**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

Lab6的整体要求是使我们理解异常和中断的软硬件协调处理机制，理解精确异常的概念和处理方法，并掌握在流水线CPU中添加异常和中断支持的方法。

实验12要求所设计的CPU能够支持简单的异常和中断处理。需要增加csr控制状态寄存器CRMD、PRMD、ESTAT、ERA、EENTRY和SAVE0~3；实现对csr的三个读写命令csrrd、csrwr和csrxchg；实现中断返回指令ertn和系统调用指令syscall。

实验13要求在实验12的基础上增加对取指地址错（ADEF）、地址非对齐（ALE）、断点（BRK）和指令不存在（INE）这几种异常处理的支持；增加对2个软中断、8个硬件中断和定时器中断的支持；在csr中增加控制状态寄存器ECFG、BADV、TID、TCFG、TVAL和TICLR。

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EXE级，如果EXE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 12
2. 增加控制状态寄存器CRMD、PRMD、ESTAT、ERA、EENTRY和SAVE0~3

在本实验中，只有触发例外或中断，以及遇到csrrd、csrwr和csrxchg指令才会对控制状态寄存器中所存数据进行读取和修改，因此在csr模块中设计控制状态寄存器的读写逻辑可以从这两方面进行考虑。对于要增加的状态寄存器，参考指令集手册和讲义，各控制状态寄存器对应的读写条件如下所述：

CRMD（当前模式信息）：该寄存器的PLV域刻画了CPU当前所处的特权级。指令手册定义在复位时PLV域被置为0，表示最高特权级，当触发例外时该域也被置为0，从而保证trap后机器会处于最高特权级。该寄存器的IE域表示全局中断使能，发生例外时该域被置为0，从而让trap后屏蔽中断。此外，执行ertn指令从中断处理程序返回时，需要将PRMD寄存器中的PPLV和PIE域分别写入PLV和IE域。因此例外信号和ertn指令信号需要作为csr模块的输入。

PRMD（例外前模式信息）：触发例外时，PRMD寄存器中的PPLV和PIE域被赋值成CRMD寄存器的PLV和IE域中的值。

ESTAT（例外状态）：在本实验中只考虑该寄存器IS域的后两位需要被csr读写指令进行读取和修改。此外，在发生例外时，该寄存器的ecode和esubcode域会被赋值成此异常的类型代号，因此异常的类型代号需要作为csr模块的输入。

ERA（例外返回地址）：当触发例外时，将触发例外的指令的PC值，即当前WB级的PC值保存在该寄存器中，因此wb\_pc需要作为csr模块的输入。

EENTRY（例外入口地址）：该寄存器的后6位恒为0，前26位可被csr读写指令进行读取和修改。

SAVE0~3（数据保存）：数据保存寄存器可以存放通用寄存器中的数据，只能由csr读写指令进行读取和修改。

1. 增加csrrd、csrwr和csrxchg指令

csrrd、csrwr和csrxchg指令用于软件访问csr模块，需要通过num来指定被操作的控制状态寄存器。其中，num可以从指令类型（如ertn指令）或者指令本身来定义。由于csrwr和csrxchg指令需要用到寄存器rj和rd中的数据，因此在取数时需要考虑数据相关的冲突。

此外，这三条指令都需要写rd寄存器，会与后续指令产生数据相关，流水线需要阻塞。这三条csr读写指令的写数据只能在WB级访问csr寄存器后才能获得，所以后续指令如果与这些指令产生了数据相关，需要阻塞一拍或者两拍，才能从WB级前递的数据中读出需要的数据。阻塞逻辑的更改：在EXE级和MEM级增加前递到ID级的res\_from\_csr信号，表示在EXE或MEM级中的指令需要对csr进行读写。如果ID级发现上述两个res\_from\_csr信号只要有一个为1，就需要将自己的ready\_go置0，从而实现阻塞，如下所示：

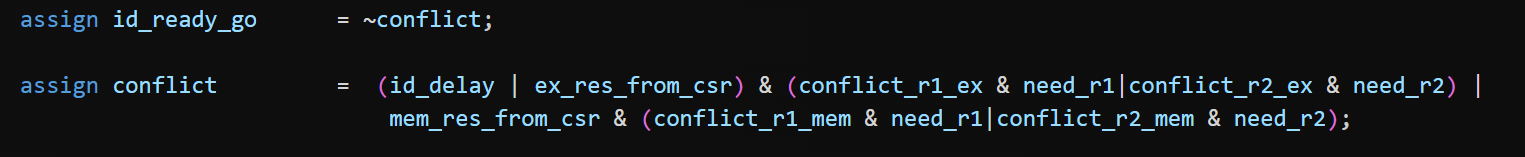


图1 exp12对ID的阻塞逻辑

1. 增加ertn指令和syscall指令

如果当前执行的指令是ertn指令，那么WB流水级会从csr模块中读出ERA的值，并传递给IF流水级更新nextpc，在下一时钟周期跳转到返回地址继续执行。同时，WB流水级要拉高向前四个流水级发出的flush信号，从而清空流水线。

如果当前执行的指令是syscall指令，ID级在进行译码时会触发中断并产生例外编码，中断信号顺着流水线传递到WB级时，WB级会把产生例外的PC存入控制状态寄存器中的ERA寄存器作为中断处理完毕后的返回地址，同时根据例外编码eecode和esubcode从EENTRY寄存器中读出中断处理程序的入口地址，并传递给IF级更新nextpc，在下一时钟周期跳转到中断处理程序开始执行。此外，WB级也要拉高flush信号来清空流水线，从而实现精确异常。

清空流水线的具体操作是根据WB级传递的flush信号，将每一级的allow\_in信号置1，同时将每一流水级向下一级传递的xx\_to\_yy\_valid信号置0(即“等效”把各级的valid标志置0）。以EXE级为例，代码实现如下：

