实验 11 报告

学号：2017K8009922027 2017K8009929011
姓名：张磊、郭豪
箱子号：39

一、实验任务（10%）

将 lab10 部分完成的转接桥连接到 lab9 的 cpu 上，顺利通过仿真和随机延迟的测试。

二、实验设计（40%）

（一）总体设计思路

1. 将 cpu_core 模块的输入输出修改为类 SRAM 端口，然后编写 top 模块，连接 cpu 和 lab10 完成的 cpu_axi_interface。
2. 对于 IF 阶段，由于发送取指请求后不会当拍返回读数据，需要根据类 sram 的端口的信号的含义，编写两个状态机。第一个状态机用于 pre_IF 欲取指阶段，用于发送请求和地址，状态机二则用于接收取指的指令。
3. 对于 EXE 阶段，通过修改 ready_go 信号，只有当访存的地址和数据发送完成（data_addr_ok 拉高后）才能跳转到 MEM 阶段，同样的，MEM 阶段会接收访存数据，只有当转接桥拉高 data_data_ok 时才代表访存指令的 MEM 阶段的结束，可以跳转到 WB 阶段。
4. 对 ID，EXE，WB 阶段，当例外/清流水到来时，需要将 eret_flush 和 ex 信号使用寄存器保存，用于阻塞该流水级，阻塞的方式是通过拉低 valid 信号实现的，只有当来自上一级的数据更新该级时才停止阻塞。
5. ID，EXE，WB 阶段的阻塞都通过修改 ready_go 信号，当例外到来时拉低 ready_go 信号，当来自前一阶段的数据到来时正常执行停止阻塞。
6. 对于 IF 阶段的例外处理，需要考虑在例外到来时，IF 阶段的两个寄存器正处于哪个阶段，根据不同阶段判断处理的流程一级 fs_pc 的更新。

（二） 重要模块 1 设计：pre_IF 欲取指模块

1. 工作原理

在状态一，当 to_fs_valid 和 fs_allowin 握手成功时，代表上一次取指的结束，此时更新 pc 并发出下一次取指的请求。

进入状态二后，当 inst_addr_ok 和 inst_req 握手成功时，代表 inst 的读请求握手成功，即 pre_IF 阶段结束，拉高 pre_IF_ready_go 信号，当 inst_data_ok（读指令握手成功）时，拉低 pre_if_ready_go，回到状态一进行下

一次取指请求。

```
always@(posedge clk) begin
    if(reset) begin
        cstate_inst_req <= `STATE_ONE;
        pre_IF_ready_go <= 1'b0;
        inst_req_r <= 1'b0;
    end
    else if(cstate_inst_req == `STATE_ONE) begin
        if(to_fs_valid && fs_allowin) begin
            inst_req_r <=(buf_npc[1:0] == 2'b00) ? 1'b1 : 1'b0;
            cstate_inst_req <= (buf_npc[1:0] == 2'b00) ? `STATE_TWO : `STATE_ONE
        end
    end
    else if(cstate_inst_req == `STATE_TWO) begin
        if(inst_addr_ok &&  inst_req) begin
            inst_req_r <= 1'b0;
            pre_IF_ready_go <= 1'b1;
        end
        if(inst_data_ok) begin
            cstate_inst_req <= `STATE_ONE;
            pre_IF_ready_go <= 1'b0;
        end
    end
end
```

图 1 pre_IF_stage

2. 接口定义

名称	位宽	功能描述
pre_IF_ready_go	1	欲取指阶段的 ready_go 信号，用于代表读请求的结束。
cstate_inst_req	2	欲取指状态机的状态

