```

```
            );
  Mux8T1_32 u7( .I0(And),
            .I1(Or),
            .I2(Sum),
            .I3(Xor),
            .I4(Nor),
            .I5(Srl),//nand
            .I6(Sum),
            .I7(Slt),
            .s(ALU_Ctr),
            .o(res)
            );
endmodule
```

其中，小模块代码分别如下：

① and32

```
module and32( input [31:0] A,
          input [31:0] B,
          output reg [31:0] rst
        );
always @*begin
rst <= A & B;
end
endmodule
```

② or32

```
module or32(  input [31:0] A,
          input [31:0] B,
          output wire [31:0] rst
        );

assign rst = A | B;
endmodule
```

③ xor32

```
module xor32( input [31:0] A,
          input [31:0] B,
          output reg [31:0] rst
        );
always @*begin
rst <= A ^ B;
```

```
end
endmodule
```

④ nor32

```
module nor32( input [31:0] A,
          input [31:0] B,
          output reg [31:0] rst
        );
always @*begin
rst <= ~(A | B);
end
endmodule
```

⑤ srl32

```
module srl32( input [31:0] A,
          input [31:0] B,
          output reg [31:0] rst
        );
always @*begin
    rst <= ~(A & B);
  end
endmodule
```

⑥ slt32

```
module slt32( input [31:0] A,
        input [31:0] B,
        output reg [31:0] rst
    );
always @* begin
  if(A<B) rst <= 1;
  else rst <= 0;
end
endmodule
```

3.UCF引脚定义

```
NET "clk_100mhz"  LOC=AC18  | IOSTANDARD=LVCMOS18;
NET "clk_100mhz"  TNM_NET=TM_CLK;
```

```
TIMESPEC TS_CLK_100M = PERIOD "TM_CLK" 10 ns HIGH 50%;

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

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

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

NET "readn" LOC=U21 | IOSTANDARD=LVCMOS33;
NET "RDY" LOC=U22 | IOSTANDARD=LVCMOS33;
NET "CR"  LOC=V22 | IOSTANDARD=LVCMOS33;

NET "SEGCLK"  LOC=M24 | IOSTANDARD=LVCMOS33;
NET "SEGCLR"  LOC=M20 | IOSTANDARD=LVCMOS33;
NET "SEGDT" LOC=L24 | IOSTANDARD=LVCMOS33;
NET "SEGEN" LOC=R18 | IOSTANDARD=LVCMOS33;

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

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

```
NET "SW[13]"  LOC=AE13  | IOSTANDARD=LVCMOS15;
NET "SW[14]"  LOC=AF13  | IOSTANDARD=LVCMOS15;
NET "SW[15]"  LOC=AF10  | 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=AD21  | IOSTANDARD=LVCMOS33;
NET "AN[1]" LOC=AC21  | IOSTANDARD=LVCMOS33;
NET "AN[2]" LOC=AB21  | IOSTANDARD=LVCMOS33;
NET "AN[3]" LOC=AC22  | IOSTANDARD=LVCMOS33;

NET "LED[0]"  LOC=AB26  | IOSTANDARD=LVCMOS33;
NET "LED[1]"  LOC=W24 | IOSTANDARD=LVCMOS33;
NET "LED[2]"  LOC=W23 | IOSTANDARD=LVCMOS33;
NET "LED[3]"  LOC=AB25  | IOSTANDARD=LVCMOS33;
NET "LED[4]"  LOC=AA25  | IOSTANDARD=LVCMOS33;
NET "LED[5]"  LOC=W21 | IOSTANDARD=LVCMOS33;
NET "LED[6]"  LOC=V21 | IOSTANDARD=LVCMOS33;
NET "LED[7]"  LOC=W26 | IOSTANDARD=LVCMOS33;
```

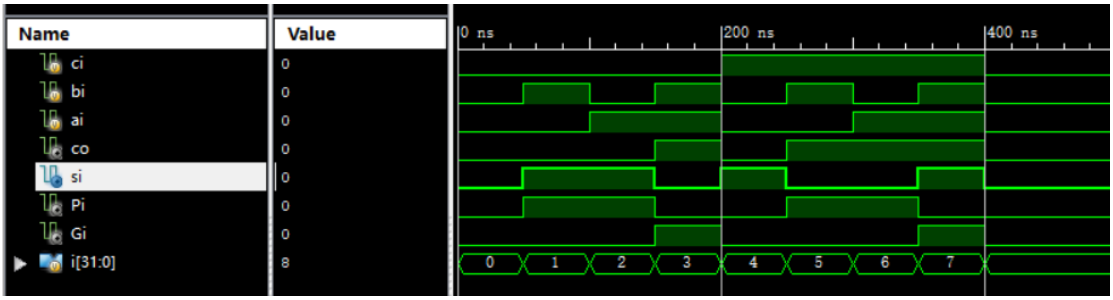
4.综合

五、实验结果与分析

5.1 工程一：Exp081-ADC32

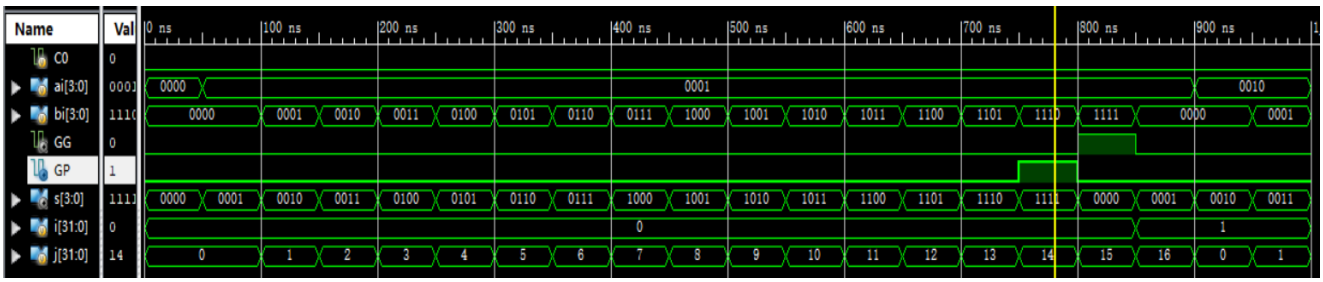
实验结果

① 1位全加器仿真结果



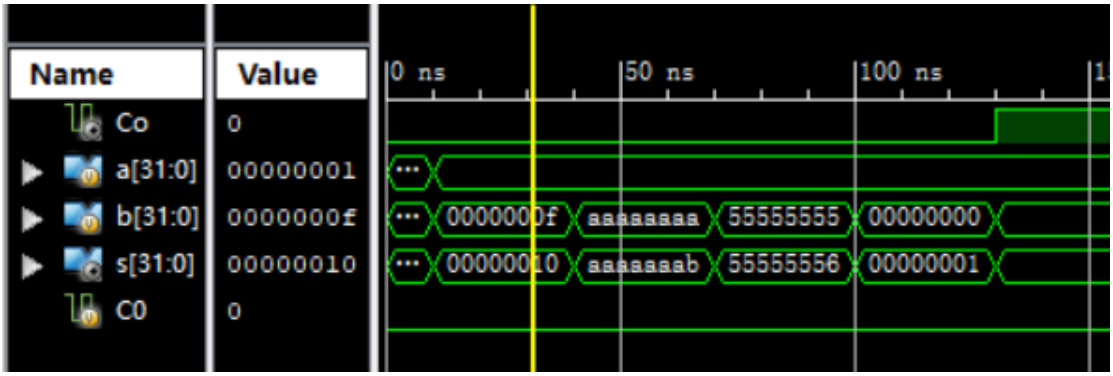
实验分析：co、si 作为本实验的输出，输出结果依次为00,01,10,11，符合一位加法器基本要求。

② 4位全加器仿真结果



实验分析：P3P2P1P0 = 1传递Co，输出结果正确显示 4 位数据加法器结果。当 ai = 1，bi = 0000，0001，0010 逐次累加时，输出结果正常显示。当黄色标线处，GP 产生一个进位。

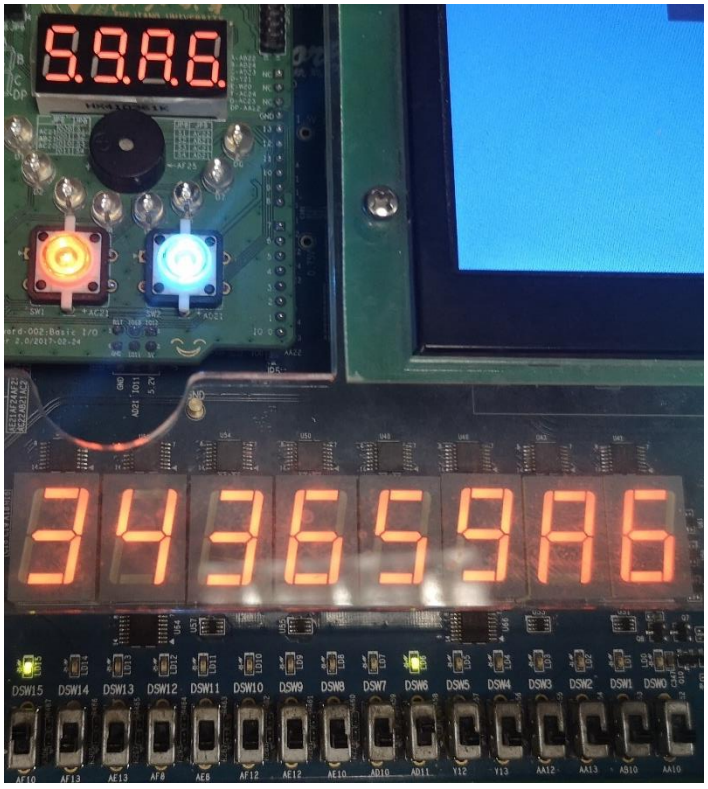
③ 32位全加器仿真结果



实验分析：对于巨量输入信号，本实验采取特征抽样法进行检验。输入ai = 1，bi = 0000 000f，aaaa aaaa，5555 5555，0000 0000，ffff ffff。S输出结果分别为0000 0010，aaaa aaab，5555 5556，0000 0001，0000 0000。输出结果正确，表明本加法器设计正确。