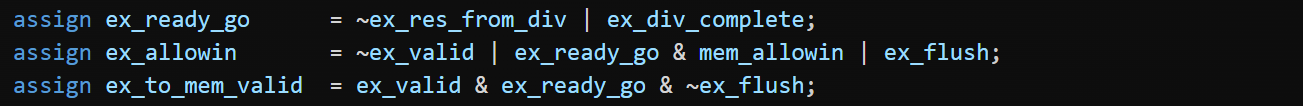


图2 exp12对流水线的清空操作

2、Exp 13

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

在本实验中，需要修改寄存器的值的情况与exp12中类似，包括触发例外或中断，以及执行csrrd、csrwr和csrxchg指令。结合上述条件，并参考指令集手册和讲义，得到新增的控制状态寄存器的读写条件如下：

ECFG（例外配置）：ECFG的第0~9位和第11~12位均为局部中断使能位LIE，每位控制一个中断源，中断源与ESTAT的对应位一一对应；其余位为保留位。对于LIE域，其读写由CSR指令实现，因此只需对此添加相关支持；其余保留位不允许写入，读出的值为0，因此直接用0填充即可。

BADV（出错虚地址）：本实验中的BADV共32位，用于存储触发地址相关例外时的虚地址。在本实验中相关的例外只有ADEF和ALE。如果是ADEF，则把wb\_pc写入BADV；如果是ALE，则把访存地址传到wb级并写入BADV。

TID（定时器编号）：TID寄存器存储定时器编号，需要支持CSR指令读写。本实验中可由rdcntid指令读出。

TCFG（定时器配置）：软件配置定时器的接口，从低到高依次为定时器使能位、循环模式控制位、初始值、只读位。需要支持CSR指令的读写，并用对应位控制定时器TVAL。

TVAL（定时器值）：TVAL存储定时器的值，初值为TCFG中的InitVal域，在TCFG的En域为1时递减。减到0后，如果TCFG的Periodic域为1，则从初始值开始重新倒计时，否则保持不变。

TICLR（定时中断清除）：从该寄存器中读出的值始终为0，通过对该寄存器最低位写1的动作来清除时钟中断标记。因此只需要用一个1bit信号csr\_ticlr\_clr来实现该寄存器，并始终赋值为0，如果要读则在高位拼接上31bit的0。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。

另外，exp12中只考虑了ESTAT的最低2位，本实验中需要将ecode和subcode写入对应域，并给硬件中断、软件中断、时钟中断的中断状态位赋值，其中时钟中断状态位在倒计时到0时拉高。

（2）增加异常支持ADEF、ALE、BRK、INE

ADEF：PC值后两位非0时发生异常。在pre-IF级对异常进行判定，如果存在异常则把adef\_excep拉高，并逐级传递下去，直到在WB级拉高flush来刷掉前面的流水级，并跳转到异常处理程序入口，这部分操作可以复用exp12中的通路。此外，还要在WB级把出错的PC存入BADV，并把ecode和subcode存入ESTAT的对应域。

ALE：访存地址未对齐时发生ALE异常。ALE在EXE级检测，如果发现异常则把ale\_excep拉高，并把出错的访存地址传到WB级写入BADV，其余的异常处理操作和ADEF使用相同的通路。

BRK：如果在译码级发现当前的指令是break，则拉高inst\_break，并逐级传递到WB级，在WB级把ecode和subcode存入ESTAT的对应域，然后通过拉高flush来刷新流水级，并跳转到异常处理程序。break指令是新增的指令，需要在ID级添加对应的译码逻辑。

INE：如果在译码级发现指令不属于任何一条已经实现的指令，则拉高ine\_excep，其余操作同BRK，包括在WB级保存ecode、subcode，刷新流水级，跳转到异常处理程序等。

（3）增加中断支持

本实验需要支持硬件中断、软件中断与时钟中断。软件中断由软件控制，通过对ECFG的使能位写入1来实现，只需支持CSR指令读写。时钟中断在TVAL倒计时到0且TCFG使能拉高时触发，中断发生时将ESTAT中对应的状态位拉高，通过对TICLR最低位位写1的动作来将该状态位清0。硬件中断则是通过hw\_int\_in信号传入csr模块，把ESTAT的对应IS域值拉高即可。

如果发生以上任何一种中断，且没有被屏蔽中断（CRMD的IE域为1），将csr模块的has\_int信号拉高，传到ID级处理。只要发生中断或异常，对应流水级的对应excep信号就会拉高，并逐级传递到WB级处理。

（4）新增计时器相关指令rdcntvl.w、rdcntvh.w、rdcntid

在新增的三天计时器相关指令中，rdcntvl.w和rdcntvh.w读取一个64位计数器的低32位和高32位。这个计数器**并不是时钟中断中的倒计时计数器，而是另一个正向计数的计数器**。本组成员将这个计数器设置在EXE级，其初值为0，每拍加一。指令在EXE级得到结果后在WB级写回寄存器。如果发生数据相关，可以将EXE级的结果前递到ID级，这部分通过复用已有的数据通路实现。

rdcntid读取TID控制状态寄存器中的内容，可以复用exp12中三条CSR指令的通路。由于它在写回级才能拿到从TID中读出的数据，因此一旦它后面的指令和它发生数据相关，需要阻塞后面的指令，直到它在写回级读数据。这部分也通过复用已有CSR指令的数据通路实现。

* 1. 重要模块1设计：CSR模块
     1. 工作原理

在写回级处理异常，将信号传入CSR模块。如果发生异常，就把异常码和出错的地址写入对应寄存器，并把异常处理程序的入口地址传给pre-IIF级。如果写回级的是CSR读写指令，就对csr\_num指明的寄存器进行对应操作。如果是ertn指令，就把保存的地址传给pre-IF级。此外，还要在发生中断时拉高has\_int信号。。

* + 1. 接口定义

表1 CSR模块接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clock | IN | 1 | 时钟输入 |
| resetn | IN | 1 | 复位信号 |
| csr\_num | IN | 14 | 控制状态寄存器读写地址 |
| csr\_we | IN | 1 | 控制状态寄存器写使能 |
| csr\_wmask | IN | 32 | 控制状态寄存器写掩码 |
| csr\_wvalue | IN | 32 | 控制状态寄存器写数据 |
| ertn\_flush | IN | 1 | ertn指令刷新信号 |
| wb\_ex | IN | 1 | 例外发生信号 |
| wb\_ecode | IN | 6 | 例外发生ecode代码 |
| wb\_esubcode | IN | 9 | 例外发生esubcode代码 |
| wb\_pc | IN | 32 | WB级PC值，用于记录例外发生的PC |
| csr\_rvalue | OUT | 32 | 控制状态寄存器读数据 |
| ex\_entry | OUT | 32 | 例外跳转入口地址 |
| has\_int | OUT | 1 | 发生中断信号 |
| ipi\_int\_in | IN | 1 | 核间中断输入 |
| coreid\_in | IN | 32 | 核编号，也是TID初值 |
| hw\_int\_in | IN | 8 | 硬件中断输入 |
| wb\_vaddr | IN | 32 | 出错地址 |

