实验报告

2021 年 4 月 27 日　　成绩：

姓名	刘合	学号		班级	
专业	计算机科学与技术		课程名称	计算机组成原理课程设计	
任课老师		指导老师		机位号	
实验序号	2	实验名称			
实验时间	2021.4.27	实验地点		实验设备号	

一、实验目的与要求

1、 **实验目的：**

1.学习多功能 ALU 的工作原理,掌握多功能 ALU 的设计方法

2.掌握暂存器的设计方法,及其与 ALU 的连接方法

3.掌握运用 Verilog HDL 语言进行组合逻辑电路与时序逻辑电路混合设计的方法:

4.学会输入输出设备不足时的处理方法

2、 实验要求：

1. 使用行为描述方式建模多功能 ALU 模块，并添加图 11.7 中的标志位产生电路 FlagU。模块引脚如图所示，仿真验证其功能。

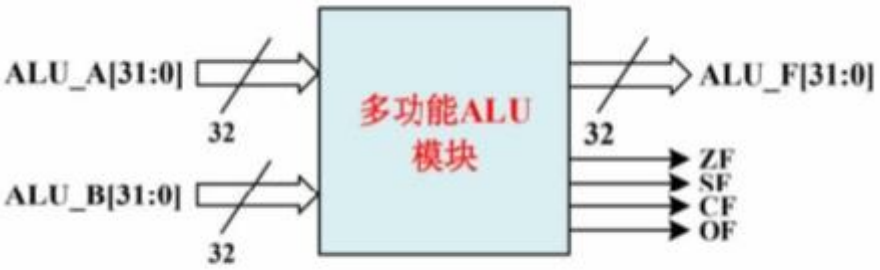


图11.13 多功能 ALU 模块引脚图

2. 设计一个暂存器模块，包含 A,B,F 和标志寄存器 FR 的寄存电路及控制逻辑（这里也可设计为各自独立的 4 个模块）。可以总结出暂存器模块信号，如图 11.14 所示。

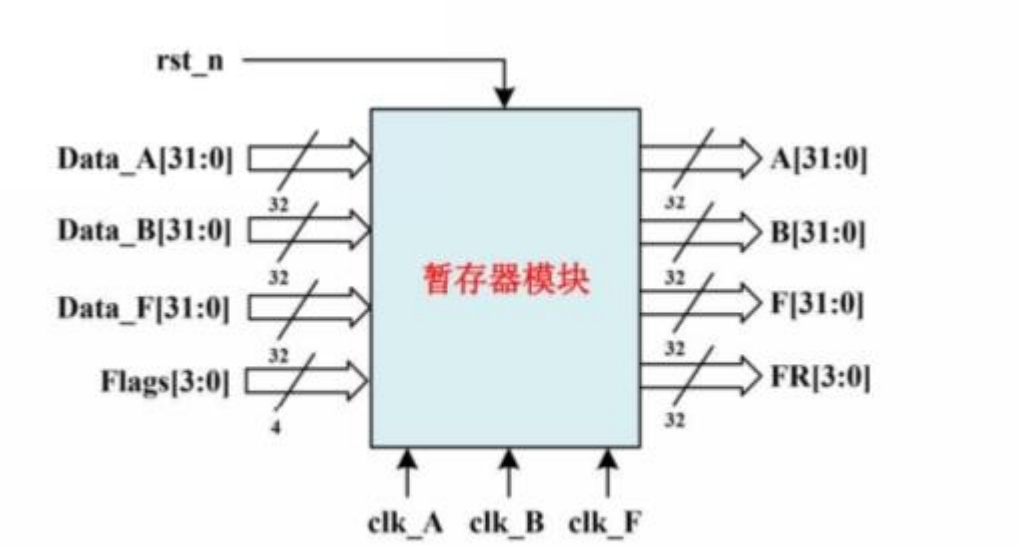


图11.14 暂存器模块引脚图

3.编写一个带暂存器的多功能 ALU 模块,按照图 11.7 结构,将多功能 ALU 模块与暂存器模块连接在一起,该模块的输入输出信号请参照图 11.7 自行整理。

4.针对使用的实验板卡,设计带暂存器的多功能 ALU 模块的板级验证实验方案,编写顶层测试模块。

5.选择典型的输入数据,验证你的 ALU 的各种功能,将实验结果记录到表中,要求所有运算功能和标志位都被有效测试

6.撰写实验报告,格式见附录,重点内容包括:对仿真结果进行分析:描述你设计的板级验证实验方案、模块结构与连接:说明你的板级操作过程:分析记录下来的板级实验结果、得到有效结论。请力所能及回答或实践本实验的“思考与探索”部分

3.实验步骤

1.新建工程,编写多功能 ALU 模块的代码,包括标志位的产生电路,模块端口信号如图 11.13 所示。

2.编写激励代码,仿真测试多功能 ALU 模块:分析仿真结果,确保模块运算逻辑正确标志位产生逻辑正确

3.新建一个暂存器模块,包含 A、B、F 三个 32 位暂存器和一个 4 位的标志寄存器 FR,模块端口信号如图 11.14 所示

4.新建一个带暂存器的多功能ALU模块,将上述两个模块连接起来,可参照图 11.7。

5.设计板级验证的实验方案,然后据此编写一个顶层测试模块

6.可以依据实际需要,对顶层测试模块进行仿真测试,或者直接进入管脚约束环节

7. 新建管脚约束文件,依据设训的板级验证实验方案,进行相应的引脚配置

8. 生成 bit 文件,下载到实验设备的 FPGA 芯片中。

9. 板级实验:按照你所设计的实验方案,操作输入设备、观察输出设备,一般过程为:

(1)按 rst_n 键复位;

(2)拨动开关输入数据 A,按下时钟键 clk_A,打入暂存器 A:

(3)拨动开关输入数据 B,按下时钟键 clk_B,打入暂存器 B

(4)拨动开关选择运算功能 ALU OP,按下时钟键 clk_F,保存结果到 F:

