中国科学院大学计算机组成原理实验课

实 验 报 告

学号：_2018K8009909006___ 姓名：_唐宇菲_____ 专业： 数学与应用数学

实验序号：_04___ 实验名称：__定制 RISC-V 功能型处理器设计

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。

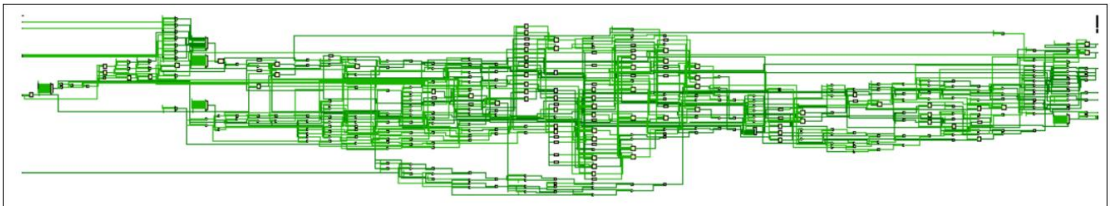
注 2：使用 git add 及 git commit 命令将实验报告 PDF 文件添加到本地仓库 master 分支，并通过 git push 推送到 GitLab 远程仓库 master 分支（具体命令详见实验报告）。

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。

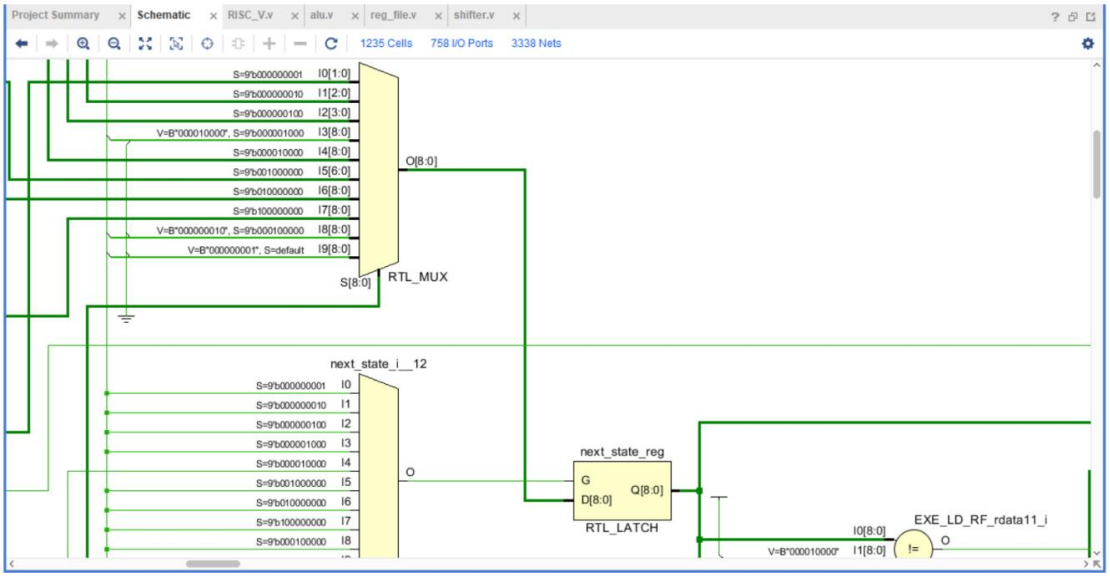
一、 逻辑电路结构与仿真波形的截图及说明（比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构图、相应信号的仿真波形和信号变化的说明等）

解:

采用 Vivado 的 RTL 详细描述和分析（Elaboration),选择“Open Elaborated Design”，自动打开 Schematic 界面。图像展开如下：



具体细节如下:



代码编写与实验三基本一致，区别在于译码的部分。

(1) 关于立即数的部分:

MIPS 指令集中立即数部分相对来说同一，但 RISCV 中，由于分出了

```
assign R_Type_judge = (Opcode == 7'b0110011);
assign I_Type_Compute_judge = (Opcode == 7'b0010011);
assign U_Type_judge = (Opcode == 7'b0110111 || Opcode == 7'b0010111);
assign J_Type_judge = (Opcode == 7'b1101111);
assign I_Type_jump_judge = (Opcode == 7'b1100111);
assign B_Type_judge = (Opcode == 7'b1100011);
assign I_Type_load_judge = (Opcode == 7'b0000011);
assign S_Type_judge = (Opcode == 7'b0100011);
```