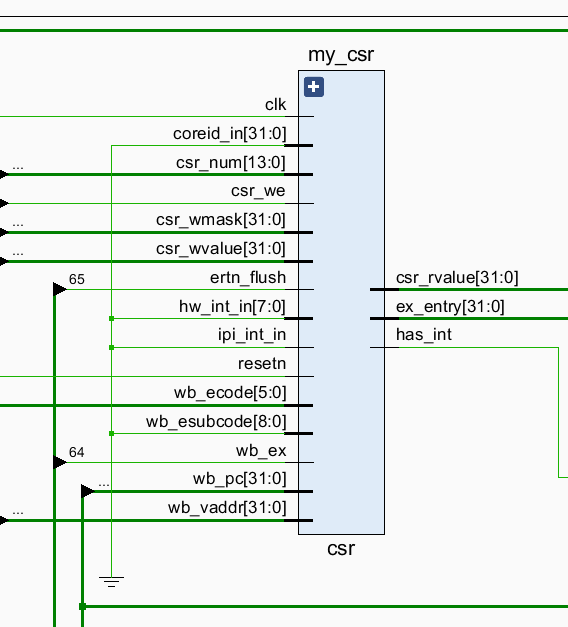


图3 CSR模块接口

* + 1. 功能描述

csr模块包含控制状态寄存器CRMD、PRMD、ESTAT、ERA、EENTRY、SAVE0~3、ECFG、BADV、TID、TCFG、TVAL、TICLR。每个模块的每个域分别用一个寄存器实现，在读出时再拼接为完整的寄存器，通过多路选择器赋值给csr\_rvalue。每个域在wmask中对应的bit，以及每个寄存器的csr\_num，都用宏在csr.h中定义。

各寄存器的含义与功能以及其更新逻辑已在总体设计思路中阐明，在此不再赘述了。

* 1. 重要模块2设计：IF流水级
     1. 工作原理

在exp13中，IF流水级可以判断的异常类型为ADEF（取指地址错）。

* + 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\_en | OUT | 1 | 指令ram片选信号 |
| inst\_sram\_we | OUT | 4 | 指令ram写使能 |
| inst\_sram\_addr | OUT | 32 | 指令ram地址信号 |
| inst\_sram\_wdata | OUT | 32 | 指令ram写数据 |
| inst\_sram\_rdata | IN | 32 | 指令ram读数据 |
| 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\_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 |

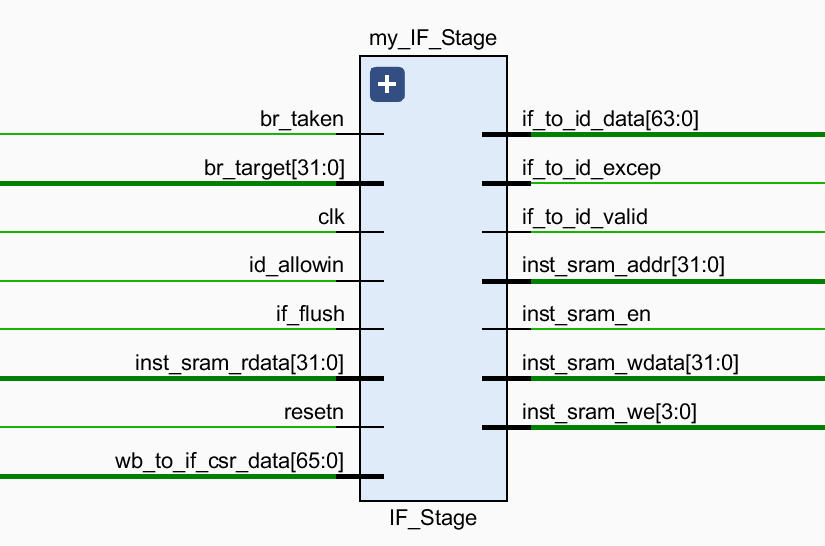


图4 IF模块接口

* + 1. 功能描述

在exp13中，若取指地址最低两位不为0，则认为取指地址错。此时判断异常类型为ADEF，还需要写入错误PC，而PC已经通过之前的数据通路进行传递，因此不需要额外实现。

* 1. 重要模块3设计：ID流水级
     1. 工作原理

为新增加的指令添加译码信号，将它们传给EXE阶段。修改与alu、立即数有关的信号，实现算术逻辑运算指令。对于转移指令则需要判断是否进行跳转，并计算跳转地址。

在exp13中，可以在ID流水级判断的一异常信息有BRK（断点）和INE（指令不存在）。

* + 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\_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 | EXE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EXE流水级的数据 |
| id\_to\_ex\_excep | OUT | 86 | ID流水级传入EXE流水级的异常信息 |
| id\_to\_ex\_valid | OUT | 1 | 标记ID流水级传入EXE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 | EXE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 |

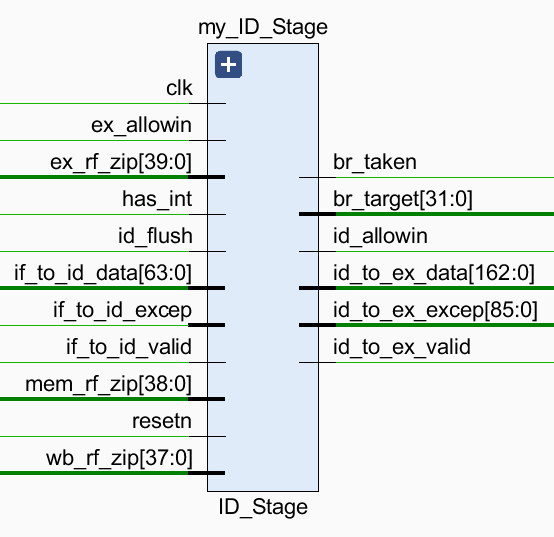


图5 ID流水级接口

* + 1. 功能描述

在exp13中，若译码出break指令，则判断异常信息为BRK，将异常类型向后传递；若发现指令不属于现在支持指令的任何一种，则判断异常信息为INE，并将异常类型向后传递。