(5)拨动开关选择输出的数据,观察 LED 灯或者数码管,记录实验结果到表中并分析实验结果是否正确

二、实验设计与程序代码

1、 模块设计说明

程序包含了 9 个模块：

1.顶层模块　　2.ALU_A 暂存器模块　　3.ALU_B 暂存器模块

4.F 暂存器模块　5.FR 标志寄存器模块　　6. ALU 功能模块

7.分频器模块　　8.三选一模块　　9.数码管扫描显示模块

1.顶层模块：有序调用其他模块的程序；

2. ALU_A 暂存器模块：为了暂时储存输入数据 A[31:0];

3. ALU_B 暂存器模块：为了暂时储存输入数据 B[31:0];

4. ALU_F 暂存器模块：为了暂时储存输出数据 F[31:0];

5. FR 标志寄存器模块：为了存储并以 LED 灯形式输出 ZF,SF,CF,OF;

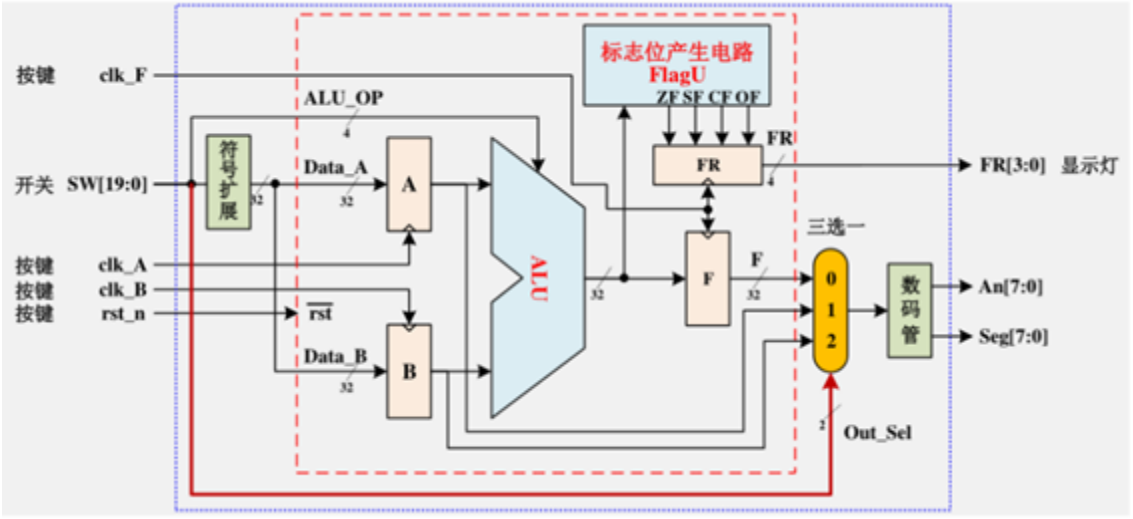
6. ALU 功能模块：实现 10 个计算功能逻辑的模块；

7.分频器模块：是为了给数码管扫描显示模块分配时钟频率；

8.三选一模块：选择显示 A,B,F 的值；

9.数码管扫描显示模块：是为了把 BCD 码数据以十进制数字显示出来

（2）板级验证：



(1)按 rst_n 键复位;

(2)拨动开关输入数据 A,按下时钟键 clk_A,打入暂存器 A:

(3)拨动开关输入数据 B,按下时钟键 clk_B,打入暂存器 B

(4)拨动开关选择运算功能 ALU OP,按下时钟键 clk_F,保存结果到 F:

(5)拨动开关选择输出的数据,观察 LED 灯或者数码管,记录实验结果到表中并分析实验结果是否正确

2.实验程序源代码及注释等

```
module dingceng(clk_1M,CLR,rst_n,clk_A,clk_B,clk_F,SW,FR,AN,seg);//顶层模块
    input clk_1M;
    input CLR;
    input rst_n;
    input clk_A;
    input clk_B;
    input clk_F;
    input [19:0]SW;
    output [3:0]FR;
    output [7:0]AN;
    output [7:0]seg;
    wire [31:0]Data_32;
    wire CLK;
```

```
    wire [31:0]ALU_A;
    wire [31:0]ALU_B;
    wire [31:0]ALU_F;
    wire [31:0]F;
    wire [31:0]Choice;
    wire ZF,SF,CF,OF;
    assign Data_32={{12{SW[19]}},SW[19:0]};
    zcA aa(rst_n,Data_32,clk_A,ALU_A);
    zcB bb(rst_n,Data_32,clk_B,ALU_B);
    ALU cc(SW[3:0],ALU_A,ALU_B,ALU_F,ZF,SF,CF,OF);
    zcF dd(rst_n,ALU_F,clk_F,F);
    jcqFR ee(clk_F,rst_n,ZF,SF,CF,OF,FR);
    fpq gg(CLR,clk_1M,CLK);
    threeone kk(SW[19:18],ALU_A,ALU_B,F,Choice);
    smgsm ff(CLK,CLR,Choice,AN,seg);
endmodule

module threeone(Out_Sel,ALU_A,ALU_B,F,Choice);//三选一 A,B,F 输出到数码管显示
     input [1:0]Out_Sel;
     input [31:0]ALU_A;
     input [31:0]ALU_B;
     input [31:0]F;
     output reg [31:0]Choice;
     always@(*)
     begin
          case(Out_Sel)
                    2'b00:begin Choice=F; end
                    2'b01:begin Choice=ALU_A; end
                    2'b10:begin Choice=ALU_B; end
                    default:begin Choice=F; end
```

```
            endcase
        end
endmodule

module zcA(rst_n,Data_A,clk_A,ALU_A);//A 暂存器
    input rst_n;
    input [31:0]Data_A;
    input clk_A;
    output reg [31:0]ALU_A;
    always@(negedge rst_n or posedge clk_A)
    begin
        if(!rst_n)
            ALU_A<=32'b0;
        else
            ALU_A<=Data_A;
    end
endmodule

module zcB(rst_n,Data_B,clk_B,ALU_B);//B 暂存器
    input rst_n;
    input [31:0]Data_B;
    input clk_B;
    output reg [31:0]ALU_B;
    always@(negedge rst_n or posedge clk_B)
    begin
        if(!rst_n)
            ALU_B<=32'b0;
        else
            ALU_B<=Data_B;
    end
```

```
endmodule

module zcF(rst_n,ALU_F,clk_F,F);//F 暂存器
    input rst_n;
    input [31:0]ALU_F;
    input clk_F;
    output reg [31:0]F;
    always@(negedge rst_n or posedge clk_F)
    begin
        if(!rst_n)
            F<=32'b0;
        else
            F<=ALU_F;
    end
endmodule

module jcqFR(clk_F,rst_n,ZF,SF,CF,OF,FR);//FR 标志寄存器
    input clk_F;
    input rst_n;
    input ZF,SF,CF,OF;
    output reg [3:0]FR;
    always@(negedge rst_n or posedge clk_F)
    begin
        if(!rst_n)
            FR<=4'b0;
        else
            begin
                FR[3]=ZF;
                FR[2]=SF;
                FR[1]=CF;
```

```
                FR[0]=OF;
            end
    end
endmodule

module ALU(ALU_OP,ALU_A,ALU_B,ALU_F,ZF,SF,CF,OF); //ALU 功能模块
    input [3:0]ALU_OP;
    input [31:0]ALU_A;
    input [31:0]ALU_B;
    output [31:0]ALU_F;
    output   ZF;//是否为 0，结果为 0 时，ZF=1;否则，ZF=0
    output   SF;//符号标志位，与运算结果的最高位相同，正数时，SF=0
    output   CF;//进位/借位，最高位产生的进位 C32，加法：C32=1 时，CF=1;减法：C32=0 时，CF=1   无符号
    output   OF;//溢出标志位，有溢出，OF=1；带符号
    reg [31:0]ALU_F;
    reg C32;//最高进位
    reg ZF,SF,CF,OF;
    always@(*)
    begin
        C32=0;
        case(ALU_OP)
            4'b0000:begin {C32,ALU_F}=ALU_A+ALU_B; end                          //加法
            4'b0001:begin ALU_F=ALU_A<<ALU_B; end                               //左移
            4'b0010:begin  ALU_F=($signed(ALU_A)<$signed(ALU_B))?1:0;  end      //有符号数比较
            4'b0011:begin ALU_F=(ALU_A<ALU_B)?1:0; end                  //无符号数比较
            4'b0100:begin ALU_F=ALU_A^ALU_B; end                                //异或
            4'b0101:begin ALU_F=ALU_A>>ALU_B; end               //逻辑右移，高位补零
```

```
            4'b0110:begin ALU_F=ALU_A|ALU_B; end                    //按位或
            4'b0111:begin ALU_F=ALU_B&ALU_A; end                    //按位与
            4'b1000:begin {C32,ALU_F}=ALU_A-ALU_B; end              //减法
            4'b1101:begin  ALU_F=($signed(ALU_A))>>>ALU_B;  end     //算术右移，
高位补 ALU_A[31]
        endcase
          ZF = ALU_F==0;//F 全为 0，则 ZF=1
          SF = ALU_F[31];//符号标志,取 F 的最高位
          CF = C32; //进位借位标志
          OF = ALU_A[31]^ALU_B[31]^ALU_F[31]^C32;//溢出标志公式
    end
endmodule

module fpq(CLR,clk_1M,CLK);             //分频器模块
   input CLR;
   input clk_1M;
   output reg CLK;
   reg [8:0]counter;
   always@(negedge CLR or posedge clk_1M)
      begin
        if(!CLR)
          begin
              counter<=9'd0;
              CLK<=1'b0;
          end
        else if(counter==9'd2000)
              begin
                  CLK<=~CLK;
                  counter<=9'd0;
              end
```

```
        else
                counter<=counter+1'b1;
    end
endmodule

module smgsm(CLK,CLR,Choice,AN,seg);//数码管扫描显示模块
    input CLK; //扫描的间隔始终，控制数码管轮流点亮的频率
    input CLR; //复位信号，用于初始化状态
    input[31:0]Choice;
    output reg [7:0]AN;//位选
    output reg [7:0]seg; //段选
    reg [3:0] data;
    reg [2:0] bit_sel;   //数码管计数器指示，000~111  最左到最右
    always@(negedge CLR or posedge CLK)
    begin
        if(!CLR)
                bit_sel<=3'b000;
        else
            bit_sel<=bit_sel+1'b1;
    end
    always@(*)
        begin
            case(bit_sel)
                3'b000:data<=Choice[3:0];
                3'b001:data<=Choice[7:4];
                3'b010:data<=Choice[11:8];
                3'b011:data<=Choice[15:12];
                3'b100:data<=Choice[19:16];
                3'b101:data<=Choice[23:20];
                3'b110:data<=Choice[27:24];
```

```
                3'b111:data<=Choice[31:28];
                default:data<=data;
            endcase
        end
    always@(*)
        begin
            case(bit_sel)
                3'b000:AN=8'b11111110;
                3'b001:AN=8'b11111101;
                3'b010:AN=8'b11111011;
                3'b011:AN=8'b11110111;
                3'b100:AN=8'b11101111;
                3'b101:AN=8'b11011111;
                3'b110:AN=8'b10111111;
                3'b111:AN=8'b01111111;
                default:AN=8'b11111111;
            endcase
        end
    always@(*)
        begin
            case(data)
                    0:seg<=8'b00000011;
                      1:seg<=8'b10011111;
                      2:seg<=8'b00100101;
                      3:seg<=8'b00001101;
                      4:seg<=8'b10011001;
                      5:seg<=8'b01001001;
                      6:seg<=8'b01000001;
                      7:seg<=8'b00011111;
                      8:seg<=8'b00000001;
```

```
                9:seg<=8'b00001001;
                10:seg<=8'b00010001;
                11:seg<=8'b11000001;
                12:seg<=8'b01100011;
                13:seg<=8'b10000101;
                14:seg<=8'b01100001;
                15:seg<=8'b01110001;
            default:seg<=8'b11111111;
        endcase
    end
endmodule
```

