中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2020K8009907032　姓名：唐嘉良　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——寄存器堆和算数逻辑单元

注 1：撰写此 Word 格式实验报告后以 PDF 格式保存在~/COD-Lab/reports 目录下。文件命名规则：prjN.pdf，其中“prj”和后缀名“pdf”为小写，“N”为 1 至 4 的阿拉伯数字。例如：prj1.pdf。PDF 文件大小应控制在 5MB 以内。此外，实验项目 5 包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：prj5-projectname.pdf，其中“-”为英文标点符号的短横线。文件命名举例：prj5-dma.pdf。具体要求详见实验项目 5 讲义。 注 2：使用 git add 及 git commit 命令将实验报告 PDF 文件添加到本地仓库 master 分支，并通过 git push 推送到 GitLab 远程仓库 master 分支（具体命令详见实验报告）。 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图、相应信号的仿真波形和信号变化的说明等）

```
reg [`DATA_WIDTH-1:0] REG_FILE [`REG_NUM-1:0];

always @(posedge clk) begin
    if(wen && (waddr!=5'b0)) begin
        REG_FILE[waddr] <= wdata;
        end
end

assign rdata1 = (raddr1==5'b0)?32'b0:REG_FILE[raddr1];
assign rdata2 = (raddr2==5'b0)?32'b0:REG_FILE[raddr2];
```

图 1 reg_file 关键代码段

如图 1 代码段，我们首先定义一个 32*32bit 的寄存器 REG_FILE 作为目标实现寄存器。

为实现同步写，在 always 模块中，每个时钟上升沿都会判断写使能 wen 是否 valid，如是则在寄存器的 waddr 地址处写入 wdata。注意到 0 号寄存器只能存储 0，我们加上 waddr!=5’b0 的判断条件。

为实现异步读，我们直接采用 assign 语句完成，此时仍然要注意 0 号寄存器只能读出 0 来，所以采用三目运算符实现读数据功能。

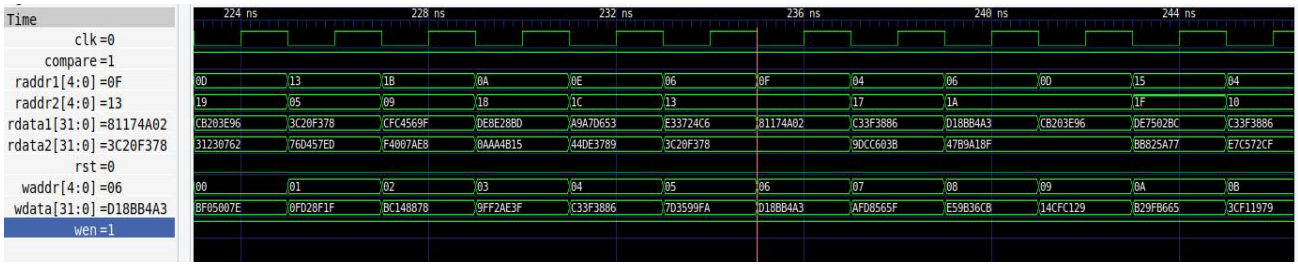

图 2　reg_file 波形图 1

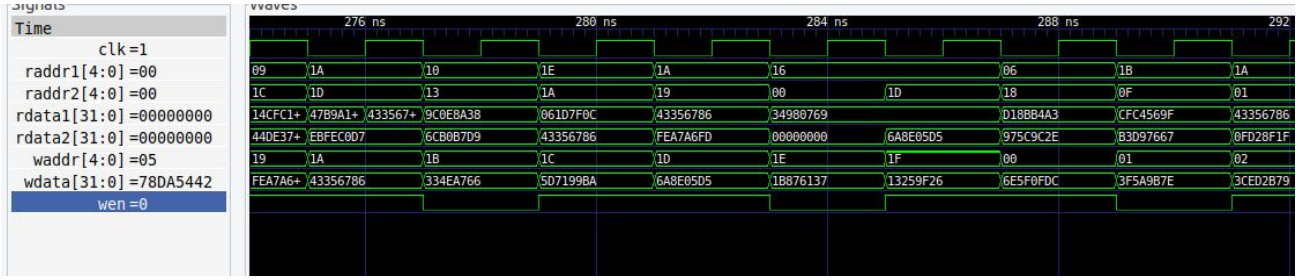

图 3　reg_file 波形图 2

可以看到，在图 2 光标所指处，因为 wen=1，于是在下一个时钟上升沿同步在 06 号寄存器写入数据 D18BB4A3，之后在 239ns 处可以清楚地看见异步读到 06 号寄存器的数据 D18BB4A3。这便是从写入数据到读出数据的过程。不难观察到，波形图图 1 其它地方也满足该过程。