在ID阶段得到csrrd、csrwr、csrxchg、ertn、syscall（exp12）与break、rdcntid、rdcntvl、rdcntvh（exp13）指令的译码信号，并由此对WB阶段接入CSR模块的csr\_num、csr\_we、csr\_wmask、csr\_wvalue、ertn\_flush信号进行赋值并向后传递。同时向后传递的还有判断寄存器堆wdata是否来自csr的res\_from\_csr信号，与ADEF、INE、SYSCALL、BREAK四种异常类型，以及WB阶段传回的has\_int信号（表示发生中断，本流水级内的内容无效）。

此外，为了译码逻辑的简洁性，将指令分为以下几类：calculate\_register、calculate\_immediate、load、store、unconditional\_branch、conditional\_branch、exception、others，分别产生一个类信号，用于对大量同类指令信号相或的逻辑进行简化。

* 1. 重要模块4设计：EX流水级
     1. 工作原理

EX阶段从ID阶段取得有效操作数和操作码后，进行算术逻辑运算，以及在EX流水级向数据ram发送读写请求，读写地址以及写数据。

在本次实验中，EXE流水级需要根据WB流水级级传回来的flush信号刷新流水线。根据从MEM流水级传回的mem\_to\_ex\_excep拉低data\_sram\_en，以避免在异常或etrn指令后的指令向内存中写入数据。

在exp13中，考虑到对中断的支持，在清空流水线时，需要保证乘法器和除法器也被复位。对于乘法器来说，其无论何时传入操作数，下一拍均能获得结果，无需更改；对于除法器来说，需要更改传入它的复位信号，从而保证中断后除法器可以正常运行。

在exp13中，可以在EX阶段判断的异常类型为ALE（地址非对齐）。

* + 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 | EXE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\_en | OUT | 1 | 数据ram片选信号 |
| data\_sram\_we | OUT | 4 | 数据ram写使能 |
| data\_sram\_addr | OUT | 32 | 数据ram地址信号 |
| data\_sram\_wdata | OUT | 32 | 数据ram写数据 |
| 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 |

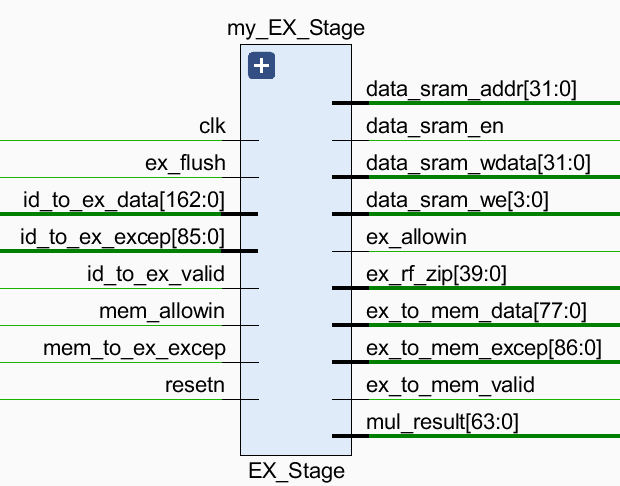


图6 EX流水级接口

* + 1. 功能描述

在ex\_flush拉高时将ex\_to\_mem\_valid置为0，同时，ex\_allowin也需要拉高，从而等效清空EXE流水级。

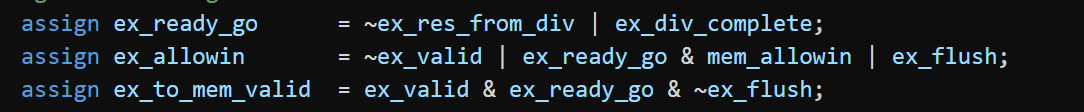


图 7 EX流水级清空流水线操作

如果当前EX级中指令的前一条指令异常，那么需要阻止EX级中的指令向数据ram发出写信号。为此，将mem\_to\_ex\_excep从MEM级传到EX级。MEM级中的指令正是EXE级中指令的前一条指令，如果MEM级中是ertn或异常指令，并且MEM级中的数据有效，则将mem\_to\_ex\_excep拉高，data\_sram\_en会随之变为0。

要将data\_sram\_en置0的还有一种情况——EX级指令会发出写请求，并且存在异常，这种情况在本实验中只可能是store指令发生ALE异常，因此在data\_sram\_en中加入& ~ex\_excp\_ale。

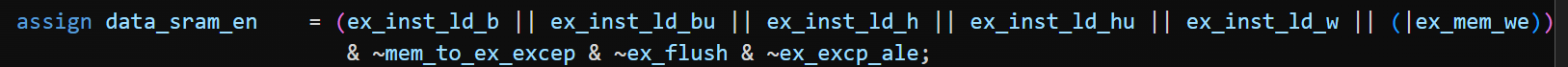


图 8 EX流水级对data\_sram\_en的处理

在EX级还需要完成load和store指令的访存地址异常检测。如果读写对象是word，则地址后两位必须为全0；如果读写对象是half word，则地址末尾必须是0。如果不是0，则拉高ex\_excp\_ale信号，表示存在异常。对于ALE异常，除了需要传递异常类型为ALE外，还需要传递虚地址。考虑到虚地址即alu\_result，且这一信号已经在ex\_to\_mem\_data中包含了，因此不在ex\_to\_mem\_excep中包括。后续在MEM阶段中，alu\_result将被替换掉，届时会将这个数据包含在mem\_to\_wb\_excep中。这一过程比较简单，后文不再提及。