④物理验证

以上为两个输入数据，输出数据见下

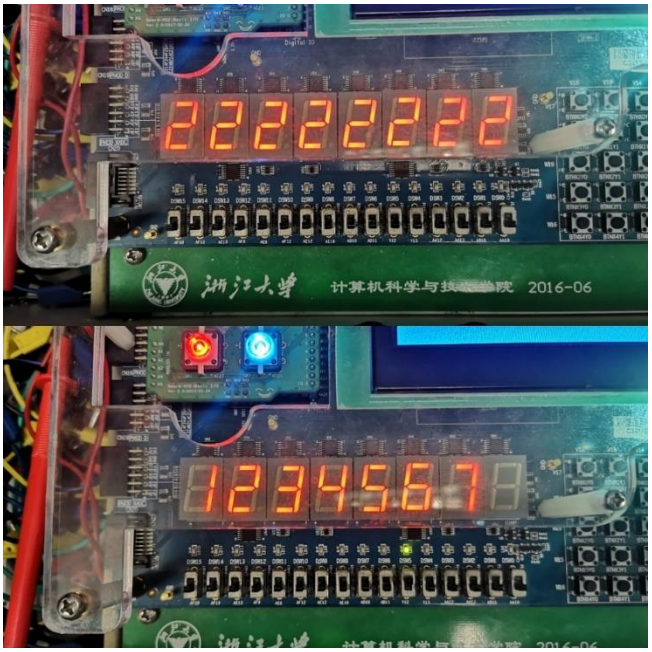


实验结果分析：SW[7:5]控制输入加数与被加数，本实验以输入1111 1111与2325 4895为例，输出结果为3436 59A6，经检验，结果正确。

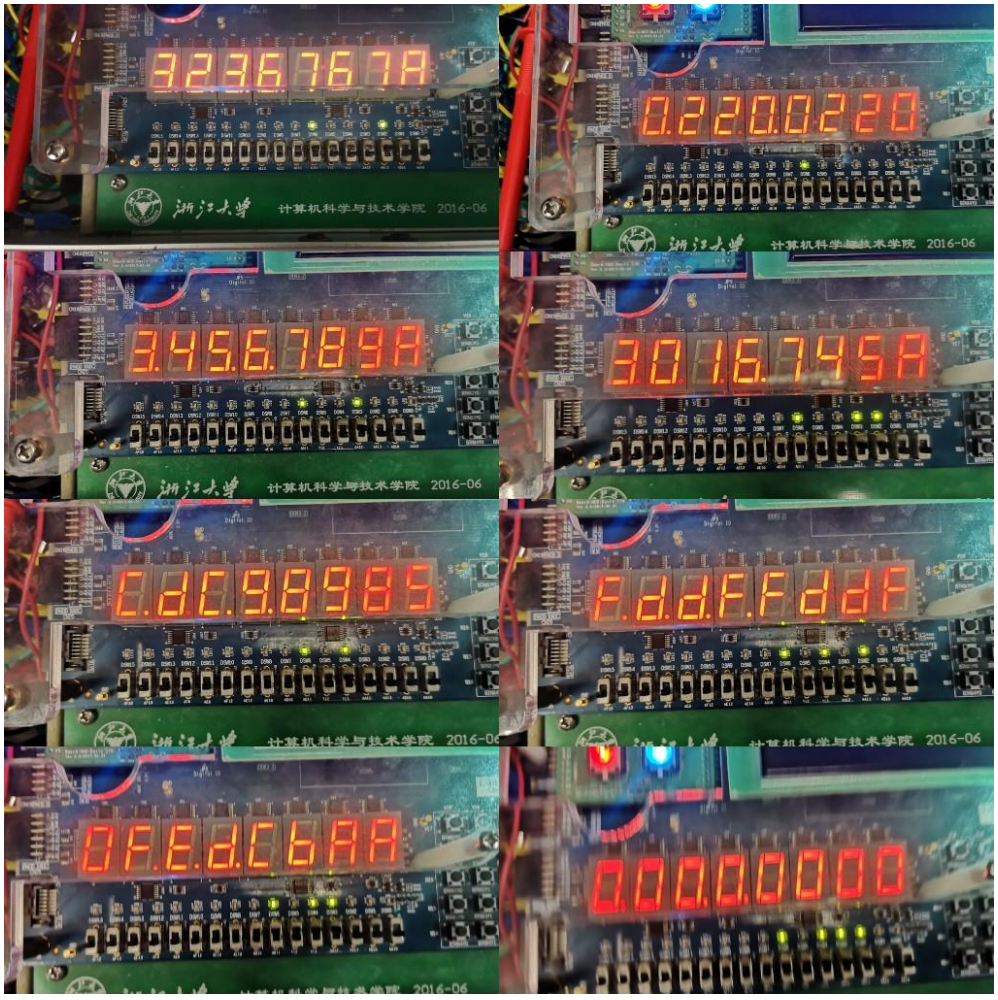
5.2 工程二：Exp082-ALU

5.2.1.实验结果

首先，输入数据Ai = 2222 2222, Bi = 1234 5678



其次，通过控制开关SW[4:2],分别实现与，或，加，异或，或非，与非，减法，比较功能，实验结果如下：



5.2.2 实验结果分析

输入Ai = 2222 2222, Bi = 1234 5678

ALU功能选择	=0（XXX）	=1(功能)	输出结果验证
SW[4:2] = ALU_Ctr(2:0)	000	AND	0220 0220
	001	OR	3236 767A
	010	ADD	3456 789A
	011	XOR	3016 745A
	100	NOR	CDC9 8985
	101	NAND	FDDF FDDF
	110	SUB	OFED C6AA
	111	SLT(A < B,A = 1, otherwise, A = 0)	0000 0000

经检验，所有输出结果均正确。

六、讨论与心得

本实验设计了 ALU 模块，在实验 7 framework 的基础上本实验设计并不复杂。最大的问题还是在于实验 7 framework 的搭建。一开始，我的显示很不正常，每次 8 位数码管都输出奇怪的结果，而非预想中的 1234 5678。经过仔细的分析和研究，我发现 4 位数码管显示正常，因此认定为数据通路数据传输正常。最后发现，是我在设计显示模块时，习惯性思维将低位与低位相接。但是sword 板上，需要将最低位与高位相接，一个相反的逻辑构造，最后输出显示正常。

有了实验 7 框架作为基础，实验 8 先是设计一个 32 位全加器，通过学习写仿真代码，我掌握了全加器的原理，并最终取得了预想的结果。后需要通过 3 位开关控制，实现 ALU 的 8 个功能。从实验 8 开始，我逐渐脱离了原理图输入，转而用代码实现预想的结构。通过设计与分析，最终顺利完成实验。

通过本次实验，我进一步理解了加法器的原理以及组合电路设计的原理及步骤。

实验 9—锁存器与触发器实验报告

姓名： 施含容 **学号：** 3180103497 **专业：** 计算机科学与技术

课程名称： 逻辑与计算机设计基础实验 **同组学生姓名：** 刘晓钦

实验时间： 2019-11-21 **实验地点：** 紫金港东 4-509 **指导老师：** 洪奇军、施青松

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. 了解静态存储器 SRAM 存储单元结构。
4. 掌握基本 RS 锁存器的基本功能与使用要点。
5. 掌握 RS、D 触发器的基本功能及基本应用。
6. 掌握集成触发器的使用和异步清零的作用
7. 了解用 D 触发器实现分频电路；
8. 了解用 D 触发器构成单稳态电路(开关去抖动)。

二、实验内容和原理

2.1 实验内容

1. 用原理图实现RS锁存器并仿真验证；
2. 实现门控RS锁存器、D锁存器并仿真验证；
3. 用RS锁存器实现RS主从触发器并仿真验证；
4. 用D锁存器和用RS锁存器实现主从D触发器并仿真验证;
5. 用原理图实现维持阻塞型D触发器；
6. 触发器物理测试

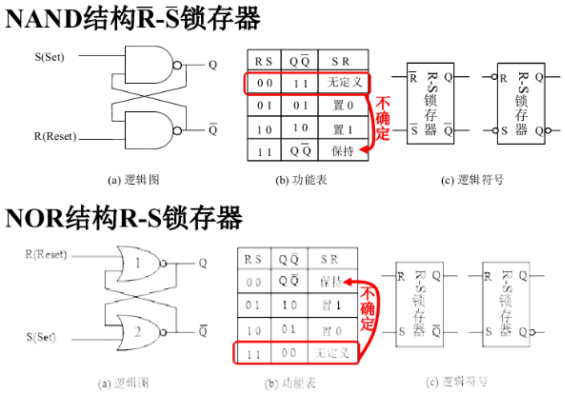
2.2 实验原理

1.什么是锁存器

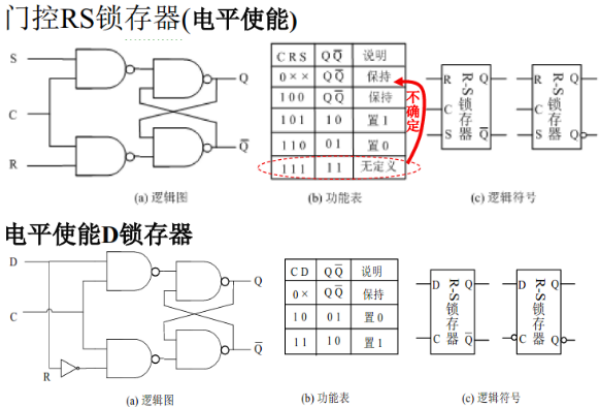
① 锁存器三个基本条件：能长期保持给定的某个稳定状态；有两个稳定状态，“0”、“1”；在一定条件下能随时改变状态，即：set 置“1” 或 reset 置“0”。

② 最基本的锁存器有 R-S 锁存器和 D 锁存器。用反向门互锁：实现长期保持稳定状态；使能(enable)NAND 或 NOR 可实现；破坏互锁实现 set 或 reset；禁止(disable) NAND 或 NOR 可实现；只有二个状态。

2.最基本 RS 锁存器



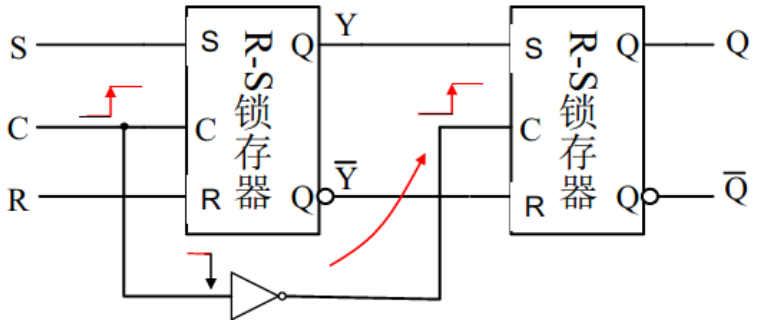
3.使能控制



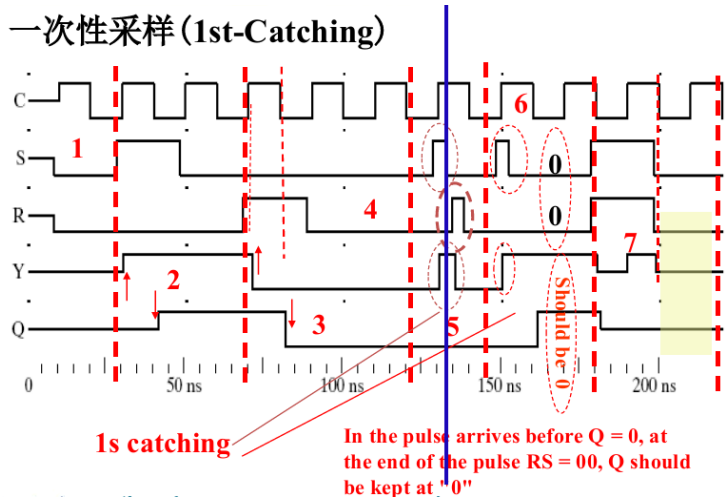
4.主从触发器

①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” 无法控制；解决思想：切断回路，同步变化一次→触发

②主从触发器：用使能信号控制回路，切断直接通路

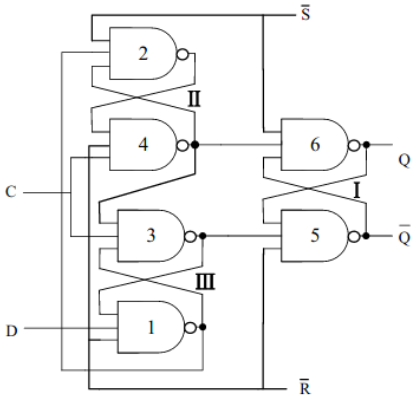


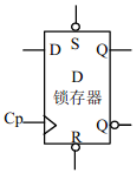
5. RS主从触发器存在的问题



6. 边沿触发器

维持阻塞型边沿触发器（正边沿）。缩小采用窗口：边沿采样。



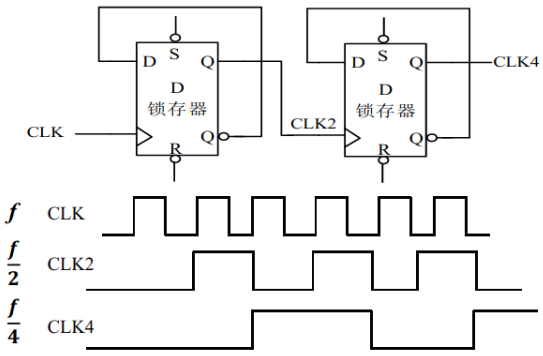


异步控制		上升沿触发			
$\overline{R}$	$\overline{S}$	Cp	D	Q	$\overline{Q}$
0	1	X	X	0	1
1	0	X	X	1	0
1	1	↑	0	0	1
1	1	↑	1	1	0

7. D触发器应用

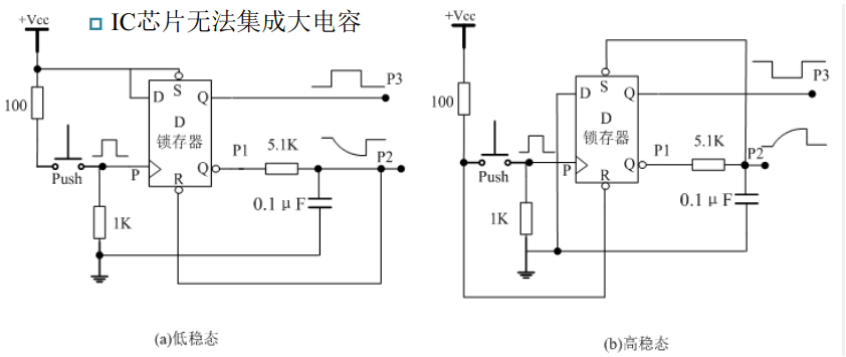
①分频电路

多个触发器一维调用（串联）。时钟作为边界条件：前面输出 Q 接入后面时钟输入。



②单稳态电路

开头去抖动。需要电阻、电容构成充放电延时。FPGA 只能用计数延时。



三、主要仪器设备

1.计算机(Intel Core i5以上，4GB内存以上)系统

2.计算机软硬件课程贯通教学实验系统(Sword)

