中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2020K8009970001　姓名：金扬　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——寄存器堆和算术逻辑单元

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。

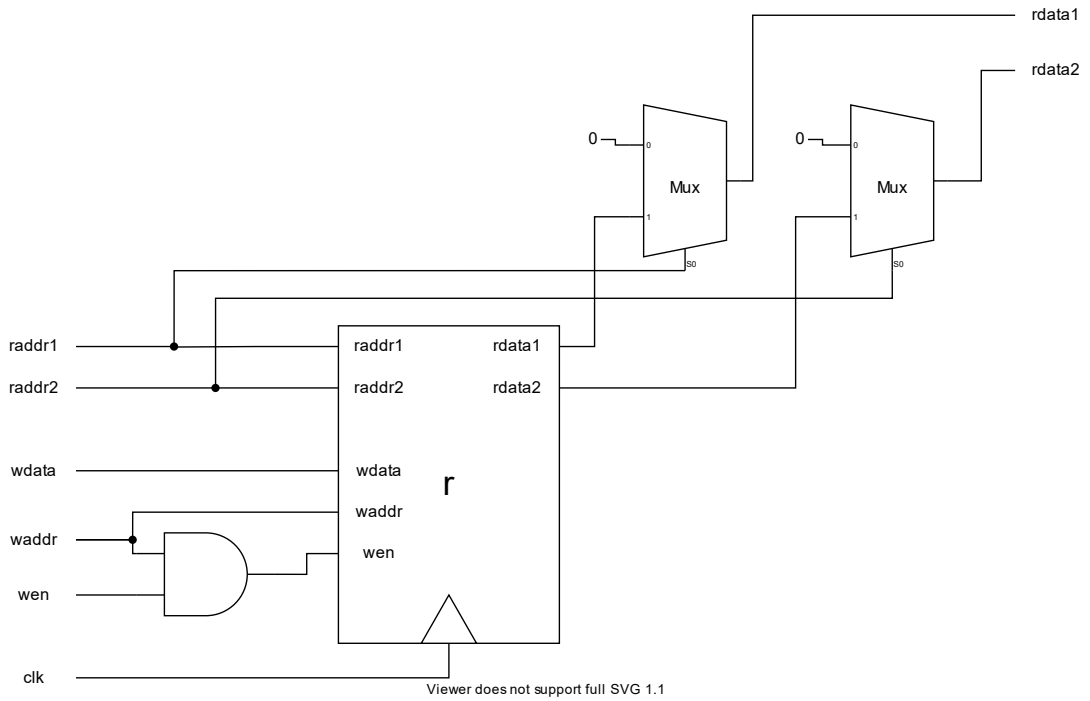
一、 逻辑电路结构与仿真波形的截图及说明（比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构图、相应信号的仿真波形和信号变化的说明等）

1. reg_file

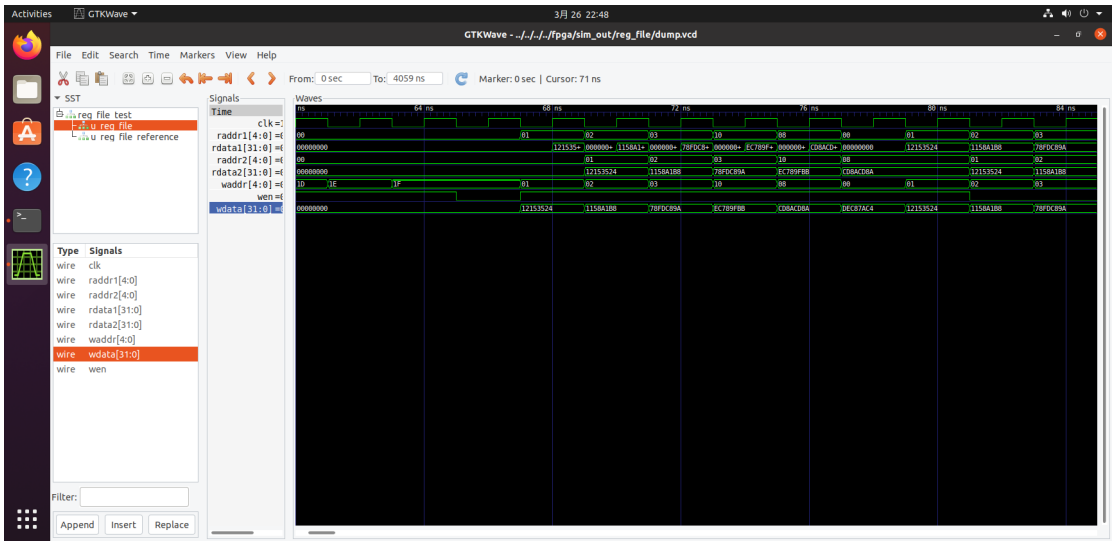
```
reg [`DATA_WIDTH - 1:0] r [(1<<`ADDR_WIDTH) - 1:0];
always @(posedge clk) begin
    if(wen==1'b1 && waddr) r[waddr] <= wdata;
end
assign rdata1 = raddr1 ? r[raddr1] : `DATA_WIDTH'b0;
assign rdata2 = raddr2 ? r[raddr2] : `DATA_WIDTH'b0;
```