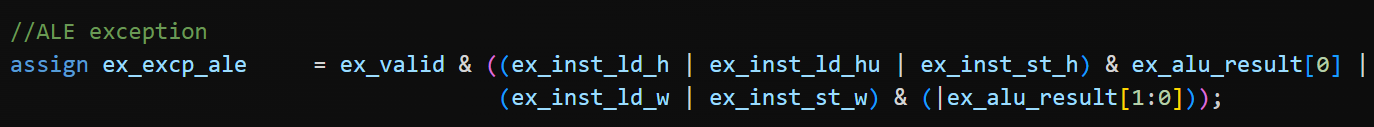


图 9 EX流水级对ALE异常的判断

除此之外，针对inst\_rdcntvh和inst\_rdcntvl指令，在EXE流水级新添一个计数器counter，若为inst\_rdcntvh，取counter的高32位，若为inst\_rdcntvl，取counter的低32位，如下所示：

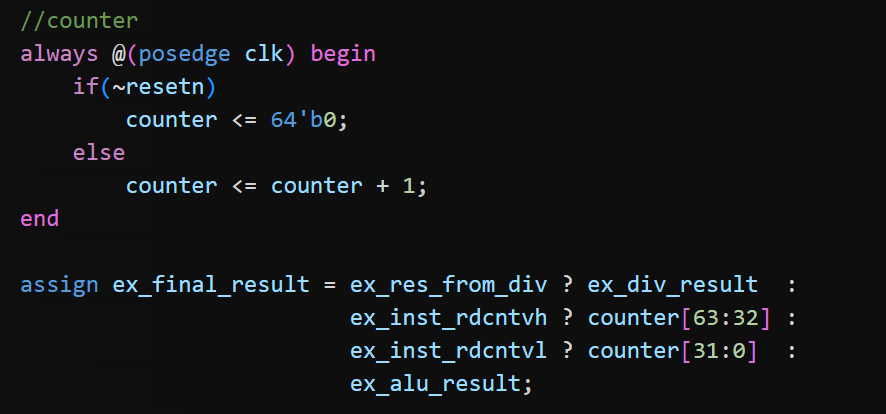


图10 对inst\_rdcntvh和inst\_rdcntvl指令的处理

在连例化除法器时，考虑到对中断的支持，将连接到除法器的复位信号更改为resetn & ~ex\_flush，即当有复位信号或需要清空流水线时，将除法器复位。

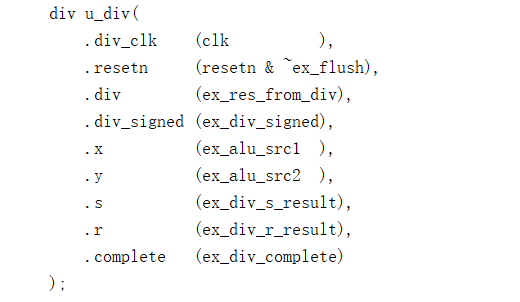


图11除法器的例化

* 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EXE模块传入数据 |
| ex\_to\_mem\_data | IN | 78 | EXE模块传入MEM模块的数据 |
| ex\_to\_mem\_excep | IN | 87 | EXE模块传入MEM模块的异常或中断数据 |
| ex\_to\_mem\_valid | IN | 1 | 标记EXE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\_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EXE级传递异常或中断信号 |

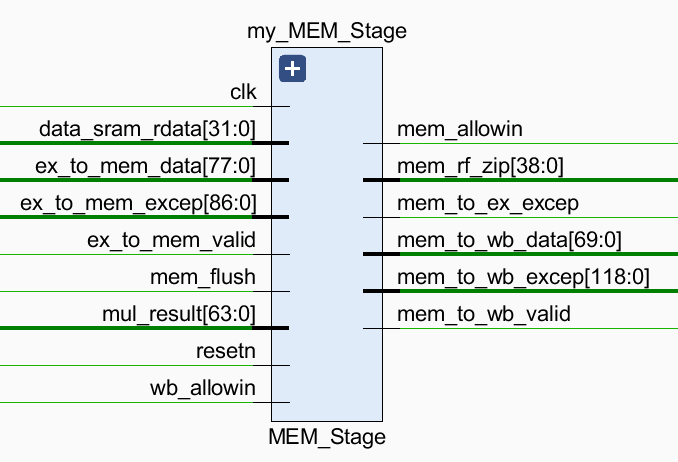


图12 MEM流水级接口

* + 1. 功能描述

根据EXE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EXE级传来的异常信号生成mem\_to\_ex\_excep传递到EXE流水级，并把EXE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EXE级前面有指令出现异常或者中断，防止EXE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 |

