中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2017K8009929044　姓名：李昊宸　专业：计算机科学与技术

实验序号：1　实验名称：基本功能部件设计 ——Register File & ALU

注 1：请在实验项目个人本地仓库中创建顶层目录 doc。撰写此 Word 格式实验报告后以 PDF 格式保存在 doc 目录下。文件命名规则：学号-prjN.pdf，其中学号中的字母“K”为大写，“-”为英文连字符，“prj”和后缀名“pdf”为小写，“N”为 1 至 5 的阿拉伯数字。例如：2015K8009929000-prj1.pdf。PDF 文件大小应控制在 5MB 以内。 注 2：使用 git add 命令将 doc 目录下的实验报告 PDF 文件添加到本地仓库，然后 git push 推送提交。 注 3：实验报告模板下列条目仅供参考，可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

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．reg_file 部分

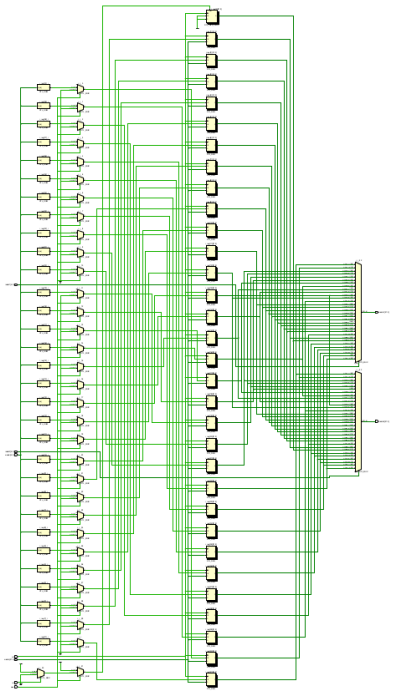
1）关键代码段：

```
always@(posedge clk)begin
 if(rst==0)
r[0]<=0;
if(wen == 1)begin
r[waddr]<= wdata;
r[0]<=32'b0;
end
end
assign rdata1 = r[raddr1];
assign rdata2 = r[raddr2];
```

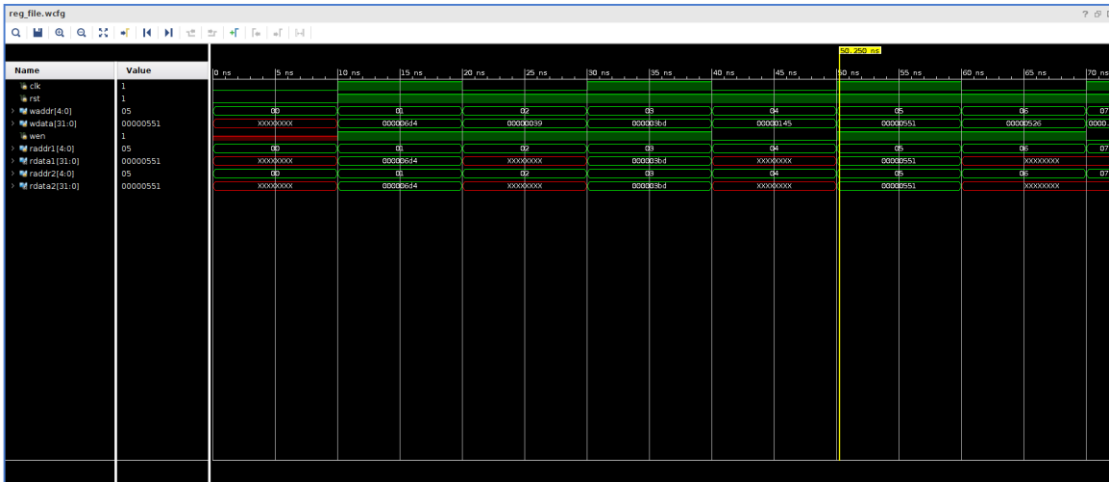
实验的第一个关键在于使内置寄存器中标号为 0 的寄存器读出数值始终为 0，于是我们采用持续赋值方式。

第二个关键在于同步写，异步读。同步写的部分使用 always 过程块与 clk 时钟相连接，异步读的部分使用组合逻辑电路 assign 赋值。

2）　逻辑电路



3）仿真波形



该仿真的 test-bench 采用时钟同步，每次时钟反转两个读头和一个写头的地址同步加一，wen 写使能信号随机取 0 或 1，data 为生成的随机数，通过检验读头读出数据是否与写入信号相等检验电路的正确性。