表 1 模块一重要信号

3. 功能描述

用于发送读指令的请求和地址。

(三)重要模块 2 设计：IF 模块

1. 工作原理

当没有例外产生时，在状态一，当 pre_IF_readygo 拉高时说明读请求已经发出，可以进入状态二等待指令，在状态二时，当 inst_data_ok 拉高时接收指令 ram 的数据，并拉高 if_readygo 信号等待和 ID 阶段的握手，fs_to_ds_valid 和 ds_allowin 握手成功时，拉低 if_readygo 信号并返回到状态一。

```
always@(posedge clk) begin
    if(reset) begin
        cstate_fs_readygo <= `STATE_ONE;
        fs_ready_go_r <= 1'b0;
        inst_sram_rdata_r <= 32'b0;
        mark_for_if <= 3'b0;
        mark_for_fs_ex <= 1'b0;
        mark_fs_ex_repeat <= 1'b0;
    end
    else if (cstate_fs_readygo == `STATE_ONE) begin
        if(to_fs_valid && fs_allowin) begin//fs_pc更新后无需阻塞
            mark_fs_ex_repeat <= 1'b0;
        end
        //状态转换
        if(pre_IF_ready_go) begin
            cstate_fs_readygo <= `STATE_TWO;
            mark_for_if <= 2'b01;
            mark_fs_ex_repeat <= 1'b0;
        end
        //地址错误
        else if(fs_addr_error && ~mark_fs_ex_repeat) begin
            cstate_fs_readygo <= `STATE_TWO;
            mark_for_if <= 2'b01;
            mark_for_fs_ex <= 1'b1;
        end
    end
    else if (cstate_fs_readygo == `STATE_TWO) begin
        if(mark_for_fs_ex) begin
            fs_ready_go_r <= 1'b1;
            inst_sram_rdata_r <= 32'b0;
            mark_for_fs_ex <= 1'b0;
        end
        else if(inst_data_ok) begin
            fs_ready_go_r <= 1'b1;
            inst_sram_rdata_r <= inst_sram_rdata;
            mark_for_if <= 2'b10;
        end
        else if((fs_to_ds_valid && ds_allowin && fs_ready_go) ||
        mark_to_fs_valid == 5'b00100 || mark_to_fs_valid == 5'b00011) begin
            fs_ready_go_r <= 1'b0;
            cstate_fs_readygo <= `STATE_ONE;
            mark_for_if <= 2'b00;
            if(mark_to_fs_valid == 5'b00100) begin
                mark_fs_ex_repeat <= 1'b1;//避免例外多次进行
            end
        end
    end
end
```

图 2 IF 模块状态机

2. 接口定义

名称	位宽	功能描述
fs_ready_go_r	1	IF 阶段的 ready_go 信号

名称	位宽	功能描述
inst_sram_rdata_r	32	用于保存 ram 发送的指令的内容
mark_for_if	2	用于标记 IF 状态机所处状态，便于例外处理
mark_for_fs_ex	1	发生取指地址错例外的处理
mark_fs_ex_repeat	1	用于防止取指例外处理时重复执行同一条 PC 的流程
cstate_fs_readygo	2	取指状态机的状态信号

表 2 模块二重要信号

3. 功能描述

取指模块，用于完成对 ram 发送的指令内容的接收和拉高 ready_go 信号

（四）重要模块 3 设计：EXE 阶段访存发送模块

1. 工作原理

1) 当 EXE 阶段的指令为 L/S 类指令时，需要发出访存请求，如图所示，只有 EXE 阶段是访存指令时才会经过此状态机,并使用 es_ready_go_r 的信号作为 ready_go 信号
2) 判定是 L//S 类指令后，EXE 阶段拉高 data_req 信号发出请求，并进入状态而。
3) 在状态二，当 data_req 和 data_addr_ok 握手成功时，代表 data 的请求结束，此时 es_ready_go 拉高代表可以进入下一个流水级。
4) 当 es_to_ms_valid 和 ms_allowin 握手成功时，EXE 阶段的内容被传输到 MEM 阶段，代表 EXE 阶段处理该指令的结束，此时应该拉低 es_ready_go 信号等待下一条指令的数据。

```
always @(posedge clk) begin
    if(reset) begin
        cstate_data_req <= `STATE_ONE;
        data_req_r <= 1'b0;
        es_ready_go_r <= 1'b0;
    end
    else if(cstate_data_req == `STATE_ONE) begin
        if((es_load_op || es_store) &&
        ~forbid_store && has_inst_exe && ~es_ex) begin
            data_req_r <= 1'b1;
            cstate_data_req <= `STATE_TWO;
        end
    end
    else if(cstate_data_req == `STATE_TWO) begin
        if(data_req && data_addr_ok) begin
            es_ready_go_r <= 1'b1;
            data_req_r <= 1'b0;
        end
        else if(es_to_ms_valid && ms_allowin) begin
            es_ready_go_r <= 1'b0;
            cstate_data_req <= `STATE_ONE;
        end
    end
end
```

