**LAB6报告**

学号： 2021K8009929010

姓名： 贾城昊

学号： 2021K8009929016

姓名： 李金明

学号： 2021K8009929007

姓名： 牛浩宇

箱子号： 13

1. 实验任务

Lab7的要求为给CPU增加AXI总线接口，令CPU能够通过总线与系统中的内存、外设进行交互。主要分为3个阶段：

Exp14：将CPU原有的访问SRAM的接口改为类SRAM总线接口，增加握手信号、随机延迟。

Exp15：通过转接桥将类SRAM接口转接为AXI接口，并实现固定延迟的验证。

Exp16：随机改变种子，实现随机延迟的验证。

1. 实验设计
   1. 总体设计思路

对CPU发生中断和异常时进行状态控制的信息存放在csr寄存器中，这部分内容由硬件电路自动完成，不属于用户可见的软件上下文。为实现这部分电路，代码中新增一个csr模块，其内包含了所有的csr寄存器以及对csr寄存器进行读写的控制电路，并例化在WB流水级中。CPU在WB流水级收集到本条指令在所有流水级中发生的异常或中断信息后，对csr模块中的控制状态寄存器进行集中读写。

为了实现精确异常，首先需要保证异常指令前面所有的指令全部执行完毕，因此在WB级之后进行中断和异常处理，这样前面的指令已经完成了所有的操作。此外，还需要保证异常指令后面“错取”的指令不能改变机器的状态，所以如果WB级发现本级指令出现异常时，必须将前面各个流水级清空，即通过向前面各流水级传递一个flush信号来“等效”把各级的valid标志置0（因为valid是一个寄存器，下一拍值才能改变，所以这里的“等效”指的是与valid信号相关的判断逻辑改为valid & ~flush，这样看起来就像当拍将valid清0了）。

需要注意的是，如果“错取”的指令包含store类型指令，那么该指令在清空各流水级之前已经将数据写入到内存当中，导致机器的状态出现了更改。为了解决这一问题，WB级和MEM级都需要把当前所执行的指令是否发生异常的判断信号传递到EX级，如果EX级发现后面两个流水级中存在出现异常的指令，便将数据ram的使能置0，防止误写数据破坏了机器的状态。

除此之外，如果“错取”的指令包含div类型指令，那么该指令在清空各流水级时需要将除法器复位，以防止下次处理除法指令时出错。

1. Exp 14
2. 类SRAM接口的定义

对于指令RAM与数据RAM，CPU分别通过以下的一套接口连接：

CPU =>RAM

req：访存请求信号

wr：读写控制（1为写，0为读）

size：请求传输的字节数（00为1字节，01为2字节，10为4字节）

wstrb：写使能信号，4位分别对应4字节是否有效。

addr：请求地址

wdata：写数据

RAM=>CPU

addr\_ok：地址传输握手完成

data\_ok：数据传输握手完成

rdata：返回读数据

1. 类SRAM接口握手流程

1）读

发送读请求：req拉高，wr设为0，读地址输入addr，size按照字节数赋值。

地址握手：SRAM将addr\_ok拉高，表示接收到读地址。

数据握手：SRAM将data\_ok拉高，同时返回读数据。

2）写

发送写请求：req拉高，wr设为1，写地址输入addr，写数据输入wdata，size与wstrb对应赋值。

地址握手：SRAM将addr\_ok拉高，表示接收到写地址与数据。

数据握手：SRAM将data\_ok拉高，表示数据写入完成。

（3）CPU对SRAM访问的处理

共有3种情况需要处理：指令RAM读（取指），数据RAM读（load指令），数据RAM写（store指令）。

1. 取指

在Pre-IF阶段发送读请求，等待地址握手。

地址握手完成（addr\_ok & req）后进入IF阶段等待返回指令，同时下一个PC进入Pre-IF阶段发送读请求。

数据握手完成（data\_ok）收到指令后进入ID阶段译码。

因为对指令RAM没有写操作，因此wr，wstrb与wdata恒为0。

1. load指令

在EX阶段发送读请求，等待地址握手。

地址握手完成（addr & req）后进入MEM阶段等待返回读数据。

数据握手完成（data\_ok）收到读数据后进入WB阶段写回寄存器堆。

1. store指令

在EX阶段发送写请求，等待地址握手。

地址握手完成（addr & req）后进入MEM阶段等待。

数据握手完成（data\_ok）完成写入后再进入WB阶段。

若不在数据握手完成前在MEM阶段进行阻塞，可能出现load指令紧接store指令，load指令收到store指令的data\_ok并误认为是自己的data\_ok的情况。

1. Exp 15

（1）增加控制状态寄存器ECFG、BADV、TID、TCFG、TVAL、TICLR，修改ESTAT

* 1. 重要模块1设计：IF流水级
     1. 工作原理

在本次实验中，IF流水级完成类SRAM接口，随机延迟的取指设计。

假想的Pre-IF流水级发送取指请求并等待地址握手。完成地址握手后进入IF流水级等待数据握手收到指令。

* + 1. 接口定义

表2 IF流水级接口定义

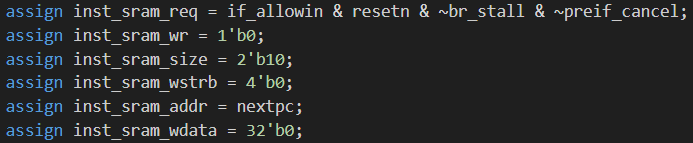
| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| resetn | IN | 1 | 复位信号 |
| inst\_sram\_req | OUT | 1 | 指令ram请求信号 |
| inst\_sram\_wr | OUT | 1 | 指令ram读写控制信号 |
| inst\_sram\_size | OUT | 2 | 指令ram请求字节数 |
| inst\_sram\_wstrb | OUT | 4 | 指令ram写使能信号 |
| inst\_sram\_addr | OUT | 32 | 指令ram请求地址 |
| inst\_sram\_wdata | OUT | 32 | 指令ram写数据 |
| inst\_sram\_addr\_ok | IN | 1 | 指令ram地址握手信号 |
| inst\_sram\_data\_ok | IN | 1 | 指令ram数据握手信号 |
| inst\_sram\_rdata | IN | 32 | 指令ram读数据 |
| axi\_arid | IN | 4 |  |
| id\_allowin | IN | 1 | ID流水级是否允许IF流水级传入数据 |
| br\_taken | IN | 1 | ID流水级传来的跳转信号 |
| br\_stall | IN | 1 | ID流水级传来的跳转阻塞信号 |
| br\_target | IN | 32 | ID流水级传来的跳转地址 |
| if\_to\_id\_valid | OUT | 1 | 标记IF流水级向ID流水级传递的数据是否有效 |
| if\_to\_id\_data | OUT | 64 | IF模块向ID模块传递的数据 |
| if\_to\_id\_excep | OUT | 1 | IF模块向ID模块传递的异常信息 |
| wb\_to\_if\_csr\_data | IN | 66 | WB模块传给IF模块的csr数据 |
| if\_flush | IN | 1 | 传给IF模块的清空流水线信号 |

图4 IF模块接口

* + 1. 功能描述

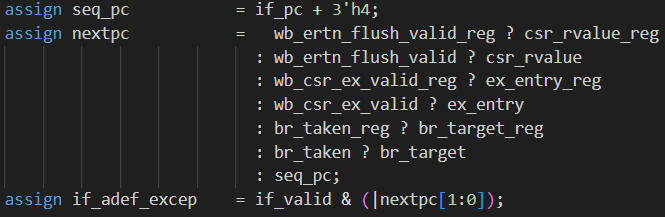
1. 接口赋值

面向指令RAM的类SRAM接口赋值如下：



由于只进行读访问，wr、size和wdata恒置0。指令长度始终为32位，size恒为2’b10。访问地址为预测的nextpc。访问请求在非复位，IF流水级allowin，ID阶段没有被阻塞的条件分支指令，且Pre-IF流水级内容未被取消的情况下发出。

其中nextpc当前赋值逻辑如下：



优先级从高到低为：ertn指令返回 > 出现例外进入例外处理入口 > 分支指令跳转到目标地址 > pc+4。

由于ertn\_valid、ex\_valid与br\_taken信号仅拉高一个时钟周期，分别对前三种情况的valid信号与nextpc值用寄存器进行缓存，优先从寄存器中取缓存的值。