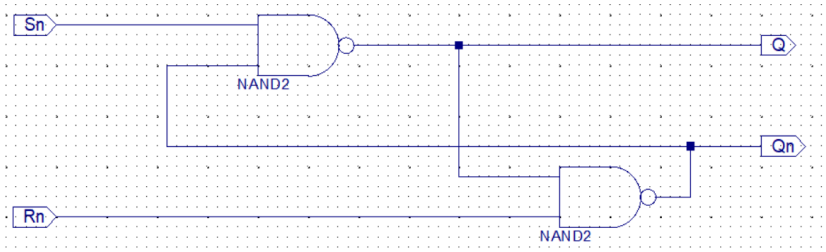
3.Xilinx ISE14.4及以上开发工具

四、操作方法与实验步骤

4.1 工程一：Locker

1.设计RS锁存器

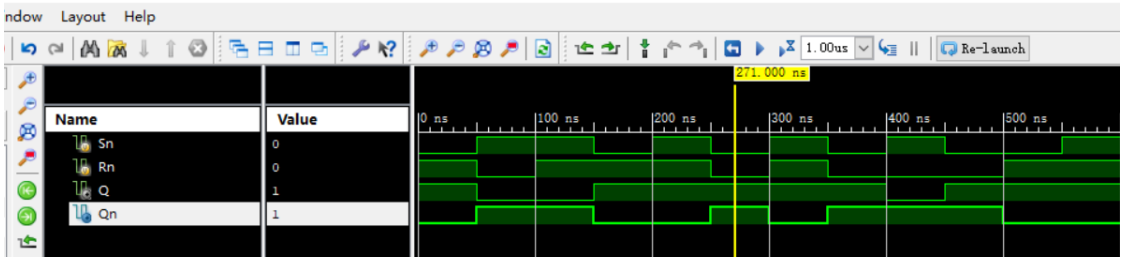
①调用与非门用原理图实现。



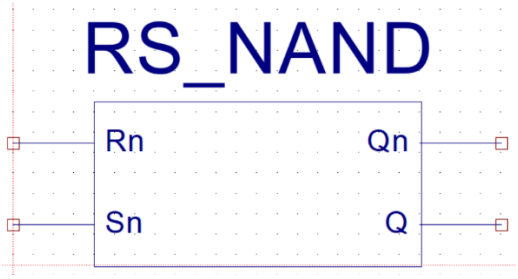
②设计激励代码并仿真测试

```
initial begin
    Rn = 1;
    Sn = 0;
    #50;
    Rn = 0;
    Sn = 1;
    #50;
    Rn = 1;
    Sn = 1;
    #50;
    Rn = 1;
    Sn = 0;
    #50;
    Rn = 1;
    Sn = 1;
    #50;
    Rn = 0;
    Sn = 0;
    #50;
    Rn = 1;
    Sn = 1;
    #50;
    Rn = 0;
    Sn = 0;
    #50;
    Rn = 0;
    Sn = 1;
    #50;
    Rn = 0;
    Sn = 0;
    #50;
    Rn = 1;
    Sn = 0;
    #50;
    Rn = 1;
    Sn = 1;
    #50;
  end
```

③仿真测试结果如下

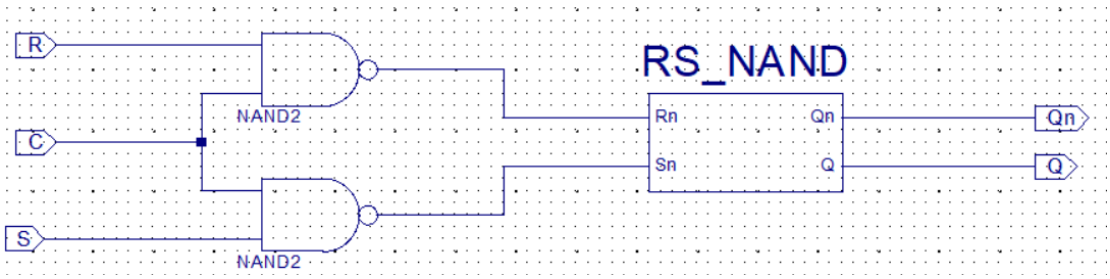


④仿真测试通过后封装逻辑符号



2. RS锁存器增加电平使能控制

①调用 RS_NAND 实现，用原理图实现



②设计仿真激励代码

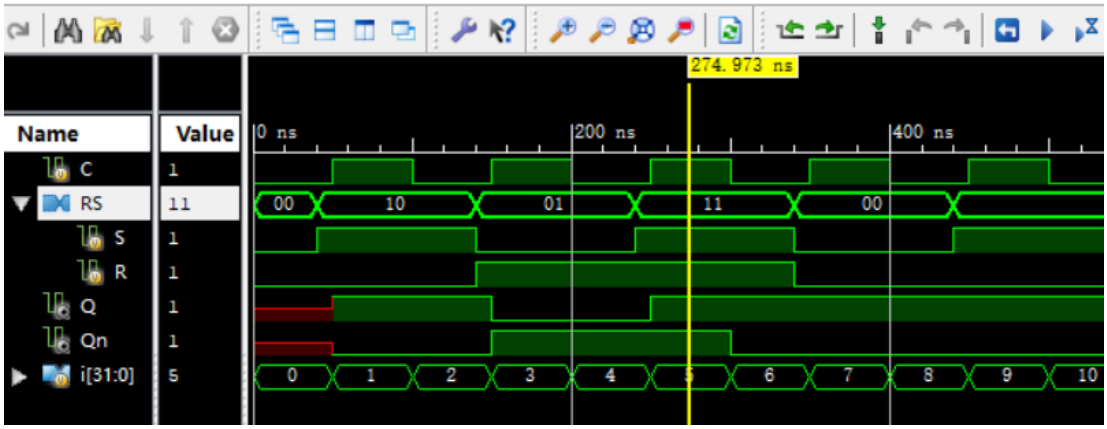
使能控制采用方波信号，功能测试输入：set, reset, hold

特殊状态输入：undefined, unknown

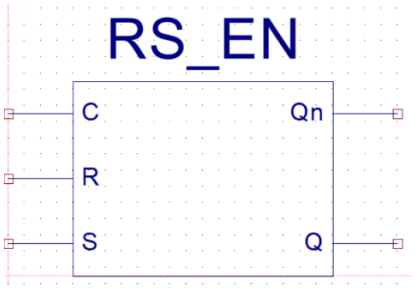
```
integer i = 0;
   initial begin
    C = 0;
    R = 0;
    S = 0;
    #40;
    S = 0;
    R = 0;
    S = 1;
    R = 0;
    #100;
```

```
    S = 0;
    R = 1;
    #100;
    S = 1;
    R = 1;
    #100;
    S = 0;
    R = 0;
    #100;
    S = 1;
    R = 0;
  end
always @*
    for(i = 0; i<20; i = i + 1)begin
    #50;
    C <= ~C;
end
```

③带使能控制 C 的 RS 锁存器仿真结果

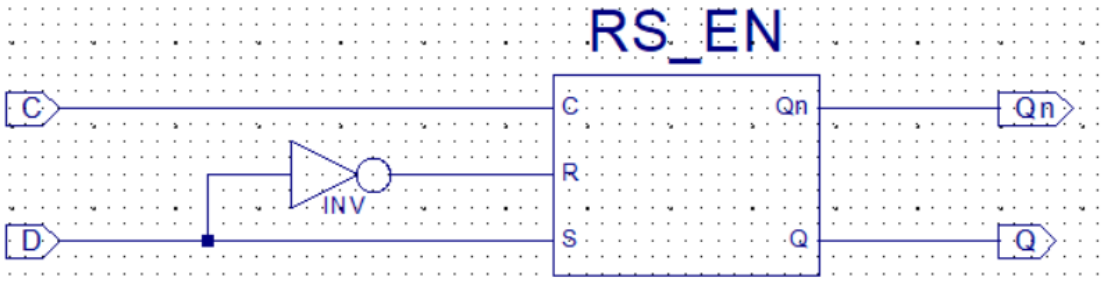


④仿真通过后封装成逻辑符号



3.设计电平控制 D 锁存器

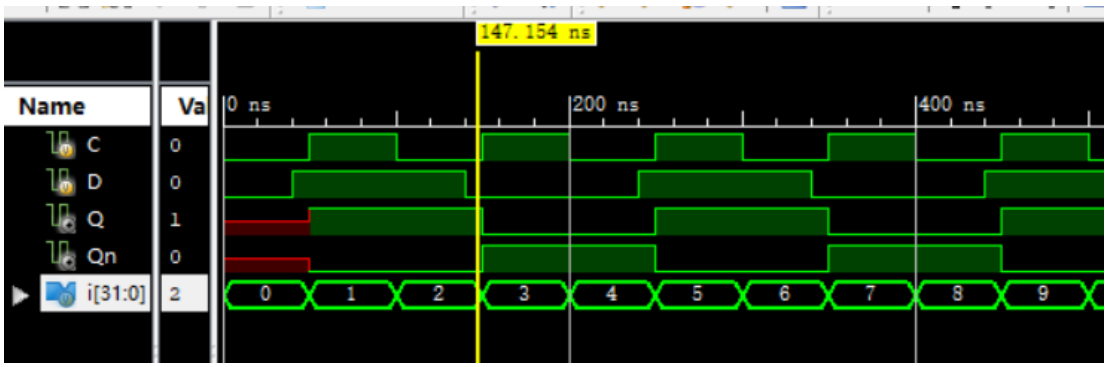
①调用 RS_EN 实现，命名为 D_EN



②仿真激励代码设计

```
integer i = 0;
   initial begin
    D = 0;
    C = 0;
    #40;
    D = 1;
    #100;
    D = 0;
    #100;
    D = 1;
    #100;
    D = 0;
    #100;
    D = 1;
    #100;
   end
  always @*
    for(i = 0; i<20; i = i + 1)begin
    #50;
    C <= ~C;
  end
```

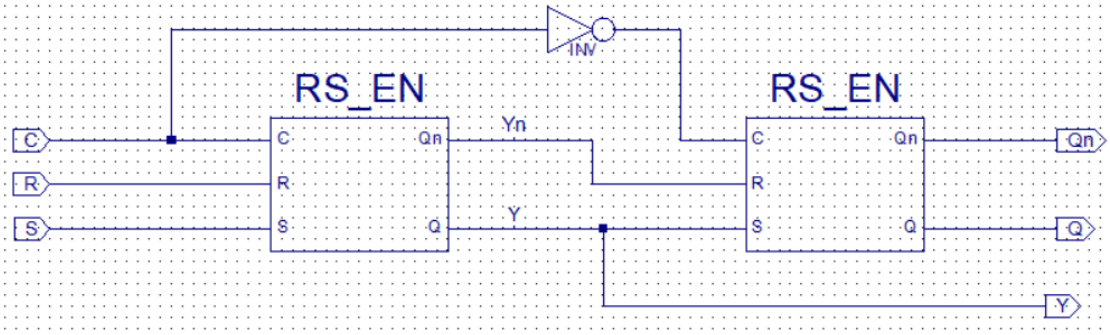
③电平控制 D 锁存器仿真结果



4.2 工程二：Trig

1.设计 RS 主从触发器

①调用工程一 RS_EN 锁存器实现

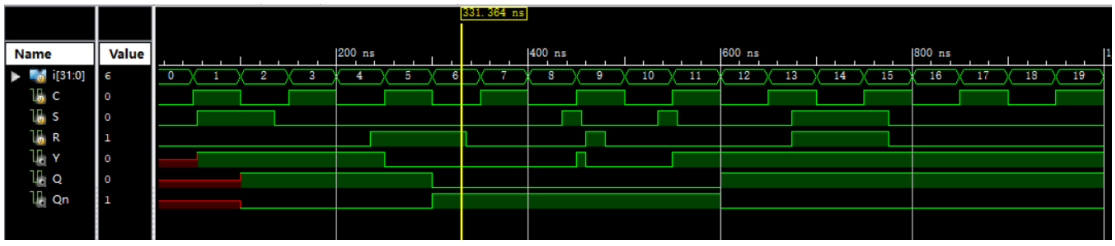


②仿真激励代码设计

```
integer i = 0;
   initial begin
    C = 0;
    R = 0;
    S = 0;
    #55;
    S = 1;
    R = 0;
    #80;
    R = 0;
    S = 0;
    #100;
    R = 1;
    S = 0;
    #100;
    R = 0;
    S = 0;
    #100;
    R = 0;
    S = 1;
    #20;
    R = 0;
    S = 0;
    #5;
    R = 1;
    S = 0;
    #20;
    R = 0;
```

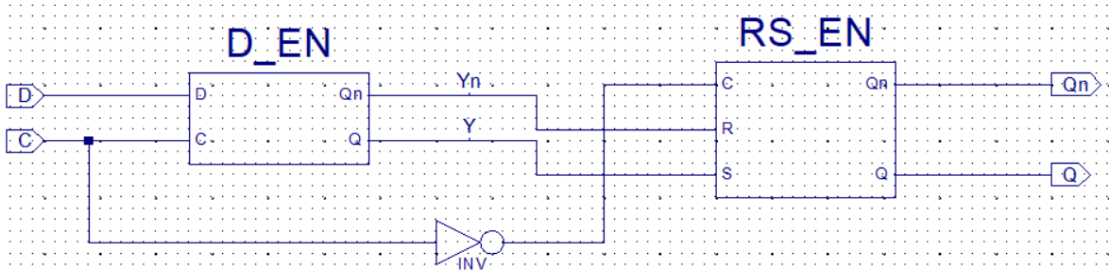
```
    S = 0;
    #55;
    R = 0;
    S = 1;
    #20;
    R = 0;
    S = 0;
    #120;
    R = 1;
    S = 1;
    #100;
    R = 0;
    S = 0;
    #100;
  end
 always@*
   for( i = 0; i<20; i = i+1) begin
     #50;
     C <= ~C;
   end
```