图 3 data 请求模块

2. 接口定义

名称	位宽	功能描述
data_req_r	1	EXE 阶段发送读请求信号
es_ready_go_r	1	S/L 类指令采纳的 readygo 信号
es_load_op	1	EXE 阶段为 L 类指令
es_store	1	EXE 阶段为 S 类指令
cstate_data_req	1	访存请求状态机状态

表 3 模块 3 重要信号

3. 功能描述

EXE 阶段用于发送访存请求的状态。

（五）重要模块 4 设计：MEM 阶段访存接收模块

1. 工作原理

1) 如图, 当 MEM 阶段是 L/S 类指令时, ms_readygo 采用 ms_ready_go_r 的信号。
2) 对 L/S 类指令，其在 MEM 阶段的任务是接收 EXE 请求的读/写响应，具体的，当 data_ok 发送来时，说明访存的完成，此时保存读数据并拉高 ms_ready_go_r。
3) 当 ws_allowin 和 ms_to_ws_valid 握手成功时，代表 MEM 阶段和 WB 握手成功狗，需要拉低 ms_ready_go_r, 等待下一条指令。

```
reg ms_lw_valid_r;
always @(posedge clk) begin
    if(reset) begin
        ms_ready_go_r <= 1'b0;
        data_sram_rdata_r <= 32'b0;
    end
    else if(data_data_ok) begin
        ms_ready_go_r <= 1'b1;
        data_sram_rdata_r <= data_sram_rdata;
    end
    else if(ws_allowin && ms_to_ws_valid) begin
        ms_ready_go_r <= 1'b0;
    end
```

图 4 写响应状态机代码

2. 接口定义

名称	位宽	功能描述
ms_ready_go_r	1	MEM 阶段是访存指令时采用的 ready_go 信号
data_sram_rdata_r	32	用于保存 S 类指令读取的数据

表 4 模块 4 重要信号

3. 功能描述

写响应通道状态机，接收 axi_slave 的写响应信号。

（六）重要模块 5 设计：IF 阶段例外处理

1. 工作原理

1) 当例外(ws_ex_in)或清流水(eret-flush)到来时，此时的 IF 阶段可能处 pre_IF 和 IF 状态机的任意一种状

态，因此，例外处理的讨论也需要充分考虑多种情况。

2) mark_to_fs_vliad 是用于标记例外到来时的状态的信号，具体的情况包括下表几种：

	mark_to_fs_vliad	具体情形
1	5'b01000	发送处 data_req 但是尚未获得
2	5'b00001	PRE_IF 刚发出 addrok， IF 仍处于状一
3	5'b00010	IF 状态机进入状态二但未收到 data_ok
4	5'b00011	Data_ok 信号刚好和 ex 信号同一拍到来
5	5'b00100	IF 阶段接收到 data_ok，尚未和 ID 握手
6	5'b00101	ex 到来和 IF 与 ID 的握手同一拍
7	5'b10000	取指地址错例外，单独考虑
8	5'b00000	例外处理结束

表 5 例外到来时的具体情况的讨论

a) 当 eret_flush 和 exception 到来时，即刻更新 buf_pc 用于下一次取指。

b) 针对情况 1， 2， 3， 6，即当先的 PC 处于正在取指的阶段，处理方式是令 IF 阶段完成该次取指，当第一次 data_ok 到来时，说明第一次取指结束，该次取指的指令需要被舍弃。舍弃的方式是在 eret_flush 和 exception 到来时拉低 to_fs_valid 的方式。

c) 第一次 Data_ok 到来后， 1， 2， 3， 6 状态 5，状态 5 转换到状态 8 的条件是 mark_for_if == 2'b00，即 IF 状态机回到状态一等待 buf_pc 的取指请求时，进入状态 8 后会拉高 to_fs_valid，表示下一次取指有效。

d) 对于取指错例外的处理，由于当 PC = A 时发生取指错例外，其后的所有 PC 都会发生该例外，因此需要单独考虑。首先，当 fs_pc 产生取指错例外时，依旧会沿用 PRE_IF 和 IF 阶段的状态机，只是不会发出请求也无需进行握手就可以进行状态转换，因此使用 mark_for_fs_ex 作为标记，当其拉高时代表 IF 阶段流程运行到 STATE_TWO，此时转换到状态 5。

e) 对于状态 4,data_ok 和 ex 信号同时到来，下一个 PC 即为 bfc00380/cp0_epc，可以直接转换到状态 8 进行取指。

2. 接口定义

名称	位宽	功能描述
mark_to_fs_valid	5	例外处理状态机的状态
mark_for_if	2	用于标记 IF 状态机所处的阶段
inst_req	1	指令请求
pre_IF_ready_go	1	欲取指阶段 ready_go 信号
Inst_data_ok	1	读数据返回信号
mark_for_fs_ex	1	用于标记取指地址错例外发生时所处的阶段