特殊地，wen=0 时不写入数据。如果我们考察图 3 中 278ns 处的写入信号，理应在 1B 号寄存器写入 334EA766，但是 289ns 处上升沿读出 1B 号寄存器

的内容是 CFC4569F。这便是由于写使能信号 wen=0，不进行写入操作。

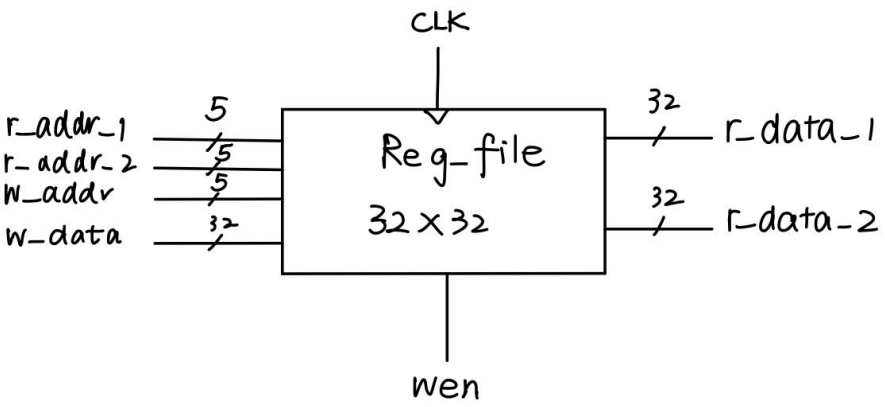

图 4　reg_file 寄存器堆逻辑电路图

Reg_file 中的核心电路是时钟控制的时序逻辑元件：32*32bit 的寄存器堆，其逻辑电路结构图如图 4。这里采用了总线的画法，下面的 ALU 核心元件逻辑电路图也将采取一定的总线画法。

```
//translate the ALUop
wire add;
wire sub;
wire orr;
wire andd;
wire slt;
//represent the result of the adder
wire [`DATA_WIDTH-1:0] add_sub_res;
//the mark of add flow or sub flow
wire add_flow;
wire sub_flow;
//mark of carryout
wire ca_out;
//the details of combination circuit
assign andd = (!ALUop[2])&(!ALUop[1])&(!ALUop[0]);//valid when ALUop=000
assign orr = (!ALUop[2])&(!ALUop[1])&(ALUop[0]);//valid when ALUop=001
assign add = (!ALUop[2])&(ALUop[1])&(!ALUop[0]);//valid when ALUop=010
assign sub = (ALUop[2])&(ALUop[1])&(!ALUop[0]);//valid when ALUop=110
assign slt = (ALUop[2])&(ALUop[1])&(ALUop[0]);//valid when ALUop=111
assign {ca_out,add_sub_res} = {1'b0,A} + {1'b0,(ALUop[2]?~B:B)} + ALUop[2];//*****The onlyyyy adder
//valid when A and B are same sign but the sign of result are diferent from those of A and B, such as 1+1=0
assign add_flow = (A[`DATA_WIDTH-1] ^ (!B[`DATA_WIDTH-1])) & (A[`DATA_WIDTH-1] ^ add_sub_res[`DATA_WIDTH-1]);
//valid when res and B are same sign but the sign of A are diferent from those of res and B, such as 0-1=1
assign sub_flow = (A[`DATA_WIDTH-1] ^ B[`DATA_WIDTH-1]) & (A[`DATA_WIDTH-1] ^ add_sub_res[`DATA_WIDTH-1]);
//handle the CarryOut and Overflow
assign CarryOut = (add)?ca_out:(!ca_out);
assign Overflow = (add)?add_flow:sub_flow;
//give the output result
/*to judge the slt, we assume that slt = 1 iff the sub mode of adder gives the result of sign 1 and a sub overflow didn't happen,
or the sub mode of adder gives the result of sign 0 and a sub overflow happened*/
assign Result = ({32{andd}} & {A & B}) | ({32{orr}} & {A | B}) | ({32{add | sub}} & add_sub_res) | ({32{slt}} & {add_sub_res[`DATA_WIDTH-1] ^ sub_flow});
//easy to judge the value of output Zero
assign Zero = (Result==0)?1:0;
```