1. 流水级控制信号设计
2. Pre-IF流水级

ready\_go信号：req & addr\_ok握手成功。

to\_if\_valid信号：Pre-IF的ready\_go与IF的allowin握手成功，且没有被取消或刷新。

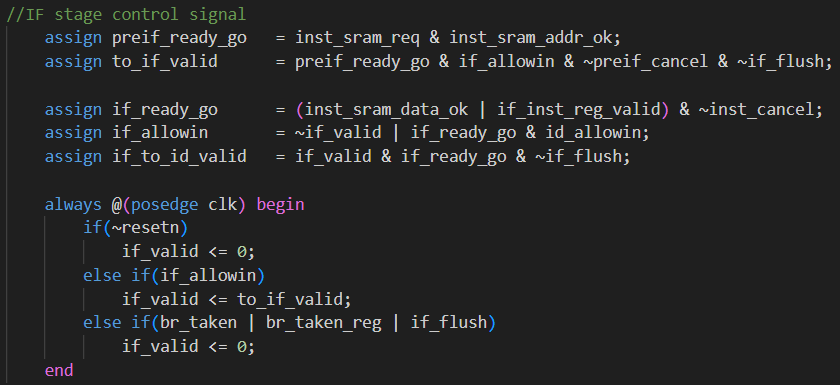
1. IF流水级

ready\_go信号：收到指令（data\_ok或指令缓存有效）且没有被取消。

allowin信号：在IF级内容无效或已经与ID级握手完成时允许进入。

valid信号：在allowin时根据to\_if\_valid赋值，在分支成立或刷新时无效。

if\_to\_id\_valid信号：IF级内容有效，ready\_go拉高，且没有被刷新。



1. 取消设计
   1. 重要模块3设计：ID流水级
      1. 工作原理

增加br\_stall信号，发送给IF流水级。用途为在ID级条件分支指令因为写后读冲突被阻塞时，暂停IF级向指令RAM发送取指请求。其余不变。

* + 1. 接口定义

表3 ID流水级接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| resetn | IN | 1 | 复位信号 |
| id\_allowin | OUT | 1 | ID流水级允许IF流水级传入数据 |
| br\_taken | OUT | 1 | 传给IF流水级的跳转信号 |
| br\_stall | OUT | 1 | 传给IF流水级的跳转阻塞信号 |
| br\_target | OUT | 32 | 传给IF流水级的跳转地址 |
| if\_to\_id\_valid | IN | 1 | 标记IF流水级传入ID流水级的数据是否有效 |
| if\_to\_id\_data | IN | 64 | IF流水级传给ID流水级的数据 |
| if\_to\_id\_excep | IN | 1 | IF流水级传给ID流水级的异常信息 |
| ex\_allowin | IN | 1 | EX流水级允许ID流水级传入数据 |
| id\_to\_ex\_data | OUT | 163 | ID流水级传入EX流水级的数据 |
| id\_to\_ex\_excep | OUT | 86 | ID流水级传入EX流水级的异常信息 |
| id\_to\_ex\_valid | OUT | 1 | 标记ID流水级传入EX流水级的数据是否有效 |
| wb\_rf\_zip | IN | 38 | WB流水级向ID流水级传递的regfile的写回信息和前递数据 |
| mem\_rf\_zip | IN | 39 | MEM流水级前递到ID流水级的数据 |
| ex\_rf\_zip | IN | 40 | EX流水级前递到ID流水级的数据 |
| id\_flush | IN | 1 | ID模块收到的清空流水线信号 |
| has\_int | IN | 1 | 若WB模块判断有中断，将ID阶段的指令进行标记 |

图7 ID流水级接口

* + 1. 功能描述

br\_stall信号逻辑如下，在条件分支指令因为写后读冲突被阻塞时拉高：

br_stall

功能为暂停取值请求的发送，等待得到条件分支指令的判断结果后再决定是否取消当前已发送取值请求进入IF级的指令。

* 1. 重要模块4设计：EX流水级
     1. 工作原理

EX阶段访问数据RAM的接口改变为类SRAM接口。

* + 1. 接口定义

表4 EX流水级接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| resetn | IN | 1 | 复位信号 |
| ex\_allowin | OUT | 1 | EX模块允许ID模块传入数据 |
| id\_to\_ex\_data | IN | 163 | ID模块传入EX模块的数据 |
| id\_to\_ex\_excep | IN | 86 | ID模块传入EX模块的异常信息 |
| id\_to\_ex\_valid | IN | 1 | 标记ID模块传入EX模块的数据是否有效 |
| mem\_allowin | IN | 1 | MEM模块允许EX模块传入数据 |
| ex\_to\_mem\_data | OUT | 78 | EX模块传入MEM模块的数据 |
| ex\_to\_mem\_excep | OUT | 87 | EX模块传入MEM模块的异常信息 |
| ex\_to\_mem\_valid | OUT | 1 | 标记EX模块传入MEM模块的数据是否有效 |
| ex\_rf\_zip | OUT | 39 | EX模块前递到ID模块的数据 |
| mul\_result | OUT | 64 | 乘法器得到的结果 |
| data\_sram\_req | OUT | 1 | 数据ram请求信号 |
| data\_sram\_wr | OUT | 1 | 数据ram读写控制信号 |
| data\_sram\_size | OUT | 2 | 数据ram请求字节数 |
| data\_sram\_wstrb | OUT | 4 | 数据ram写使能信号 |
| data\_sram\_addr | OUT | 32 | 数据ram请求地址 |
| data\_sram\_wdata | OUT | 32 | 数据ram写数据 |
| data\_sram\_addr\_ok | IN | 1 | 数据ram地址握手信号 |
| ex\_flush | IN | 1 | 清空流水线信号 |
| mem\_to\_ex\_excep | IN | 1 | MEM模块向EX前传的异常信息 |

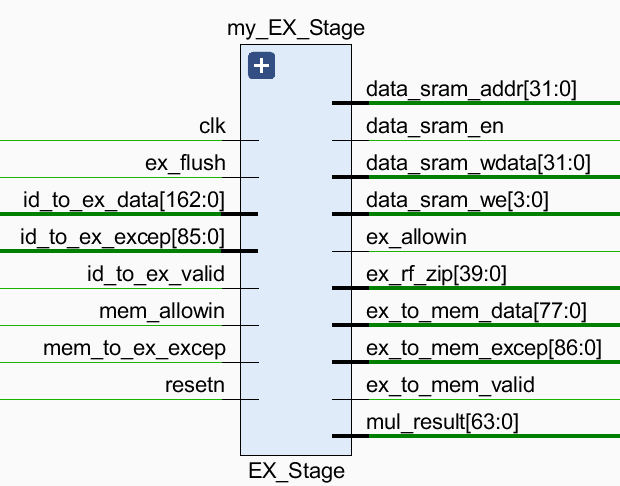
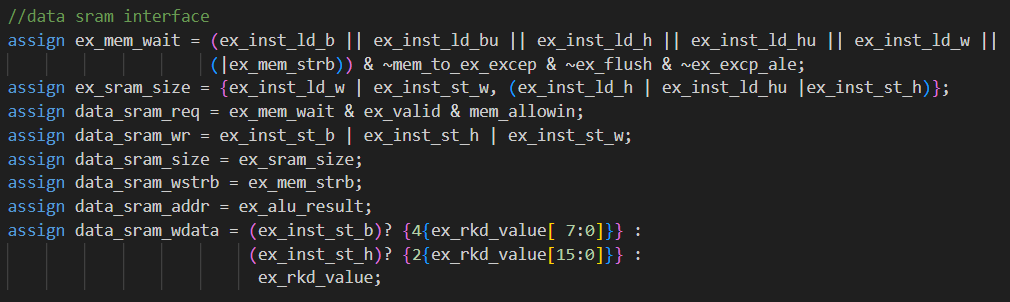


