

# Lab 1 报告

## 1 实验任务

由给出的单周期 CPU 的 Verilog 代码, 根据仿真波形对照金标准, 找出其中的错误并进行修正, 最后完成上板验证。

## 2 实验过程

### 2.1 实验流水账

8.30(周五)下午, 经过约四个小时的 debug, 成功找出了所有错误, 之后花费约一小时进行综合、生成比特流文件, 最后在板子上验证通过。

### 2.2 错误记录

#### 1. 信号的命名、声明错误。

这一类的错误十分好找, 只需要观察 vivado 报错, 然后根据报错信息进行修改即可。在原代码中即有好几处这类错误, 如 `rj_eq_rd`、`final_result` 未声明, `debug_wb_rf_we` 误写作 `debug_wb_rf_wen`。

#### 2. 模块引脚接线错误。

在这里我并没有遇到什么困难。定睛一看, 我便发现 ALU 的实例化中出现了错误:

```
alu u_alu(  
    .alu_op  (alu_op  ),  
    .alu_src1 (alu_src2 ),  
    .alu_src2 (alu_src2 ),  
    .alu_result (alu_result)  
);
```

显然, `alu_src1` 应该接 `alu_src1`, 而不是 `alu_src2`。不过这一类错误也不会个个都这么明显, 有时候需要根据波形图进行分析。

#### 3. 寄存器写使能逻辑错误。

这一类的错误需要根据波形图进行分析, 找出出错时究竟在执行哪一条指令, 但是指令信号并不在波形图中显示, 于是我在文件中加入了一些输出信号, 以便观察。下面是我自行添加的输出信号:

```
//add some signals  
output wire [31:0] debug_inst,  
output wire [31:0] debug_alu_result,
```

之后添加赋值语句:

```
//add some signals
assign debug_inst = inst;
assign debug_alu_result = alu_result;
```

当然, testbench 中也需要添加这些信号, 过程与之类似, 这里不再赘述。(注: 查看指令更方便的办法是直接查看反汇编文件, 找到对应 pc 值的指令即可)

这个错误就出现在指令 BL 上。在 LoongArch 手册中, BL 指令为无条件转移指令, pc 直接加上偏移量, 并将该指令的 pc 值加 4 的结果写入到 1 号通用寄存器 r1 中。最开始的时候, 我以为是分支条件的问题, 查了又查, 没发现问题, 后来仔细阅读了手册才明白原因。原代码中对寄存器写使能的赋值如下:

```
assign gr_we = ~inst_st_w & ~inst_beq & ~inst_bne & ~inst_b & ~inst_bl;
```

可见 BL 指令时, gr\_we 被赋值为 0, 这显然是错误的。我将其改为:

```
assign gr_we = ~inst_st_w & ~inst_beq & ~inst_bne & ~inst_b;
```