这里通过 always 语句块，使用时序逻辑对寄存器堆进行写入；通过 assign 语句，使用组合逻辑对寄存器堆进行读取。

电路结构图如下：



信号仿真波形图如下：



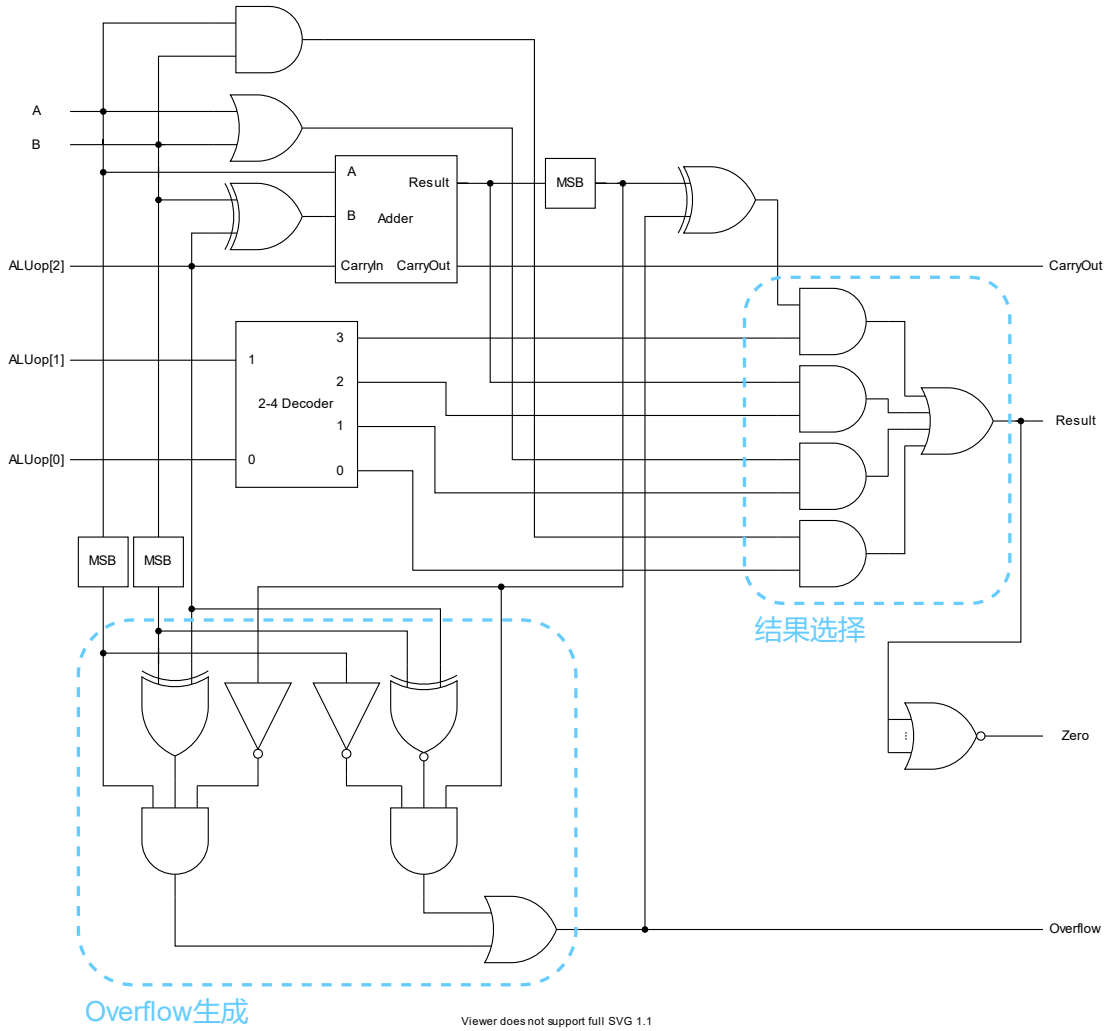
例如在 67ns 时刻，读头 1 读取了位于 01 地址的数据，此时为 00000000；与此同时，虽然 wen 为高电平，但此时 clk 并未处于上升沿，受时序逻辑控制的写头直到 clk 位于上升沿的 68ns 时刻才会在 01 地址写入 12153524 的数据；而在 68ns 时刻，可以观察到由组合逻辑控制的读头 1 的读数由 00000000 立刻转为刚写入的数据 12153524；在 69ns 时刻，同样受组合逻辑控制的读头 2 也读取了更新后的地址 2 的数据。说明寄存器堆工作正常。