图8 EX流水级接口

* + 1. 功能描述

面向数据RAM的类SRAM接口赋值如下：



Wait信号表示当前存在访存请求，赋值逻辑为存在访存指令且无例外，未被刷新。

Req信号在存在访存请求，EX流水级有效，且MEM阶段允许进入时拉高。

Wr信号在当前为store指令时拉高。

Size信号根据访存指令的访问位数进行赋值。

Strb信号不变。

Addr为alu的计算结果。

Wdata逻辑不变。

* 1. 重要模块5设计：MEM流水级
     1. 工作原理

接收数据ram返回的读出数据以及乘法器的计算结果，根据控制信号生成除读取csr寄存器之外的写回数据。

* + 1. 接口定义

表5 MEM流水级接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| resetn | IN | 1 | 复位信号 |
| mem\_allowin | OUT | 1 | MEM模块允许EX模块传入数据 |
| ex\_to\_mem\_data | IN | 78 | EX模块传入MEM模块的数据 |
| ex\_to\_mem\_excep | IN | 87 | EX模块传入MEM模块的异常或中断数据 |
| ex\_to\_mem\_valid | IN | 1 | 标记EX模块传入MEM模块的数据是否有效 |
| wb\_allowin | IN | 1 | WB模块允许MEM模块传入数据 |
| mem\_to\_wb\_data | OUT | 70 | MEM模块传入WB模块的数据 |
| mem\_to\_wb\_excep | OUT | 119 | MEM模块传入WB模块的异常或中断数据 |
| mem\_to\_wb\_valid | OUT | 1 | 标记MEM模块传入WB模块的数据是否有效 |
| data\_sram\_data\_ok | IN | 1 | 数据ram数据握手信号 |
| data\_sram\_rdata | IN | 32 | 数据ram读数据 |
| mul\_result | IN | 64 | 乘法器运算结果 |
| mem\_rf\_zip | OUT | 38 | MEM模块前递到ID模块的数据 |
| mem\_reflush | IN | 1 | 清空流水级信号 |
| mem\_to\_ex\_excep | OUT | 1 | MEM级向EX级传递异常或中断信号 |

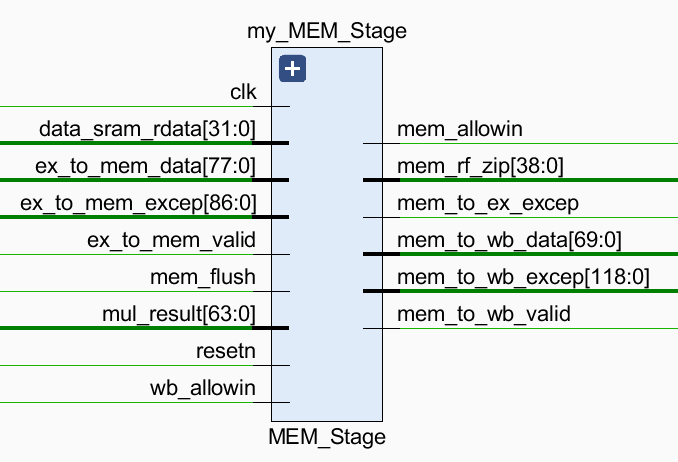


图13 MEM流水级接口

* + 1. 功能描述

根据EX模块传递来的数据执行相应的访存操作。并将访存指令结果、写回控制信号、PC等信息在下一个时钟上升沿传递给WB模块。

本实验中，MEM级利用从EX级传来的异常信号生成mem\_to\_ex\_excep传递到EX流水级，并把EX传来的异常与中断信息和信息传递到WB流水级。其中mem\_to\_ex\_excep的作用是告知EX级前面有指令出现异常或者中断，防止EX流水级的store指令将数据写入数据ram中，从而实现精确异常。此外，MEM级会根据从WB级传来的flush信号进行流水线的清空。

* 1. 重要模块6设计：WB流水级
     1. 工作原理

WB模块收到传入的异常信息数据。若有异常产生，则将flush信号拉高并向前面的模块传递，从而清空流水线。根据异常类型为csr\_ecode和csr\_esubcode赋值，令它们为各类型异常对应的值。WB模块中例化csr模块，将异常处理所需的数据传入csr模块进行处理。

* + 1. 接口定义

表6 WB流水级接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| resetn | IN | 1 | 复位信号 |
| wb\_allowin | OUT | 1 | WB模块允许MEM模块传入数据 |
| mem\_to\_wb\_data | IN | 70 | MEM模块传入WB模块的数据 |
| mem\_to\_wb\_excep | IN | 70 | MEM模块传入WB模块的异常信息 |
| mem\_to\_wb\_valid | IN | 1 | 标记MEM模块传入WB模块的数据是否有效 |
| debug\_wb\_pc | OUT | 32 | 写回指令PC值（用于debug） |
| debug\_wb\_rf\_we | OUT | 4 | 写回指令写使能（用于debug） |
| debug\_wb\_rf\_wnum | OUT | 5 | 写回指令写地址（用于debug） |
| debug\_wb\_rf\_wdata | OUT | 32 | 写回指令写数据（用于debug） |
| wb\_rf\_zip | OUT | 38 | WB模块向ID模块传递的regfile写回信息和前递数据 |
| wb\_to\_if\_csr\_data | OUT | 66 | WB模块传给IF模块的csr数据 |
| wb\_flush | OUT | 1 | WB模块输出的清空流水线信号 |
| has\_int | OUT | 1 | 将ID阶段的指令标记为中断 |

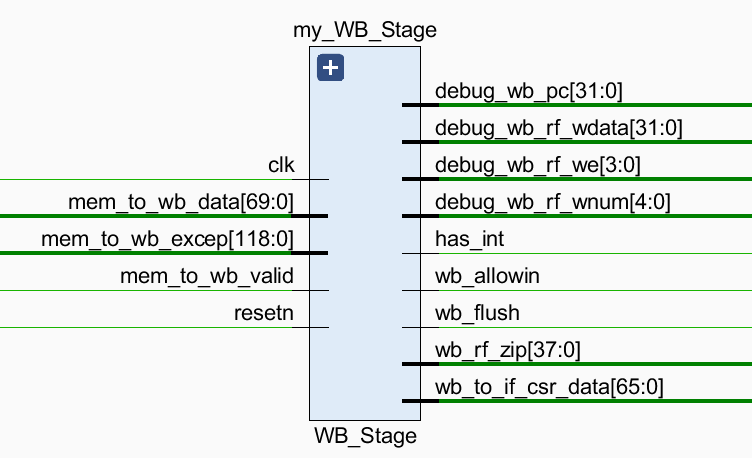


图14 WB流水级接口

* + 1. 功能描述

当WB模块识别到ertn指令或异常时，需要拉高flush信号，向前面的模块传递进行相关处理，从而清空流水线。

WB模块还会根据前面模块传递过来的异常类型信息为csr\_code和csr\_ecode赋值。在该部分实验中，所有异常类型的csr\_ecode均为0。根据指令集手册，对于该部分实验，当同时存在多个异常时，中断的优先级高于异常，中断的优先级最高。对于异常，取指阶段检测出的优先级最高，译码阶段检测出的优先级次之，执行阶段检测出的优先级再次之，因此使用如下代码逻辑进行赋值。该部分实验不涉及取值和执行阶段同时检测出多种异常的情况，译码阶段虽然会同时检测出多种异常，但其判断条件互斥。赋值代码如下：

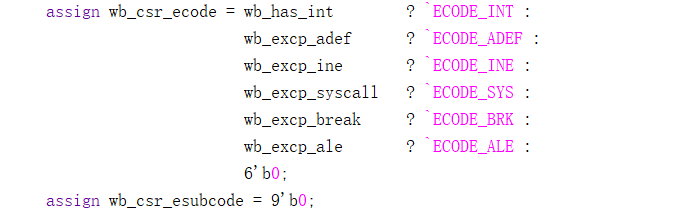


图15 csr\_code和csr\_ecode赋值

当异常到来时，不能向寄存器写入东西。其实大部分异常在译码阶段就会将寄存器堆写使能拉低而不存在这个问题。原因主要在于当指令不存在时，根据译码逻辑寄存器堆写使能此时可能被错误地拉高，因此需要保证异常时写使能为0，相关代码如下：