R_Type、I_Type(分为计算和跳转，以及 load 类型)、U_Type、B_Type、S_Type，所以相应的立即数编码也有所不同。根据 RISCV 手册中的显示，我们取到下列立即数:

```
wire [11 : 0] I_Immediate;
wire [11 : 0] S_Immediate;
wire [12 : 0] B_Immediate;
wire [19 : 0] U_Immediate;
wire [20 : 0] J_Immediate;

assign I_Immediate = IW_ID_Instruction[31 : 20];
assign S_Immediate = {IW_ID_Instruction[31 : 25], IW_ID_Instruction[11 : 7]};
assign B_Immediate = {IW_ID_Instruction[31], IW_ID_Instruction[7], IW_ID_Instruction[30 : 25], IW_ID_Instruction[11 : 8],1'b0};
assign U_Immediate = IW_ID_Instruction[31 : 12];
assign J_Immediate = {IW_ID_Instruction[31], IW_ID_Instruction[19 : 12], IW_ID_Instruction[20], IW_ID_Instruction[30 : 21], 1'b0};
```

其中 B_immediate 与 J_immediate 的立即数需要增加末尾的 0,因为在指令手册中只标注到了 immediate[1]或者 offset[1]，需要自己添加零。这在 MIPS 中是不需要的，所以又设置了对应位宽所描述的有符号扩展后的立即数:

```
wire [31 : 0] Sign_extend_I_immediate;
wire [31 : 0] Sign_extend_U_immediate;
wire [31 : 0] Sign_extend_J_immediate;
wire [31 : 0] Sign_extend_B_immediate;
wire [31 : 0] Sign_extend_S_immediate;

assign Sign_extend_I_immediate = { {20{I_Immediate[11]}}, I_Immediate};
assign Sign_extend_U_immediate = { {12{U_Immediate[19]}}, U_Immediate};
assign Sign_extend_J_immediate = { {11{J_Immediate[20]}}, J_Immediate};
assign Sign_extend_B_immediate = { {19{B_Immediate[12]}}, B_Immediate};
assign Sign_extend_S_immediate = { {20{S_Immediate[11]}}, S_Immediate};
```

(2) RISCV 中取消了分支延迟槽的设计。并且在后期与实验平台提供的仿真波形对比，发现这里并不特别区分 nop 指令，也是当作 addi 指令来处理。
由于设计尽可能保留原来对于 MIPS 处理器的代码。而 MIPS 需要在 ID 译码阶段判断出是 nop 指令后，就立即跳转到 IF 阶段，不进入 EXE 阶段。所以在实验三的代码中，我们在 IF 阶段就执行对 PC + 4 这样的赋值过程。
这样的赋值对于有分支延迟槽的设计来说是不会出错的，因为分支延迟槽导致对于 jump 类型或者 branch 类型的指令，实际地址的 PC + 4 + offset 可以直接在 IF 阶段 PC + 4 的基础上，由现在的 PC + offset.
但 RISCV 中取消了分支延迟槽的设计，所以原先的时序逻辑赋值：

```
// IF
    reg [31 : 0] cpu_PC;
    wire [31 : 0] cpu_PC_next;
    wire cpu_PC_next_enable;

    always@(posedge clk)begin
        if(rst)
            cpu_PC <= 32'b0;
        else if(current_state == IF && next_state == IW)
            cpu_PC <= cpu_PC + 4;
        else if(current_state == EXE && cpu_PC_next_enable)
            cpu_PC <= cpu_PC_next;
        else
            cpu_PC <= cpu_PC;
    end

    assign PC = cpu_PC;
```

在后面对于 cpu_PC_next 的部分需要减去这个提前添加的 4,但同时需要注意，对于 jalr 这类原先在 MIPS 中属于 R-Type 类型的指令，它的地址值是直接从寄存器中取得，然后再加上 offset，这里就不需要再进行 -4 的操作。

```
assign cpu_PC_next = ( {32{current_state == EXE && cpu_PC_branch_enable}} & (ID_EXE_cpu_PC_branch - 4) ) |
                     ( {32{current_state == EXE && ID_EXE_cpu_PC_jump_enable}} & (ID_EXE_cpu_PC_jump - 4)) |
                     ( {32{current_state == EXE && ID_EXE_cpu_PC_R_enable}} & (ID_EXE_cpu_PC_R));
assign cpu_PC_next_enable = (cpu_PC_branch_enable || ID_EXE_cpu_PC_jump_enable || ID_EXE_cpu_PC_R_enable);
```

需要对此进行区分。

（3）RISCV 指令集中，load 和 store 类型的指令，其地址并不相同。实际上，由于指令格式的不同，两者内容显然也不一定相等，所以需要对 Address 在 LD 和 ST 阶段都分别赋值，以保证输入或输出的地址正确。

```
wire [31 : 0] Address_ST;
assign Address_ST = ({32{current_state == ST }} & (EXE_ST_RF_rdata1 + EXE_ST_Sign_extend_S_immediate));
assign Address = ( {32{current_state == ST}} & {Address_ST[31 : 2], 2'b0}) |
                 ( {32{current_state == LD}} & {Address_real[31 : 2], 2'b0});
```

