实验 6 报告 2

第 32 小组
袁峥

一、实验任务（10%）

1、CPU 顶层修改为 AXI 接口。CPU 对外只有一个 AXI 接口，需内部完成取指和数据访问的仲裁。

2、集成到 SoC_AXI_Lite 系统中。

3、完成功能测试。

4、在 myCPU 上运行 lab4 的电子表程序， 要求能实现相同功能。

二、实验设计（30%）

这次的实验任务主要是在原来的五级流水 CPU 中将指令和数据存储器的两个 SRAM 接口，通过一个转换桥，最终用一个 AXI 接口与外界连接。在上周的实验中已经完成了一个由两个类 SRAM 接口到一个 AXI 接口的转换桥，因此本周需要将原来 CPU 中存储器的 SRAM 接口更改为类 SRAM 接口。

主要更改的地方一共有两个，分别在取指级（IF）和访存级（MEM）。

在取指级，由于原来的取指一拍就可以完成，但是现在完成时间不定，因此只有在取指完成时才能将 PC 和指令向译码级传递，在取指完成之前需要将取指级阻塞。这里还有一个细节的地方是，如果在执行级有中断例外需要响应，此时需要将 IF、ID、EXE 级都清空，但是该清流水信号只存在一拍，因此在 IF 级需要增加一个寄存器，记录在每次取指命令发出到取回指令的过程中是否有清空取指级流水的信号，如果有，那么取回的指令不能向译码级传递。另外，在取指级还增加了一个寄存器来记录是否有在进行取指的操作，在 inst_addr_ok 时则置 1，在每次完成取指操作后，即 inst_data_ok 时置 0。inst_req 信号在没有进行取指操作时置 1。

在访存级，首先在类 SRAM 接口的基础上，还添加了一个字节写使能控制信号，由访存级直接通过译码结果发出，而不用再在转换桥中通过访存地址和读写字节长度进行转换，这样可以方便 SWL、SWR 指令的实现。其余修改主要还有，在访存结束即 data_data_ok 为 1 前，将访存级进行阻塞。同时由于之前的 CPU 设计中，访存级不会发生阻塞，因此此处需要增加一个控制访存级阻塞的信号，并修改相关信号向写回级传递的控制。

三、实验过程（60%）

（一）实验流水账

12 月 3 日晚上 8 点至 11 点，编写代码。

12 月 4 日下午 3 点至晚上 11 点，调试代码，并通过仿真和上板测试。

12 月 5 日上午 9 点至 11 点，完成实验报告。

（二）错误记录

1、错误 1

（1）错误现象

仿真一开始就卡住，不再取出新的指令。

（2）分析定位过程

查看波形后发现在取出第一条指令后，无法取出第二条指令。

（3）错误原因

由于发生在测试一开始，因此没有通过跟踪波形的方法来改错，而是重新分析了一遍取指级（IF）的逻辑，发现在逻辑上存在问题，对于 inst_addr 信号，即取指地址，应该用寄存器先将其保存下来，而原来的设计由于取指保证一拍完成，因此为了减少阻塞而采用了提前送信号的方法，所以原来这里用的是 wire 型。

（4）修正效果

修改代码后重新仿真，能够连续取出指令。

（5）归纳总结

这个错误主要是由于 wire 型和 reg 型的变量混用导致的，在增加总线后修改设计时没有想清楚取指级的逻辑，所以导致写代码时时序上出现了错误，inst_addr 信号送的不对。

2、错误 2

（1）错误现象

仿真进行到一半时不再向前动，时钟也停住。

（2）分析定位过程

这个问题以前碰到过，当时通过查看文档知道了是因为组合回路造成的。而且可以通过看这时候代码中的指针位置来了解大概是哪句代码有问题。

（3）错误原因

查看错误代码后根据以前经验，知道了错误的原因。产生组合回路的原因是在访存级（MEM）中用来生成阻塞信号（MEM_stall）的信号在产生时又用了阻塞信号，这样导致了组合回路。这个错误发生的原因是因为在原来的设计中访存级没有阻塞信号，因此直接用切分流水时用来接收信号的寄存器用作输出信号，但增加阻塞信号后，

输出信号需要另外用 wire 型进行输出，同时用来进行阻塞判断的是接受信号的 reg 型信号，这里如果错用就会产生回路。

（4）修正效果

将代码进行如下修改。

```
assign MEM_pc_out       = (MEM_stall) ? 32'd0 : MEM_pc;
assign MEM_inst_out     = (MEM_stall) ? 32'd0 : MEM_inst;
assign MEM_src2_out     = (MEM_stall) ? 5'd0  : MEM_src2;
assign MEM_dest_out     = (MEM_stall) ? 5'd0  : MEM_dest;
assign MEM_memtoreg_out = (MEM_stall) ? 1'd0  : MEM_memtoreg;
```

同时将 MEM_ pc、MEM_inst、MEM_src2 等信号输出到 stall.v 用来生成 MEM_stall 信号，这样就不会导致组合回路。

（5）归纳总结

这个问题是由于不同流水级之间的传递信号的代码很久没有修改过，因此对其中的逻辑有些遗忘。其实这个部分与前面流水级的重复度较大，都是用一个寄存器先接收从上一级传来的信号，然后再用 wire 信号加上阻塞判断后把信号往下一级流水传递。因此发现问题后修改起来比较快。

