中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2019K8009907015 姓名：高鸣驹　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

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 4 的阿拉伯数字。例如：2019K8009929000-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：学号-prj5-projectname.pdf，例如：2019K8009929000-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 doc 目录下的实验报告 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、相应信号的仿真波形和信号变化的说明等）

寄存器堆实验（见下）：

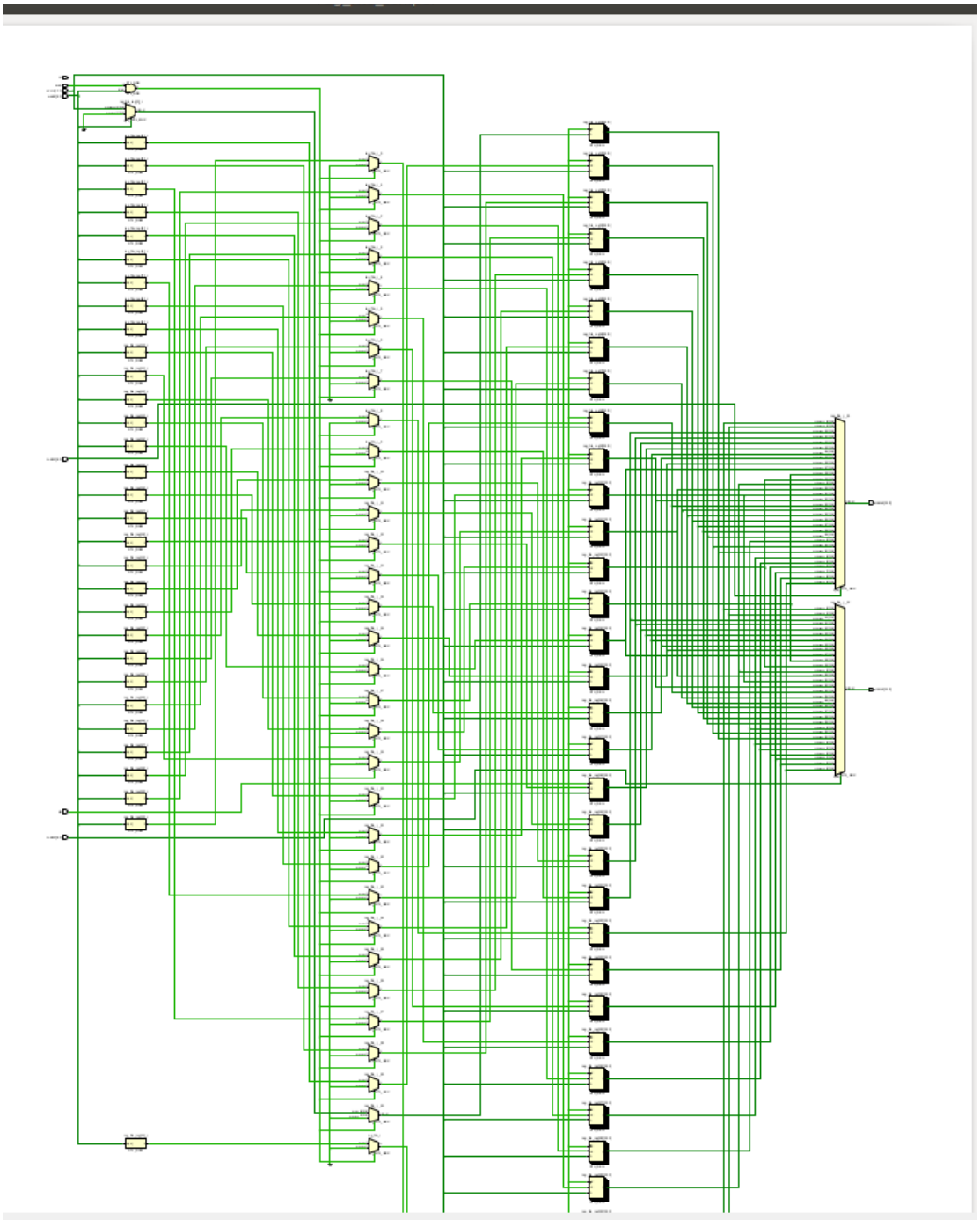


图 1.1 RF 逻辑电路结构

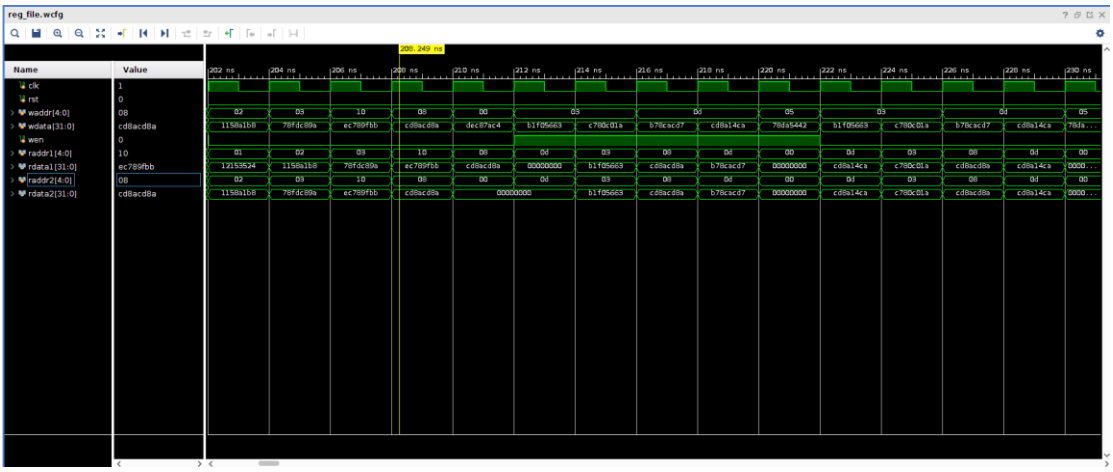
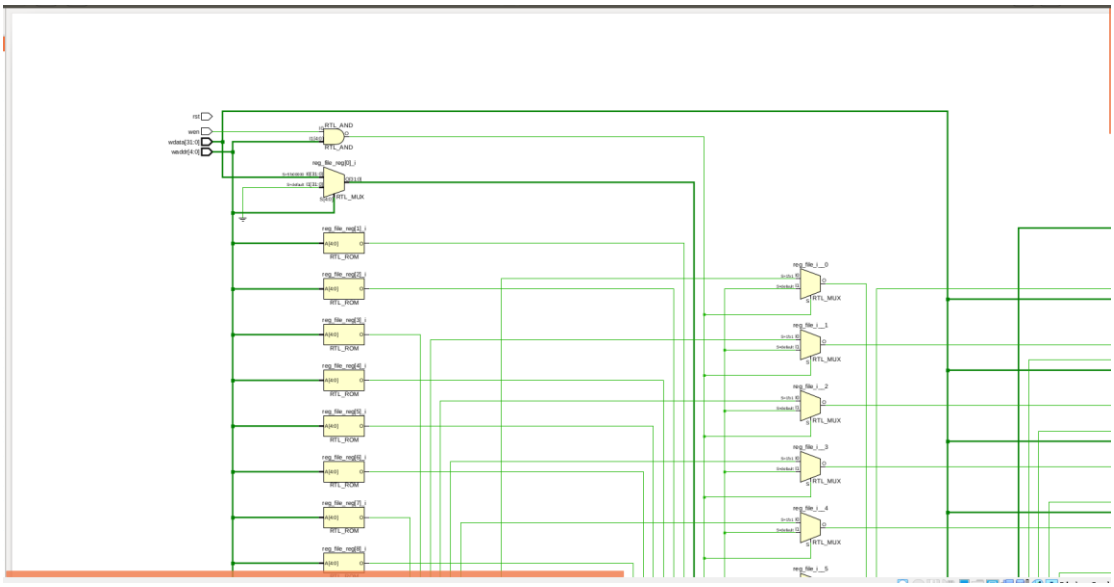