表 6 IF 阶段例外处理重要信号

```
always @(posedge clk) begin
    if(reset) begin
        mark_to_fs_valid <= 5'b00000;
    end
    else if(ws_ex_in || eret_flush) begin
        if(fs_addr_error && ~pre_IF_ready_go) begin
            mark_to_fs_valid <= 5'b10000;
        end
        else if(inst_req  && ~pre_IF_ready_go) begin
            mark_to_fs_valid <= 5'b01000; //发出请求但是未收到addr_ok
        end
        else if(pre_IF_ready_go && mark_for_if == 2'b00) begin
            mark_to_fs_valid <= 5'b00001;//刚发出addr_ok
        end
        else if(pre_IF_ready_go && mark_for_if == 2'b01 && inst_data_ok) begin
            mark_to_fs_valid <= 5'b00011;//data_ok和例外一起来
        end
        else if(pre_IF_ready_go && mark_for_if == 2'b01) begin
            mark_to_fs_valid <= 5'b00010;//未等到data_ok
        end
        //刚好赶上握手
        else if(mark_for_if == 2'b10 && (fs_to_ds_valid && ds_allowin)) begin
            mark_to_fs_valid <= 5'b00101;
        end
        else if(mark_for_if == 2'b10) begin
            mark_to_fs_valid <= 5'b00100;//data_ok已经来了,还没有传给id
        end
        else if(~pre_IF_ready_go && mark_for_if == 2'b00) begin
            mark_to_fs_valid <= 5'b00000;//当前指令已经传到ds
        end
    end
    else if(mark_to_fs_valid == 5'b00001 || mark_to_fs_valid == 5'b00010
         || mark_to_fs_valid == 5'b01000 || mark_to_fs_valid == 5'b00101) begin
        if(inst_data_ok ) begin//第一个data_ok,无效
            mark_to_fs_valid <= 5'b00100;
        end
    end
    else if(mark_to_fs_valid == 5'b10000) begin
        if(mark_for_fs_ex) begin
            mark_to_fs_valid <= 5'b00100;
        end
    end
    else if(mark_to_fs_valid == 5'b00100) begin
        if(mark_for_if == 2'b00) begin //例外时IF的PC的取指结束，且无效
            mark_to_fs_valid <= 5'b00000;
        end
    end
    else if(mark_to_fs_valid == 5'b00011) begin
        mark_to_fs_valid <= 5'b00000;
    end
end
```

图 4 IF 阶段例外处理状态机

4. 功能描述

IF 阶段例外处理的状态机。

三、实验过程（50%）

（一）实验流水账

1. 2019 年 11 月 19 日 22:00-23:00 ， 接口信号修改；
2. 2019 年 11 月 20 日 13:00-19：00，仿真发现理解错误，rtl 代码编写
3. 2019 年 11 月 21 日 13:00-22:00，成功连接 cpu_axi_interface,并进行 debug。
4. 2019 年 11 月 22 日，10:00- 23:00，debug 并最终 pass， 修改随机种子后失败
5. 2019 年 11 月 23 日，10:00- 次日两点，IF 阶段凑波形写的太乱 debug 很心烦，于是重新从头开始编写 IF 阶段，debug， 并于次日两点顺利通过随机数测试
6. 2019 年 11 月 25 日，20:00- 23:00，实验报告撰写-

（二）错误记录（只记录部分有代表性的错误）

1. 错误 1：握手理解有误

（1）错误现象

crossbar 输入的 s_axi_araddr 有地址，但是输出的 m_axi_araddr 地址为 0，导致输入到 ram 的 ram_araddr 也一直为 0。