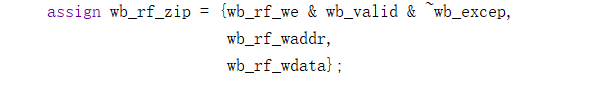


图16 寄存器堆相关信号赋值

1. 实验过程
   1. 实验流水账

贾城昊

2023年10月25日 8：00-10：00 阅读讲义

2023年10月25日 15：20-17：30 完成exp12的CSR模块

2023年10月25日 19：00-22：00 完成exp12的流水线阻塞与控制逻辑，完善exp12的新增的数据通路

2023年10月25日 22：00-次日0：00对exp12进行debug

2023年 11月 1日 10：00-11：40 完成exp13的CSR模块

2023年11月 1日 17：30-19：00 完成exp13对inst\_rdcntvh和inst\_rdcntvl指令的处理

2023年11月 1日 20：00-次日0：00 完成exp13的debug

李金明

2023年10月26日 8：00-10：00 阅读讲义

2023年10月26日 14：00-18：00 完成exp12的debug

2023年11月 1日 14：00-16：00 增加exp13对异常支持的部分

2023年11月 1日 16：00-19：00 增加exp13对中断支持的部分

2023年11月 1日 20：00-次日1：00完成exp13的debug

牛浩宇

2023年10月25日晚实现exp12的译码逻辑与流水级间的数据通路。

之后实验任务被上面两个人抢完了，被迫摸鱼。

* 1. 错误记录
     1. 错误1：exp12中WB模块给IF模块的csr数据位宽未定义
        1. 错误现象

运行仿真后，console报错如下：

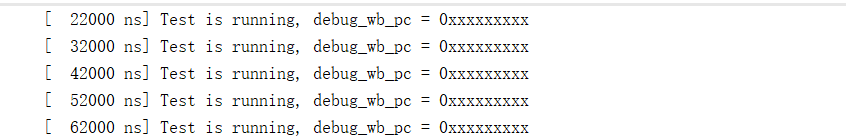


图17 错误1 对应的Console报错

* + - 1. 分析定位过程

可见从一开始进行仿真，PC更新便出错，于是对波形图进行查看，发现IF流水级接受到的从WB流水级传回的csr数据有异常，如下所示：

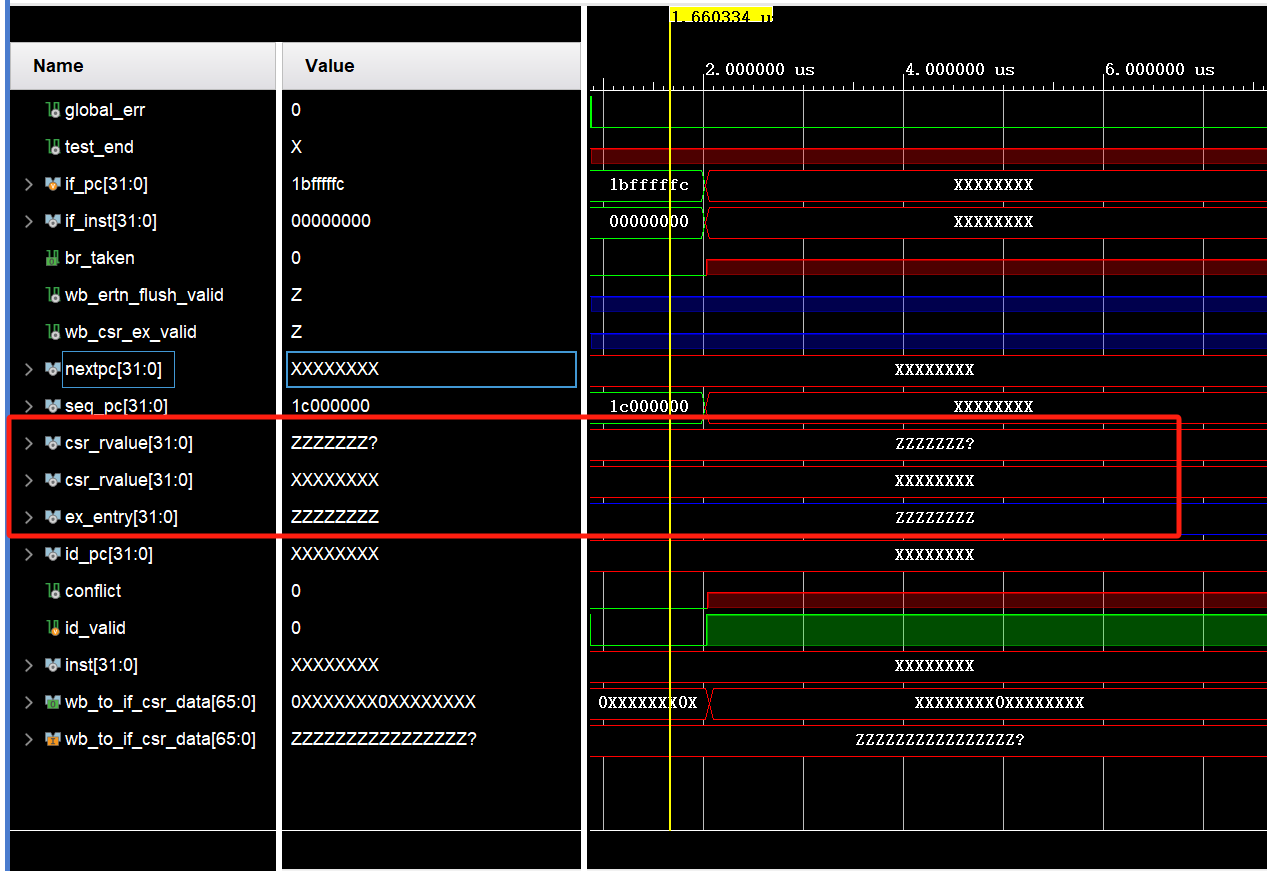


图18 错误1中IF接受到的从WB传回的csr数据有异常

上图第一个csr\_rvalue是csr模块里的值，第二个csr\_rvalue是IF译码得到的值，可以发现两者有区别，且IF译码出的csr\_rvalue的高31位全为高阻态，于是本人很容易想到，可能是位宽定义的问题，查看mycpu\_top.v，果然wb\_to\_if\_csr\_data的位宽没有定义，导致其默认为1。

* + - 1. 错误原因

mycpu\_top.v中wb\_to\_if\_csr\_data的位宽没有定义，导致其默认为1，IF接受到的从WB传回的csr数据有异常，进而导致PC的跳转出现问题。

* + - 1. 修正效果

对mycpu\_top.v中wb\_to\_if\_csr\_data的位宽进行定义即可：

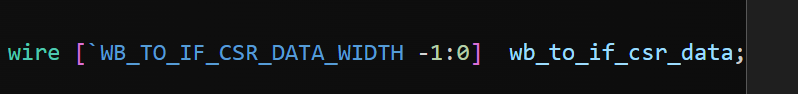


图19 错误1修改代码

该方法有效，来到下一个bug。

* + 1. 错误2：exp12 EX模块与MEM模块的异常或中断数据通路位宽出错
       1. 错误现象

运行仿真后，console报错如下：

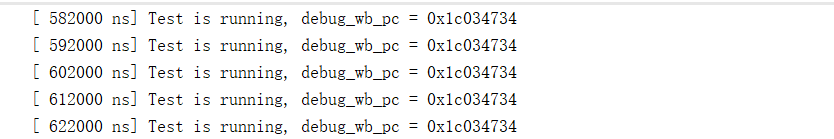


图20 错误2对应的Console报错

* + - 1. 分析定位过程

可见PC更新还是出错，于是对波形图进行查看，发现可能是因为debug\_wb\_rf\_we为X导致的问题，如下所示：



图21 错误2中debug\_wb\_rf\_we为X

于是进一步查看debug\_wb\_rf\_we为X的原因，发现是id\_to\_ex\_valid的问题，进一步查看原因，发现是ID阶段判断conflic信号的问题，如下所示：