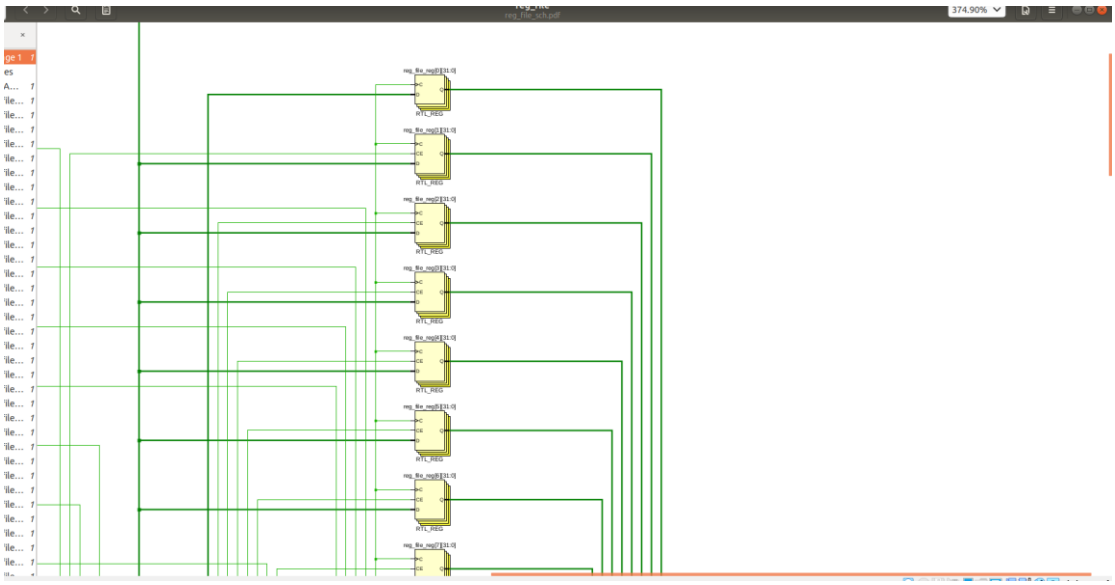
图 2.2 RF 仿真波形图

解释：

（1）原图过大，摘取两个重要的部分



仿真图中，当 wen 和 waddr 都不为 0 的时候，才能执行写的操作，一是由于 wen 是写的控制信号，二是由于当 waddr 等于 0 的时候无法修改寄存器的数据，即 0 号寄存器的数据不能修改。当两者都不为 0 的时候，根据 waddr 的地址选取要写入数据的寄存器。