三、实验仿真

1、 仿真代码

这里只是 ALU 模块的仿真激励代码，分别验证 8 种逻辑功能

```
module ALUfz();
//INPUT
reg [3:0]ALU_OP;
reg [31:0]ALU_A;
reg [31:0]ALU_B;
//OUTPUT
wire [31:0]ALU_F;
wire ZF,SF,CF,OF;
initial
    begin
        //加法
          ALU_OP=4'b0000;ALU_A=32'h0000_0000; ALU_B=32'h0000_0000;#50;
          ALU_OP=4'b0000;ALU_A=32'h0000_0001; ALU_B=32'h0000_0001;#50;
          ALU_OP=4'b0000;ALU_A=32'h5555_5555; ALU_B=32'h9999_9999;#50;
          ALU_OP=4'b0000;ALU_A=32'h00FF_00FF; ALU_B=32'h5F5F_5F5F;#50;
```

```
        ALU_OP=4'b0000;ALU_A=32'hF0F0_F0F0; ALU_B=32'h0000_0004;#50;
        ALU_OP=4'b0000;ALU_A=32'hF000_0004; ALU_B=32'h1000_000F;#50;
    //左移
        ALU_OP=4'b0001;ALU_A=32'h0000_0000; ALU_B=32'h0000_0001;#50;
        ALU_OP=4'b0001;ALU_A=32'h0000_0000; ALU_B=32'h0000_0000;#50;
        ALU_OP=4'b0001;ALU_A=32'h5555_5555; ALU_B=32'h9999_9999;#50;
        ALU_OP=4'b0001;ALU_A=32'h00FF_00FF;ALU_B=32'h5F5F_5F5F;#50;
        ALU_OP=4'b0001;ALU_A=32'hF0F0_F0F0; ALU_B=32'h0000_0004;#50;
       ALU_OP=4'b0001;ALU_A=32'h0000_0004;ALU_B=32'hEFEF_EFEF;#50;
     //有符号比较
        ALU_OP=4'b0010;ALU_A=32'h0000_0000; ALU_B=32'h0000_0001;#50;
        ALU_OP=4'b0010;ALU_A=32'h0000_0000; ALU_B=32'h0000_0000;#50;
        ALU_OP=4'b0010;ALU_A=32'h5555_5555; ALU_B=32'h9999_9999;#50;
        ALU_OP=4'b0010;ALU_A=32'h00FF_00FF;ALU_B=32'h5F5F_5F5F;#50;
        ALU_OP=4'b0010;ALU_A=32'hF0F0_F0F0; ALU_B=32'h0000_0004;#50;
        ALU_OP=4'b0010;ALU_A=32'h0000_0004ALU_B=32'hEFEF_EFEF;#50;
     //无符号比较
        ALU_OP=4'b0011;ALU_A=32'h0000_0000; ALU_B=32'h0000_0001;#50;
        ALU_OP=4'b0011;ALU_A=32'h0000_0000; ALU_B=32'h0000_0000;#50;
        ALU_OP=4'b0011;ALU_A=32'h5555_5555; ALU_B=32'h9999_9999;#50;
        ALU_OP=4'b0011;ALU_A=32'h00FF_00FF;ALU_B=32'h5F5F_5F5F;#50;
        ALU_OP=4'b0011;ALU_A=32'hF0F0_F0F0; ALU_B=32'h0000_0004;#50;
       ALU_OP=4'b0011;ALU_A=32'h0000_0004;ALU_B=32'hEFEF_EFEF;#50;
     //异或
       ALU_OP=4'b0100;ALU_A=32'h7FFF_FFFF;ALU_B=32'h7FFF_FFFF;#50;
       ALU_OP=4'b0100;ALU_A=32'hFFFF_FFFF;ALU_B=32'hFFFF_FFFF;#50;
       ALU_OP=4'b0100;ALU_A=32'h5555_5555; ALU_B=32'h9999_9999;#50;
       ALU_OP=4'b0100;ALU_A=32'h00FF_00FF; ALU_B=32'h5F5F_5F5F;#50;
       ALU_OP=4'b0100;ALU_A=32'hF0F0_F0F0; ALU_B=32'h0000_0004;#50;
       ALU_OP=4'b0100;ALU_A=32'h0000_0004; ALU_B=32'hEFEF_EFEF;#50;
```

