中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2016K8009915009　姓名：钟赟　专业：计算机科学与技术

实验序号：___　实验名称：基本功能部件设计—Register File & ALU

一、　逻辑电路结构与仿真波形的截图及说明(比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构、相应信号的仿真波形和信号变化的说明等)

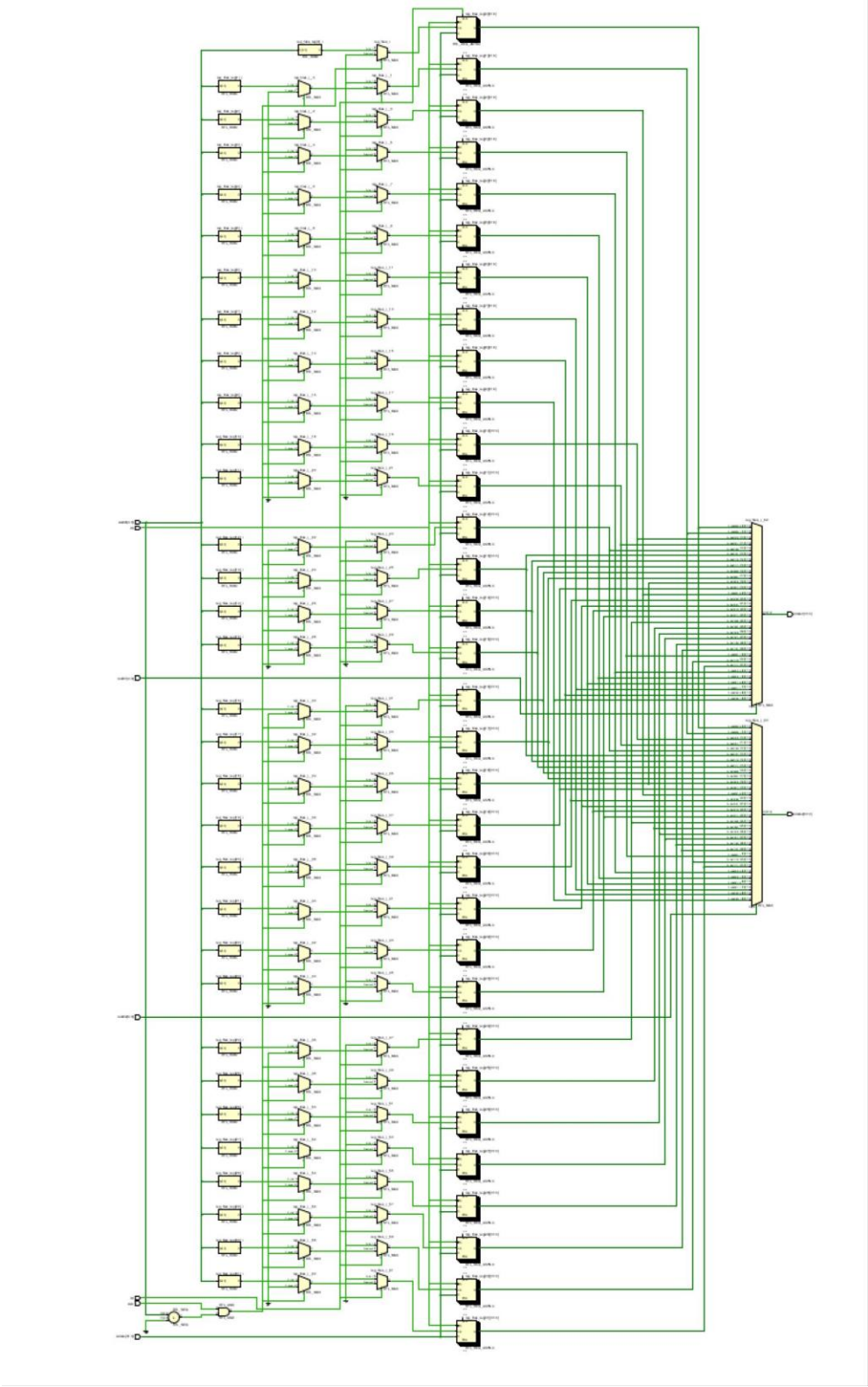
1. 通用寄存器堆功能部件设计

（1） 关键 RTL 代码段

reg_file.v 的关键 RTL 代码段如下图所示

```
reg [`DATA_WIDTH - 1:0] reg_files [0:`DATA_WIDTH - 1];//定义32个32位寄存器组

//时钟逻辑实现同步写
always @(posedge clk or posedge rst)//
begin
    if(rst)//读到复位信号，将0号寄存器置为0
        begin
            reg_files[0] <= 32'b0;
        end
    else
    begin
        if(wen && waddr != 5'b0)//对于0号寄存器以外的寄存器，当写使能为1时，读入数据
            reg_files[waddr] <= wdata;
    end
end

//组合逻辑实现异步读
assign rdata1 = reg_files[raddr1];//读口1读入数据
assign rdata2 = reg_files[raddr2];//读口2读入数据
```