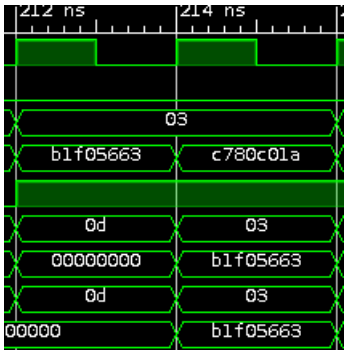


写入寄存器的堆的时候利用了触发器，代表着写的过程受时钟的控制，体现了“同步写”。

此部分对应的代码和波形：

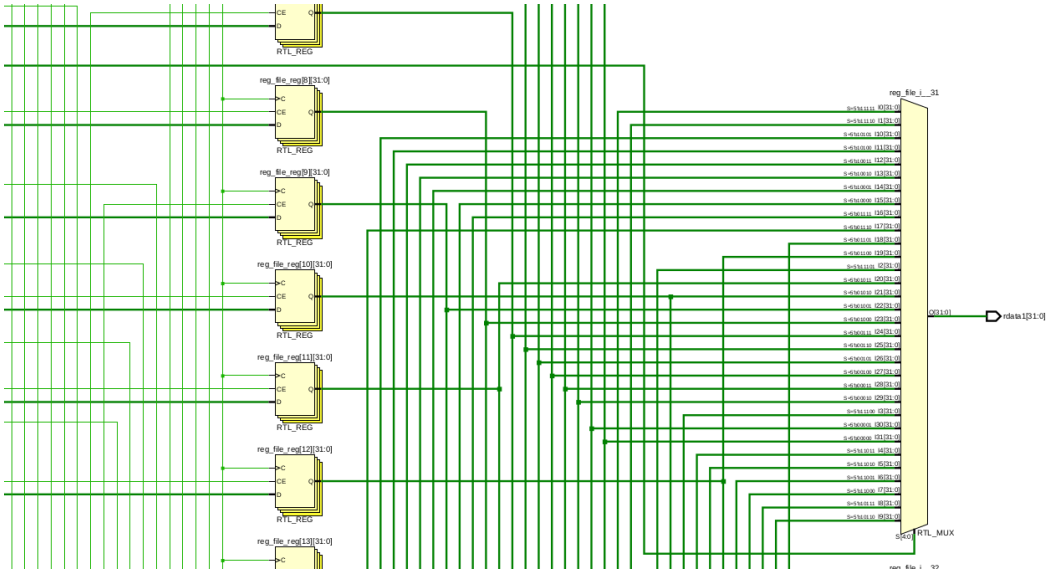
```
if(wen && waddr)
        begin
        reg_file[waddr] <= wdata;
        end
end
```

这一部分采用了时序逻辑



抽取波形的一小部分，我们可以看到，03 号寄存器在图中第二个时钟上升沿到来时候被赋值 b1f05663，在下面读取寄存器 03 号存储数据的过程中，读的数据也是 b1f05663。

(2)

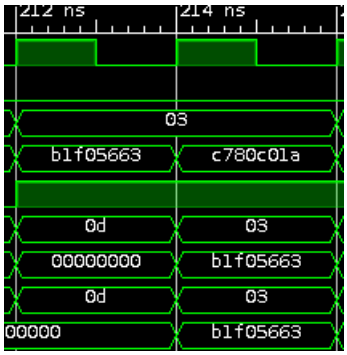


这一段主要部分是一个数据选择器，数据选择端由读地址信号 raddr 控制，从寄存器堆中读出数据，我们可以看出，读数据时是直接由组合逻辑控制，因此时“异步读”。

对应的代码和波形：

```
assign rdata1 = reg_file[raddr1];
assign rdata2 = reg_file[raddr2];
```