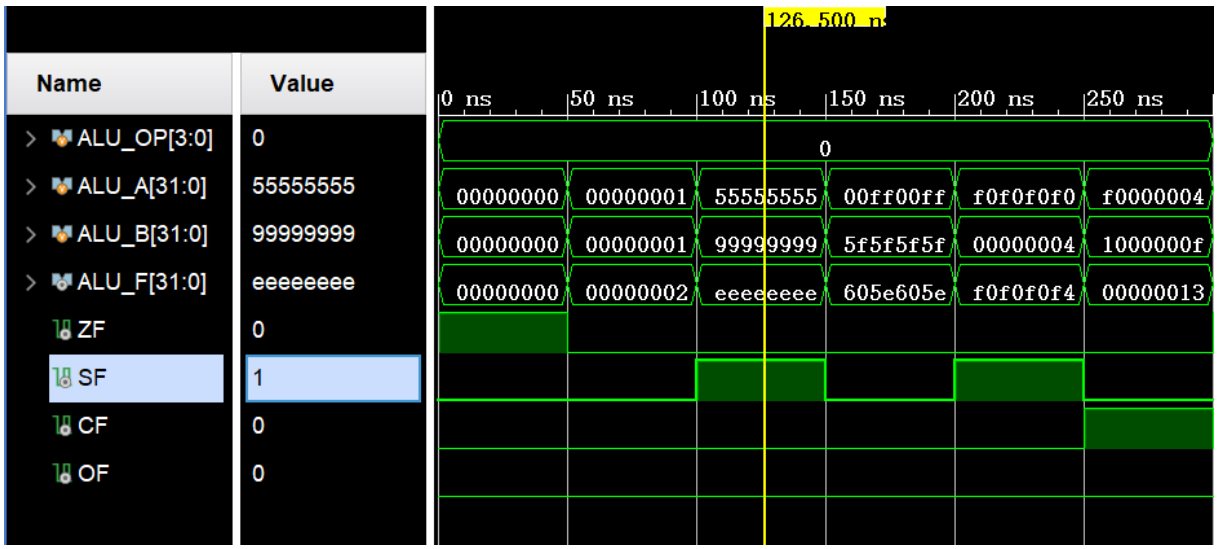
```
//逻辑右移，高位补零
  ALU_OP=4'b0101;ALU_A=32'h7FFF_FFFF; ALU_B=32'h7FFF_FFFD;#50;
  ALU_OP=4'b0101;ALU_A=32'h7FFF_FFFF; ALU_B=32'hFFFF_FFFF;#50;
  ALU_OP=4'b0101;ALU_A=32'h5555_5555; ALU_B=32'h9999_9999;#50;
  ALU_OP=4'b0101;ALU_A=32'h00FF_00FF; ALU_B=32'h5F5F_5F5F;#50;
  ALU_OP=4'b0101;ALU_A=32'hF0F0_F0F0; ALU_B=32'h0000_0004;#50;
  ALU_OP=4'b0101;ALU_A=32'h0000_0004; ALU_B=32'hEFEF_EFEF;#50;
//按位或
    ALU_OP=4'b0110;ALU_A=32'h7FFF_FFFF; ALU_B=32'h8FFF_FFFF;#50;
    ALU_OP=4'b0110;ALU_A=32'hFFFF_FFFF;ALU_B=32'h7FFF_FFFF;#50;
    ALU_OP=4'b0110;ALU_A=32'h5555_5555; ALU_B=32'h9999_9999;#50;
    ALU_OP=4'b0110;ALU_A=32'h00FF_00FF; ALU_B=32'h5F5F_5F5F;#50;
    ALU_OP=4'b0110;ALU_A=32'hF0F0_F0F0; ALU_B=32'h0000_0004;#50;
    ALU_OP=4'b0110;ALU_A=32'h0000_0004; ALU_B=32'hEFEF_EFEF;#50;
//按位与
    ALU_OP=4'b0111;ALU_A=32'h0000_0001; ALU_B=32'h0000_0001;#50;
    ALU_OP=4'b0111;ALU_A=32'h0000_0001; ALU_B=32'h0000_0008;#50;
    ALU_OP=4'b0111;ALU_A=32'h5555_5555; ALU_B=32'h9999_9999;#50;
    ALU_OP=4'b0111;ALU_A=32'h00FF_00FF; ALU_B=32'h5F5F_5F5F;#50;
    ALU_OP=4'b0111;ALU_A=32'hF0F0_F0F0; ALU_B=32'h0000_0004;#50;
    ALU_OP=4'b0111;ALU_A=32'h0000_0004; ALU_B=32'hEFEF_EFEF;#50;
//减法
    ALU_OP=4'b1000;ALU_A=32'h0000_0001; ALU_B=32'h0000_0001;#50;
    ALU_OP=4'b1000;ALU_A=32'h0000_0001; ALU_B=32'h0000_0008;#50;
    ALU_OP=4'b1000;ALU_A=32'h5555_5555; ALU_B=32'h9999_9999;#50;
    ALU_OP=4'b1000;ALU_A=32'h00FF_00FF; ALU_B=32'h5F5F_5F5F;#50;
    ALU_OP=4'b1000;ALU_A=32'hF0F0_F0F0; ALU_B=32'h0000_0004;#50;
    ALU_OP=4'b1000;ALU_A=32'h0000_0004; ALU_B=32'hEFEF_EFEF;#50;
//算术右移，高位补 ALU_A[31]
    ALU_OP=4'b1101;ALU_A=32'h0000_0001; ALU_B=32'h0000_0001;#50;
```

```
        ALU_OP=4'b1101;ALU_A=32'h0000_0001; ALU_B=32'h0000_0008;#50;
        ALU_OP=4'b1101;ALU_A=32'h5555_5555; ALU_B=32'h9999_9999;#50;
        ALU_OP=4'b1101;ALU_A=32'h00FF_00FF; ALU_B=32'h5F5F_5F5F;#50;
        ALU_OP=4'b1101;ALU_A=32'hF0F0_F0F0; ALU_B=32'h0000_0004;#50;
        ALU_OP=4'b1101;ALU_A=32'h0000_0004; ALU_B=32'hEFEF_EFEF;#50;
    end
ALU fz(ALU_OP,ALU_A,ALU_B,ALU_F,ZF,SF,CF,OF);
endmodule
```