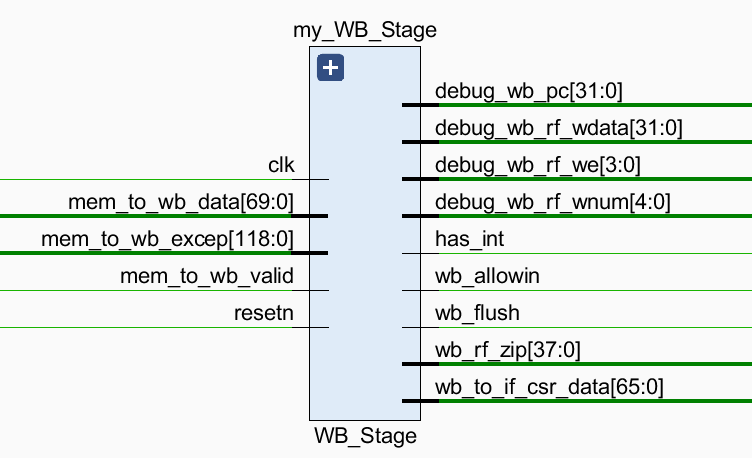


图12 WB流水级接口

* + 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：

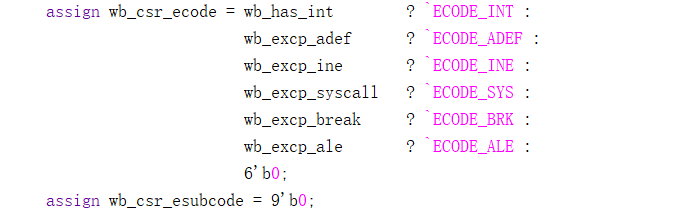


图13 csr\_code和csr\_ecode赋值

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：

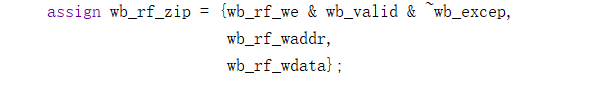


图14 寄存器堆相关信号赋值

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：

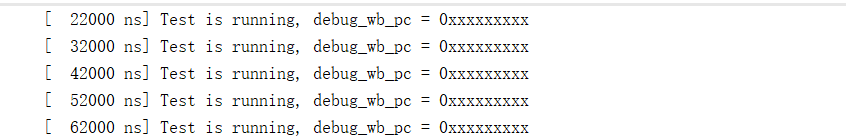


图15 错误1 对应的Console报错

* + - 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：

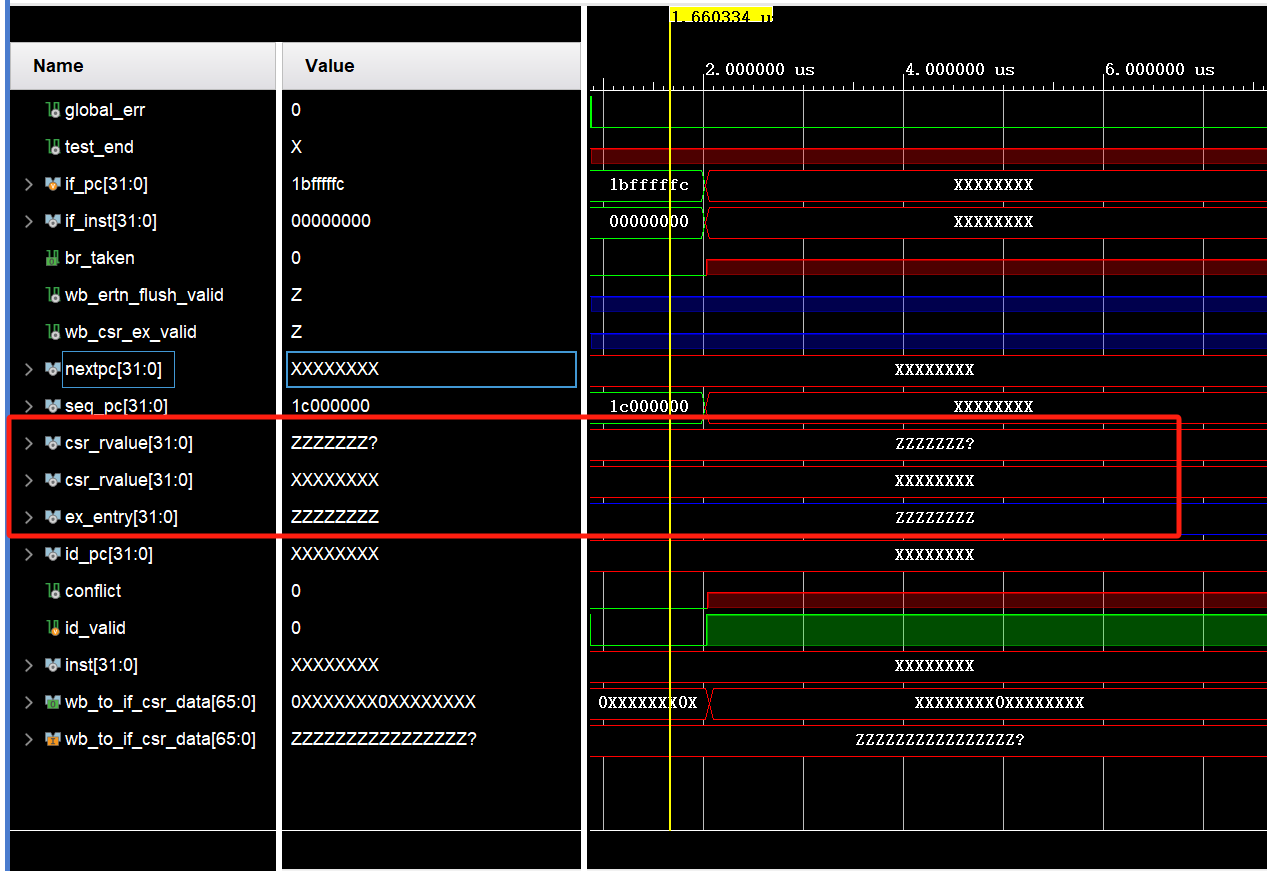


图16 错误1中IF接受到的从WB传回的csr数据有异常

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：

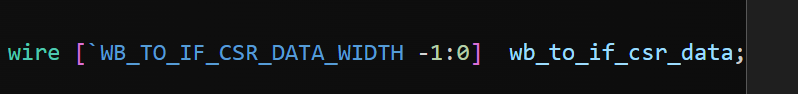


图17 错误1修改代码

该方法有效，来到下一个bug。

* + 1. 错误2：exp12 EXE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：

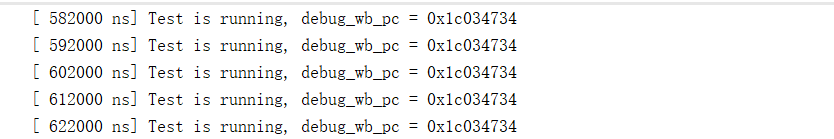


图18 错误2对应的Console报错

* + - 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19 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：