可以看出，读的过程用的是 assign 组合逻辑



仍然用上面的波形一小部分的图，我们看出，在 raddr 提供地址之后，rdata 立马读出了相应地址寄存器堆中存储的数据而没有必要等到下一个时钟信号到来。

ALU：

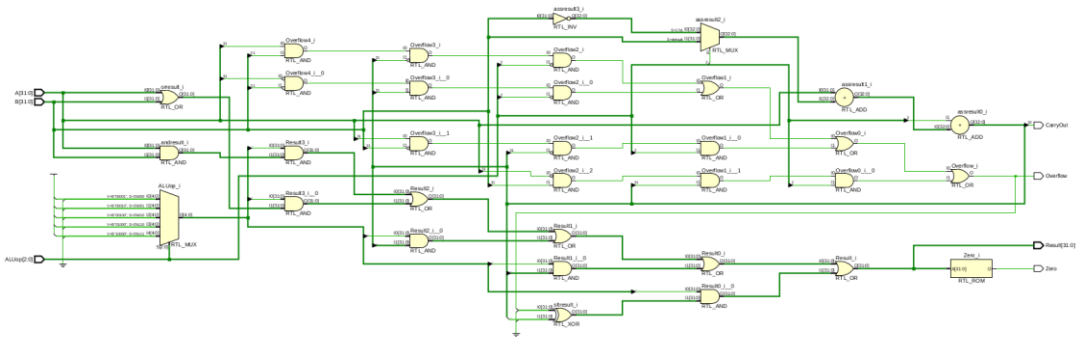


图 2.1 ALU 逻辑电路结构

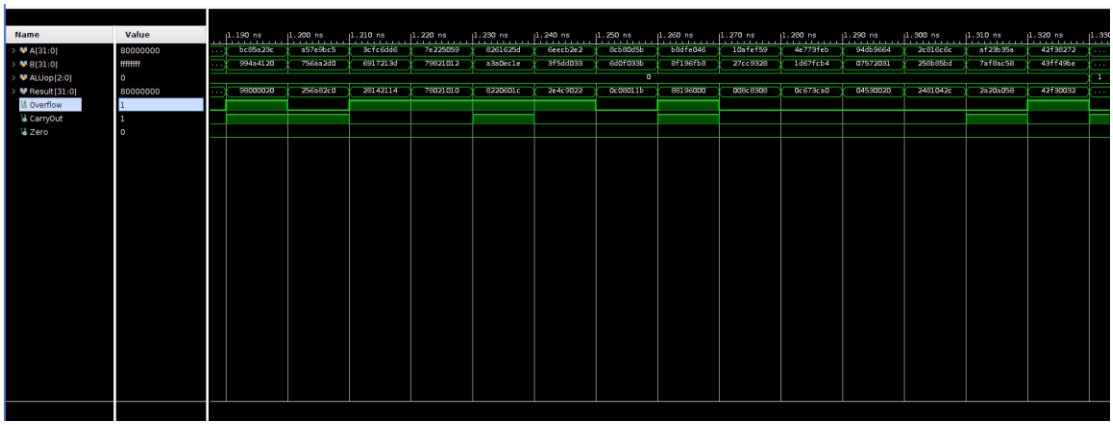
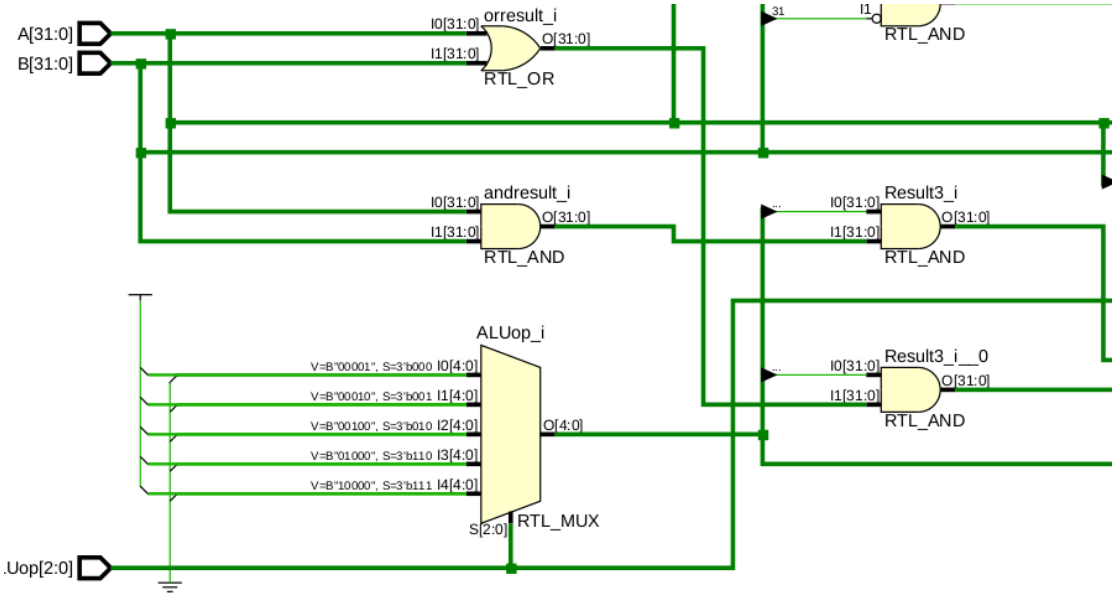