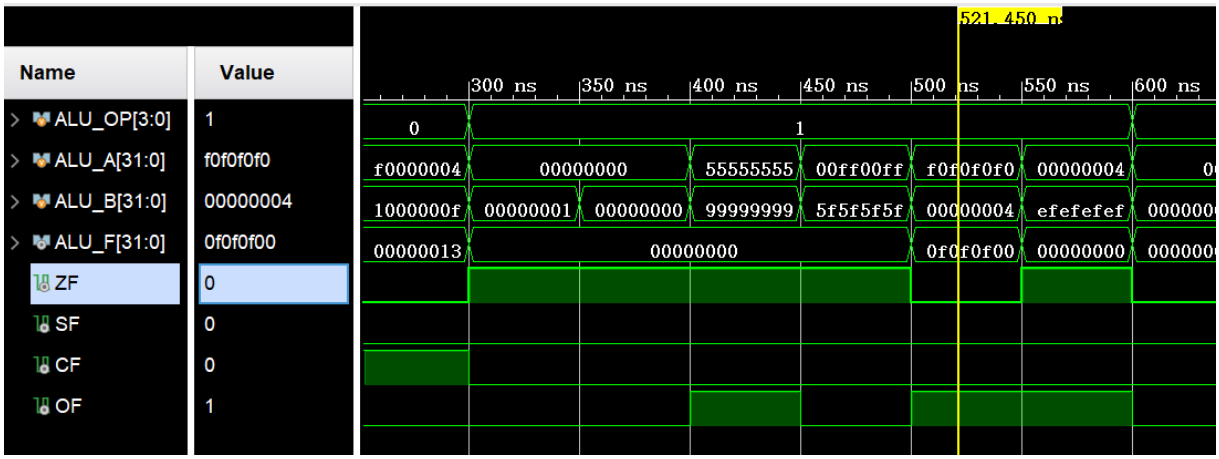
2、 仿真波形

（运行仿真时，波形截图）

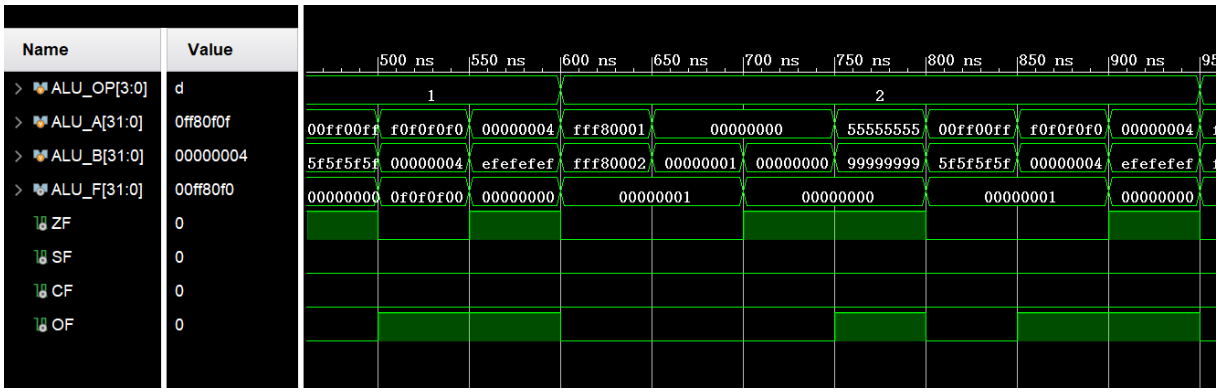
加法：



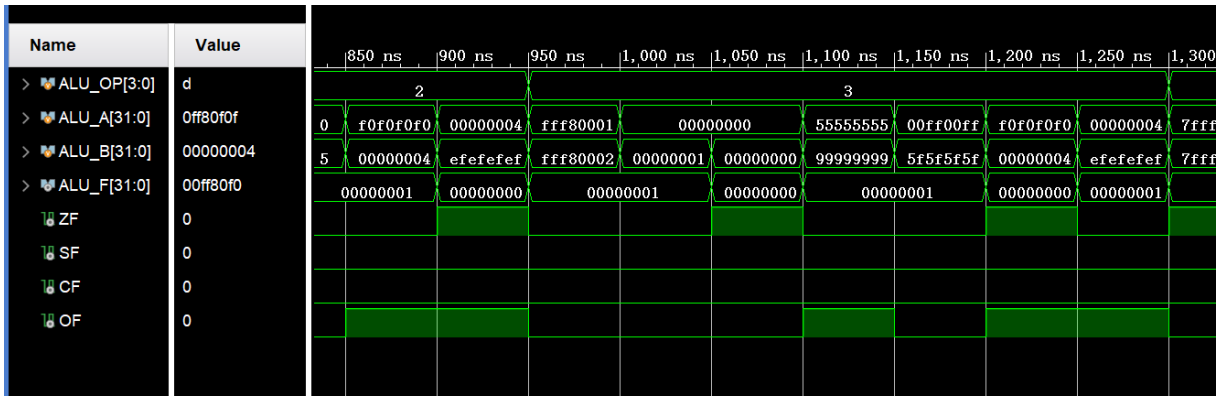
左移：



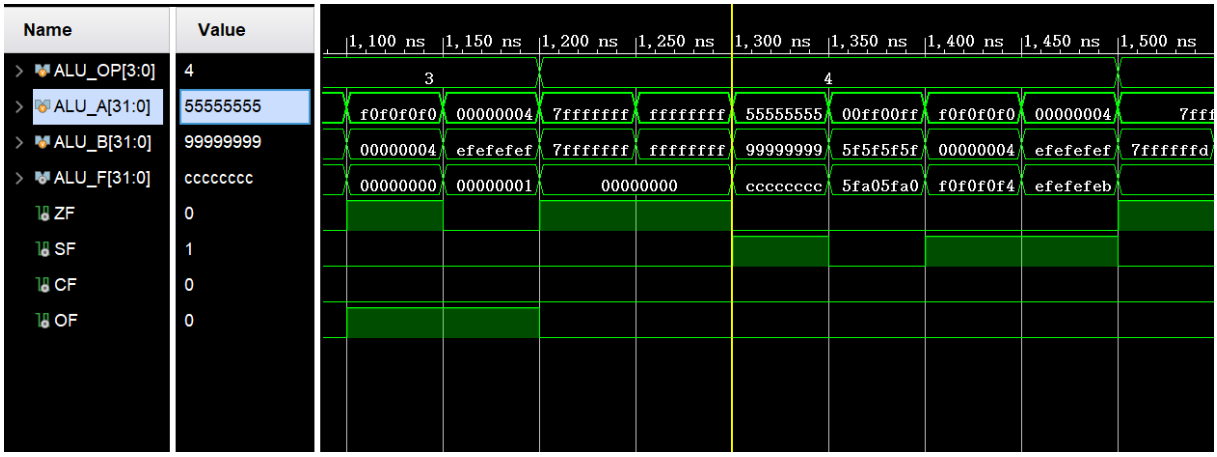
有符号数比较：



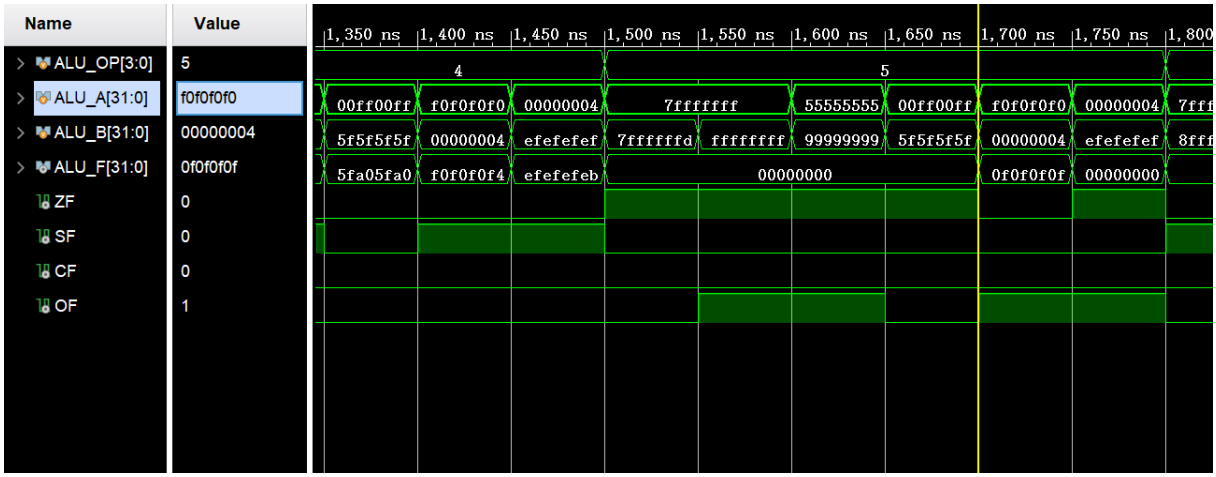
无符号数比较：



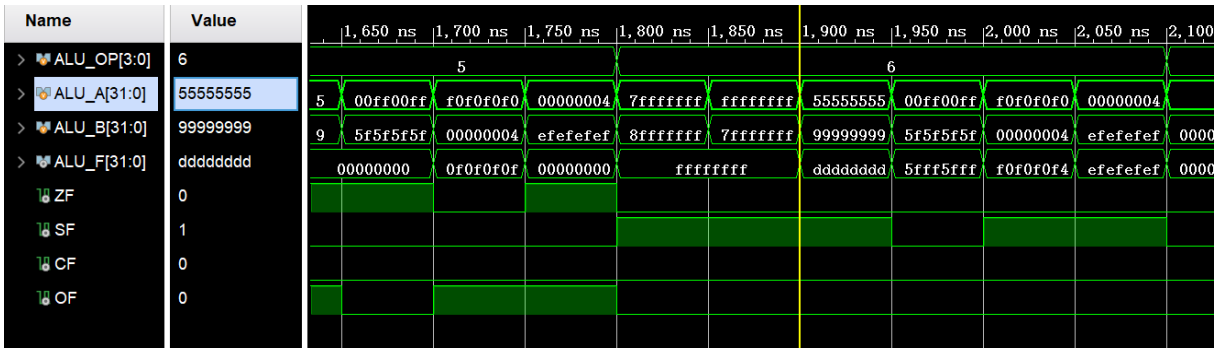
异或：



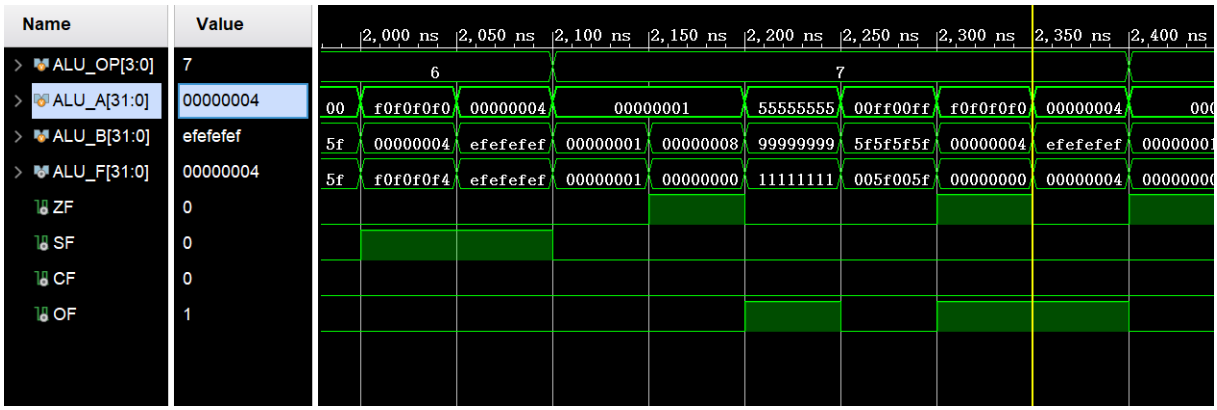
逻辑右移：高位补零：


Name
Value
ALU_OP[3:0]
ALU_A[31:0]
ALU_B[31:0]
ALU_F[31:0]
ZF
SF
CF
OF
5
f0f0f0f0
00000004
0f0f0f0f
0
0
0
1
1,350 ns
1,400 ns
1,450 ns
1,500 ns
1,550 ns
1,600 ns
1,650 ns
1,700 ns
1,750 ns
4
5
00ff00ff
f0f0f0f0
00000004
7fffffff
55555555
00ff00ff
f0f0f0f0
00000004
5f5f5f5f
00000004
efefefef
7ffffffd
ffffffff
99999999
5f5f5f5f
00000004
efefefef
5fa05fa0
f0f0f0f4
efefefeb
00000000
0f0f0f0f
00000000

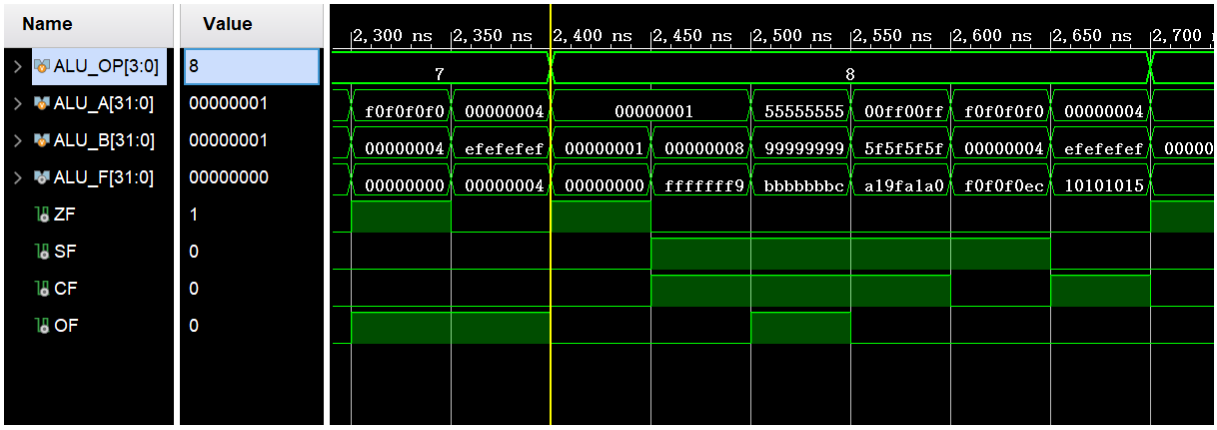
按位或：


Name
Value
ALU_OP[3:0]
ALU_A[31:0]