（2） 逻辑电路结构图

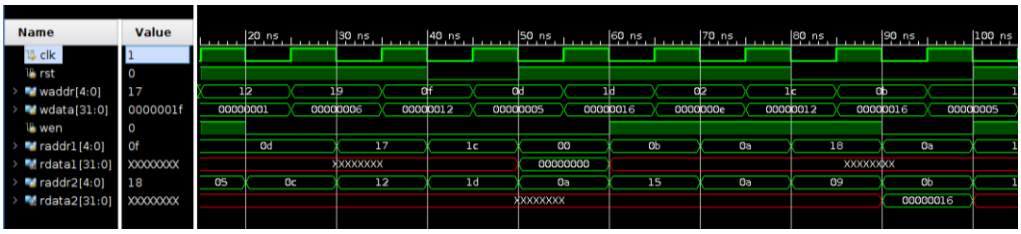


（3） 仿真波形

testbench 代码如下：

```
reg reg_files;//定义寄存器寄存器变量

initial begin
        //设置初值
        clk = 0;
        rst = 0;
        raddr1 = 0;
        raddr2 = 0;
        waddr = 0;
        wdata = 0;
        wen = 0;
        forever begin//对写使能，复位信号，寄存器堆，读口地址1，读口地址2赋值随机数
          begin
          #10
                rst = {$random} % 2;
                wen = {$random} % 2;
                reg_files = {$random} % 32;
                raddr1 = {$random} % 32;
                raddr2 = {$random} % 32;
  end
end
end

//设置时钟信号
always begin
        #5 clk = ~clk;
end

always @(posedge clk)//时钟信号变化时，给写口地址和写入的数据赋值随机数
begin
        waddr = {$random} % 32;
        wdata = {$random} % 32;
end
```

仿真波形如下：



由波形可看出，在 50ns 时，复位信号 rst=1，此时 raddr1=0，读出 0 号寄存器的值为 0；

在 10ns 时，在 12 号寄存器读入 data 00000001，但此时写使能 wen=0，未能读入，故在 30ns 时，通过读口 2 读 12 号寄存器的值，结果不存在。