（2）分析定位过程

查看波形图发现当 araddr 在 arvalid 和 arready 握手成功的当拍即便为 0，导致最终发送的地址错误。

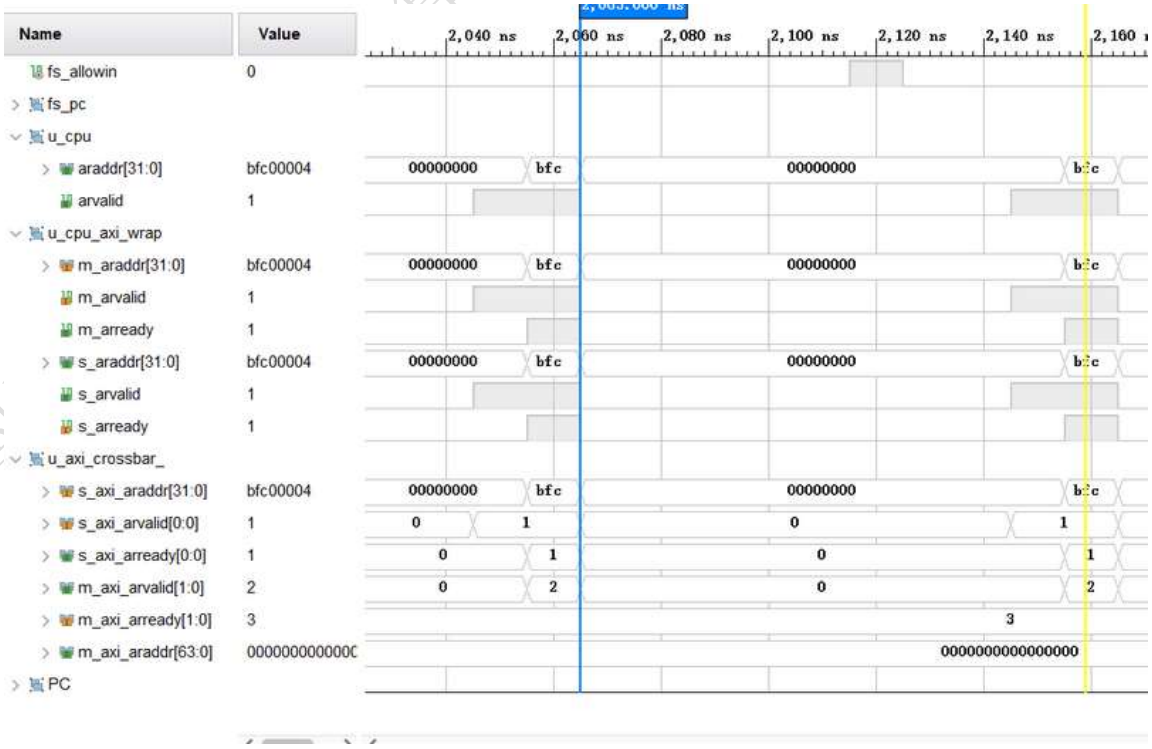

图 5 错误 1 波形图

（3）错误原因

axi 要求 request 拉高后不能变请求，也就是 arvalid 为 1 后，araddr 应该一直保持不变。因为 addr 使用了 araddr&arvlaid&arready，导致最终 ram 获取的地址为 0。