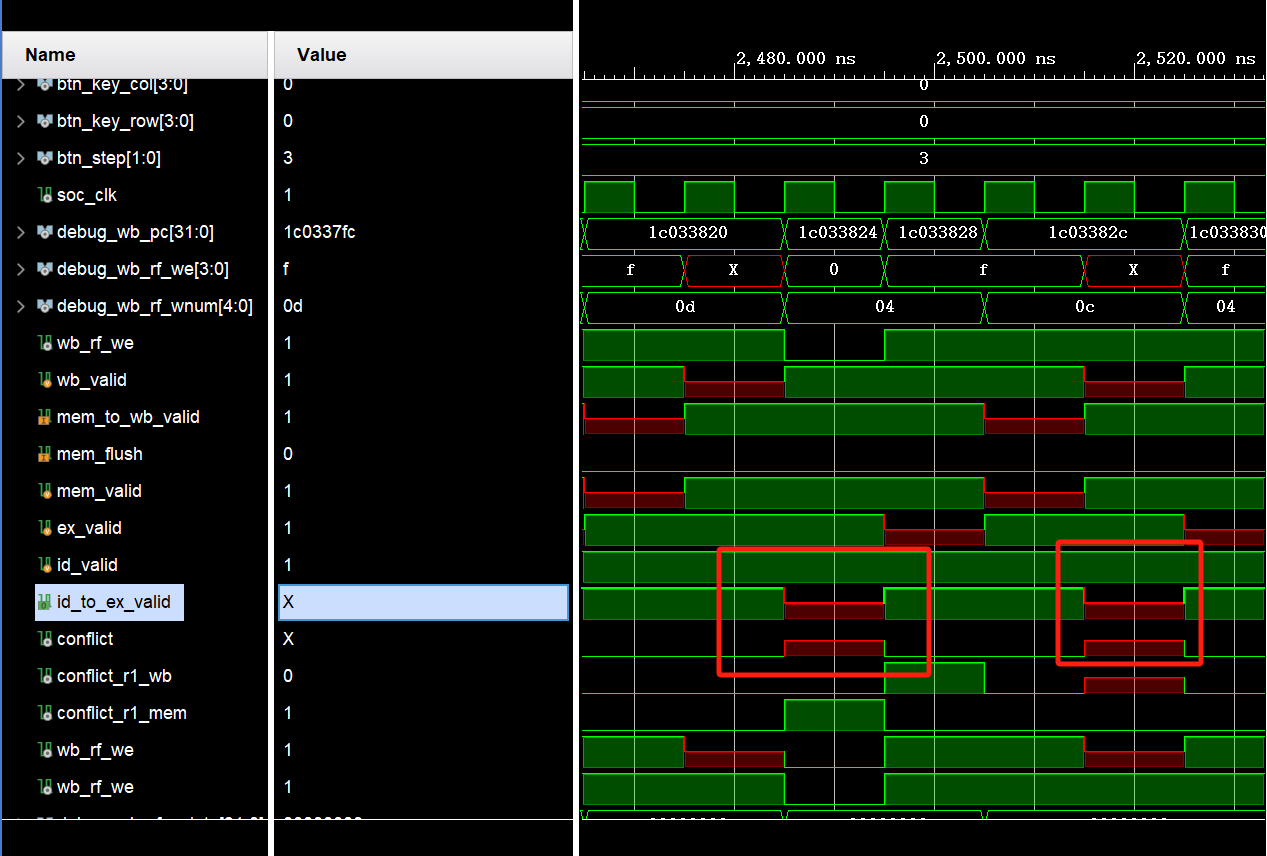


图22 错误2中ID模块的conflict为X

ID模块的conflict的赋值逻辑如下：

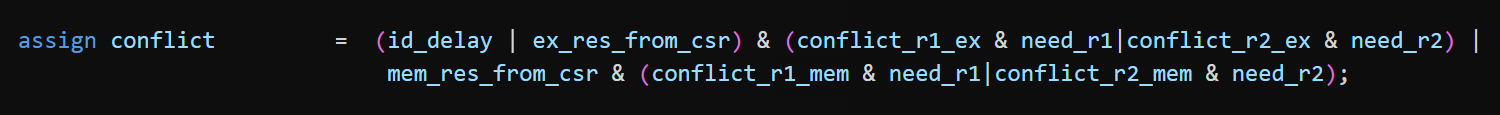


图23 错误2中ID模块的conflict的赋值逻辑

于是进一步查看与其相关的信号，发现是MEM传给ID的mem\_res\_from\_csr信号的问题：

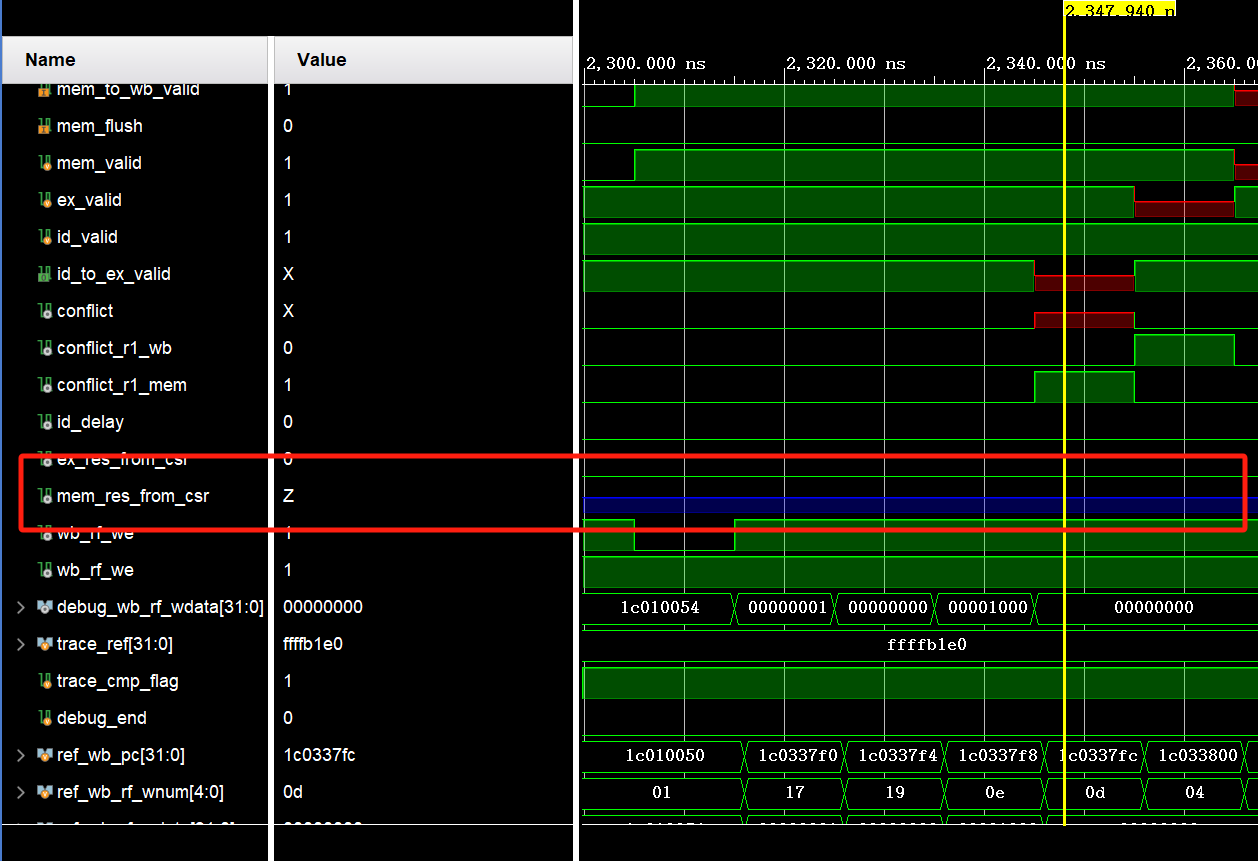


图24 错误2中ID模块的mem\_res\_from\_csr信号为Z

最后查看其赋值逻辑，发现是因为EX模块与MEM模块的异常或中断数据通路位宽少了一位，导致其解析出来的信号为Z

* + - 1. 错误原因

EX模块与MEM模块的异常或中断数据通路位宽少了一位，导致其解析出来的mem\_res\_from\_csr信号为Z，导致MEM传给ID的mem\_res\_from\_csr信号出错，进而导致id\_to\_ex\_valid为X，传递到WB流水级后debug\_wb\_rf\_we为X，导致出错。

* + - 1. 修正效果

修正EX模块与MEM模块的异常或中断数据通路位宽即可



图25 错误2修改代码

该方法有效，来到下一个bug。

* + 1. 错误3：exp12中Input/Output搞反
       1. 错误现象

仿真可以通过，但是上板无法通过。上板后发现数码管没有任何累加。

* + - 1. 分析定位过程

阅读讲义，查找“仿真通过，上板不过”的可能原因。一开始以为时时序过差导致的，但询问同学后发现有时序更差但能正常通过的，于是判断是否是其他原因。后来考虑到多驱动的可能性，打开综合报告，发现有critical\_warning，提示EX\_stage中的ex\_rf\_zip有多驱动问题：

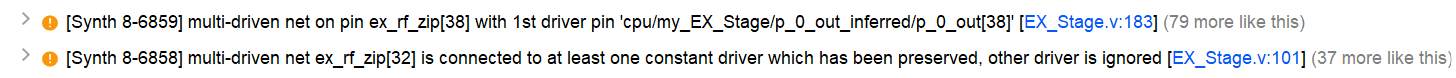


图26 错误3 Tcl\_console的critical\_warning

查看相关信号：

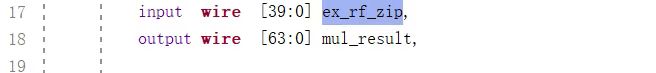


图27 ex\_rf\_zip信号的定义

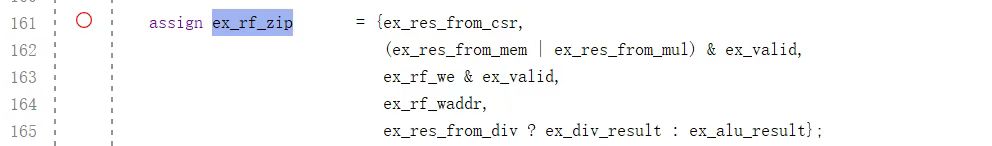


图28 ex\_rf\_zip信号的赋值

* + - 1. 错误原因

在EX模块中，ex\_rf\_zip应是输出信号，其代表寄存器相关信息，在EX模块中也赋了值，但在定义信号接口时将其定义为了INPUT接口，导致出现仿真能过，上板却过不了的问题。