③RS 主从触发器仿真结果



2. 设计主从 D 触发器

①调用 D_EN 和 RS_EN 用原理图设计实现

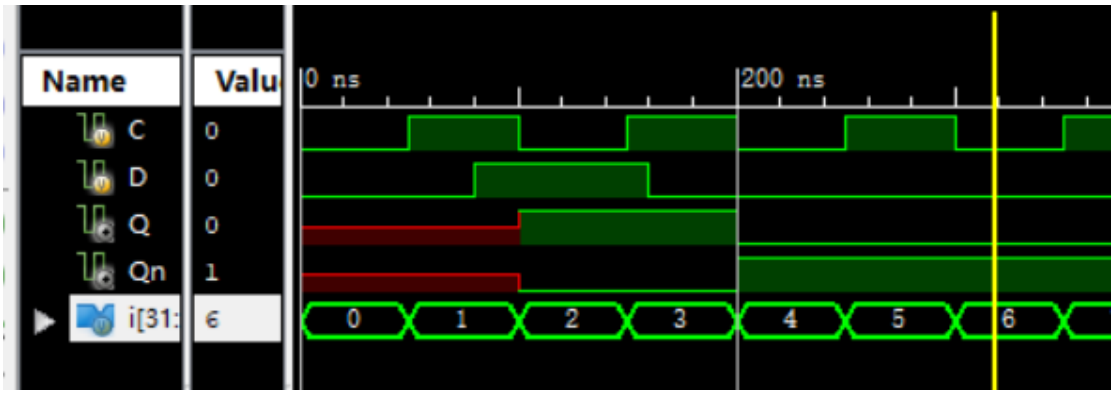


②仿真激励代码设计

```
initial begin
    D = 0;
    C = 0;
```

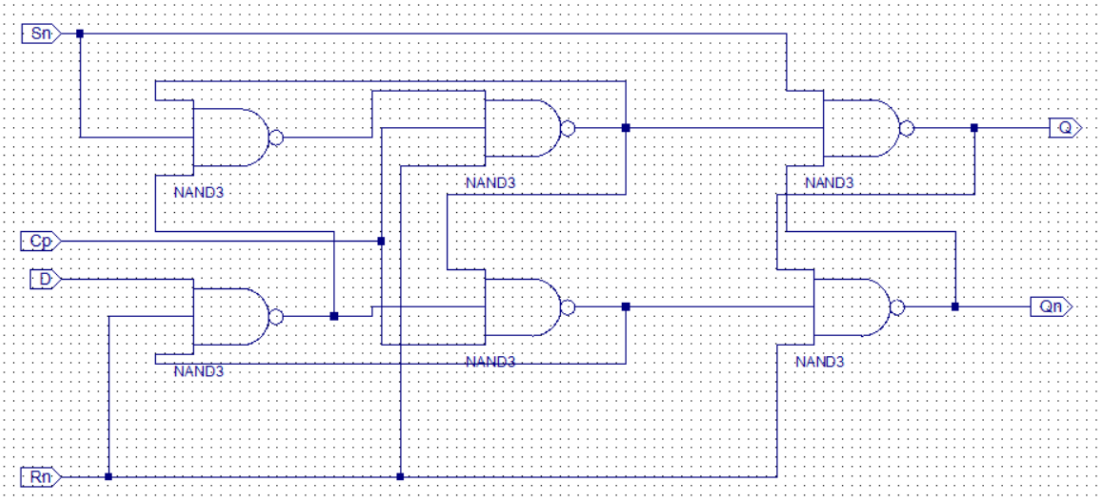
```
    #20;
    D = 0;
    C = 1;
    #20;
    D = 0;
    C = 0;
    #20;
    D = 1;
    C = 1;
    #20;
    D = 1;
    C = 0;
    #20;
    D = 1;
    C = 1;
    #20;
  end
```

③仿真结果



3. 设计维持阻塞型正边沿 D 触发器

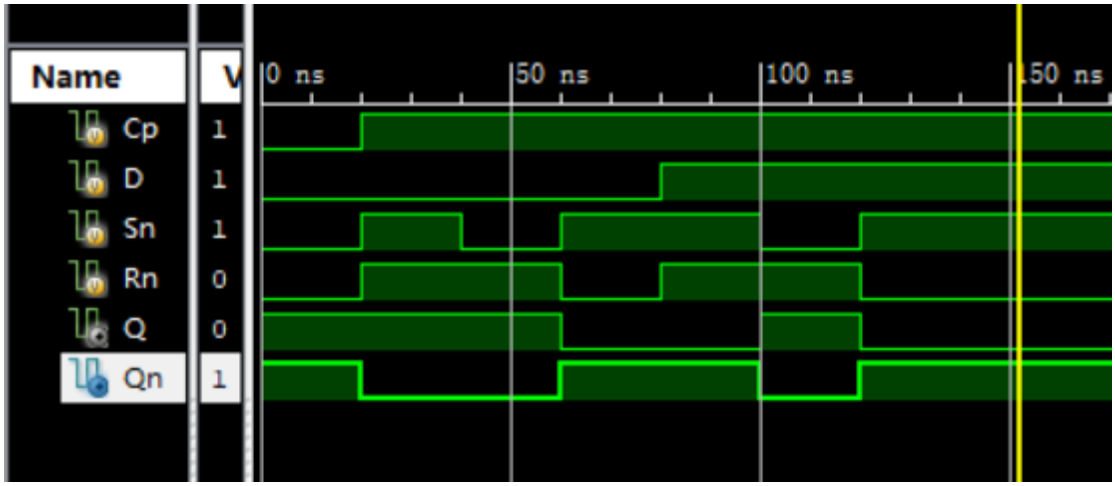
①增加异步初始输入功能：Rn， Sn， 调用与非门用原理图设计实现



②仿真激励代码设计

```
initial begin
    Sn = 0;
    Cp = 0;
    Rn = 0;
    D = 0;
    #20;
    Cp = 1;
    Rn  =1;
    Sn = 1;
    D = 0;
    #20;
    Cp = 1;
    Rn  =1;
    Sn = 0;
    D = 0;
    #20;
    Cp = 1;
    Rn  =0;
    Sn = 1;
    D = 0;
    #20;
    Cp = 1;
    Rn  =1;
    Sn = 1;
    D = 1;
    #20;
    Cp = 1;
    Rn  =1;
    Sn = 0;
    D = 1;
    #20;
    Cp = 1;
    Rn  =0;
    Sn = 1;
    D = 1;
    #20;
   end
```

③仿真结果



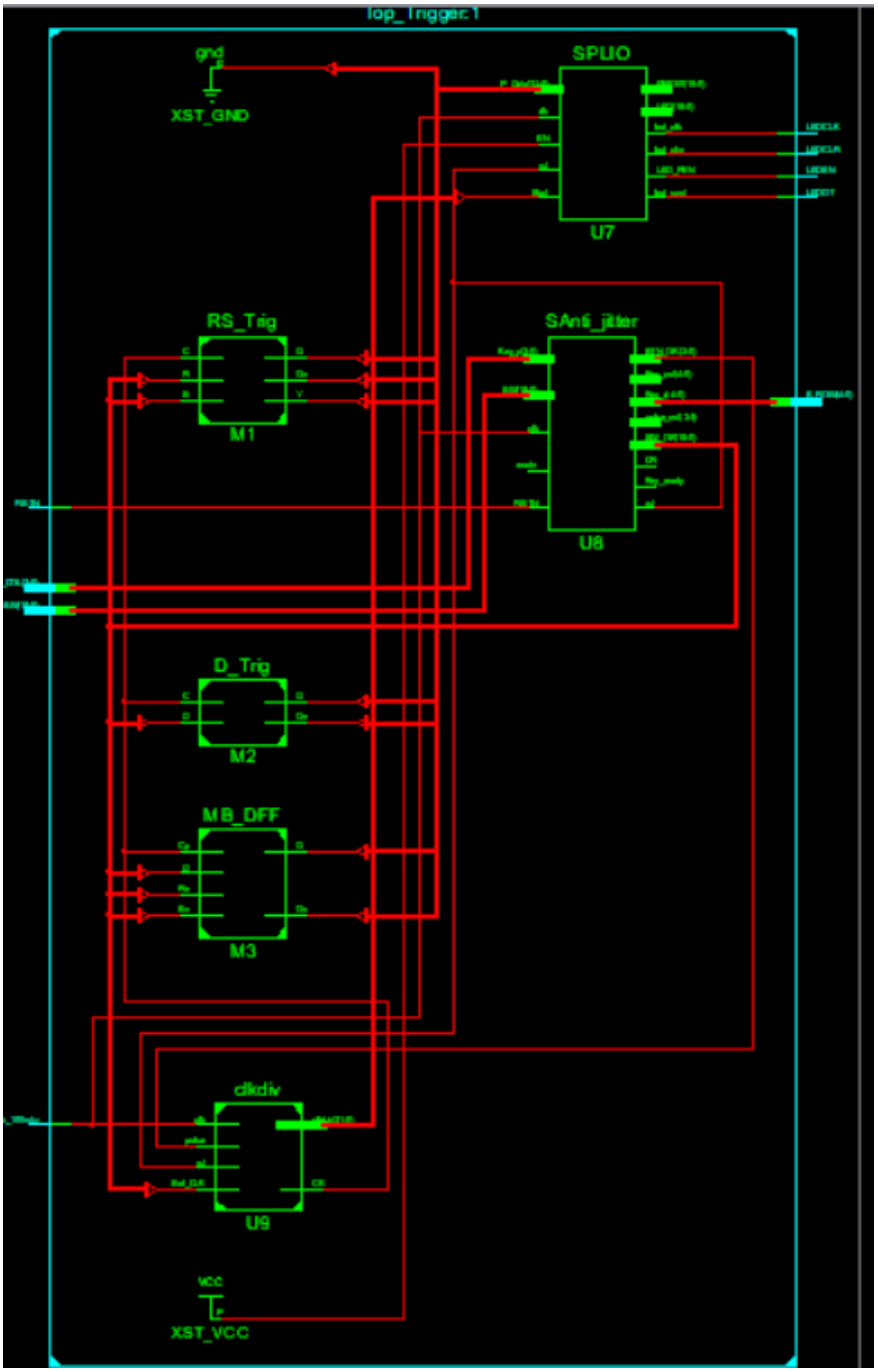
4. 顶层模块设计

```
module Top_Trigger( input clk_100mhz,
                input wire RSTN,
                input wire [3:0] K_COL,
                input wire [15:0] SW,
                output wire [4:0] K_ROW,

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

wire [31:0] Div, PD;
wire [15:0] SW_OK;
wire [3:0] BTN_OK, pulse_out;
wire rst, CK;
assign clk = clk_100mhz;

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

D_Trig M2(    .D(SW_OK[3]),
          .C(CK),
```

```
          .Q(PD[3]),
          .Qn(PD[4])
          );

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

SAnti_jitter  U8( .clk(clk),
              .RSTN(RSTN),
              .readn(),
              .Key_y(K_COL),
              .Key_x(K_ROW),
              .SW(SW),
              .Key_out(),
              .Key_ready(),
              .pulse_out(),
              .BTN_OK(BTN_OK),
              .SW_OK(SW_OK),
              .CR(),
              .rst(rst)
              );

clkdiv      U9( .clk(clk),
              .rst(rst),
              .Sel_CLK(SW_OK[2]),
              .pulse(BTN_OK[0]),
              .clkdiv(Div),
              .CK(CK)
              );
SPLIO       U7( .clk(clk),
              .rst(rst),
              .Start(Div[20]),
              .EN(1'b1),
              .P_Data(PD),
              .LED(),
              .led_clk(LEDCLK),
              .led_sout(LEDDT),
```

```
                .led_clrn(LEDCLR),
                .LED_PEN(LEDEN),
                .GPIOf0()
                );
Endmodule
```

综合后，看RTL：



5.设计引脚约束

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

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

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

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

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

NET "SW[0]" LOC=AA10  | IOSTANDARD=LVCMOS15;
NET "SW[1]" LOC=AB10  | IOSTANDARD=LVCMOS15;
NET "SW[2]" LOC=AA13  | IOSTANDARD=LVCMOS15;
NET "SW[3]" LOC=AA12  | IOSTANDARD=LVCMOS15;
NET "SW[4]" LOC=Y13 | IOSTANDARD=LVCMOS15;
NET "SW[5]" LOC=Y12 | IOSTANDARD=LVCMOS15;
NET "SW[6]" LOC=AD11  | IOSTANDARD=LVCMOS15;
NET "SW[7]" LOC=AD10  | IOSTANDARD=LVCMOS15;
NET "SW[8]" LOC=AE10  | IOSTANDARD=LVCMOS15;
NET "SW[9]" LOC=AE12  | IOSTANDARD=LVCMOS15;
NET "SW[10]"  LOC=AF12  | IOSTANDARD=LVCMOS15;
NET "SW[11]"  LOC=AE8 | IOSTANDARD=LVCMOS15;
NET "SW[12]"  LOC=AF8 | IOSTANDARD=LVCMOS15;
NET "SW[13]"  LOC=AE13  | IOSTANDARD=LVCMOS15;
NET "SW[14]"  LOC=AF13  | IOSTANDARD=LVCMOS15;
NET "SW[15]"  LOC=AF10  | IOSTANDARD=LVCMOS15;