2. 错误 2：流水线阻塞出错

（1）错误现象

Rf_wdata 写入数据错误

（2）分析定位过程

查看汇编文件得知是写后读相关前递数据的错误， 黄线时 lw 指令到 WB 阶段写回数据，但是前一条指令的 EXE 阶段已经执行结束了，前递的数据出错。

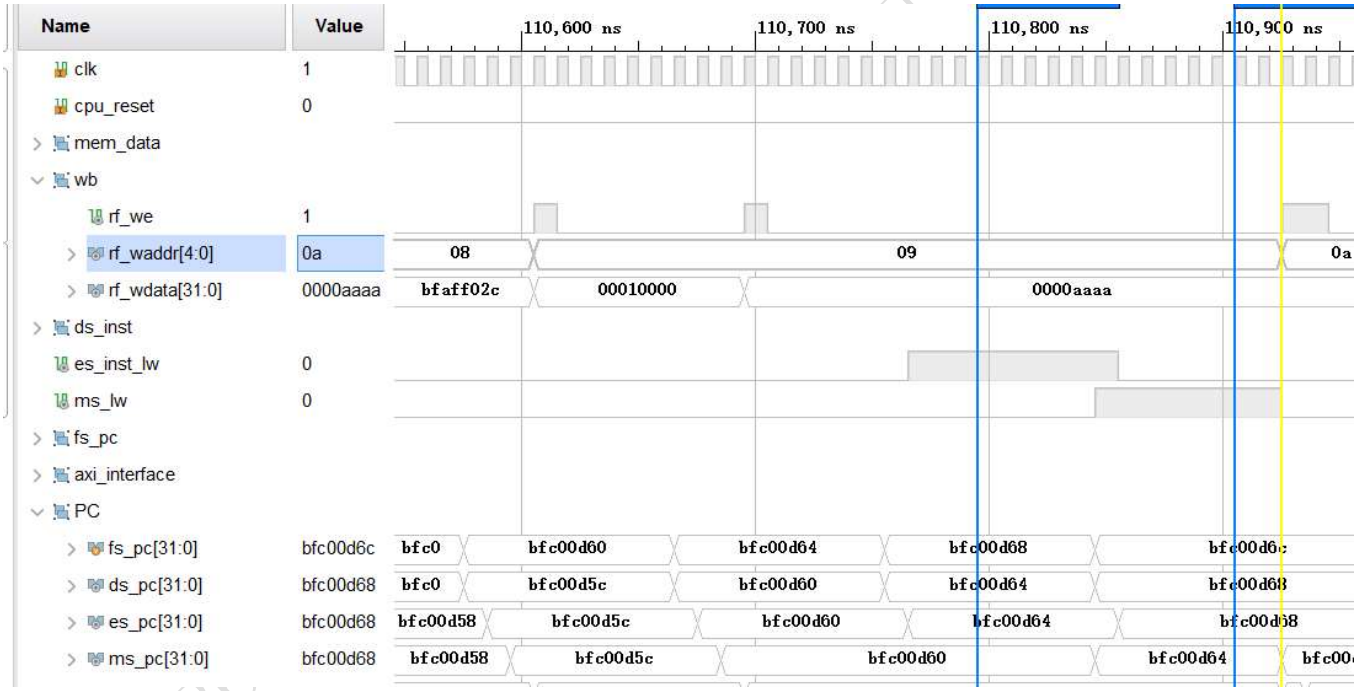

图 6 错误二波形图

（3）错误原因

使用 wire 信号进行阻塞只阻塞了一拍，当 MEM 阶段位 LW 指令，此时产生写后读相关时，阻塞信号也需要使用一个 reg 信号保存，知道 MEM 阶段收到 data_ok，数据前递后才能停止阻塞。

（4）修正效果

修正后顺利通过前 68 个测试。

3. 错误 3：例外处理更新 pc

（1） 错误现象

如图，例外到来后 fs_pc 没有完成到 bfc00380 的跳转。

```
[13774907 ns] Error!!!
    reference: PC = 0xbfc00380, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x00004000
    mycpu    : PC = 0xbfc1e964, wb_rf_wnum = 0x09, wb_rf_wdata = 0x45000000
```

（2）分析定位过程

开始编写时没有考虑例外到来时 buf_pc 的问题，由于当例外到来时, next_pc 更新了，buf_valid 为 1， 但此时 buf_pc 并没有更新,导致下一次 fs_to_ds_valid 和 ds_allowin 握手成功时没有跳转成功。