3、错误 3

（1）错误现象

异常处理时 mfc0 $14 的值与 golden trace 不一致，即 CP0_EPC 的值不正确。

（2）分析定位过程

在波形中定位到最后一次更新 CP0_EPC 的位置，查看 CP0_EPC 信号后发现相应的异常类型不正确。

（3）错误原因

查看了测试程序的汇编代码后，发现这里真正应该相应的是取指地址错。在处理取指地址错的时候，是在取指级将其标记，但是继续往后执行，直到执行级再响应。但是这次产生的问题是，在继续执行的时候，用该地址进行取指后，取出的错误指令又触发了另一个异常，而且该异常的优先级比取指地址错异常的优先级，因此发生了异常的错误相应。这里修改的方法是，在取指级标记了取指地址异常错误后，不再将从错误地址取出的指令向后传递，而是将该指令放空。具体修改如下：

```
assign IF_inst = (inst_data_ok) ? ((is_clear || IF_inst_addr_err) ? 32'd0 : inst_rdata) : 32'd0;
```

（4）修正效果

重新运行仿真后通过该测试点。

（5）归纳总结

这个错误应该是属于在实验 5 的例外与中断的支持中遗留下来的错误，只不过当时由于使用的 inst_ram 不同，从而恰巧没有碰到这个错误。这个问题是由于在细节上考虑的不够细致严谨，对于产生了错误的指令，应该尽量清空不让其向后传递从而引起其他意想不到的错误。

4、错误 4

（1）错误现象

在第 81 个测试点（硬件中断——时钟中断）中，跑了很久也没有触发时钟中断。

（2）分析定位过程

查看波形后发现在前面其实已经有过 CP0_COUNT 等于 CP0_COMPARE 的情况，但是没有能够成功响应中断并跳转到异常处理入口地址。

（3）错误原因

在查看取指级（IF）的信号后发现，在触发中断后没有能够成功清掉这一级的流水。取指的过程由于随机延时可能会持续多个时钟周期，因此在取指级的设计中加入了一个寄存器来记录每一次取指的过程中是否有异常导致的清流水，如果有那么即使取出了新的地址也不要向后传递，而是跳转到异常处理入口地址。这里是由一个 is_clear 寄存器控制的。该寄存器应该在每次取指结束（inst_data_ok 到达）后清 0，在每次取指过程中如果有 IF_clear 传入则将寄存器置 1。但是原来的设计中错误地在取指的 inst_addr_ok 到达时就将该寄存器清 0，导致错过了一个 IF_clear 信号，因此没有能够执行清流水并响应异常。

（4）修正效果

is_clear 信号的修改如下：

```
always @(posedge clk)
begin
    if (!resetn)
        is_clear <= 1'b0;
    else if (inst_data_ok)
        is_clear <= 1'b0;
    else if (IF_clear)
        is_clear <= 1'b1;
end
```

修改代码后，仍然出现一样的情况，但导致错误的原因不同，错误原因见错误 5。

（5）归纳总结

这个错误可以属于笔误，但是由于比较深层，因此跟踪并发现起来比较不容易，以后写代码要细心，特别是名称相似的信号不要弄混。

5、错误 5

（1）错误现象

在第 81 个测试点（硬件中断——时钟中断）中，跑了很久也没有触发时钟中断。

（2）分析定位过程

查看波形发现标记中断的 interrupt 信号从取指级往上传递的过程中丢失，导致在执行级处理异常中断的时候没有接收到 interrupt 信号，也就没有真正响应时钟中断。

（3）错误原因

正常在流水级间传递信号应该在每一级的任务完成后，但是现在的 interrupt 信号从取指级发出时没有控制好，没有等到指令取回再传，而是一旦等到 CP0_CAUSE 修改 TI 位后就传递，导致传递的中断信号没有真正标记在指令上，所有在后序流水中中断信号丢失。

（4）修正效果

在取指级的指令取回时再向译码级传递 interrupt 信号，将 IF_interrupt 修改如下：

```
assign IF_interrupt = (inst_data_ok && !is_clear) ? interrupt : 1'b0;
```

修改后时钟中断可以正常响应了。

四、实验总结

（一）组员：袁峥

这次的实验看上去比较容易，主要想法就是在取指和访存完成之前，分别将取指级和访存级进行阻塞，等到 inst_data_ok 和 data_data_ok 到来时再向下一级传递。但在实现上遇到了许多问题，在仿真调试上花费了整整一晚上。一开始在取指级的处理有问题，导致 CPU 直接无法跑起来，在重新思考了取指级的信号处理后才让 CPU 重新跑起来。中间还发现了前面实验遗留下来的一些错误，由于种种原因现在才碰到。最后在上板的时候，发现计时器的硬件中断部分出了一些问题，响应起来很迟钝，按下一个键需要过了一两秒才有反应。思考后发现了问题，在以前的设计中为了防止一按键不停的响应中断，因此在中断处理程序中每次处理完后空跑了大约 0.1 秒，这个 0.1 秒当时是按照每个指令一个周期来计算的重复次数，但是这次的 CPU 在处理每条指令时，由于取指的延时，完成时间远远不止一个周期，因此导致每次中断处理花费的时候变得更长，所以每次按键后需要一两秒才响应。清楚问题后减少了中断处理中空跑的周期数，重新上板后果然解决了问题。