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）

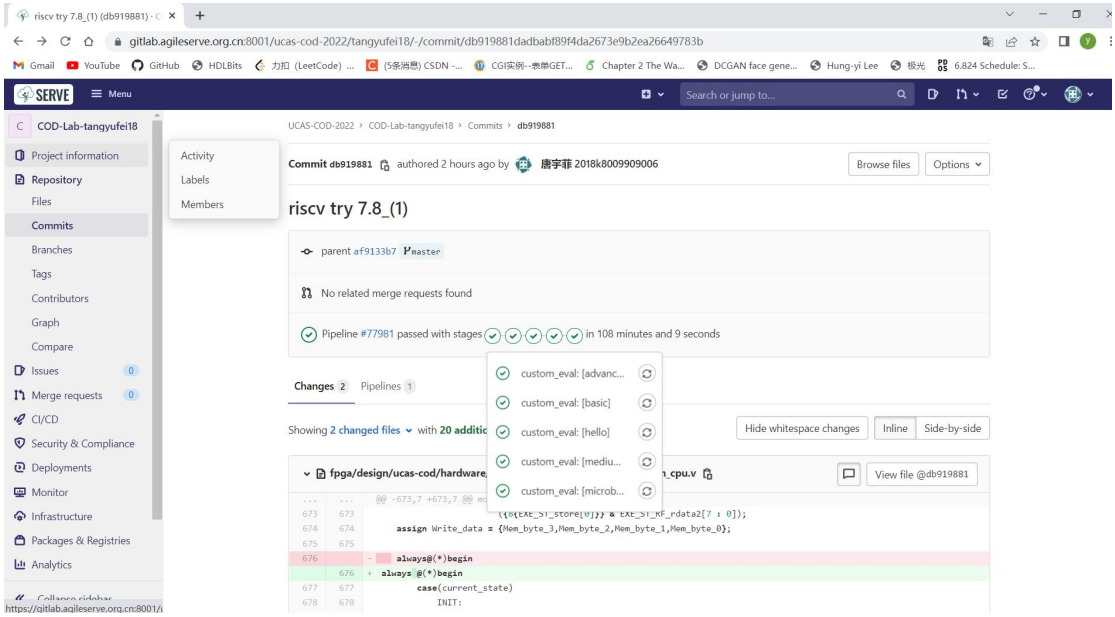
1. 首先对于 J_immediate 和 B_immediate,注意到其编码格式中，只标注到 immediate[1]或者 offset[1]，所以最后一位应该补充零，实际位宽为 21 位或者 13 位。

2. 对于 RISCV 指令集中，没有了分支延迟槽，所以原先在实验项目三中设计的，在 IF 阶段对于 PC + 4 的操作，在 branch 和 jump 指令中，现在的目的地址，由于减去了分支延迟槽的设计，由 PC + 4 + offset 改为 PC + offset，所以在后续的操作中，需要再减去 4。

3. NOP 指令没有特殊处理，不需要像实验项目三一样对 NOP 指令进行特殊处理。

4. 出现了 bhv_sim 阶段完全正确，但是 fpga_eval 阶段出现错误的情况。后来发现出错的原因是：在 next_state 的跳转阶段，写了 if 条件判断句和 4 个 else if 语句，并没有添加 else 语句，导致判断条件不完全，出现锁存器。锁存器的出现不会导致仿真出错，但是会导致上板时的概率出错。

最后一个的 else 没有，综合跑起来之后，根据不同器件，会概率的出错。仿真不会出现问题。但教训是，以后 if case 默认的必须写全。这也是老师在上课强调过，但实际上通过很长时间的调试自己才切身体会到的错误。



三、 对讲义中思考题（如有）的理解和回答

四、 在课后，你花费了大约___20_____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）

感谢助教陈飞宇同学对我的提醒，解决了仿真通过但是上板没有过的情况。
整个学期的实验到这里就结束了，感觉学会了看布局布线，学会了阅读指令集，整个 cpu 的实验使得我对于计算机指令的执行有了更深的体会。
建议:由于云平台设计得非常好，可能在省略了大部分不属于课程内容的麻烦之后,也会带来一些知识上的茫然。比如不知道自己的程序是怎么烧写到板子上的，比如不同型号的板子对于输入输出管脚的绑定。
之前在数字电路课上使用过 Vivado 平台，不过对于其大部分的功能没有用过，只使用了 run simulation，后来发现可以用到上面的 create block design,run systhensis,run implemenation,generate bit stream,和硬件相连，或者直接通过 export hardware 之后 launch SDK，然后通过 C 代码来控制程序。建议增加半个课时介绍一些和板子相连的知识,不过还是感谢云平台省去了很多操作上的麻烦。