* + - 1. 修正效果

修改为



图29 错误3修改后代码

该方法有效，exp12仿真和上板均通过。

* + 1. 错误4：exp13中优化数据通路出错
       1. 错误现象

运行仿真后，console报错如下所示：：

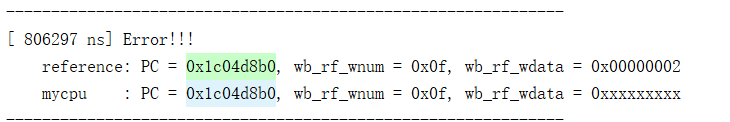


图 30 错误4对应的Console报错

* + - 1. 分析定位过程

首先通过查看反汇编代码，发现对应的指令为div指令，接着查看波形图，发现是MEM流水级的rf\_wdata出错，进一步查看原因发现是因为rf\_wdata的选择逻辑出错，具体波形图如下所示：

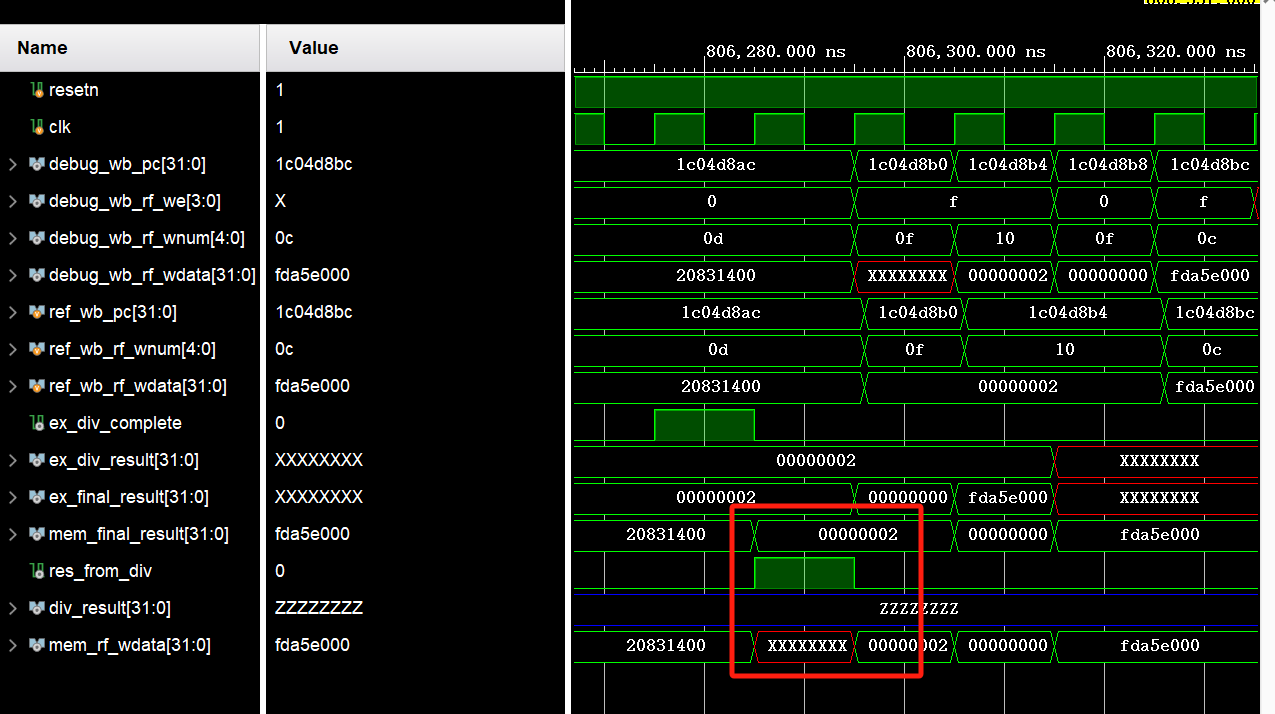


图31 错误4中alu\_src全为高阻态

可以看到，这里MEM流水级多了一个全为Z的div\_result，而mem\_rf\_wdata没有选择正确的mem\_final\_result，而是div\_result。这是因为本次实验对数据通路进行了优化，在EX流水级对div\_result进行了选择，而不是将信号传递给在MEM流水级再进行选择，但修改过程中遗漏了对mem\_rf\_wdata的赋值逻辑的修改导致出错，出错前代码如下：

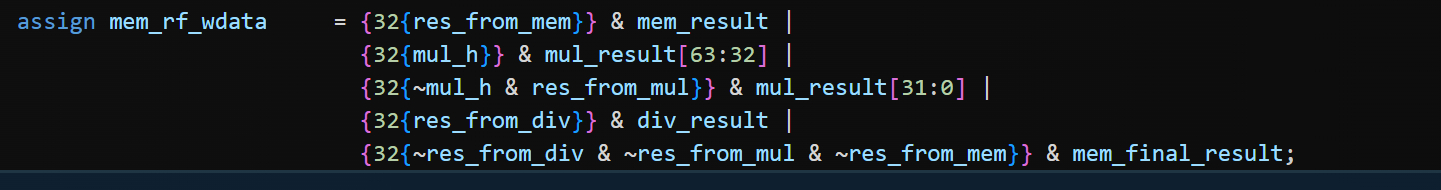


图32 错误4修改前代码

* + - 1. 错误原因

本次实验对数据通路进行了优化，在EX流水级对div\_result进行了选择，而不是将信号传递给在MEM流水级再进行选择，但修改过程中遗漏了对mem\_rf\_wdata的赋值逻辑的修改导致出错

* + - 1. 修正效果

对mem\_rf\_wdata的赋值逻辑进行修改即可，修改后代码如下：

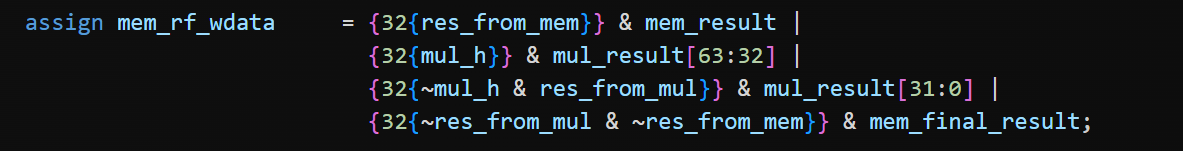


图33 错误4改后代码

* + 1. 错误5：exp13 CSR模块中TICLR寄存器实现出错
       1. 错误现象

Console报错如下：

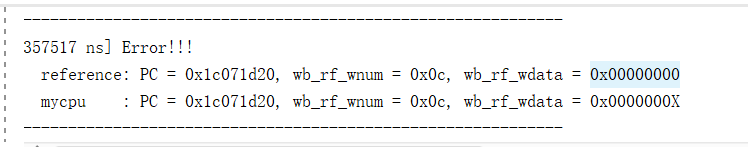


图34 错误5对应的Console报错

* + - 1. 分析定位过程

首先查看反汇编，发现是csrwr指令，接着查看波形图相关信号，发现csr\_rvalue出错，然后通过csr\_num，发现是TICLR寄存器的问题，如下所示：



图35 错误5 的csr\_ticlr\_rvalue出错

于是查看csr\_ticlr\_rvalue的赋值逻辑与指令集手册，发现ticlr寄存器读出值需要全为0，但本人csr模块中的赋值代码如下：



图36 错误5 的csr\_ticlr\_rvalue赋值逻辑

进一步发现是本人没有对csr\_ticlr\_clr信号进行赋值，导致出错。

（注：本人设计是用一个1bit信号csr\_ticlr\_clr来实现该TICLR寄存器，并始终赋值为0，如果要读则在高位拼接上31bit的0。而至于清除时钟中断，只需要捕捉对寄存器写1的动作，不需要真的写入1）