2. ALU 功能部件设计

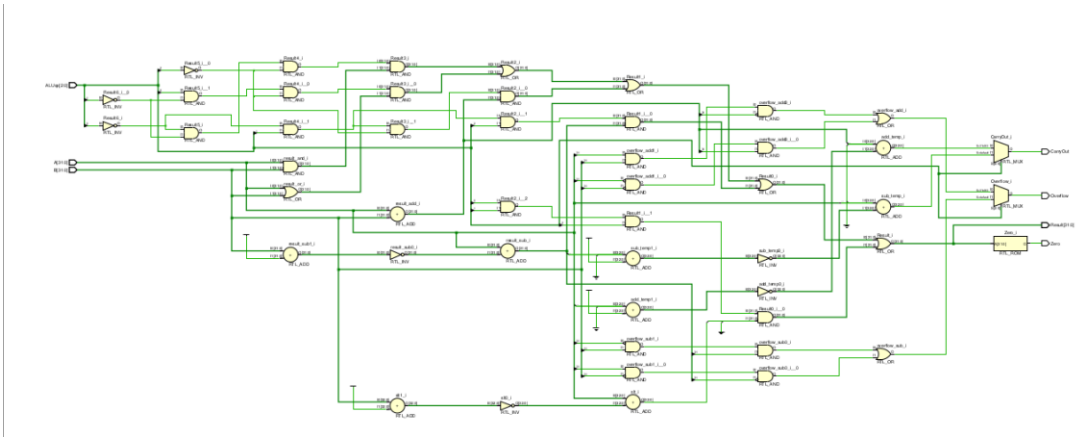
（1） 关键 RTL 代码段

alu.v 的关键代码段如下所示：

```
    //定义线型变量，记录进行and,or,add,sub,slt操作后产生的结果
    wire [`DATA_WIDTH-1:0] result_and;
    wire [`DATA_WIDTH-1:0] result_or;
    wire [`DATA_WIDTH-1:0] result_add;
    wire [`DATA_WIDTH-1:0] result_sub;
    wire [`DATA_WIDTH-1:0] result_slt;

    wire [`DATA_WIDTH:0] add_temp;//用于计算加时是否有进借位
    wire [`DATA_WIDTH:0] sub_temp;//用于计算减时是否有进借位
    wire [`DATA_WIDTH-1:0] cout_add;//记录加法时的CarryOut值
    wire [`DATA_WIDTH-1:0] cout_sub;//记录减法时的CarryOut值
    wire [`DATA_WIDTH-1:0] overflow_add;//用于计算加法时的Overflow值
    wire [`DATA_WIDTH-1:0] overflow_sub;//用于计算减法时的Overflow值
    wire [`DATA_WIDTH:0] slt;//用于比较A和B的大小

    //计算and, or, add, 和sub操作的结果
    assign result_and = A & B;
    assign result_or = A | B;
    assign result_add = A + B;
    assign result_sub = A + ~(B+32'hffffffff);//减法用补码计算

    //增加一位符号位，计算CarryOut
    assign add_temp = {0,result_add}+~({0,A}+33'h1ffffffff);//和减去加数；若溢出即产生进位，则结果的符号位为1，否则为0
    assign sub_temp = {0,A}+~({0,result_sub}+33'h1ffffffff);//被减数减去减数；若溢出，即产生借位，则结果符号位为1，否则为0


    assign cout_add = add_temp[`DATA_WIDTH];//将上一步计算结果的符号位赋给cout_add
    assign cout_sub = sub_temp[`DATA_WIDTH];//将上一步计算结果的符号位赋给cout_sub


    //计算Overflow
    assign overflow_add = ((A[`DATA_WIDTH-1]&B[`DATA_WIDTH-1]&(~result_add
[`DATA_WIDTH-1]))|
((~A[`DATA_WIDTH-1])&(~B[`DATA_WIDTH-1])&result_add[`DATA_WIDTH-1]));//两个正数的加法是一个负数，或两个负数的加法是正数的情况下，产生溢出
    assign overflow_sub = (((~A[`DATA_WIDTH-1])&B[`DATA_WIDTH-1]&(result_sub
[`DATA_WIDTH-1]))|
(A[`DATA_WIDTH-1]&(~B[`DATA_WIDTH-1])&(~result_sub[`DATA_WIDTH-1])));//正数减负数为负数，或负数减正数为正数的情况下，产生溢出

    //将上述结果赋给CarryOut和Overflow
    assign CarryOut = (ALUop == 32'b010) ? cout_add : cout_sub;
    assign Overflow = (ALUop == 32'b010) ? overflow_add : overflow_sub;


    //将A和B扩展为33位，原来的最高位做扩展后的最高位，进行A-B操作，若A比B小，则最高位为1
    assign slt = {A[`DATA_WIDTH-1],A} + ~({B[`DATA_WIDTH-1],B}+33'h1ffffffff);

    //将上述结果赋给Result_slt
    assign result_slt = slt[`DATA_WIDTH];
```

```
//将上述各种运算的结果赋给Result
assign Result = (result_and & {32{(~ALUop[0]&~ALUop[1]&~ALUop[2])}}) |
        (result_or & {32{(ALUop[0]&~ALUop[1]&~ALUop[2])}}) |
        (result_add & {32{(~ALUop[0]&ALUop[1]&~ALUop[2])}}) |
        (result_sub & {32{(~ALUop[0]&ALUop[1]&ALUop[2])}}) |
        (result_slt & {32{(ALUop[0]&ALUop[1]&ALUop[2])}});

//如果结果为0，Zero赋值为1
assign Zero = (Result == 32'b0)?32'b1:32'b0;
```

（2） 逻辑电路结构图



（3） 仿真波形

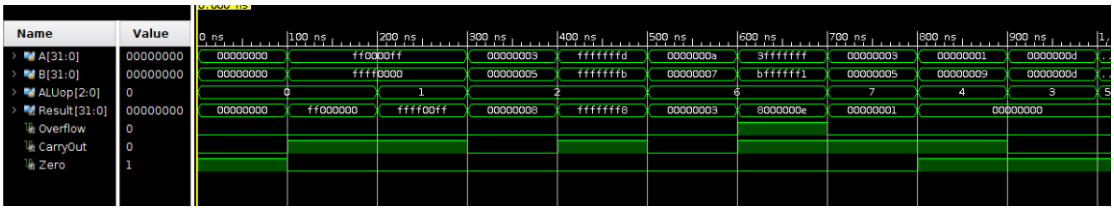
testbench 代码如下：

首先给 A，B，ALUop 赋特殊值，检测 ALU 的各个功能；再赋随机值：

```
initial
begin
        A = 0;
        B = 0;
        ALUop = 0;

        #10;
        ALUop = 3'b000;
        A = 32'hff0000ff;
        B = 32'hffff0000;

        #10;
        ALUop = 3'b001;
        A = 32'hff0000ff;
        B = 32'hffff0000;

        #10;
        ALUop = 3'b010;
        A = 32'h3;
        B = 32'h5;
```

```
        #10;
        ALUop = 3'b010;
        A = 32'hfffffffd;
        B = 32'hfffffffb;

        #10;
        ALUop = 3'b110;
        A = 32'ha;
        B = 32'h7;

        #10;
        ALUop = 3'b110;
        A = 0;
        B = 0;

        #10;
        ALUop = 3'b111;
        A = 32'h3;
        B = 32'h5;
        forever begin
                begin
                #10
                        ALUop = {$random} % 8;
                        A = {$random} % 32;
                        B = {$random} % 32;
                end
        end
    end
```