图 2.2 ALU 仿真波形图

解释：

着重对几个部分进行解释：

（1）



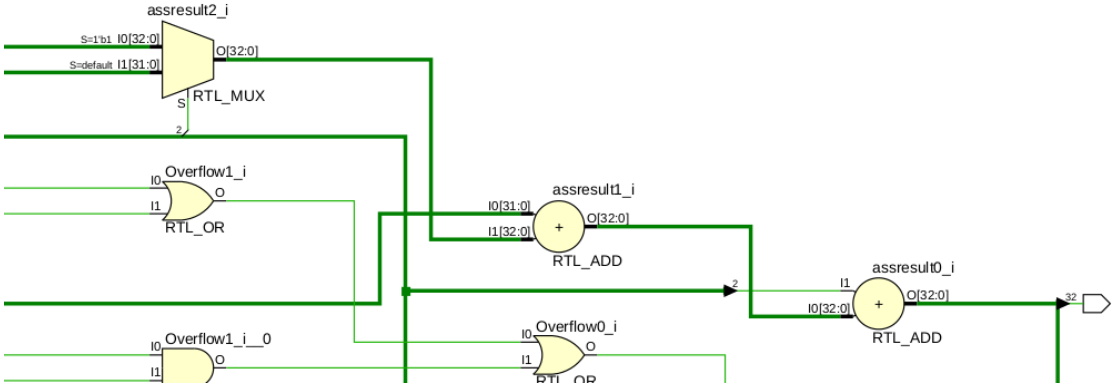
这里刚开始根据 ALUop 对操作数进行了筛选，利用数据选择器把结果输出，以“与”操作为例，对 A 与 B 进行按位与之后，将结果与对 ALUop 的筛选结果进行按位与（如果 ALUop 是“与”操作，则筛选结果为 1，不是则为 0），这样的话如果 ALUop 是与操作的话，输出结果可以输出该操作后的值，又由于这时其它操作结果经和筛选出的操作数的“与”操作后得到的结果均为 0，因此可以将相应操作数的结果准确地赋给 result。

对应的代码：

```
wire op_and = ALUop == `ALUOP_AND;
wire op_or = ALUop ==  `ALUOP_OR;
wire op_add = ALUop == `ALUOP_ADD;
wire op_sub = ALUop == `ALUOP_SUB;
wire op_stl = ALUop == `ALUOP_STL;


assign Result = {32{op_and}} & andresult |
                {32{op_or}} & orresult |
                {32{op_add}} & assresult |
                {32{op_sub}} & assresult |
                {32{op_stl}} & sltresult;
```

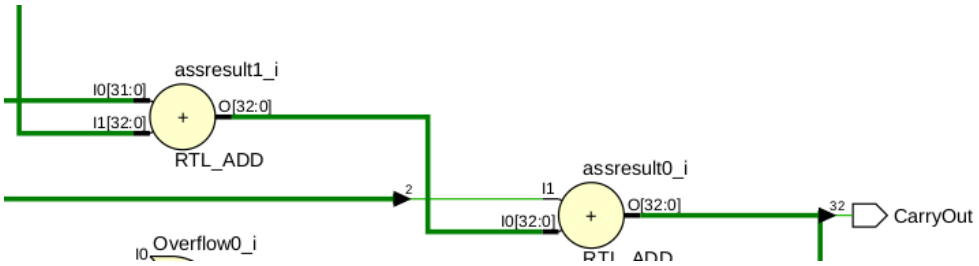
(2)