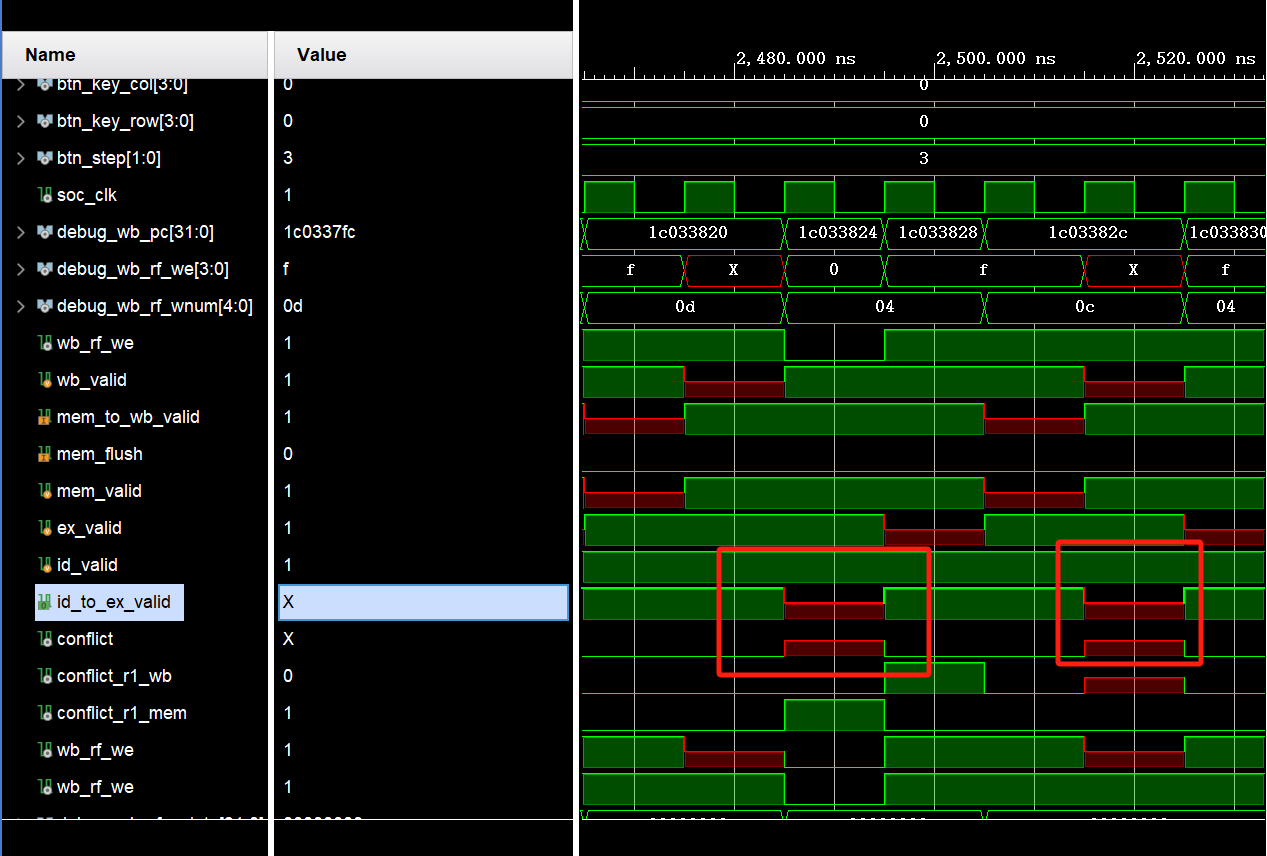


图20 错误2中ID模块的conflict为X

ID模块的conflict的赋值逻辑如下：

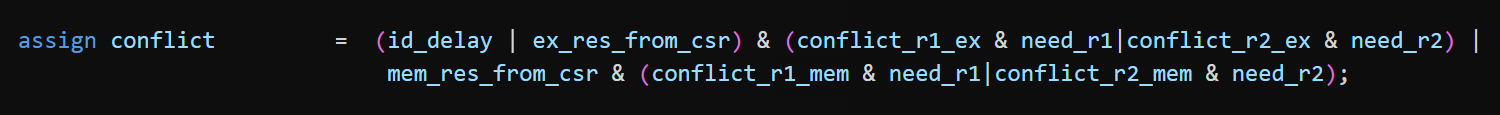


图21 错误2中ID模块的conflict的赋值逻辑

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：

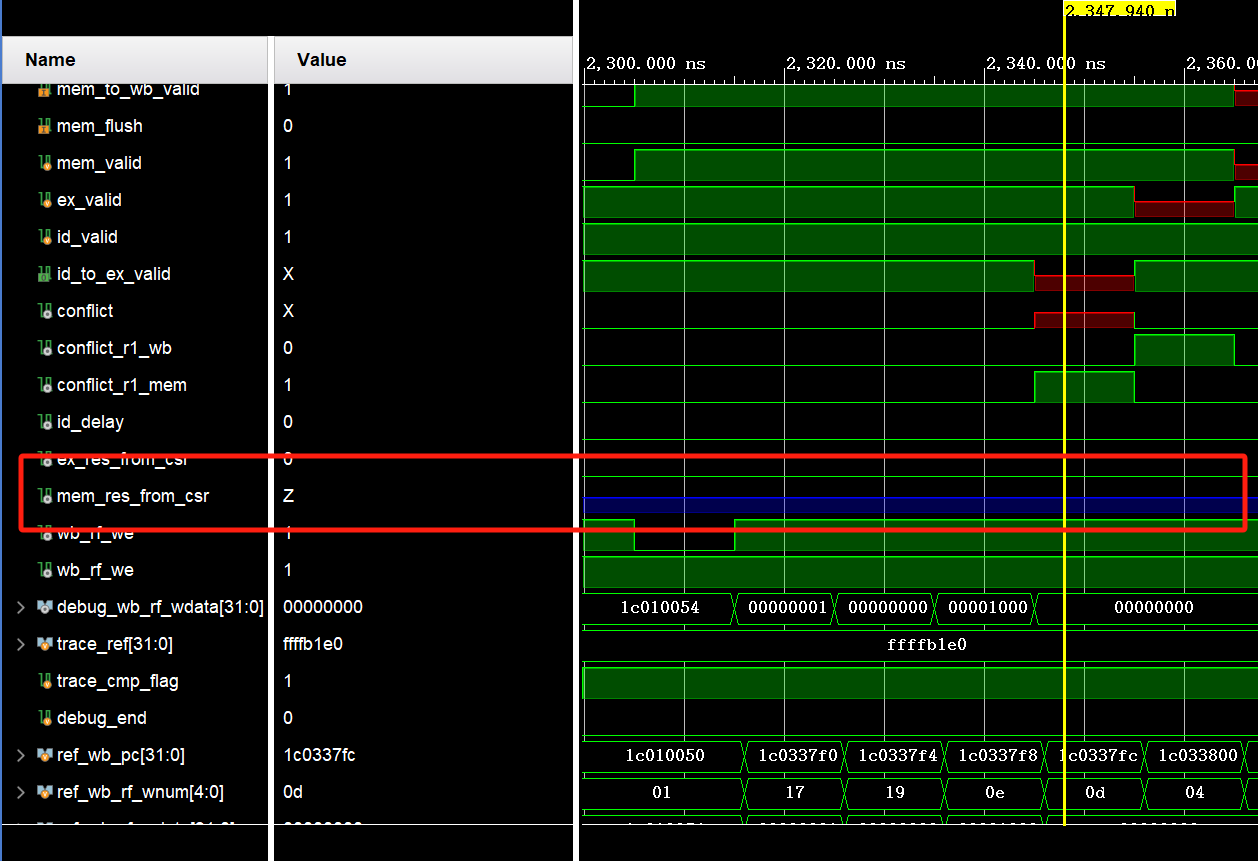


图22 错误2中ID模块的mem\_res\_from\_csr信号为Z

最后查看其赋值逻辑，发现是因为EXE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

EXE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EXE模块与MEM模块的异常或中断数据通路位宽即可



图23 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：

