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。

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。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：

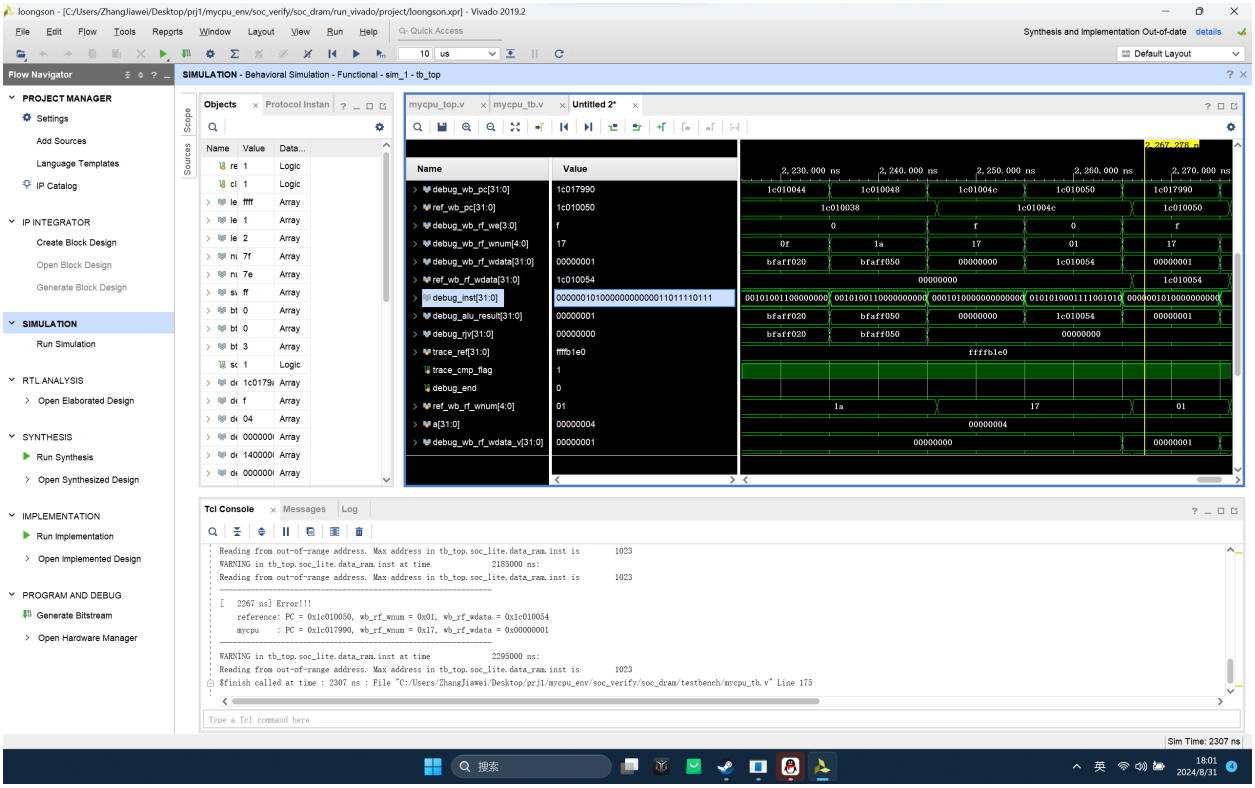

图 **1.** BL 指令处波形图

出错位置对应指令为 BL 指令。

4. 立即数错误。

   不出意外，仿真又碰到了错误，这次出错的指令是 SLLI 指令。在 LoongArch 手册中，SLLI.W 指令为逻辑左移指令，将通用寄存器 rj 中的数据逻辑左移 ui5 位，移位结果写入通用寄存器 rd 中。原代码中我们看到：

```
assign need_ui5 = inst_slli_w | inst_srli_w | inst_srai_w;
assign imm = src2_is_4 ? 32'h4            :
          need_si20 ? {i20[19:0], 12'b0}  :
/*need_ui5 || need_si12*/{{20{i12[11]}}, i12[11:0]} ;
```

   代码中贴心地给出了注释，need_ui5 为真时，应当采取最后一种情况，但是这赋值和 ui5 又有什么关系？这显然是错误的。我添加了 i5 的声明与赋值，也多加了一个判断条件，将其改为：

```
wire [ 4:0] i5;
assign i5 = inst[14:10];

assign imm = src2_is_4 ? 32'h4            :
          need_si20 ? {i20[19:0], 12'b0}  :
          need_si12 ? {{20{i12[11]}}, i12[11:0]} :
                    {{27'b0},i5[4:0]};
```

   问题得到解决。

5. ALU 内部逻辑错误。

   问题还没有得到解决，移位得到的结果仍然是错误的，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
```

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：

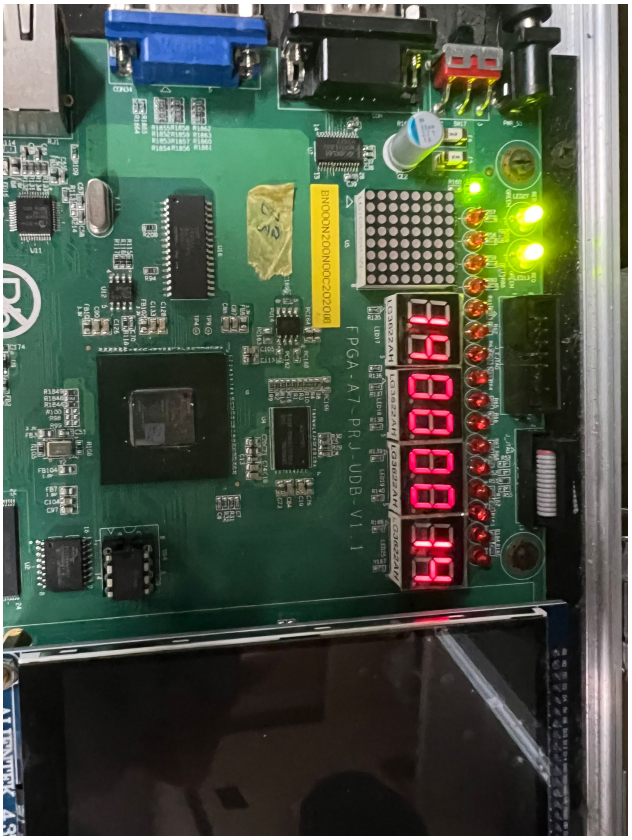

图 2. 板子上的数码管显示

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。