* + - 1. 错误原因

实现TICLR寄存器时，遗漏了对csr\_ticlr\_clr信号的赋值，导致csrwr指令出错。

* + - 1. 修正效果

添加对csr\_ticlr\_clr的赋值即可，如下所示：



图37 错误5修改后代码

* + 1. 错误6：exp13中ECFG寄存器赋值出错
       1. 错误现象

Console报错如下：

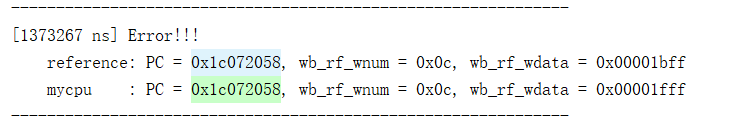


图38 错误6对应的Console报错

* + - 1. 分析定位过程

发现wb\_rf\_wdata错误，由于已经跑了很久的仿真，所以猜想是对新加的指令的处理出错。于是本人首先查看了反汇编代码，发现此时是csrwr指令：



图39 错误6对应的指令

根据金标准与输出信号比对的原理，即当写入寄存器时进行比对，可知上一次往0x4对应的寄存器写入时，寄存器内容不应该存在问题，于是我们将目光对准0x4对应的csr寄存器。查阅指令集手册可知其对应的是ECFG寄存器，查看ECFG寄存器写入的代码：