ALU_B[31:0]
ALU_F[31:0]
ZF
SF
CF
OF
6
55555555
99999999
dddddddd
0
1
0
0
1,650 ns
1,700 ns
1,750 ns
1,800 ns
1,850 ns
1,900 ns
1,950 ns
2,000 ns
2,050 ns
5
6
00ff00ff
f0f0f0f0
00000004
7fffffff
ffffffff
55555555
00ff00ff
f0f0f0f0
00000004
5f5f5f5f
00000004
efefefef
8fffffff
7fffffff
99999999
5f5f5f5f
00000004
efefefef
00000000
0f0f0f0f
00000000
ffffffff
dddddddd
5fff5fff
f0f0f0f4
efefefef

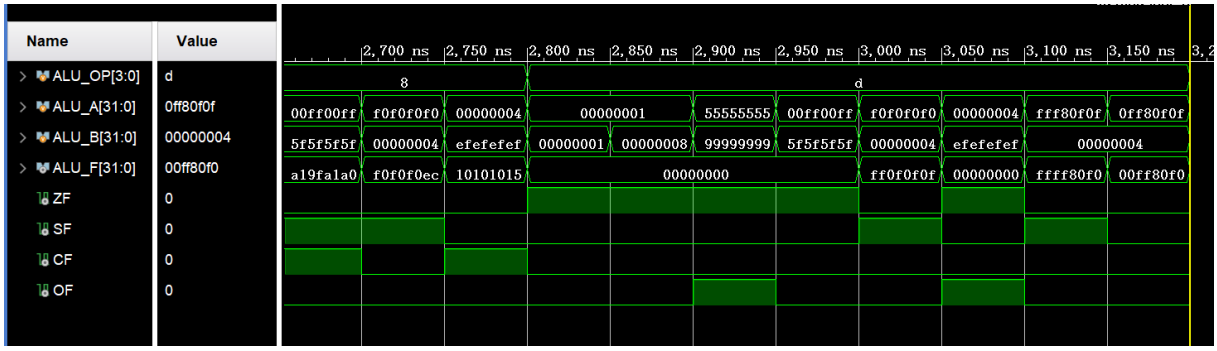
按位与：


Name
Value
ALU_OP[3:0]
ALU_A[31:0]
ALU_B[31:0]
ALU_F[31:0]
ZF
SF
CF
OF
7
00000004
efefefef
00000004
0
0
0
1
2,000 ns
2,050 ns
2,100 ns
2,150 ns
2,200 ns
2,250 ns
2,300 ns
2,350 ns
2,400 ns
6
7
00
f0f0f0f0
00000004
00000001
55555555
00ff00ff
f0f0f0f0
00000004
5f
00000004
efefefef
00000001
00000008
99999999
5f5f5f5f
00000004
efefefef
5f
f0f0f0f4
efefefef
00000001
00000000
11111111
005f005f
00000000
00000004