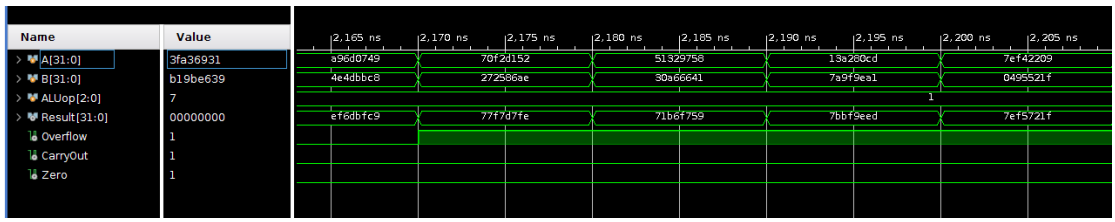
这是整个设计中比较重要的部分，是处理减法的，对 ALUop 的第二位操作数为高还是低电平进行是否对 B 取反的筛选，如果其第二位操作数为 1，即要进行减法和比较运算，则需要对 B 取反，然后加上其第二位操作数即 1，等于对 B 取了补码。如果第二位操作数为 0，则进行加法操作，B 取原码即可，不需要取反加 1，其对应的代码如下：

```
assign {CarryOut,assresult} = A + ((ALUop[2])? ~B:B) + ALUop[2];
```

（3）

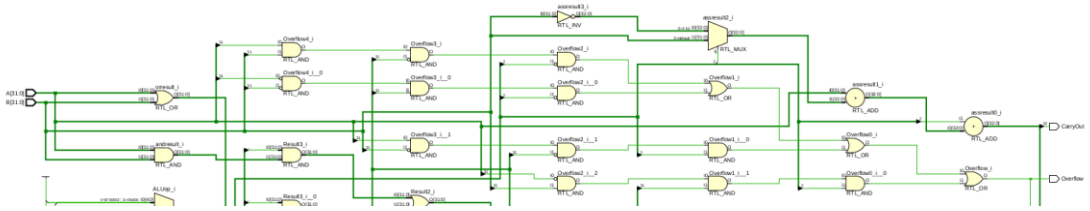


CarryOut 信号如图，其被定义为或者减法的进位



从下面这张波形图也能看出来

（4）



overflow 信号的处理如上图，逻辑链比较长，具体思路是分了四种情况，当为加的时候，两加数为正，得数为负；两加数为负，得数为正；或者当为减法的时候：被减数为负，减数为正，得数为负；或者是被减数为正，减数为负，得数为负，具体在代码中体现如下：

```
assign Overflow = (A[`DATA_WIDTH - 1] & B[`DATA_WIDTH - 1] & ~assresult[`DATA_WIDTH - 1] & ~ALUop[2]) | (~A[`DATA_WIDTH - 1] & ~B[`DATA_WIDTH - 1] & assresult[`DATA_WIDTH - 1] & ~ALUop[2])|
(A[`DATA_WIDTH - 1] & ~B[`DATA_WIDTH - 1] & ~assresult[`DATA_WIDTH - 1] & ALUop[2]) | (~A[`DATA_WIDTH - 1] & B[`DATA_WIDTH - 1] & assresult[`DATA_WIDTH - 1] & ALUop[2]);
```

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、云平台调试过程中的难点等）

问题 1：编译问题，看报错位置，根据语法修改即可；

问题 2：刚开始没搞清楚 zero 的定义，zero 赋值不是判断 result 是否为 0

问题 3：没搞清 overflow 的定义，在 ffffffff+00000001 时候，会产生进位，但是并不会产生 overflow，因为按照补码运算原则，这个结果并没有溢出，但是刚开始时候把它误判为溢出，后来仔细思索溢出的仔细定义后修好了 bug

三、 在课后，你花费了大约 2 小时完成此次实验。

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）

这次实验感觉相对较好，在实验开始前老师都提供了近似的案例给了实验写代码

的思路，知道了大体的实验框架怎么写之后后面的东西就相对简单多了。感觉在 debug 过程中由于云平台运行时间较长，有时候修改完一个很小的错误后，就要登上较长的一段时间，可能导致 debug 的效率相对较低，但可能这也是不可以避免的吧。