#NET "LED[0]" LOC=AB26  | IOSTANDARD=LVCMOS33;
#NET "LED[1]" LOC=W24 | IOSTANDARD=LVCMOS33;
#NET "LED[2]" LOC=W23 | IOSTANDARD=LVCMOS33;
#NET "LED[3]" LOC=AB25  | IOSTANDARD=LVCMOS33;
#NET "LED[4]" LOC=AA25  | IOSTANDARD=LVCMOS33;
#NET "LED[5]" LOC=W21 | IOSTANDARD=LVCMOS33;
```

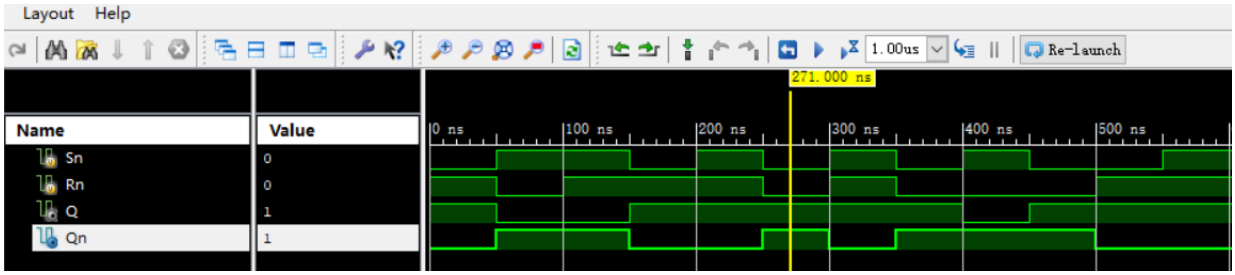
```
#NET "LED[6]" LOC=V21 | IOSTANDARD=LVCMOS33;
#NET "LED[7]" LOC=W26 | IOSTANDARD=LVCMOS33;
```

五、实验结果与分析

5.1 工程一：Locker

实验结果

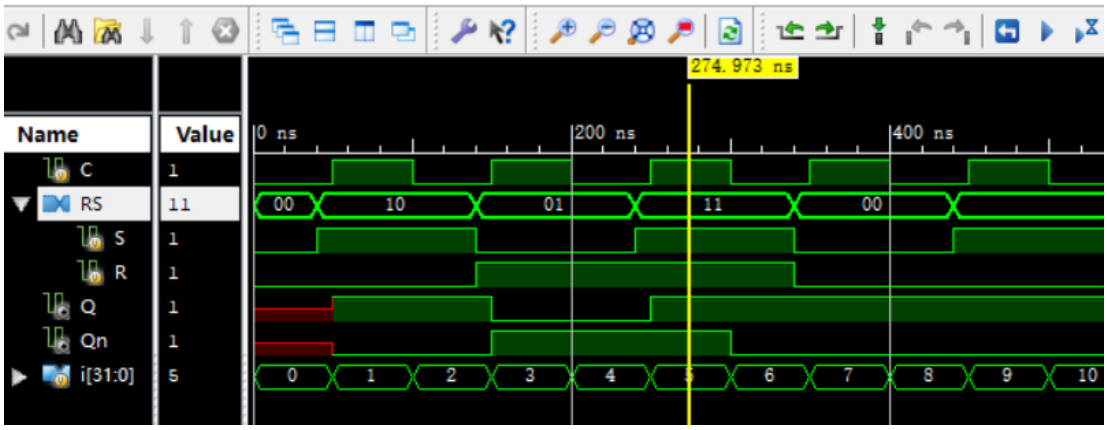
① NAND RS 锁存器仿真结果分析



实验结果分析：

时间	输入Sn	输入Rn	输出Q	输出Qn	状态
0-50	0	1	1	0	设置 Q 为1
50-100	1	0	0	1	重设 Q 为0
100-150	1	1	1	0	保持
150-200	0	1	1	0	设置 Q 为1
200-250	1	1	1	0	保持
250-300	0	0	1	1	未定义
300-350	1	1	1	0	保持，确定但是未知，因为不清楚在未定义状态Q的确定值
350-400	0	0	1	1	未定义
400-450	1	0	0	1	重设 Q 为0
450-500	0	0	1	1	未定义
500-550	0	1	1	0	设置 Q 为1
550-600	1	1	1	0	保持

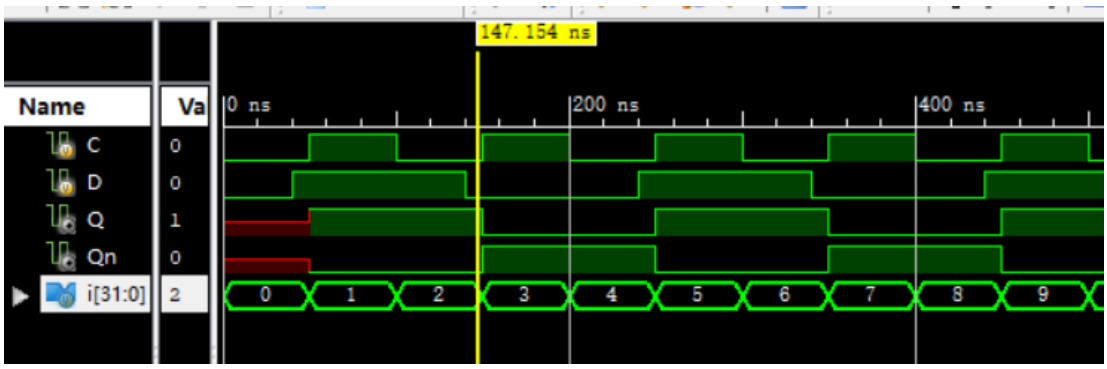
②带使能控制 C 的 RS 锁存器仿真



实验结果分析：

时钟	输入RS	输出Q	输出Qn	状态
0	00	x	X	未知
1	10	1	0	设 Q 为1
0	10	1	0	保持
1	01	0	1	设 Q 为0
0	01	0	1	保持
1	11	1	1	未定义
0	11	1	0	虽然时钟为0，但是 Qn 的结果却发生了改变，是因为前一状态为未定义状态
1	00	1	0	保持

③电平控制 D 锁存器仿真



实验结果分析

时钟	输入D	输出Q	输出 Qn	状态
0	0	X	X	未定义

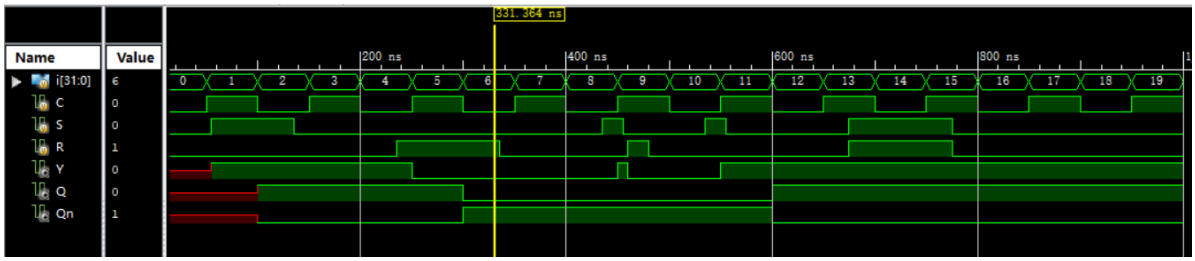
1	1	1	0	设 Q 为1
0	1	1	0	保持
1	0	0	1	设 Q 为0
0	0	0	1	保持
1	1	1	0	设 Q 为1
0	1	1	0	保持

D 触发器实验结果非常清晰，即 Q 的输出与 D 相同。且避免了出现未定义的状态。不过也是时钟 C 控制，即只有当 C 为1时，结果才可能改变。

5.2 工程二：Trig

实验结果

① RS 主从触发器激励结果



结果分析：

1）50-140ns,数据采样段，S = 1, R = 0,所以置 Q 为1

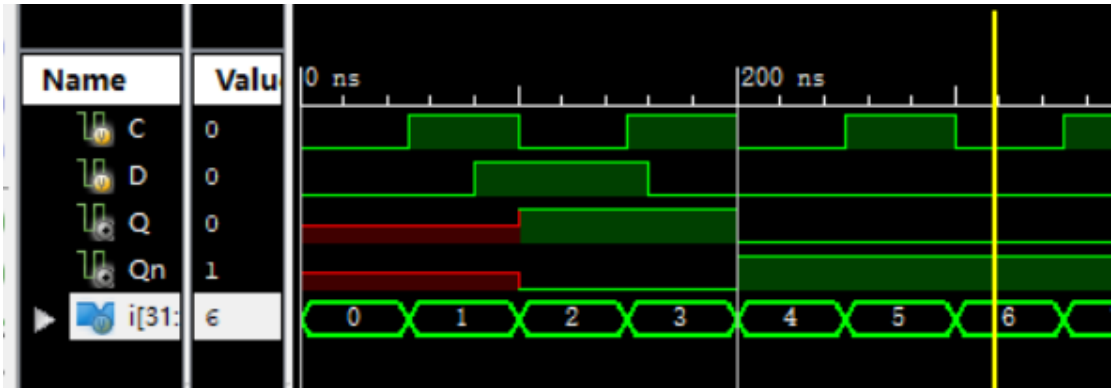
2）150-250ns,数据保持，S = 0, R = 0，所以 Q 输出不变

3）250-350ns，数据采样段，S = 0, R = 1,所以 Q 输出0。考虑到为主从触发器，所以 Q 的输出比 Y 慢一个周期。

4）350-450ns，数据保持，所有数据输出均保持不变。

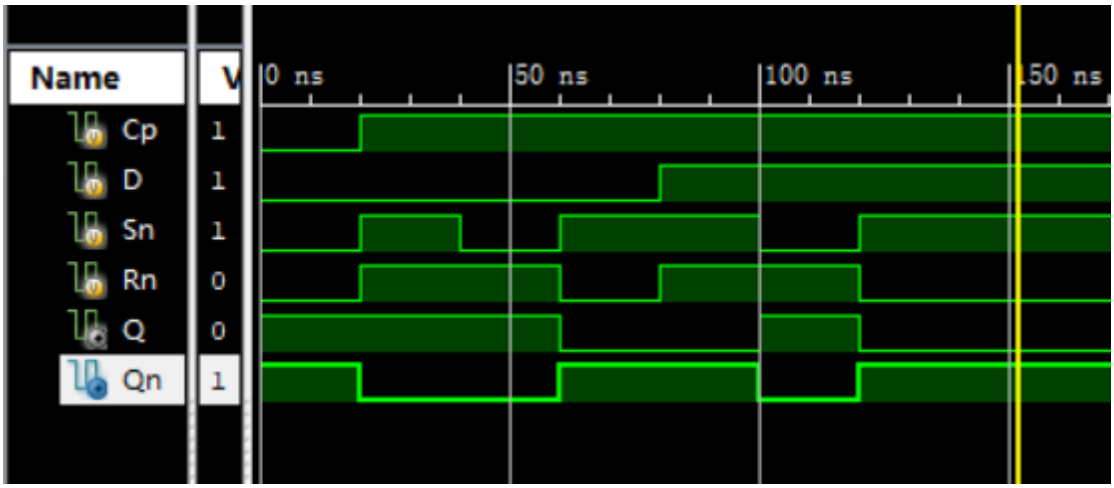
5）450-550ns，这一段中 S 出现一个信号为 1 的尖脉冲，这个脉冲传导到 Y,但是一开始数据可以实现正常保持。但是几个脉冲后，再度对数据采样，Q 出现错误保持现象。即一次性采样。该问题为 RS 主从触发器存在的问题，由于锁存器的输出端可以直接反应出输入端的数据，在时序电路存储状态时出现“空翻”无法控制。

②主从 D 触发器仿真激励结果



输入C	输入D	输出Q	输出Qn
0	0	X	X
1	0	X->1	X->0
0	1	1	0
1->0	0	1->0	0->1
0	0	0	1

③维持阻塞正边沿 D 触发器激励结果



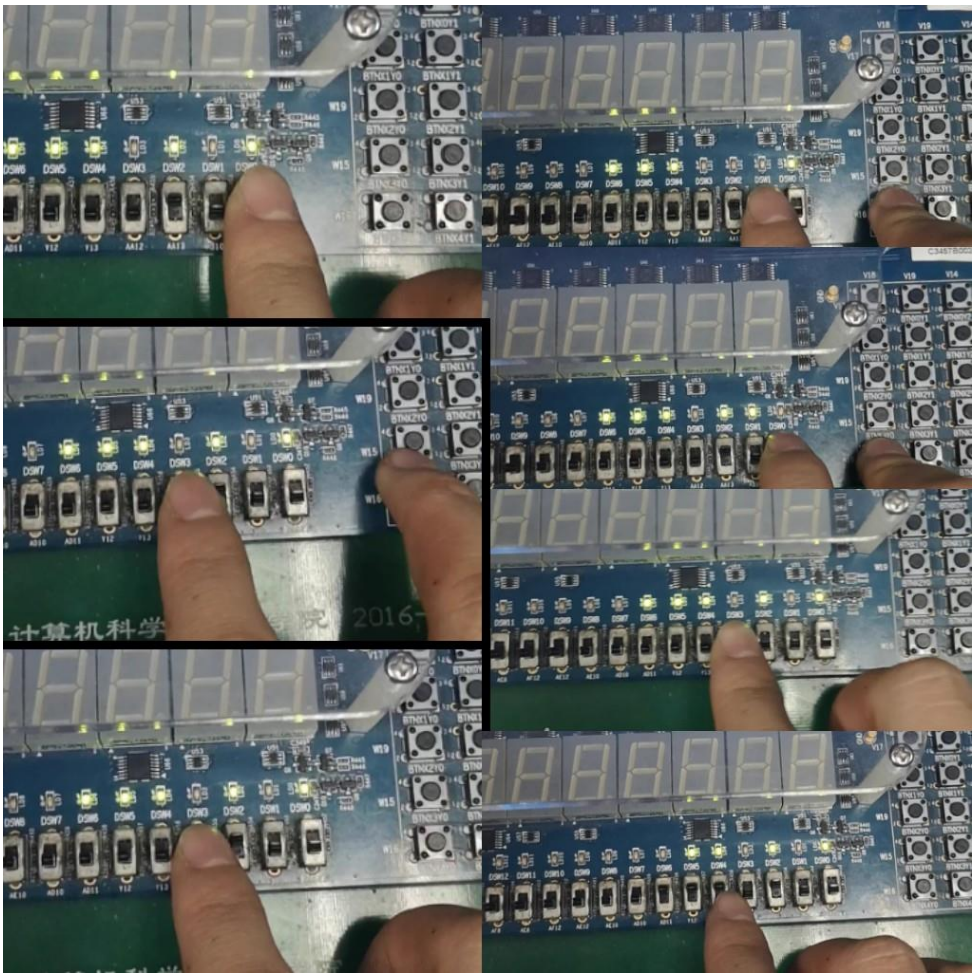
结果分析：

输入Cp	输入D	输入Sn	输入Rn	输出Q	输出Qn
0	0	0	0	1	1
1	0	1	1	1	0
1	0	0	1	1	0
1	0	1	0	0	1

1	1	1	1	0	1
1	1	0	1	1	0
1	1	1	0	0	1

3.物理板验证结果

物理板上实验结果



接口说明：

RS_Trig:输入RS = SW[1:0],输出{Y, Q, Qn} = LED[2:0]

D_TrIG:输入D = SW[3],输出{Q, Qn} = LED[4:3]

MB_DFF:{Sn, Rn, D} = SW[6:4],输出{Q, Qn} = LED[7:6]

公共控制接口信号为时钟选择SW_OK[2]:单步为1，连续为0；使能控制时钟为BTN_OK[0]

实验结果分析：

①主从触发器测试

功能	R	S	C单步	Y	Q	Qn	备注

置1	0	1	0	1	X	X	未知
	0	1	1->0	1	X->1	X->0	Q 置1
保持	0	0	0	1	1	0	保持
	0	0	1->0	1	1	0	保持
置0	1	0	0	1->0	1	0	Y 置0
	1	0	1->0	0	1->0	0->1	Q 置0
保持	0	0	0	0	0	1	保持
	0	0	1->0	0	0	1	保持
无定义	1	1	0	1	0	1	无定义
	1	1	1->0	X	X	X	无定义
保持	0	0	0	X	X	X	未知
	0	0	1->0	X	X	X	未知

②一次性采样测试

功能		**R**	**S**	**C单步**	**Y**	**Q**	**Qn**	**备注**
一次性采样	置1	0	1	1->0	1	0	1	置1
	采样RS	0	0	1	1	0	1	保持
	R = 脉冲	1	0	1	0	0	1	正确保持
	保持	0	0	1->0	0	0	1	正确保持
一次性采样	置1	0	1	1->0	0->1	0	1	正确保持
	采样RS	0	0	1	1	0	1	正确保持
	S = 脉冲	0	1	1	1	0	1	采样
	保持	0	0	1->0	1	1	0	错误保持，第一次采样

③D 触发器测试

主从 D 触发器				**边沿 D 触发器**					
D	C	Q	Qn	Rn	Sn	D	C	Q	Qn

0	1	x	X	0	0	0	0	1	1
0	1->0	x->1	x->0	1	0	0	1	1	0
1	1	1	0	0	1	0	1	0	1
1	1->0	1->0	0->1	1	0	1	1	1	0
1	1	0	1	0	1	1	1	0	1

六、讨论与心得

本次实验好多仿真激励代码需要自己设计。本来觉得很难，但通过仔细分析电路图，了解清楚每一个输入，去分析它对应的输出。再去根据预想的仿真图像去设计仿真输入结果。经过细心的比对，发现仿真与预想的差异，最终顺利完成仿真激励图像的输出。

本次实验由简到难，从锁存器的设计再到触发器的设计，从最基本的 RS 锁存器到加使能控制端再到设计 RS 主从触发器，从 D 触发器再到维持阻塞正边沿 D 触发器的设计。一步步深入，让我加深了对于锁存器以及触发器概念的理解。通过分析仿真图像，弄清楚了一次性采样的原理。为设计时序电路打下了良好的基础。

实验 10—锁存器与触发器实验报告

姓名： 施含容 **学号：** 3180103497 **专业：** 计算机科学与技术

课程名称： 逻辑与计算机设计基础实验 **同组学生姓名：** 刘晓钦

实验时间： 2019-11-28 **实验地点：** 紫金港东 4-509 **指导老师：** 洪奇军、施青松

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. 掌握有限状态机描述与电路实现方法；
3. 掌握时序电路的状态图、状态方程和触发器激励函数；
4. 掌握用 FPGA 实现有限状态机设计、仿真与调试。

二、实验内容和原理

2.1 实验内容

1. 用原理图设计基于状态方程描述的4位二进制同步计数器；
2. 用HDL 行为描述设计32位同步二进制双向计数器；
3. 集成到实验环境(混合计算器，Calculation)。

2.2 实验原理

1.4 位同步二进制计数器状态表

	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

2.4位同步二进制计数器状态方程

根据卡诺图化简，即可以得到需要的状态方程。

3.D触发器输入方程

$$D_A = \overline{Q_A}$$

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

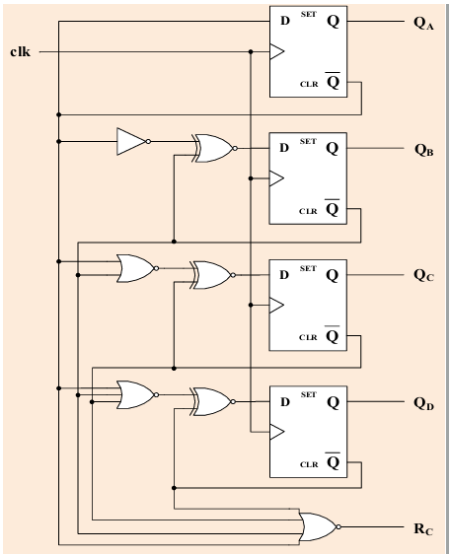
$$D_C = \overline{Q_A}Q_C + \overline{Q_B}Q_C + Q_AQ_B\overline{Q_C} = \overline{(\overline{Q_A} + \overline{Q_B}) \oplus Q_C}$$

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

增加一个 4 位进位输出Rc

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

4.4位二进制同步计数器逻辑原理图



5.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

三、主要仪器设备

1.计算机(Intel Core i5以上，4GB内存以上)系统

2.计算机软硬件课程贯通教学实验系统(Sword)

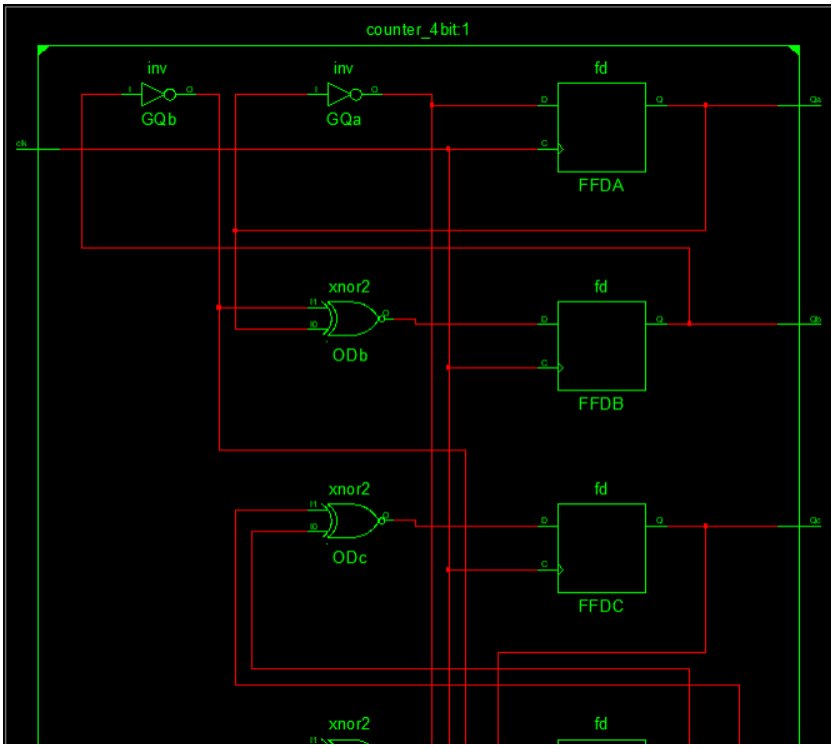
3.Xilinx ISE14.4及以上开发工具

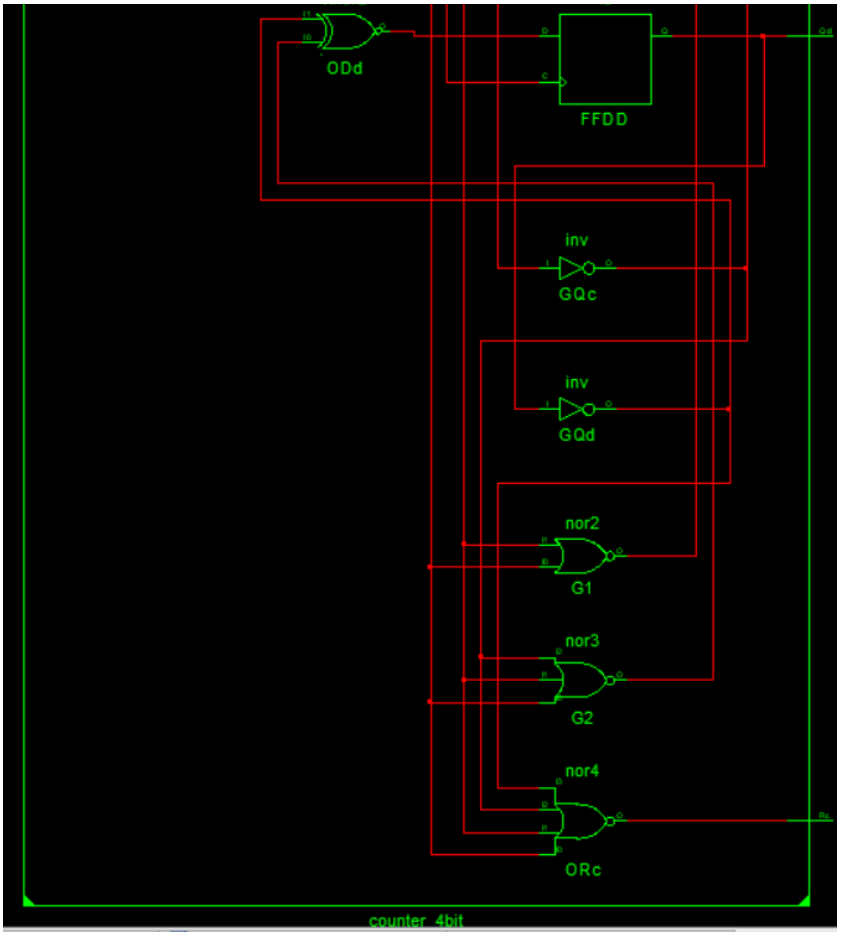
四、操作方法与实验步骤

工程：FSM

1.设计实现 4 位计数器