减法：



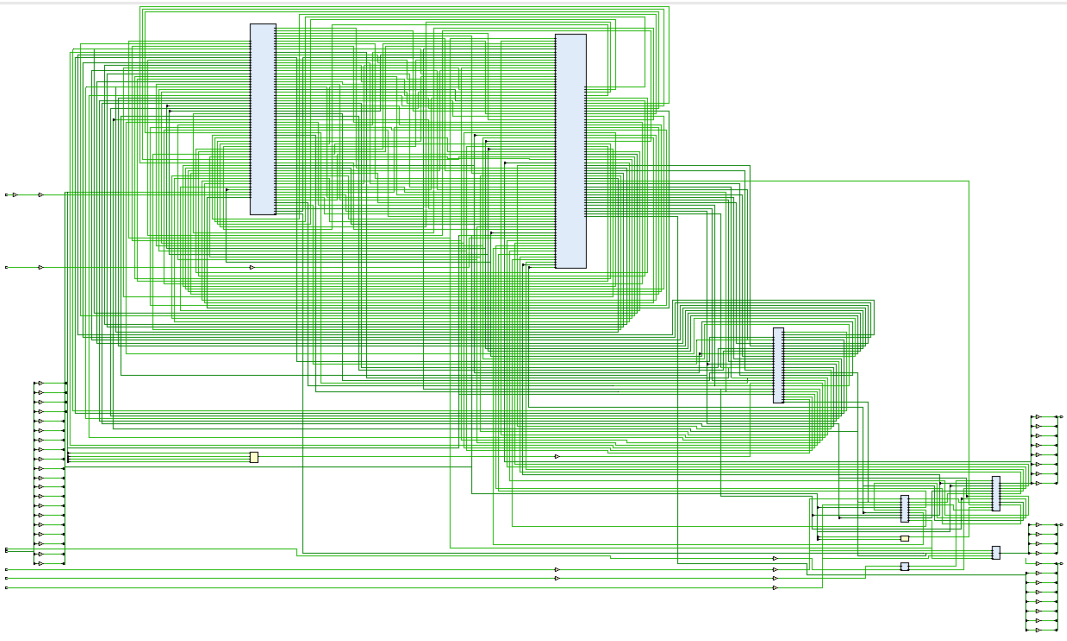
算术右移；高位补 ALU_A[31]:



3、 仿真结果分析

ALU 模块的 10 个逻辑功能都符合预期的效果。逻辑成功。

四、电路图



五、引脚配置

（引脚约束文件的内容，描述主要配置情况）

```
set_property IOSTANDARD LVCMOS18 [get_ports {seg[7]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[6]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[0]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[7]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[6]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[0]}]

set_property IOSTANDARD LVCMOS18 [get_ports {SW[19]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[18]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[17]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[16]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[15]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[14]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[13]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[12]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[11]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[10]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[9]}]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports {SW[8]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[7]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[6]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[0]}]

set_property IOSTANDARD LVCMOS18 [get_ports clk_1M]
set_property IOSTANDARD LVCMOS18 [get_ports clk_A]
set_property IOSTANDARD LVCMOS18 [get_ports clk_B]
set_property IOSTANDARD LVCMOS18 [get_ports clk_F]
set_property IOSTANDARD LVCMOS18 [get_ports CLR]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n]

set_property IOSTANDARD LVCMOS18 [get_ports {FR[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {FR[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {FR[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {FR[0]}]

set_property PULLDOWN true [get_ports {SW[19]}]
set_property PULLDOWN true [get_ports {SW[18]}]
set_property PULLDOWN true [get_ports {SW[17]}]
set_property PULLDOWN true [get_ports {SW[16]}]
set_property PULLDOWN true [get_ports {SW[15]}]
set_property PULLDOWN true [get_ports {SW[14]}]
set_property PULLDOWN true [get_ports {SW[13]}]
set_property PULLDOWN true [get_ports {SW[12]}]
```

```
set_property PULLDOWN true [get_ports {SW[11]}]
set_property PULLDOWN true [get_ports {SW[10]}]
set_property PULLDOWN true [get_ports {SW[9]}]
set_property PULLDOWN true [get_ports {SW[8]}]
set_property PULLDOWN true [get_ports {SW[7]}]
set_property PULLDOWN true [get_ports {SW[6]}]
set_property PULLDOWN true [get_ports {SW[5]}]
set_property PULLDOWN true [get_ports {SW[4]}]
set_property PULLDOWN true [get_ports {SW[3]}]
set_property PULLDOWN true [get_ports {SW[2]}]
set_property PULLDOWN true [get_ports {SW[1]}]
set_property PULLDOWN true [get_ports {SW[0]}]

set_property PULLDOWN true [get_ports clk_1M]
set_property PULLDOWN true [get_ports clk_A]
set_property PULLDOWN true [get_ports clk_B]
set_property PULLDOWN true [get_ports clk_F]
set_property PULLDOWN true [get_ports CLR]
set_property PULLDOWN true [get_ports rst_n]

set_property PACKAGE_PIN E3 [get_ports clk_1M]
set_property PACKAGE_PIN N17 [get_ports clk_A]
set_property PACKAGE_PIN P18 [get_ports clk_B]
set_property PACKAGE_PIN P17 [get_ports clk_F]
set_property PACKAGE_PIN U17 [get_ports CLR]
set_property PACKAGE_PIN U18 [get_ports rst_n]

set_property PACKAGE_PIN U6 [get_ports {FR[3]}]
set_property PACKAGE_PIN R5 [get_ports {FR[2]}]
set_property PACKAGE_PIN U7 [get_ports {FR[1]}]
```

```
set_property PACKAGE_PIN R6 [get_ports {FR[0]}]

set_property PACKAGE_PIN V5 [get_ports {SW[19]}]
set_property PACKAGE_PIN T4 [get_ports {SW[18]}]
set_property PACKAGE_PIN V6 [get_ports {SW[17]}]
set_property PACKAGE_PIN T5 [get_ports {SW[16]}]
set_property PACKAGE_PIN T6 [get_ports {SW[15]}]
set_property PACKAGE_PIN V7 [get_ports {SW[14]}]
set_property PACKAGE_PIN R8 [get_ports {SW[13]}]
set_property PACKAGE_PIN U9 [get_ports {SW[12]}]
set_property PACKAGE_PIN T9 [get_ports {SW[11]}]
set_property PACKAGE_PIN V10 [get_ports {SW[10]}]
set_property PACKAGE_PIN R10 [get_ports {SW[9]}]
set_property PACKAGE_PIN U11 [get_ports {SW[8]}]
set_property PACKAGE_PIN R11 [get_ports {SW[7]}]
set_property PACKAGE_PIN U12 [get_ports {SW[6]}]
set_property PACKAGE_PIN T13 [get_ports {SW[5]}]
set_property PACKAGE_PIN V14 [get_ports {SW[4]}]
set_property PACKAGE_PIN T14 [get_ports {SW[3]}]
set_property PACKAGE_PIN V15 [get_ports {SW[2]}]
set_property PACKAGE_PIN R15 [get_ports {SW[1]}]
set_property PACKAGE_PIN U16 [get_ports {SW[0]}]

set_property PACKAGE_PIN F14 [get_ports {seg[7]}]
set_property PACKAGE_PIN N14 [get_ports {seg[6]}]
set_property PACKAGE_PIN J13 [get_ports {seg[5]}]
set_property PACKAGE_PIN G13 [get_ports {seg[4]}]
set_property PACKAGE_PIN F13 [get_ports {seg[3]}]
set_property PACKAGE_PIN G14 [get_ports {seg[2]}]
set_property PACKAGE_PIN M13 [get_ports {seg[1]}]
```