2. alu

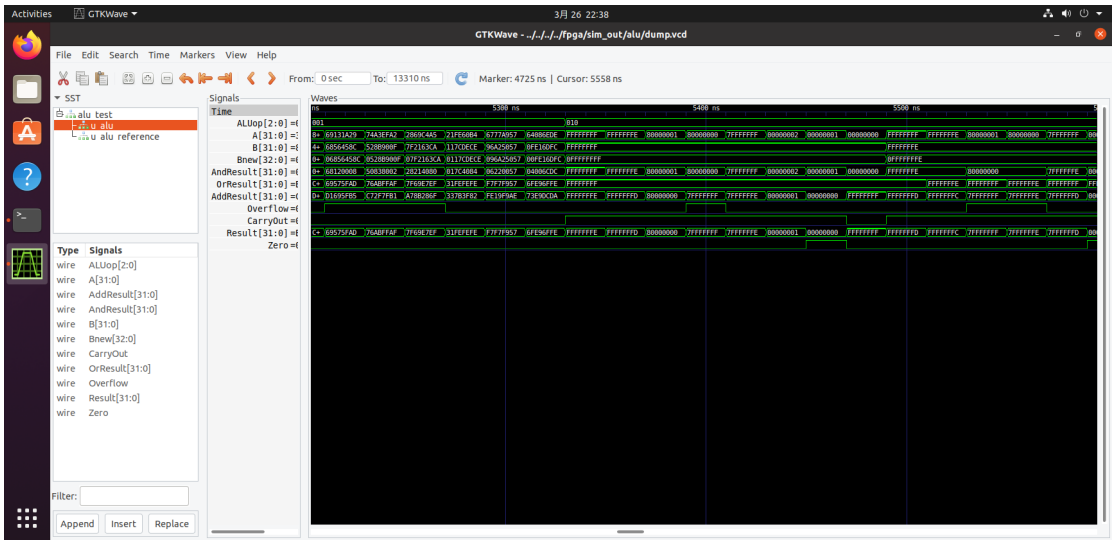
```
    wire [`DATA_WIDTH - 1:0] AndResult;
    wire [`DATA_WIDTH - 1:0] OrResult;
    wire [`DATA_WIDTH - 1:0] AddResult;
    wire [`DATA_WIDTH:0] Bnew;
    wire [`DATA_WIDTH - 1:0] sig [3:0];

    // Decode ALUop[1:0] to sig[3:0]
    assign sig[2'b00] = {(`DATA_WIDTH){!ALUop[1] & !ALUop[0]}};
    assign sig[2'b01] = {(`DATA_WIDTH){!ALUop[1] & ALUop[0]}};
    assign sig[2'b10] = {(`DATA_WIDTH){ALUop[1] & !ALUop[0]}};
    assign sig[2'b11] = {(`DATA_WIDTH){ALUop[1] & ALUop[0]}};

    // Generate results, carryout, overflow
    assign AndResult = A&B;
    assign OrResult = A|B;
    // If slt or sub, reverse B
    assign Bnew = {(`DATA_WIDTH + 1){ALUop[2]}} ^ {1'b0, B};
    // Use one adder to generate result and carryout for add and sub
    assign {CarryOut, AddResult} = {1'b0, A} + Bnew + ALUop[2];
    /* Set overflow to 1
     * if pos + pos = neg
     * or pos - neg = neg
     * or neg + neg = pos
     * or neg - pos = pos
     */
    assign Overflow = A[`DATA_WIDTH-1] & (ALUop[2]^B[`DATA_WIDTH-1]) &
~AddResult[`DATA_WIDTH-1]
        | ~A[`DATA_WIDTH-1] & (ALUop[2]^~B[`DATA_WIDTH-1]) &
AddResult[`DATA_WIDTH-1];

    // Select result according to sig[3:0]
    assign Result = sig[2'b11] & (Overflow^AddResult[`DATA_WIDTH-1])
        | sig[2'b10] & AddResult
        | sig[2'b01] & OrResult
        | sig[2'b00] & AndResult;

    // Generate zero
    assign Zero = ~|Result;
```