```
module counter_4bit(  input clk,
                output wire Qa,
                output wire Qb,
                output wire Qc,
                output wire Qd,
                output Rc
    );
wire Da,Db,Dc,Dd,nQa,nQb,nQc,nQd;
wire Nor_nQa_nQb,Nor_nQa_nQb_nQc;
FD  FFDA( .C(clk),.D(Da),.Q(Qa)),
  FFDB( .C(clk),.D(Db),.Q(Qb)),
  FFDC( .C(clk),.D(Dc),.Q(Qc)),
  FFDD( .C(clk),.D(Dd),.Q(Qd));
defparam FFDA.INIT = 1'b0;
defparam FFDB.INIT = 1'b0;
defparam FFDC.INIT = 1'b0;
defparam FFDD.INIT = 1'b0;
INV GQa(.I(Qa),.O(nQa)),
    GQb(.I(Qb),.O(nQb)),
    GQc(.I(Qc),.O(nQc)),
    GQd(.I(Qd),.O(nQd));
assign Da = nQa;
XNOR2 ODb(.I0(Qa),.I1(nQb),.O(Db)),
    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));
NOR2 G1(.I0(nQa),.I1(nQb),.O(Nor_nQa_nQb));
NOR3 G2(.I0(nQa),.I1(nQb),.I2(nQc),.O(Nor_nQa_nQb_nQc));
endmodule
```



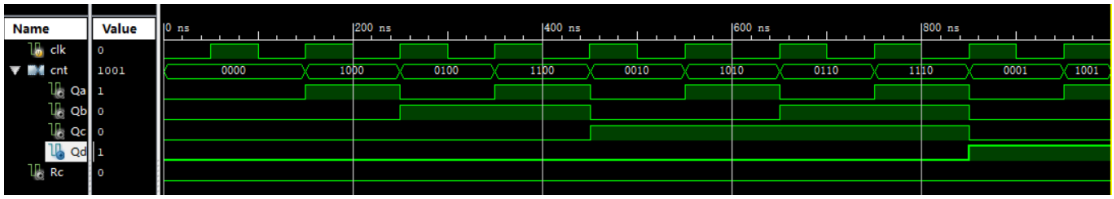


②4位同步二进制计数器激励与仿真

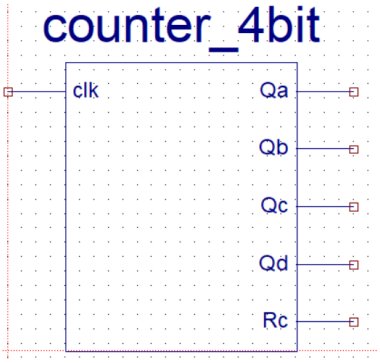
```
initial begin
    // Initialize Inputs
```