图 5　alu 核心代码（含注释）

从图 5 可以看到，我们首先对 ALUop 进行了译码操作，并拓展位宽实现了

唯一的加法器 adder（考虑到有符号数和无符号数采取统一的运算规则），这里利用三目运算符实现了 B 转补码的情况，顺便得到进位标志 ca_out。

后面我们单独判断加法溢出和减法溢出，并根据相应规则置出 CarryOut、Overflow、Result 和 Zero 的值，具体实现逻辑和思路已经在注释中详尽体现。

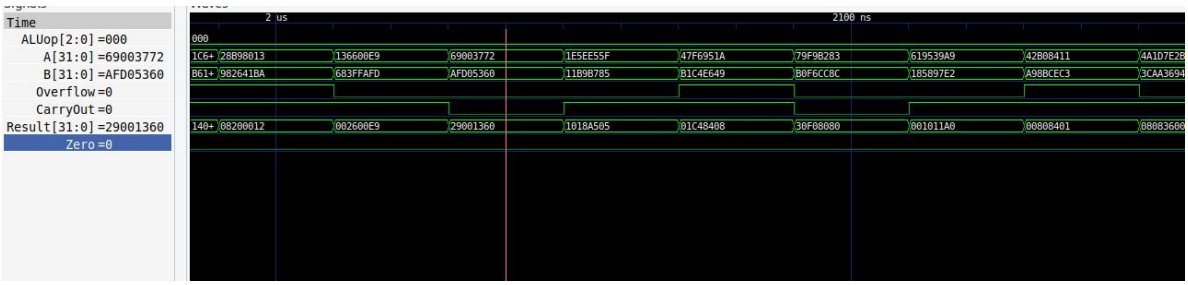

图 6　alu 波形图（ALUop=000）

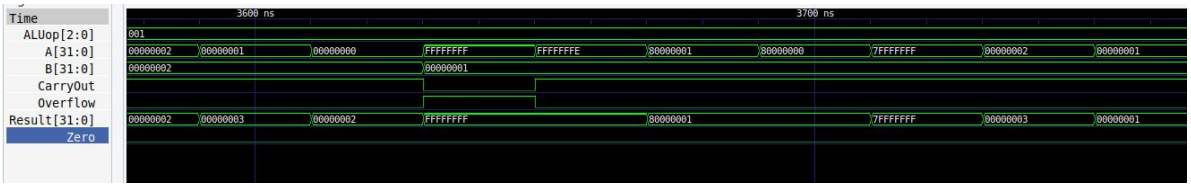

图 7　alu 波形图（ALUop=001）

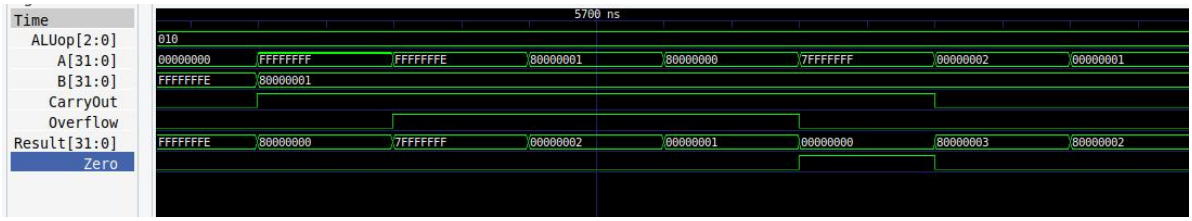

图 8　alu 波形图（ALUop=010）

图 9　alu 波形图（ALUop=110）

图 10　alu 波形图（ALUop=111）

从图 6、7、8、9、10 中容易观察到，ALUop 取各个值的时候运算均正确。

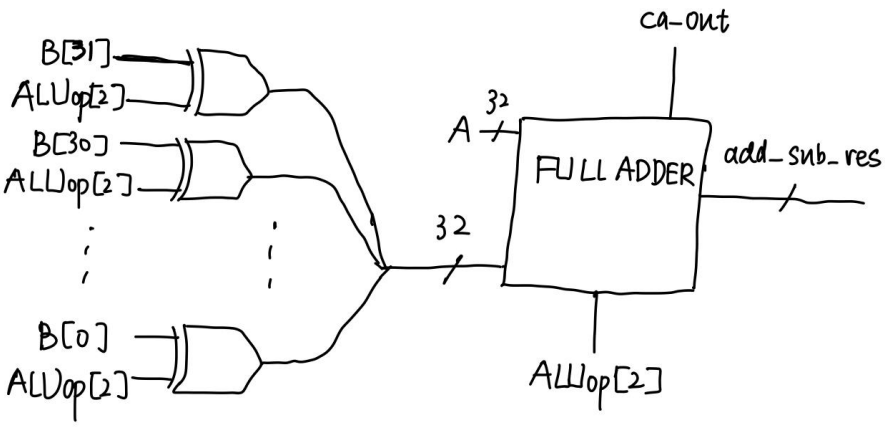

图 11　alu 中 adder 电路图

ALU 中的核心电路是三目运算符配合下的加法器（全加器）电路，逻辑电路结构图如图 11。在此我尝试将三目运算符以更细节的方式呈现，而不是直接画上一个抽象电路元件，这实际上不失为一种我们所需要的 32 位宽的三目运算符的实现方式。尽管这种三目运算符的实现方式不具备很强的普适性，因为我们的两个被选数据恰好是逐位取反的关系，但不可否认这也属于我的思考的一部分（电路的实现细节和优化）。

其余部分的电路图较为显然，在此不再赘述。

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，逻辑仿真和 FPGA 调试过程中的难点等）