2. alu 部分

1）关键代码段：

```
assign registerAND = A&B;

assign registerOR = A|B;

assign registerADD ={A[31],A}+{B[31],B} ;
assign Overflow1 = registerADD[32]^registerADD[31];
assign registerADD1={32'd0,A}+{32'd0,B};
assign CarryOut1 = registerADD1[32];

assign registerSUB = {A[31],A}+{~{B[31],B}+32'd1};
assign Overflow2 = registerSUB[32]^registerSUB[31];
assign registerSUB1 = {32'd0,A}+~{32'd0,B}+32'd1;
assign CarryOut2 = registerSUB1[32];


assign result = (ALUop==3'b000)? registerAND:(ALUop==3'b001)? registerOR:(ALUop==3'b010)?registerADD[31:0]:(ALUop==3'b110)?registerSUB[31:0]:(ALUop==3'b111)?registerSUB1[31]^Overflow2:0;
assign carryOut = (ALUop==3'b010)?CarryOut1:(ALUop==3'b110)?CarryOut2:0;
assign overflow = (ALUop==3'b010)?Overflow1:(ALUop==3'b110||ALUop==3'b111)?Overflow2:0;
assign zero = !result;

assign Result = result;
assign CarryOut = carryOut;
assign Overflow = overflow;
assign Zero = zero;
```

AND 与 OR 指令使用位运算符处理，之后存于对应寄存器；ADD 与 OR 以及 SLT 使用共同的 33 位加法器进行计算：对于无符号数的计算，我使用了并位运算：在 32 位的数据顶端并入一位 0 作为符号位，从而构建出 33 位的有符号数，运算结果如果 33 位等于 1，说明运算中有进位或者退位，从而对应 CarryOut 为 1；反之对应 0。对应有符号数的计算，同样并入一位符号位构建两位符号位的 33 位运算，若符号位为 11 或 00 则 Overflow 等于 0，反之，即最高两位的异或值等于 1，那么 Overflow 等于 1。最终 SLT 的确定由有符号数的运算结果得到：如果符号位为 00 且未溢出，或符号位为 11 且溢出，那么 slt 等于 1，即符号位与 Overflow 的异或。Zero 则对 Result 使用逻辑非运算可直接得到。

由于不能够使用 always 块，我对 ALUop 的选择性采用条件表达式，并给其他未定义的操作值统一赋值为 0。

改进版：

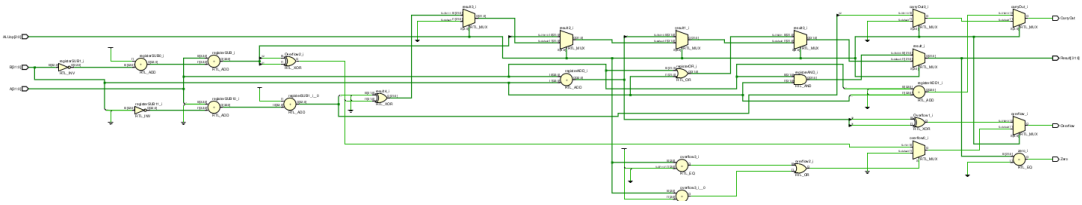
```
assign registerAND = A&B;

assign registerOR = A|B;

assign BvertSIGNED = ~{B[31],B}+33'd1;
assign BvertUNSIGNED = ~{1'b0,B}+33'd1;
assign BnumberSIGNED = (ALUop==3'b010)?{B[31],B}:BvertSIGNED;
assign BnumberUNSIGNED = (ALUop==3'b010)?{1'b0,B}:BvertUNSIGNED;


assign calculate = {A[31],A}+BnumberSIGNED;
assign calculate1 = {1'b0,A}+BnumberUNSIGNED;
assign Overflow = calculate[32]^calculate[31];
assign CarryOut = calculate1[32];



assign Result = (ALUop==3'b000)? registerAND:(ALUop==3'b001)? registerOR:(ALUop==3'b010)?calculate[31:0]:(ALUop==3'b110)?calculate[31:0]:(ALUop==3'b111)?calculate[31]^Overflow:0;

assign Zero = !Result;

endmodule
```