保证 BL 指令时, gr\_we 被赋值为 1, 即可解决问题。

现在来展示一下波形图：

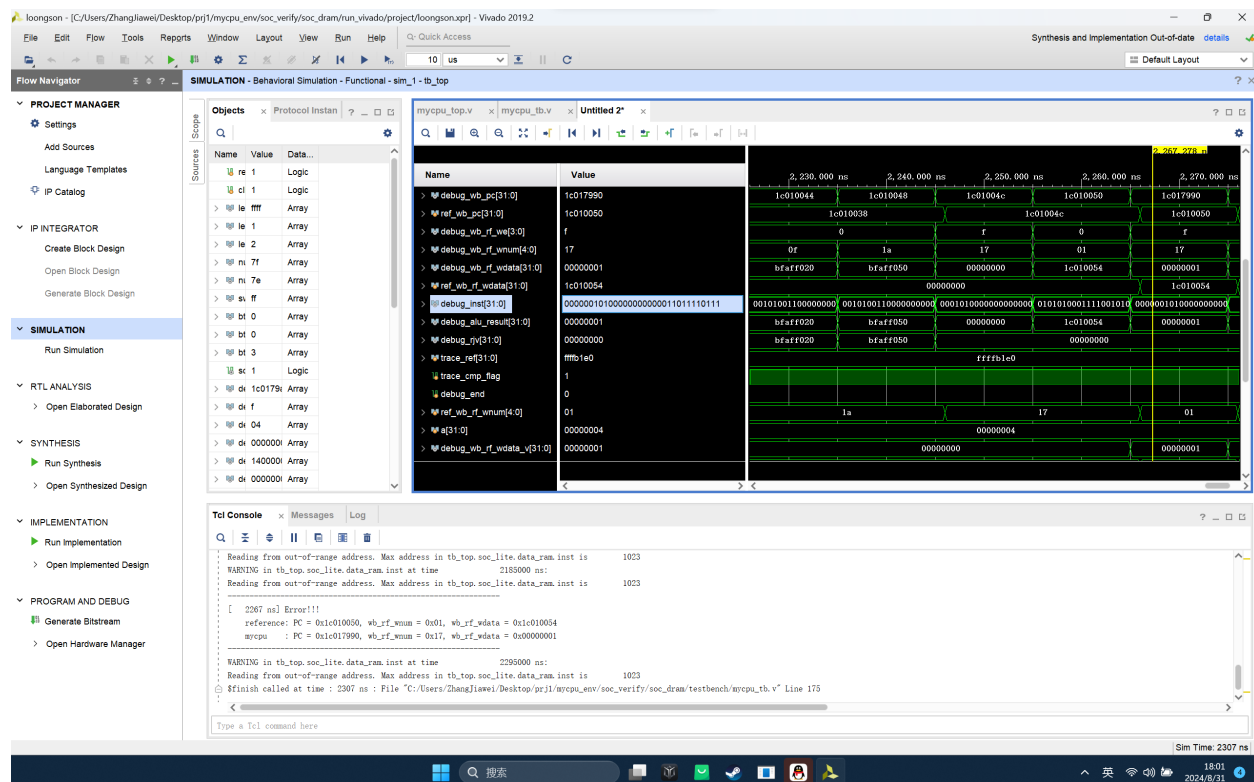


图 1. BL 指令处波形图

出错位置对应指令为 BL 指令。

#### 4. ALU 内部逻辑错误。

问题还没有得到解决,下一个出错的指令是 SLLI.W,移位得到的结果是错误的,既然 ALU 的接线没有问题,错误就只能出在 ALU 模块中了。在 ALU 内部,逻辑左移指令的实现如下:

```
assign sll_result = alu_src2 << alu_src1[4:0]; //rj << i5
```

但是仔细观察 cpu 文件,赋值下来:

```
assign alu_src1 = src1_is_pc ? pc[31:0] : rj_value;  
assign alu_src2 = src2_is_imm ? imm : rkd_value;
```

可见,alu\_src1 才是 rj,而 alu\_src2 才是立即数,ALU 里面写反了,应当改为:

```
assign sll_result = alu_src1 << alu_src2[4:0]; //rj << i5
```

同样,下面的右移逻辑也是反的,原代码作:

```
assign sr64_result = {{32{op_sra & alu_src2[31]}}, alu_src2[31:0]} >>  
alu_src1[4:0]; //rj >> i5
```

应当改为:

```
assign sr64_result = {{32{op_sra & alu_src1[31]}}, alu_src1[31:0]} >>  
alu_src2[4:0]; //rj >> i5
```

接下来是 OR 指令,错误依然在 ALU 模块中,原代码中的逻辑如下:

```
assign or_result = alu_src1 | alu_src2 | alu_result;
```

但是根据手册,OR 指令应当是将 rj 和 rk 的值进行或运算,因此应当改为:

```
assign or_result = alu_src1 | alu_src2;
```

至此,所有错误均已解决。

### 3 上板验证

经过漫长的等待,激动人心的上板验证终于开始了,按照教材指引,我将比特流文件烧录到板子上,板子上的数码管显示出如下图字样:



图 2. 板子上的数码管显示

## 4 实验总结

本次实验是对单周期 CPU 的 Verilog 代码进行 debug 的实验,其实 bug 并不多,但是都在细微之处,也是同学们经常会犯的错误。希望我以后能少犯一些吧。