仿真波形如下：



由波形图可知：

t=0ns，A=0，B=0，A&B=0，Zero=1;

t=100ns，A=ff0000ff，B=ffff0000，A&B=ff000000;

t=200ns，A=ff0000ff，B=ffff0000，A | B=ffff00ff;

t=300ns，A=3，B=5，A+B=8，不产生进位，无溢出；

t=400ns，A=fffffffd，B=fffffffb，A+B=fffffff8，产生进位，CarryOut=1，无溢出；

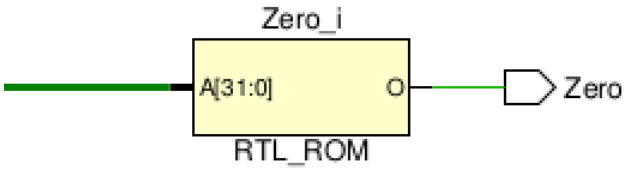
t=500ns，A=a，B=7，A-B=3，不产生借位，无溢出；

t=600ns，A=3fffffff，B=bffffff1，A-B=8000000e，产生借位，CarryOut=1，且溢出，Overflow=1；

以上检验结果均为正确的。

二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，仿真、本地上板及云平台调试过程中的难点等）

1. 开始写 alu 时，用了 always 语句和 case 语句。听闻要完全用组合逻辑写，我删去了所有的 reg 类型变量，always，case 语句等，只用 wire 类型变量和 assign 语句来写，这个过程略艰辛，相当于重写一遍。
2. 由于 alu 部分的 testbench 没有写对，在进行仿真时跑到”get waves”这步就卡住不动，但是不影响查看波形。我误以为是我的 testbench 里没有写 finish，故停不下来。导致改写了代码结果却没有变化，后来改正了 testbench，仿真可以跑完了。
3. 在实现 alu 中的 slt 功能时，开始用了比大小的方式实现。这个地方不知道如何修改，在查了一些资料，再加上反复修改、运行了几次代码之后才写好，这一块略微耗时了一些。
4. 这是一个没有解决的问题，alu 中 Zero 输出时有一个 ROM 元件（如下图），我不太明白为什么会出现 ROM。



三、 对讲义中思考题（如有）的理解和回答

思考题：当 ALUop = 3’ b011, 3’ b100, 3’ b101 时，ALU 可以实现什么功能？代码上如何实现？

答 :在本次实验中，当 ALUop = 3’ b011, 3’ b100, 3’ b101 时，我没有对 Result 进行赋值操作。从仿真波形可以看出，当 ALUop 等于这三个值时，Result 的值不发生

改变。

如果问 ALU 还能实现什么其他功能（不确定老师的问题是不是这个），选择则比较多,下面列出几种：

```
//异或操作
wire [`DATA_WIDTH-1:0] result_xor;
assign result_xor = A ^ B;

//或非运算
wire [`DATA_WIDTH-1:0] result_nor;
assign result_nor = ~(A | B);


//无符号数B右移A位，B >>> A
wire [`DATA_WIDTH-1:0] result_brau;//unsigned b>>>a
assign result_brau = B >>> A;


//有符号数B右移A位，B >> A
wire [`DATA_WIDTH-1:0] result_bra;//b>>a
assign result_bra = B >> A;


//有符号数B左移A位，B << A
wire [`DATA_WIDTH-1:0] result_bla;
assign result_bla = B << A;
```

四、 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

我认为此次试验难度适中。整个写代码的过程十分冗长，因为修改代码导致的重新检查代码、跑仿真等非常耗时，所以写作效率稍微有点低。

在验收的课上出现的一个问题是：原本跑过的程序在上板时报错，并且 reg_file 和 alu 不能同时更新、跑过。助教常老师说这是代码没有更新的问题，但我并不是很清楚其中的原因。另外，在验收的课堂上，由于许多同学同时跑程序，虚拟机运行的速度十分缓慢，生成仿真波形花费逾半小时。

写 alu 的过程收获比较大 ,加深了我对于组合逻辑的理解 ,代码书写也规范了许多。特别感谢周盈坤学长的指正 , 以及感谢助教老师张旭很耐心地解决问题 , 受益颇多,十分感谢！