处理方式是例外到来时当即更新 buf_pc 并保存例外信号，当当前取值级的指令的取值结束后，舍弃掉该取值，然后更行 fs_pc 进行跳转 PC 的取指。

（3）错误原因

例外处理时 IF 阶段的 PC 更新出错。

（4）修正效果

报错，不过是其他错误。

4. 错误 4：指令跳转时的数据前递问题

（1）错误现象

PC 错误

（2）分析定位过程

如图， ID 阶段是跳转指令，EXE 阶段为 LW 指令，EXE 阶段的执行还没有结束，ID 阶段译码过程只需要一拍，因此已经完成跳转，跳转到错误的 PC。

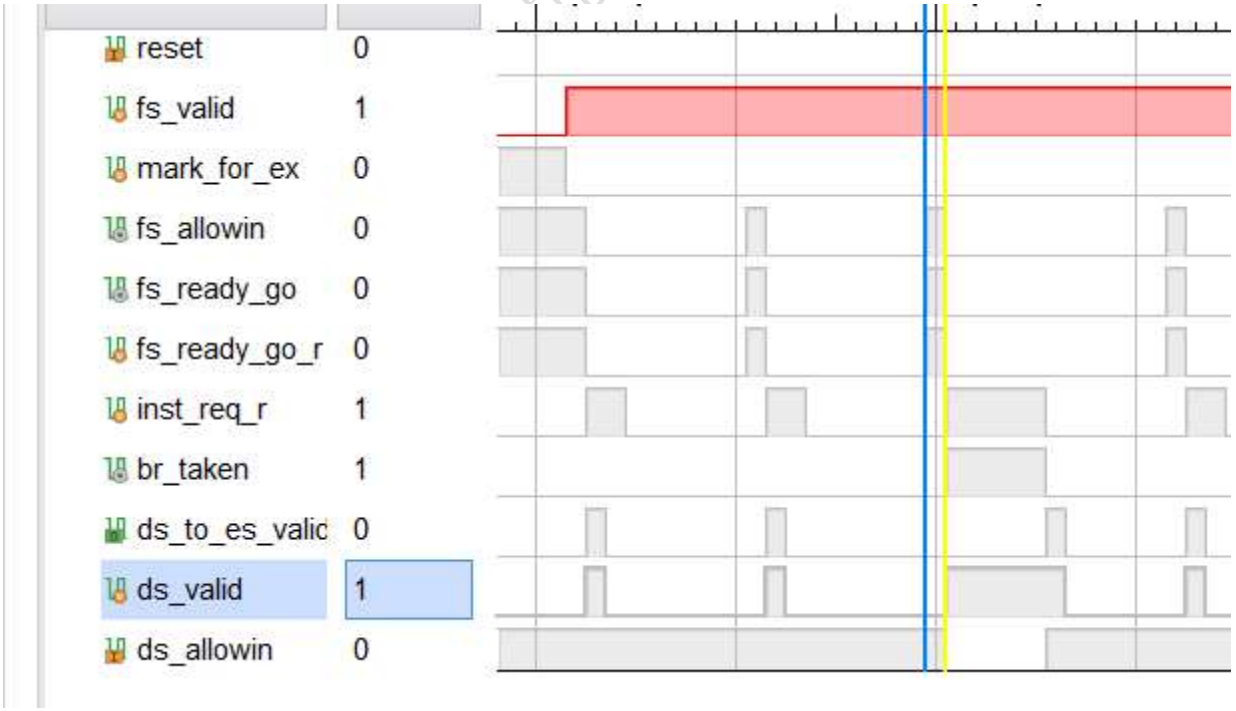

图 9 错误 4 波形图

（3）错误原因

LW 指令和跳转指令发生数据相关，但是当 LW 指令还在 EXE 阶段时，没有到达 MEM 阶段，ID 阶段的跳转

已经完成了，导致跳转到错误的地址处。

处理方式是修改阻塞的逻辑，只有当数据从 MEM 阶段前递来之后，br_taken 才能为 1 完成跳转。

（4）修正效果

修改后依旧报出错误,但是错误变了。

5. 错误 5：延迟槽标记错误

（1）错误现象

如图，时钟中断到来时，被标记例外的 PC 错误，导致 cp0_epc 保存的 PC 错误。

```
        [14322000 ns] Test is running, debug_wb_pc = 0xbfc5fa90
------------------------------------------------------------
[14326557 ns] Error!!!
    reference: PC = 0xbfc005d0, wb_rf_wnum = 0x1a, wb_rf_wdata = 0xbfc5fa8c
    mycpu    : PC = 0xbfc005d0, wb_rf_wnum = 0x1a, wb_rf_wdata = 0xbfc5fa90
```

图 10 错误 5 报错

（2）分析定位过程

错误的测试为时钟中断测试，查看波形图发现 epc 中保存的值被更新为 bfc5fa90, 查看汇编文件，发现 0xbfc5fa8c 处为一条 b 类指令，而 bfc5fa90 则为 nop 指令，位于延迟槽中，因此无论例外标记到这两条指令的哪一条 epc 都应该保存 0xbfc5fa8c，因此应该是延迟操标记的问题。