set_property PACKAGE_PIN H14 [get_ports {seg[0]}]

set_property PACKAGE_PIN C9 [get_ports {AN[7]}]
set_property PACKAGE_PIN C10 [get_ports {AN[6]}]
set_property PACKAGE_PIN D10 [get_ports {AN[5]}]
set_property PACKAGE_PIN C11 [get_ports {AN[4]}]
set_property PACKAGE_PIN M17 [get_ports {AN[3]}]
set_property PACKAGE_PIN J14 [get_ports {AN[2]}]
set_property PACKAGE_PIN K13 [get_ports {AN[1]}]
set_property PACKAGE_PIN P14 [get_ports {AN[0]}]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_1M]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_A]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_B]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_F]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets CLR]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets rst_n]

六、思考与探索

1、 实验结果记录：

（实验操作的过程及结果记录）

Data_A[31:0]	Data_B[31:0]	ALU_OP	功能	F[31:0]	FR(ZF SF CF OF)
FFF8_0F0F	FFF8_0F0F	0000	加法	FFF0_1E1E	0110
FFF8_0F0F	0000_0100	0001	左移	FF80_F0F0	0100
FFF8_0001	FFF8_0002	0010	有符号数比较	0000_0001	0000
FFF8_0001	FFF8_0002	0011	无符号数比较	0000_0001	0000
FFF8_0F0F	FFF8_0F1F	0100	异或	0000_0010	0000
FFF8_0F0F	0000_0100	0101	逻辑右移	0FFF_80F0	0001

FFF8_0F0F	FFF8_0F1F	0110	按位或	FFF8_0F1F	0101
FFF8_0F0F	FFF8_0F1F	0111	按位与	FFF8_0F0F	0101
FFF8_0F0F	FFF8_0917	1000	减法	0000_05F8	0000
FFF8_0F0F	0000_0004	1101	算数右移	FFFF_80F0	0100

2、 实验结论：

实验结果符合预期结果，逻辑正确，实验成功。

3、 问题与解决方案：

(1)板子按键与输入位数不符，采用了符号扩展的方法，当按键不足时，按键最高位补全剩下的位数。

(2)ALU 模块中有符号比较验证发现结果有异议，后想到有符号数是补码；

如 FFF8_0001，FFF8_0002 原码是 8007FFFF, 8007FFFE; 8007FFFF < 8007FFFE;

所以输出 F=1；

(3)数码管显示异常，后来发现是只写了 0-9 的显示，A-F 的显示逻辑没有写入。

4、 思考题：

1.你是如何实现有符号数的比较置位运算 slt 和无符号数的比较置位运算 sltu?你使用什么数据来验证你的 slt 和 sltu 运算是正确的?

答：ALU_F=($signed(ALU_A)<$signed(ALU_B))?1:0; //有符号数比较
ALU_F=(ALU_A<ALU_B)?1:0; //无符号数比较
使用了 FFF8_0001，FFF8_0002 表示两个负数
其原码是 8007FFFF, 8007FFFE; 8007FFFF < 8007FFFE;
所以当为有符号比较时，输出 F=1；
无符号是显然输出 F=1;

2.你是如何实现逻辑右移 srl 运算和算术右移 sra 运算的?同样,你使用什么数据来验证你的 srl 和 sra 运算是正确的?

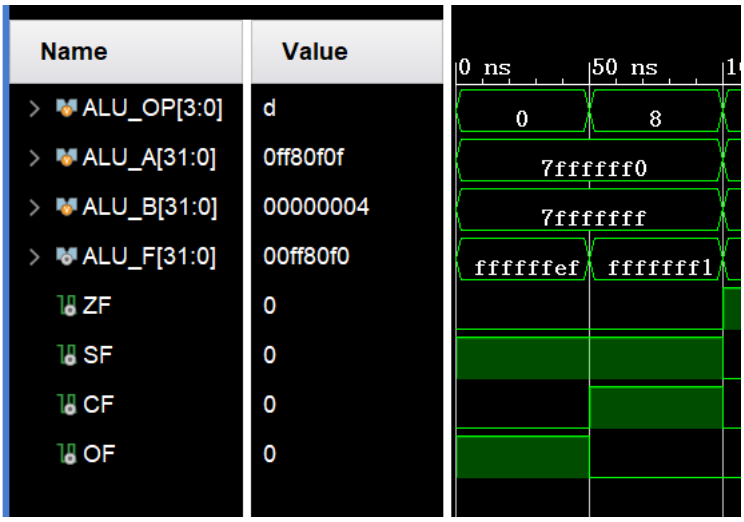
答：ALU_F=ALU_A>>ALU_B; //逻辑右移，高位补零
ALU_F=($signed(ALU_A))>>>ALU_B; //算术右移，高位补 ALU_A[31]
逻辑右移使用了 FFF8_0F0F，右移 0000_0100 位，得到结果 0FFF_80F0;
算术右移使用了 FFF8_0F0F 和 0FF8_0F0F,右移 0000_0100 位，
分别得到结果 FFFF_80F0 和 00FF_80F0;
结果显然正确。

3.式(11.6)为硬件逻辑表达式,如果采用 Verilog HDL 语言的运算符进行加法和减法运算,如下所

{C32,ALU_F}=ALU_A+ALU_B
{C32, ALU_F}=ALU_A-ALU_B

这里的 C32 是式 11.6 中的 C 吗?请通过仿真验证当 ALU_A=32'h7FFF_FFF0, ALU_B=32'h7FFF FFFF 时,加减法运算后,CF 的逻辑是否正确,从而给出答案

答：



CF 逻辑显然正确。

4.你的板级验证实验,使用了什么板卡?输入输出设备够吗?如果不够,你是如何解决的?你用了什么输出设备显示运算结果?能观察什么数据?说说你的方案

答：使用了 HCS-A01 板卡；输入输出设备不够；输入使用了符号扩展加暂存器的方法；输出使用了数码管显示;可以选择显示 A,B,F，和 ZF,SF,CF,OF;

8.谈谈你在实验中碰到了哪些问题?又是如何解决的?

答:(1)板子按键与输入位数不符，采用了符号扩展的方法，当按键不足时，按键最高位补全剩下的位数。

(2)ALU 模块中有符号比较验证发现结果有异议，后想到有符号数是补码；

如 FFF8_0001，FFF8_0002 原码是 8007FFFF, 8007FFFE; 8007FFFF < 8007FFFE;

所以输出 F=1；

(3)数码管显示异常，后来发现是只写了 0-9 的显示，A-F 的显示逻辑没有写入。