原理与之前相同，但提前用条件表达式处理好对应的 B 的取值，分别对应加法时的 B 和减法时的 B 的按位取反加 1，这样提前逻辑判断可以减少一半的

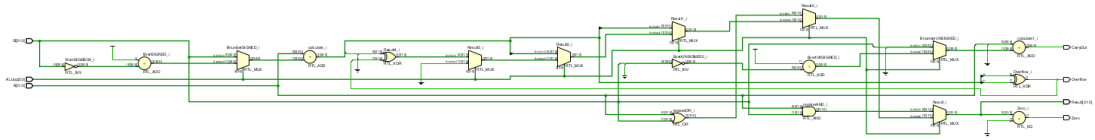
运算次数。

同理，对 carryout 和 overflow 的运算速度也得到了提高。

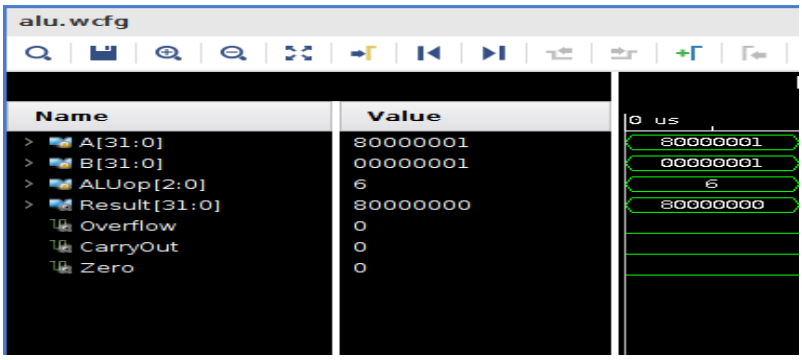
2）逻辑电路



改进后：



3）仿真波形



Test-bench 采用对 A 和 B 赋随机值，对 ALUop 赋随机值从而验证。

该样图中表现的是 A=0x80000001，B=0x00000001，ALUop=110 时的测试结果。

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）

Reg_file 部分：

对硬件作用未区分明确。第一次测试时发现全部的读写内容均为 wrong answer，后来发现是 rst 信号只负责控制第 0 个寄存器的刷新。

Alu 部分：

首先的难点是如何确定 Overflow 的值。一开始尝试使用两个加法器，即 32 位加法器和 33 位加法器确定进入最高位的进位和从最高位进出的进位，使用二者的异或进行计算，后来发现不能使用两个加法器，于是对其原理进行了化简，决定使用单一 33 位加法器和 2 位符号位的运算原理。其次是不能够使用 always 块，从而对逻辑结构有了较大的冲击。不能够使用 case，丢失了简洁性。还好最后想到了条件表达式，问题得以解决。

三、 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）

本次实验与预科实验难度跳跃较大，对于对 vivado 以及 linux 操作系统不太熟悉的同学着实不太友好。应该专门开一节预备课程，对 vivado 生成 bit-stream 的过程进行一个初步讲解，为后续程序的自我调试能够起到较大的帮助。

其次，关于不能使用 always 块的要求，还是有些疑问。助教说这样做是为了防止我们写错电路，防止将组合逻辑写成时序逻辑。但是随后查阅的大量资料显示，仍有不少逻辑器件的完成是使用了 always 块的协助实现，并且对应的块往往可以在实体电路中找到对应的现成组合器件，如本次实验中 ALUop 的控制信号，如果使用 always 块的写法，该块在逻辑图中完全可以替换成非时序的 6 选一数据选择器，所以感觉这样要求意义性不明确。

三、 思考题

需要实现的 ALU 部件应该支持五种基本操 作，但是 ALUop 信号位宽为 3 位，因此最多可以支持 2 的 3 次方种操作，即八种操作。那么，请问同学们，多出的三 种情况我们该怎么处理呢？在写 Verilog 代码时又应该注意 些什么问题呢？

不难发现，对于 ALU 运算结果的处理由运算规则决定，即在符合约定范围内的计算结果为正确即可，当 ALUop 为多出的三种信号时，外部原件将会把这几个未约定的值当作垃圾值，或者未定义值处理。所以原则上对这三种情况不需要特殊处理。在实现过程中，为了使输出结果整齐，在使用条件表达式赋值之后，统一将这三种情况的 Result 值归零。