```
    clk <= 0;
    forever #50 clk <= ~clk;
  end
```

③仿真结果



④仿真通过封装成逻辑符号Counter_4bit



2.设计 32 位同步双向计数器

①采用行为描述实现：counter_rev_32.v,双向控制信号 s。并增加计数器初始化功能。

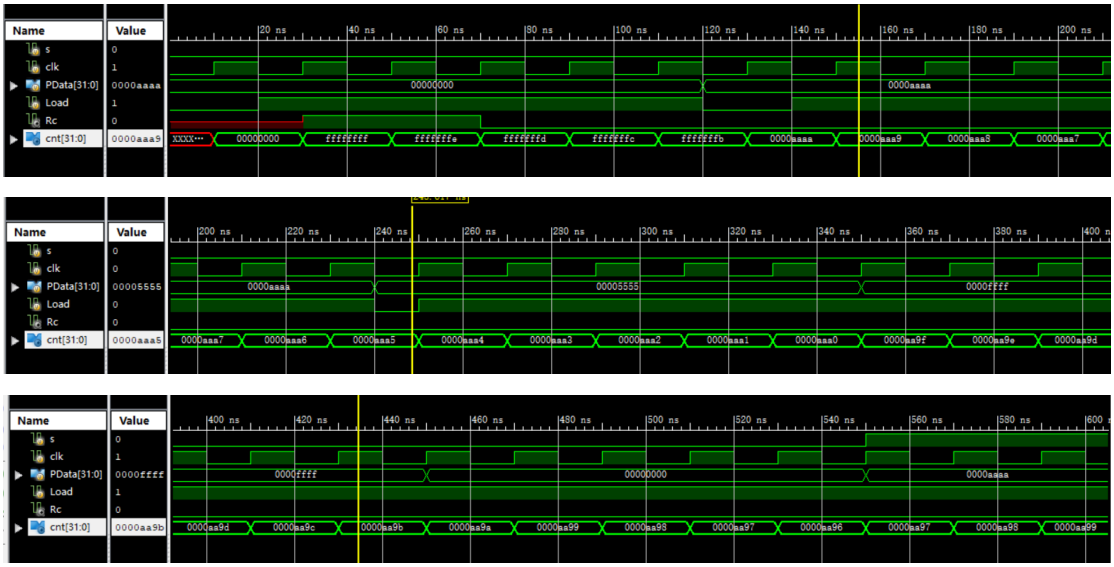
```
module counter_32_rev(  input clk,
                input s,
                input Load,
                input[31:0] PData,
                output reg[31:0] cnt,
                output reg Rc
    );
//assign Rc = (~s & (~|cnt))  | (s & (&cnt));
always@( posedge clk) begin
  if(Load == 1'b0) cnt <=  PData;
  else begin
    if(s) cnt <= cnt + 1;
    else  cnt <= cnt - 1;
    if(~|cnt == 1 || &cnt == 1) Rc <= 1;
    else Rc <= 0;
  end
end
endmodule
```

②32位同步二进制计数器激励与仿真

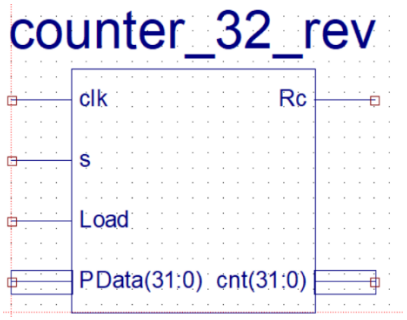
```
initial begin
    // Initialize Inputs
    s = 0;
    Load = 0;
    PData = 0;
    #20;
    Load = 1;
    #100;
    Load=0;
      PData = 16'b1010101010101010;
    #20;
    s = 0;
    Load = 1;
    #100;
    Load = 0;
    PData = 16'b0101010101010101;
    #10;
    Load = 1;
    s = 0;
    #100;
    PData = 16'b1111111111111111;
    s = 0;
    #100;
    PData = 16'b0000000000000000;
    s = 0;
    #100;
    PData = 16'b1010101010101010;
    s = 1;
    #100;
    PData = 16'b0101010101010101;
    s = 1;
    #100;
    PData = 16'b1111111111111111;
    s = 1;
    #100;
    PData = 16'b0000000000000000;
    s = 1;
    #100;
  end

    always begin
      clk = 1'b0;
```

```
        #10;
        clk = 1'b1;
        #10;
    end
```

③仿真结果



④仿真通过后封装逻辑符号



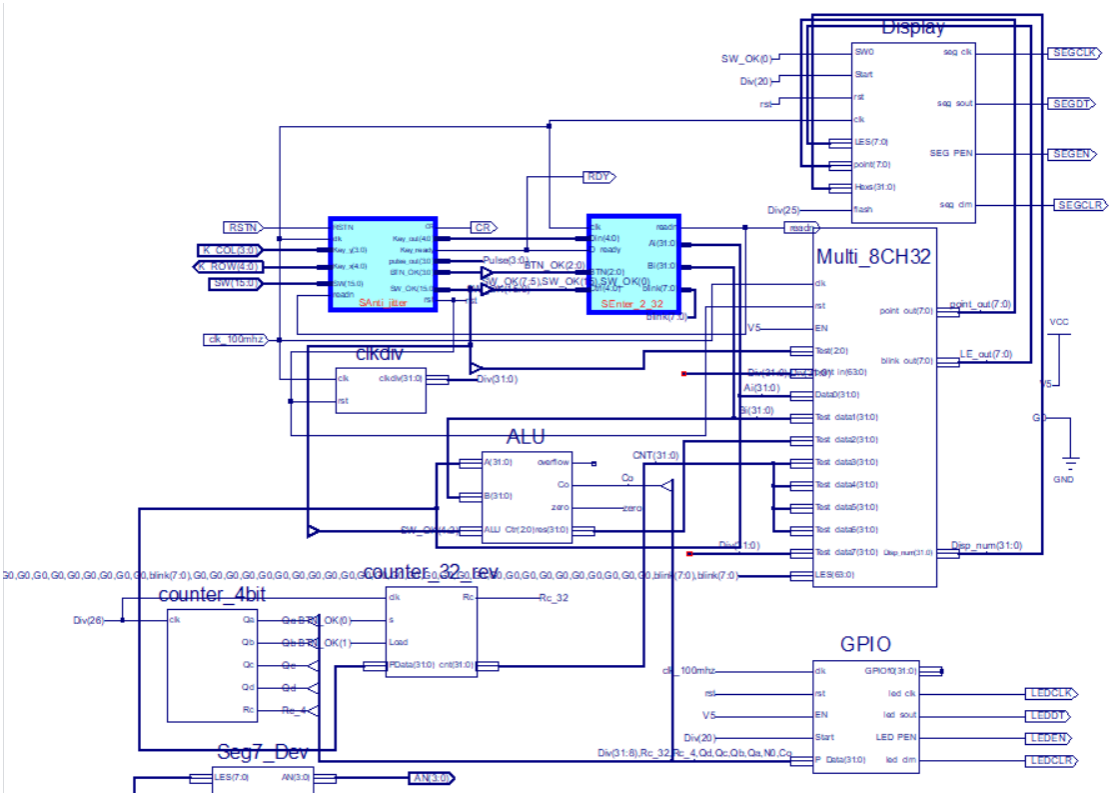
3. 设计辅助模块：1Hz的秒脉冲方波

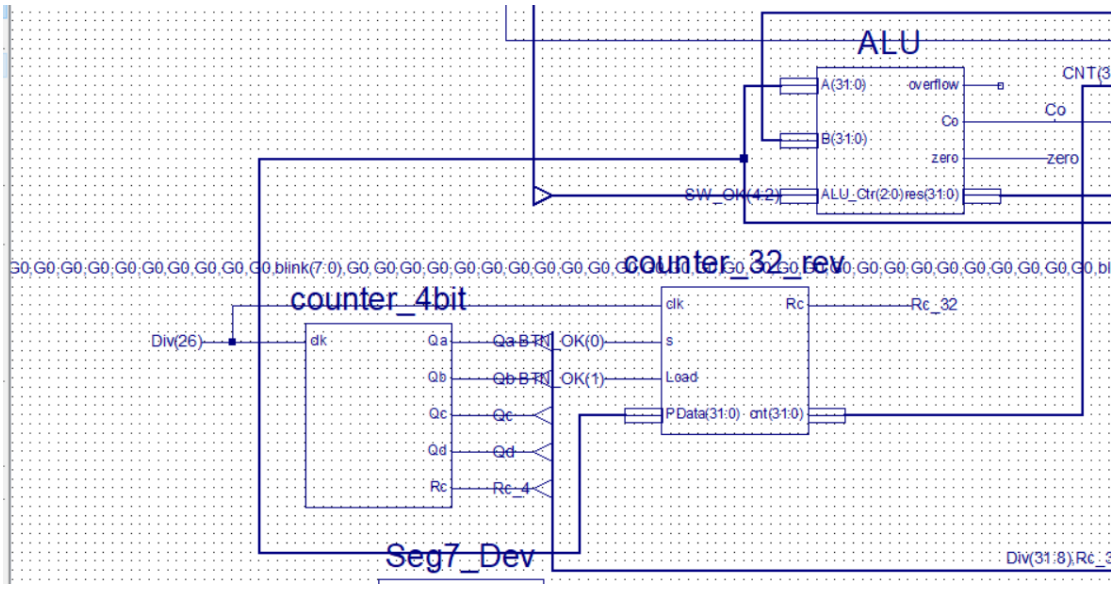
```
module counter_1s(input wire clk,
                  output reg clk_1s
    );
reg[31:0] cnt;
always@(posedge clk) begin
  if(cnt < 50_000_000)begin
    cnt <= cnt + 1;
  end
  else begin
  cnt <= 0;
  clk_1s <= ~clk_1s;
  end
```

```
  end
endmodule
```

4.集成混合计算器：计数功能

复制实验8的顶层模块，并改名为：Top-FSM.





5.UCF引脚分配。

输入接口物理映射。SW[7:5] = 通道选择

= 000：输入Ai；= 001：修改加数；=010：ALU输出；= 011：32位计数输出

完整 UCF 结构如下：

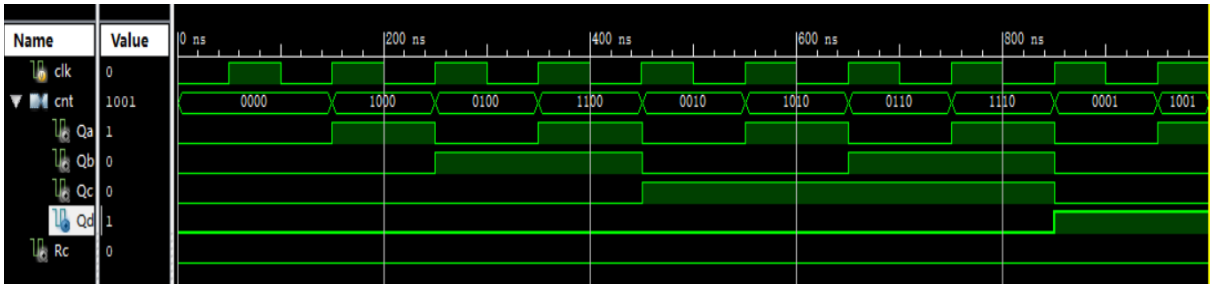
```
NET "clk_100mhz"  LOC=AC18  | IOSTANDARD=LVCMOS18;
NET "clk_100mhz"  TNM_NET=TM_CLK;
TIMESPEC TS_CLK_100M = PERIOD "TM_CLK" 10 ns HIGH 50%;