在做本实验的过程中，遇到了一个比较隐秘的 bug：alu 在 slt 模式下没有考虑到减法溢出的情况而得到了错误的结果，后来经过我缜密的分析，发现 slt 置“1”实际上对应两种情况：减法负结果且不溢出，或者减法正结果且溢出。考虑到减法溢出之后，代码利用异或符号兼并两种情况，最终得到了正确的结果。

三、 对讲义中思考题（如有）的理解和回答

无思考题。

四、 在课后，你花费了大约_____4_____小时完成此次实验。

reg_file 寄存器堆花费 5min, alu 算术逻辑单元花费近 4hours。还记得一口气写出 ALU 的那个晚上，我在隔离酒店的书桌前一动不动地从 22 点写到了 2 点。（事实上，我下定的决心是天亮之前一定要写出来）

五、 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）

本次实验磨练了我的 Verilog HDL 代码技巧，尤其是我一开始写 ALU 时比较繁琐，思路不够清晰，后来慢慢梳理电路逻辑结构，利用译码、三目运算符、宏定义等一系列操作成功将代码降至轻量级，并且拓展了它的功能性。目前的代码非常简洁，可读性非常高，能够清晰地看出逻辑思路。最终，代码已经极其简练。最后验收的时候，助教学姐没有给出更进一步的修改意见。

此外，ALU 算是不简单的一个实验，有许多细节之处考验电路实现的完备性和正确性，而且用到了学到的逻辑运算知识，个人感觉非常好。毕竟学习就像习武，不仅要懂得理论知识，还必须有足够多的练习与实践。“实践出真知”，真正上手去做的过程能让我们收获很多，对知识的理解也会深入不少。不仅如此，

debug 的过程也十分考验耐心和细心。所以这是很成功的实验设计，能够有效地锻炼同学们各方面的能力。

最重要的一点在于，计算机组成原理实验并没有“竞速”制度，而且给我们留出了充足的时间思考、学习，这是非常棒的。上个学期的程序设计实验课采取了课堂竞速写代码的制度，许多同学苦不堪言，心理压力很大。对于处于计算机入门阶段的同学们来说，竞速容易打压学习积极性，且不利于深入的思考。相比之下感觉计算机组成原理无论是理论课还是实验课都做得非常好，各方面讲解都很细致，让小白也能听懂，循序渐进，可谓一大好课！

上了这门课，我能从学习中获得许多正向的反馈，更加渴望进一步探索计算机科学这个领域了！