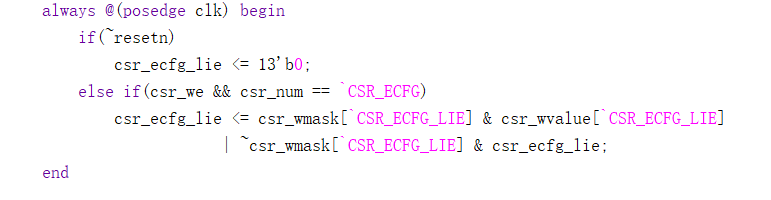


图40 ECFG寄存器复赋值代码

* + - 1. 错误原因

根据指令集手册，ECFG寄存器的第10位需要一直为0且不允许通过软件修改，ECFG的赋值有错。

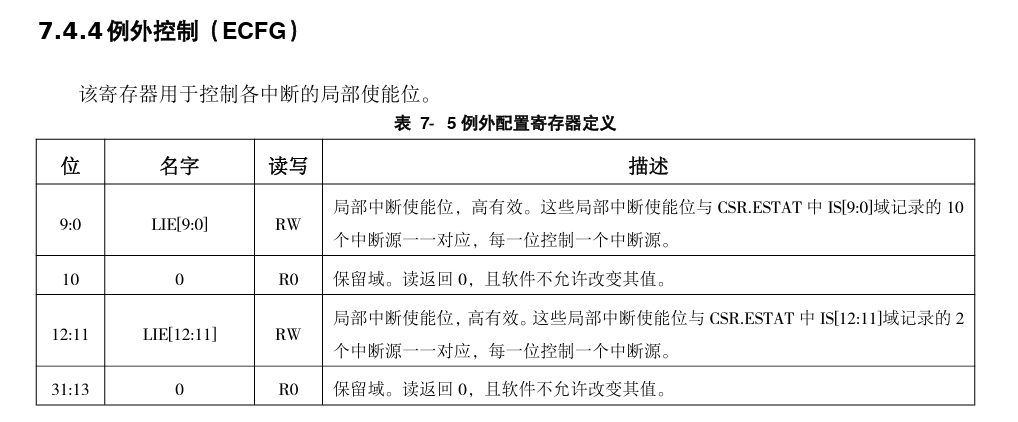


图41 指令集手册关于ECFG寄存器的介绍

* + - 1. 修正效果

将ECFG赋值逻辑进行如下修改即可：

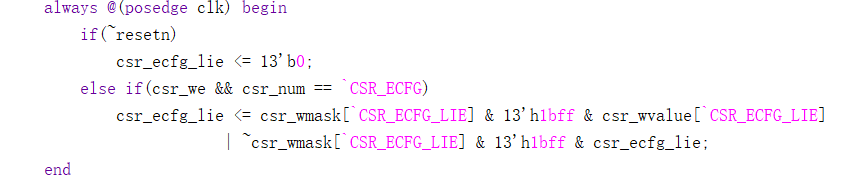


图42 错误6修改后代码

* + 1. 错误7：exp13中处理错误指令时错误拉高寄存器堆写使能
       1. 错误现象

Console报错如下：

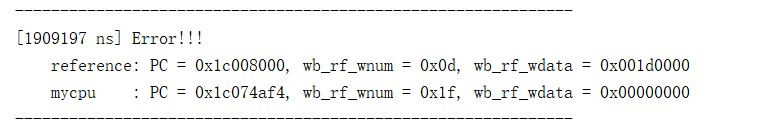


图 43 错误7对应的Console报错

* + - 1. 分析定位过程

查看汇编代码，得到报错位置的是一条错误指令：

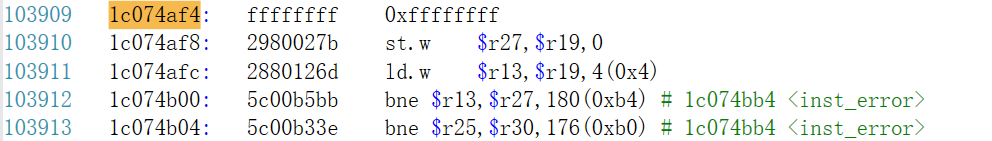


图 44 错误7对应的指令

作为错误指令，其不应当向寄存器堆写入任何内容，但其拉高了寄存器堆写使能，导致与金标准进行了比对从而出错。

* + - 1. 错误原因

WB流水级没有考虑选择出现异常时，寄存器堆写使能不应该拉高，不能向寄存器中写入数据。

* + - 1. 修正效果

修改WB模块中关于寄存器堆写使能的赋值，使其在异常时不拉高：

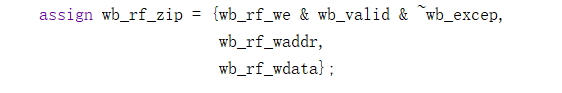


图45 错误7修改后代码（1）

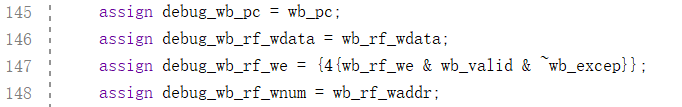


图46 错误7修改后代码（2）

* + 1. 错误8：exp13中出现异常除法器没有复位
       1. 错误现象

Console报错如下：

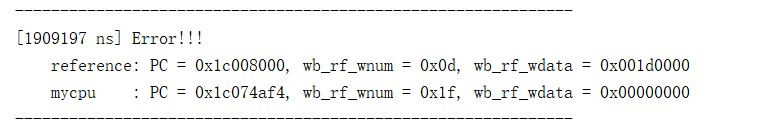


图 47 错误8对应的Console报错

* + - 1. 分析定位过程

通过查阅汇编指令可知报错位置对应的指令时除法指令，查看对该指令的计算过程对应波形图：

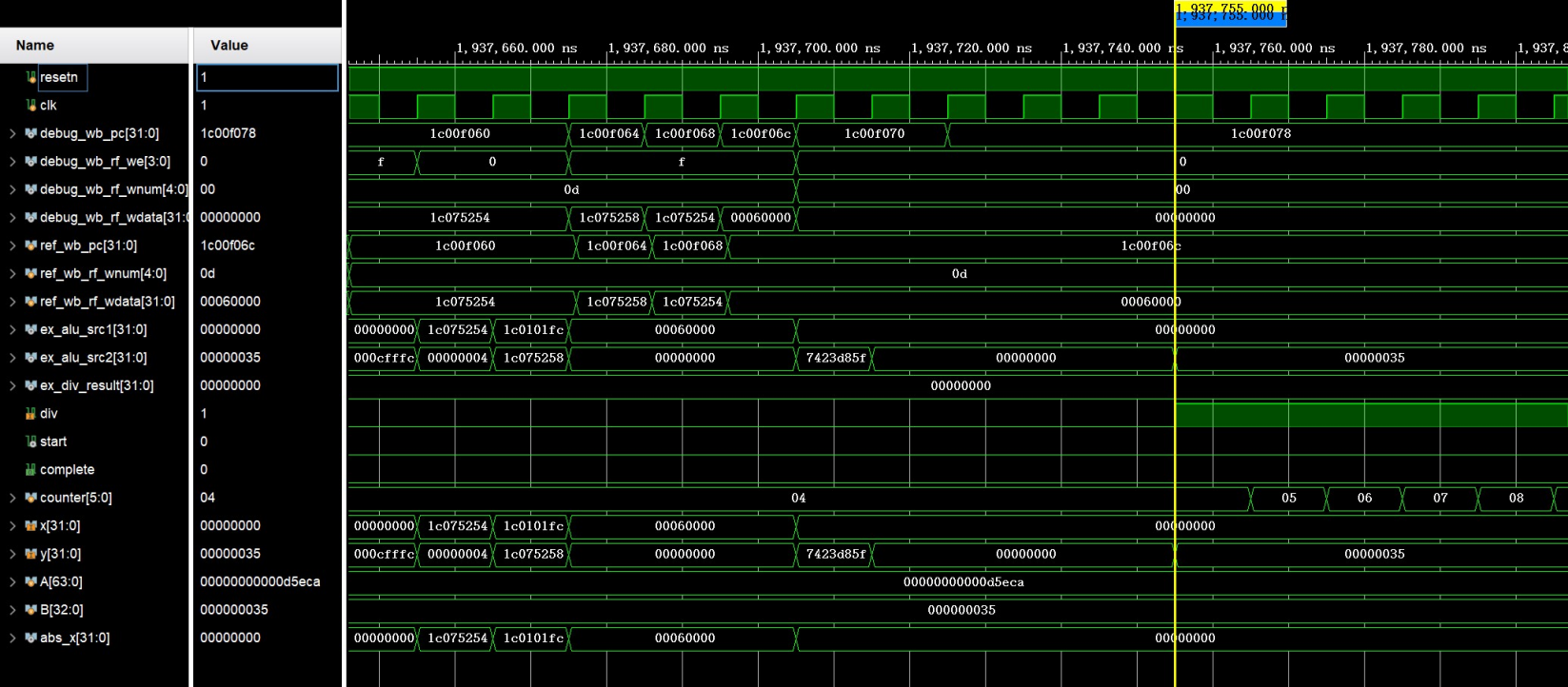


图 48 错误8处理除法时的波形图

可以看到当div信号传入时，除法器中的计数器值并不为0，说明当出现异常时，除法器并没有进行复位，但由于操作数改变，所以会出错。查看除法器上一次进行除法操作的相关波形图：



图 49 错误9处理上一次除法时的波形图

可以看到除法器工作执行到中间却被打断了，这是出现异常清空流水线导致的。

* + - 1. 错误原因

出现异常时除法器没有进行复位导致下一次再处理除法时出错。

* + - 1. 修正效果

修改EX模块连接到除法器的复位信号，使得除法器在清空流水线时也能够复位：

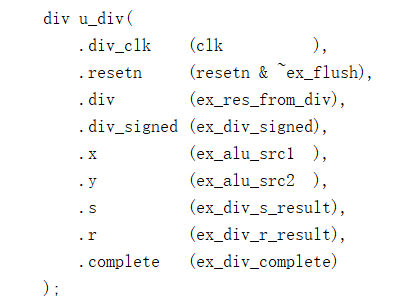


图 50 错误9修改后

修改有效，来到下一个bug。

* + 1. 错误9：exp13中rdcntid指令相关控制逻辑出错
       1. 错误现象

Console报错如下：

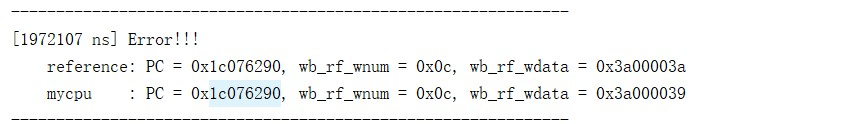


图 51 错误8对应的Console报错

* + - 1. 分析定位过程

查看报错位置的汇编指令，可知错出现位置时inst\_error部分，但这一部分并不能看出来有什么错。于是根据波形图向前追溯并与金标准进行比对：

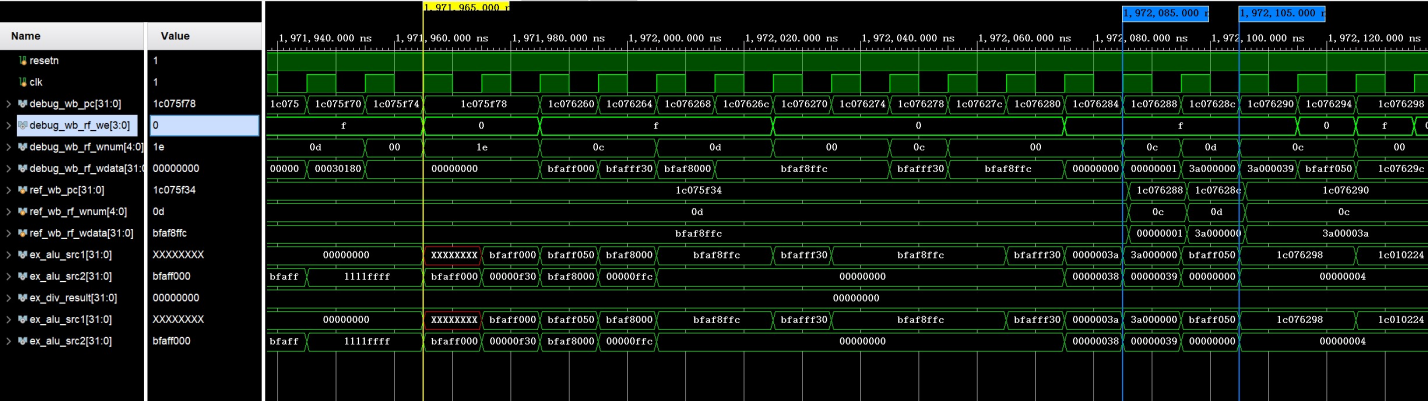


图 52 错误9处理除法时的波形图

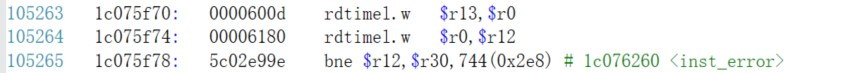


图 53 计时器相关指令部分的汇编指令

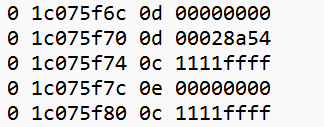


图 54 计时器相关指令部分的金标准

可以看到，在处理rdcntid指令后，指令错误跳转向了inst\_error，且根据金标准当处理地址为0x1c075f74这条指令时，指令应当向12号寄存器写入0x1111ffff，而此时不光写入寄存器号没有正常赋值，写入寄存器数据也不对。这是两个bug，我们分别来处理。

对于写入寄存器数据错误的问题，由于前面处理异常的部分已经通过，认为csr模块错误的概率不大。查看ID模块译码时为csr寄存器号赋值相关代码：



图 55 ID模块译码时为csr寄存器号赋值

可以发现并未考虑指令是计时器相关指令的情况：当指令是计时器相关指令时，应当从TID寄存器读取数据。

对于写入通用寄存器目的地址错误的问题，查看ID模块为写入通用寄存器目的地址赋值的代码：



图 56 ID模块译码时为写入通用寄存器目的地址赋值

对于伪指令rdcntid，其指令原型为rdtimel.w zero, rj。这条指令和以往指令不同：写入通用寄存器目的地址既不为31，也不为rd，而是rj，需要特别考虑。

* + - 1. 错误原因

在处理rdcntid指令时，写入寄存器号没有正确赋值，写入寄存器数据也没有正确赋值，导致指令没有立马报错，而是跳转到了inst\_error才报错。

* + - 1. 修正效果

将以上两处修改为：

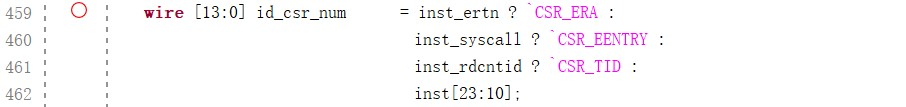


图 57 修改后ID模块为csr寄存器号的代码



图 58 修改后ID模块为写入通用寄存器目的地址赋值的代码

这两处修改后有效，修正后仿真通过，上板验证也通过。

1. 实验总结

本实验实现了控制状态寄存器、异常处理、中断处理，实现了精确异常功能，并增加了相关指令，为中断和异常处理建立了新的数据通路，修正了除法器模块，使其实现中断后能够正常运行。

该部分实验理解后实现起来较为简单直接，且debug的过程并不复杂（只要bug不多的话），一般通过波形图都能较快地定位到问题所在并进行修复。但是在exp12中遇到了一次“仿真通过，上板不过”的情况，后来发现是多驱动问题。这一过程使我们了解了再次面对这种问题的处理方法。

通过本次实验，本小组成员还进一步熟悉了git的使用，对如何使用git进行合作有了更深的理解，在这次编写代码的过程中，没有产生冲突、将别人修改的代码改回去等的情况。