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

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

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

NET "readn" LOC=U21 | IOSTANDARD=LVCMOS33;
NET "RDY" LOC=U22 | IOSTANDARD=LVCMOS33;
NET "CR"  LOC=V22 | IOSTANDARD=LVCMOS33;

NET "SEGCLK"  LOC=M24 | IOSTANDARD=LVCMOS33;
NET "SEGCLR"  LOC=M20 | IOSTANDARD=LVCMOS33;
NET "SEGDT" LOC=L24 | IOSTANDARD=LVCMOS33;
NET "SEGEN" LOC=R18 | IOSTANDARD=LVCMOS33;

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

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

```
NET "SW[10]"  LOC=AF12  | IOSTANDARD=LVCMOS15;
NET "SW[11]"  LOC=AE8 | IOSTANDARD=LVCMOS15;
NET "SW[12]"  LOC=AF8 | IOSTANDARD=LVCMOS15;
NET "SW[13]"  LOC=AE13  | IOSTANDARD=LVCMOS15;
NET "SW[14]"  LOC=AF13  | IOSTANDARD=LVCMOS15;
NET "SW[15]"  LOC=AF10  | 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=AD21  | IOSTANDARD=LVCMOS33;
NET "AN[1]" LOC=AC21  | IOSTANDARD=LVCMOS33;
NET "AN[2]" LOC=AB21  | IOSTANDARD=LVCMOS33;
NET "AN[3]" LOC=AC22  | IOSTANDARD=LVCMOS33;

NET "LED[0]"  LOC=AB26  | IOSTANDARD=LVCMOS33;
NET "LED[1]"  LOC=W24 | IOSTANDARD=LVCMOS33;
NET "LED[2]"  LOC=W23 | IOSTANDARD=LVCMOS33;
NET "LED[3]"  LOC=AB25  | IOSTANDARD=LVCMOS33;
NET "LED[4]"  LOC=AA25  | IOSTANDARD=LVCMOS33;
NET "LED[5]"  LOC=W21 | IOSTANDARD=LVCMOS33;
NET "LED[6]"  LOC=V21 | IOSTANDARD=LVCMOS33;
NET "LED[7]"  LOC=W26 | IOSTANDARD=LVCMOS33;
```

五、实验结果与分析

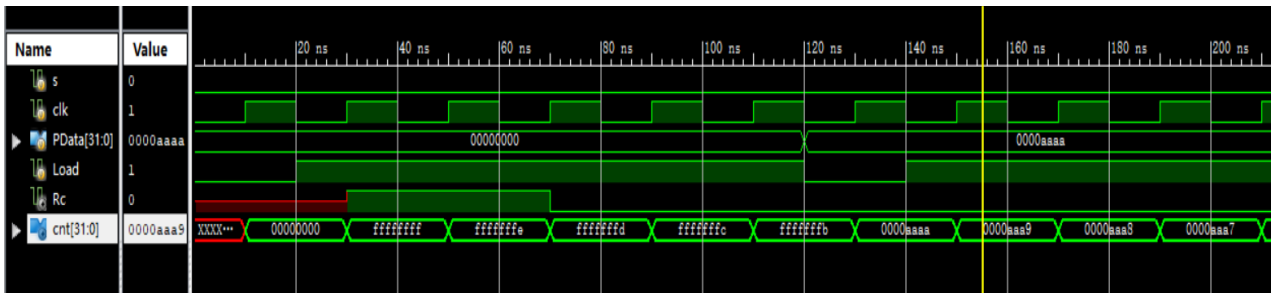
工程：FSM

①4位同步二进制计数器激励与仿真结果

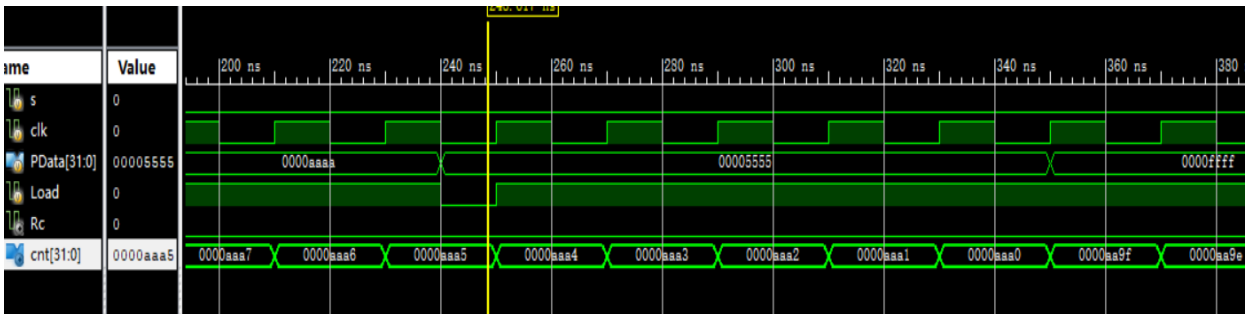


实验结果分析：Rc 代表进位，给定 clk 时钟信号，Qa，Qb，Qc，Qd正常实现计数功能，说明该模块设计正确。

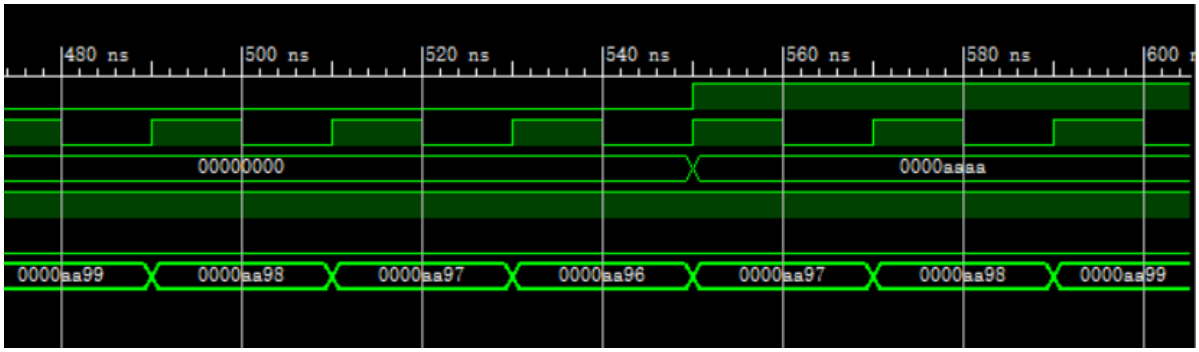
②32 位同步双向二进制计数器仿真结果



结果分析：我写的 32 位同步双向二进制计数器，通过 s 控制正向计数还是反向计数，通过 Load = 0，并入数据。一开始，当时钟上升沿来到，并入数据0000 0000，因为s = 0，所以为反向计数，cnt代表实验输出结果，输出结果为ffff ffff，ffff fffe，ffff fffd…该结果输出正常。后当load = 0，并入数据0000 aaaa，依旧为反向计数，结果正常输出。



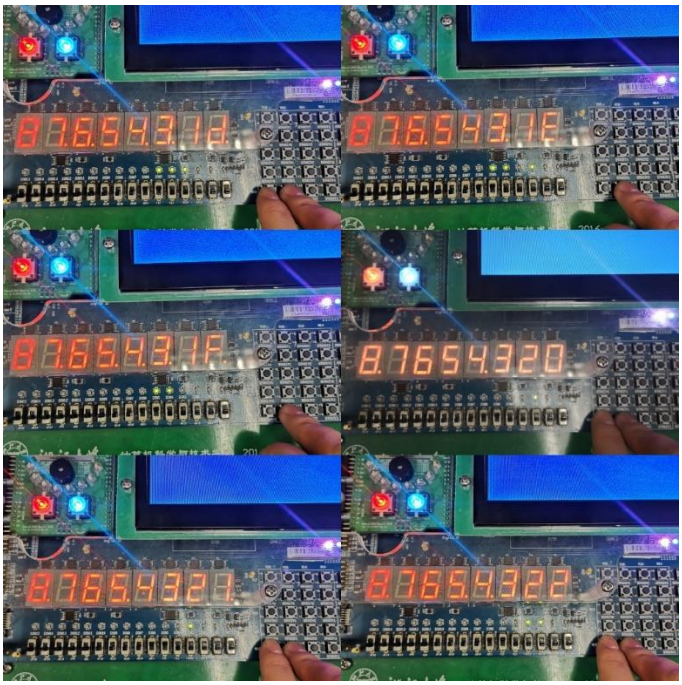
结果分析：图2并入数据0000 5555，反向计数，结果依旧正常。因为 32 位为巨量信号输入，因此在仿真中考虑到这一点，采用特征值抽样。aaaa为10101010交替出现，5555为01010101交替出现，结果均显示正常。



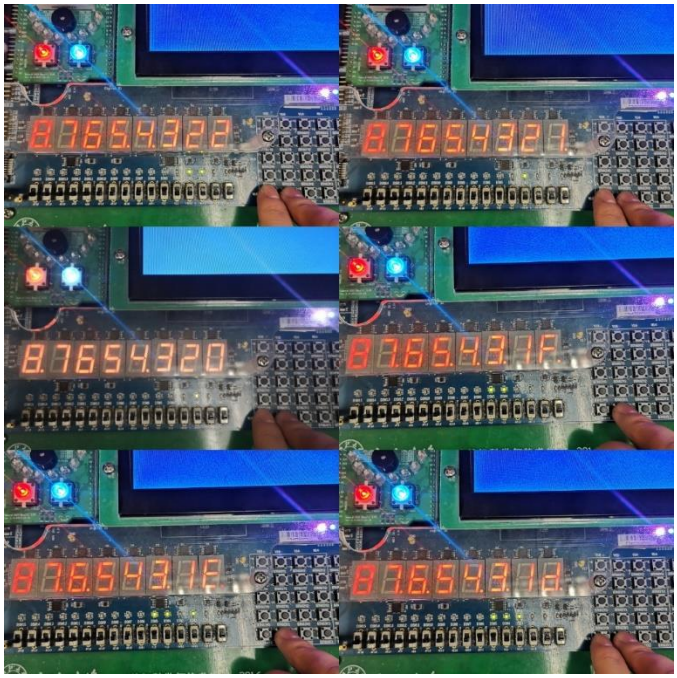
结果分析：该部分中s = 1，故为正向计数。从图中结果可以看出，输出由0000 aa96，正向加1，结果依次为0000 aa97，0000 aa98……综上所述，该模块设计正确，所有结果均可正确输出。

③物理板验证结果

正向计数检验



反向计数检验



实验结果分析：首先，SW[7:5]控制数字输入。通道选择 011 表示 32 位计数输出。BTN[1]表示 Load 信号，即控制数据并入。BTN[0]为 s 信号，即控制正向计数还是反向计数。按下BTN[0]实现正向计数，放开实现逆向计数。但原本实验存在一个问题：只有当长按BTN[0]才可以不断正向计数。因此，我对该实验做了改进优化。

④实验改进优化：

设想，按一下BTN[0]就可以自动切换正向计数还是反向计数。

解决思路1：将按钮改成开关，鉴于开关数量有限，采用其他思路。

解决思路2：用 D 触发器实现状态控制。

```
module counter_32_rev(  input clk,
                input s,
                input Load,
                input[31:0] PData,
                output reg[31:0] cnt,
                output reg Rc
    );
wire D, Q;
assign D = ~Q;
FD M1 ( .D(D),
        .C(s),
        .Q(Q));
defparam M1.INIT = 1'b0;
always@( posedge clk) begin
  if(Load == 1'b0) cnt <=  PData;
  else begin
    if(Q) cnt <= cnt + 1;
    else  cnt <= cnt - 1;
    if(~|cnt == 1 || &cnt == 1) Rc <= 1;
    else Rc <= 0;
  end
end
```

解决思路3：增加判断语句。

优化后，32 位同步双向计数器代码如下：

```
module counter_32_rev(  input clk,
                input s,
                input Load,
                input[31:0] PData,
                output reg[31:0] cnt,
                output reg Rc
    );
reg rev = 0;
always @(posedge s) begin
  rev <= ~rev;
end
//assign Rc = (~s & (~|cnt))  | (s & (&cnt));
```

```
always@( posedge clk) begin
  if(Load == 1'b0) cnt <=  PData;
  else begin
    if(s) cnt <= cnt + 1;
    else  cnt <= cnt - 1;
    if((~rev && (~|cnt)) || (rev && (&cnt)))  Rc <= 1;
    else Rc <= 0;
  end
end
```

六、讨论与心得

本实验设计了基于状态方程描述的 4 位二进制同步计数器，并用行为描述设计了 32 位同步二进制双向计数器。通过本实验，我进一步学习了如何用原语描述和行为描述来设计电路元件。以及熟练了对巨量信息输入的仿真代码的写法。

本实验最大的收获是能将实验 9 的 D 触发器运用到计数器中，改进与优化原本的实验。由于原本只能长按按钮实现正向计数与反向计数的控制，而加入了 D 触发器，使得按钮只需按一下便可实现控制。大大优化了实验性能，颇有成就感。

另外，从实验 8 开始，各实验之间的联系更为紧密，环环相扣，都是在之前实验框架的基础上不断地去补充完整。因此要重视每一次实验，为大作业做好充足的准备。

个人照片：施含容 3180103497