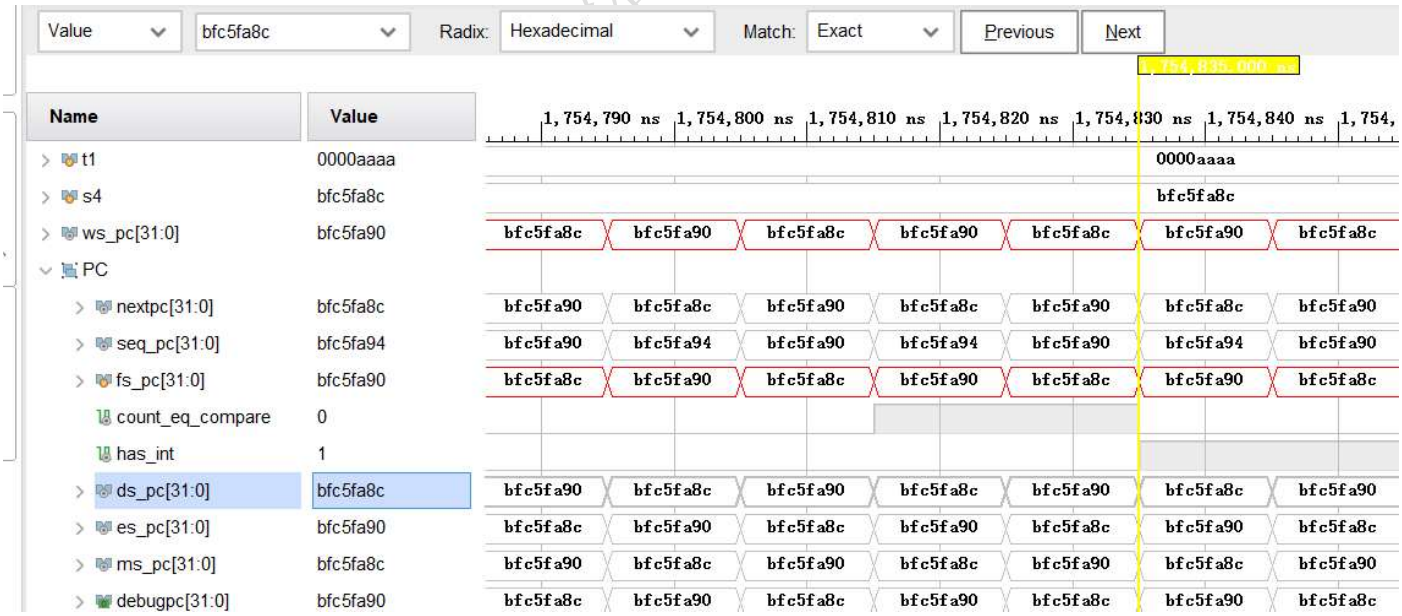

图 11 错误 5 波形图

（3）错误原因

ID 阶段是跳转指令时，传到 IF 阶段的用于标记延迟槽的标记信号也需要用 reg 信号保存，不然当 fs_to_ds_valid 和 ds_allowin 握手成功时标记已经失效了。

（4）修正效果

顺利通过时钟中断测试。

四、实验总结（可选）

1.虽然过了上周的测试，但是对 axi 通道的信号理解还不太准确，理解错误导致这周还是遇到了一些 bug 需要修改 cpu_axi_interface.

2.这一次实验开始之前没有提前构思好，只知道将转接桥连接到 cpu 上可以正常取指令，没有具体考虑到需要因此而修改的数据前递，例外处理等问题。

3. 因为对实验需要做什么有宏观的构思，前三天的实验过程到 69 之后，基本是遇到一个 bug 处理一个 bug 的凑波形过程，虽然最终 pass 了但是 IF 各种信号相互牵制已经很难读懂了。换随机数错了之后 debug 无从下手。因此周五晚上仔细去构思了整个实验，周六从头写了一遍，逻辑思维清晰后，只用了一个多态状态机就处理了所有例外到来时 IF 阶段的可能情况，这一次从编写到 debug 到换随机数通过只用了一天，实现速度和代码的可读性都高了很多。所以实验还是先有整体的公关构思，了解到具体要做到啥后再开始，避免摸黑前行浪费时间。

4.讲义上对具体实现和可能遇到的问题写的感觉没以前实验详细了，因此这周实验花了很久时间，感觉讲义不写那么详细，自己去设计思考收获挺多的，但还是感觉实验时间给两周比较合适（这句划重点）。