这里先对 ALUop 进行译码，然后同时生成各项结果，利用译码后生成的独热码对结果进行数据选择。

电路结构图如下：



信号仿真波形图部分如下：



例如当 ALUop=010，A=00000001，B=FFFFFFFF 时，对应运算为 add 加法运算。如果 A 和 B 为两个有符号数，则 A 表示 1，B 表示-1，此时结果应

为 0，无溢出，通过波形可知 Result 确实为 00000000，Overflow 为低电平；如果 A 和 B 为两个无符号数，则 A 表示 1，B 表示 32 位无符号数能表示的极大值，则结果有进位，通过波形可知 CarryOut 确实为低电平，且低 32 位如 Result 显示。

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）

1．N+N+1 型加法器的设计

对于减法运算，在最初的设计中，我先对减数取补，再与被减数相加。这样能得到正确的结果，但是使用了两次加法运算（其中一次在对减数的取补，即取反加一的运算中），延迟较高。

课前还曾设计了另一套 alu 模块，其中使用 1bit 全加器串联得到 32bit 加法器（见同一目录下 alu.v.bak 文件）。通过将“取反加一”中的“加一”作为最低位 1bit 全加器的进位输入（CarryIn），可以解决这个问题。但代码逻辑复杂。

最终的解决方案采用了 Verilog 的语法特性，使用多位+多位+1 位进位输入的方式设计加法器。当进行减法运算时，将取反后的结果（Bnew）作为加数输入，并令进位输入为高电平。这样只使用了一次加法运算，可以降低延迟。

2．CarryOut 与 Overflow 的生成

本实验遇到的另一个难点在于 CarryOut 的生成。CarryOut 为无符号数加减运算的进借位标志。解决方案是对两个运算数进行扩位（最高位补零），最终结果的最高位即为无符号数加减运算的进借位标志。

Overflow 的生成略微复杂。通过观察发现，溢出仅会发生于以下四种情况：

一个正数加一个正数，结果为负数；

一个正数减一个负数，结果为负数；

一个负数加一个负数，结果为正数；

一个负数减一个正数，结果为正数。

因此解决方案也通过判断运算数和结果的正负性生成。

3. 结果的选择

最初的设计中使用了大量三目运算符，因为不知道该如何使用单比特的数据来选择多比特的数据。在老师的启发下，发现可以通过利用 Verilog 语法特性扩展单比特数据到多比特，因此选择数据可以简化为多比特与或非运算。最终的 alu 解决方案中避免了所有三目运算和==运算。

原本多层嵌套的三目运算得出结果需要经过多级延迟，使用“与或非”代替“三目”可以将串行的链式结构转换为并行的树式结构，从而降低延迟。

三、 对讲义中思考题（如有）的理解和回答

本实验无思考题。

四、 在课后，你花费了大约 4 小时完成此次实验。

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）

这次实验的任务初看不是特别复杂，实现一个能通过测试的电路设计并不困难，但是要设计一个高效、严谨、规范的电路仍然需深入每一个细节，经过仔细的思考。非常感谢老师的帮助，让我发现了很多能进一步改进的细节。通过这次实验，我受益匪浅，对寄存器堆和算术逻辑单元有了更深的理解。此外，撰写实

验报告时我还熟悉了如何利用 VS Code 的 draw.io 绘制电路图。

另一方面，通过这次实验，我熟悉了很多 Verilog 语法上的特性，例如逐位逻辑运算（用于判断一个多位数据是否为零），generate – for 语法（用于串联 32 个 1bit 全加器），位数扩展语法（用于 1 位选择多位数据）。数字电路的理论课上，往往只会注意电路的结构，但不会特别关注对电路结构的描述，因此也忽略了这部分 Verilog 语法。然而在实际设计过程中，这些特性是不可或缺的。通过这次实验，我大大加深了硬件描述语言特性的理解。