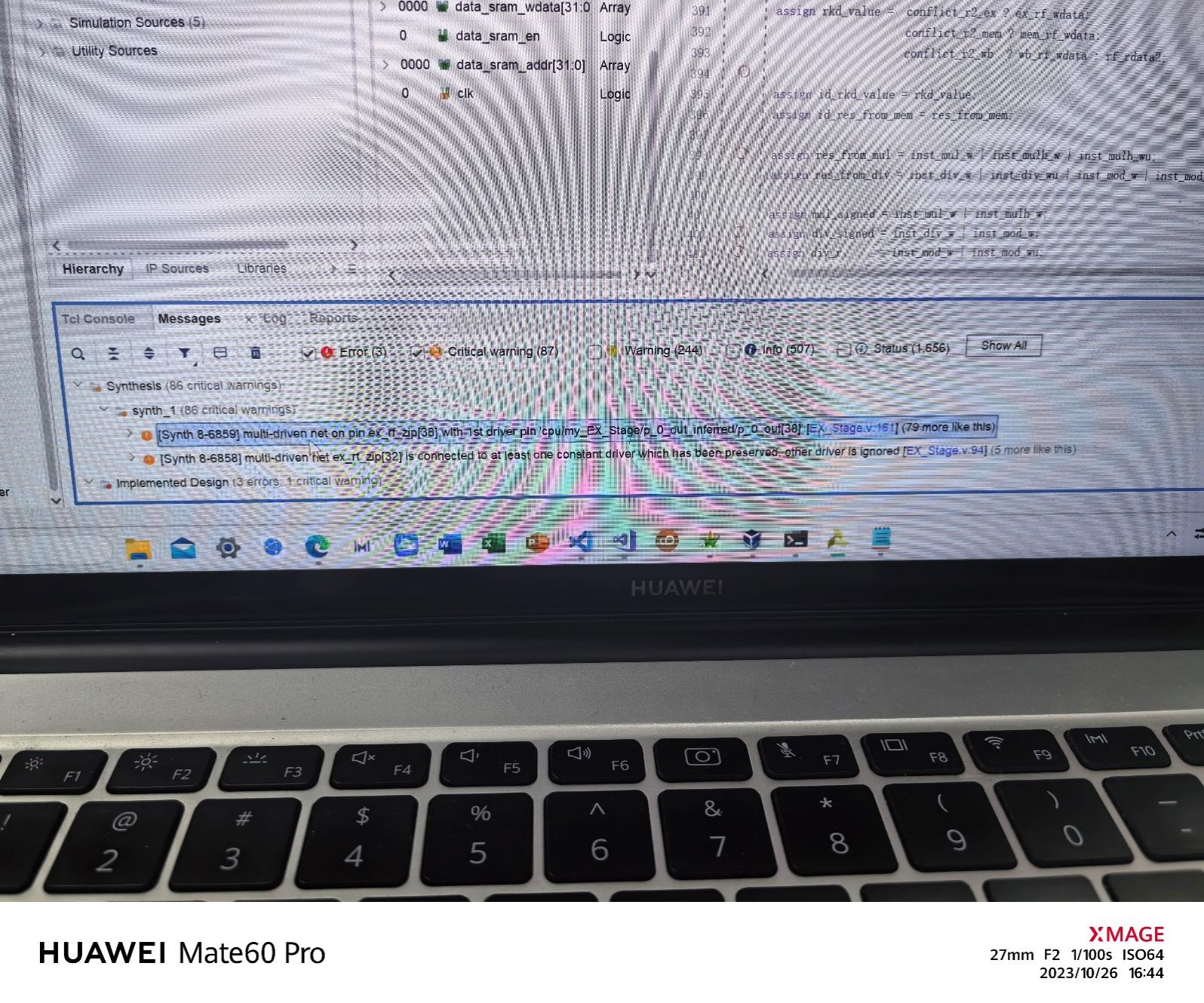


图24 错误3 Tcl\_console的critical\_warning

查看相关信号：

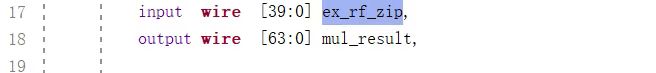


图25 ex\_rf\_zip信号的定义

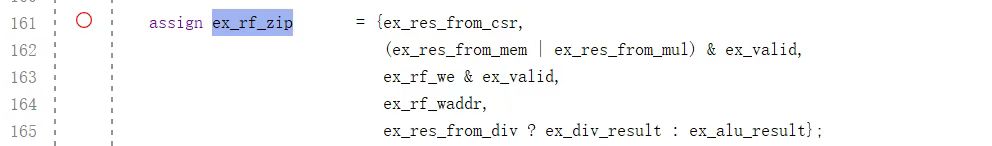


图26 ex\_rf\_zip信号的赋值

* + - 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27 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：：

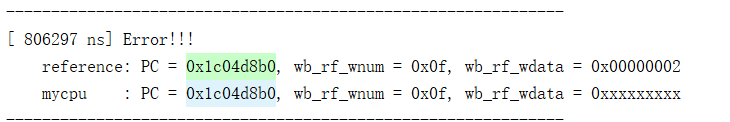


图 28 错误4对应的Console报错

* + - 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：

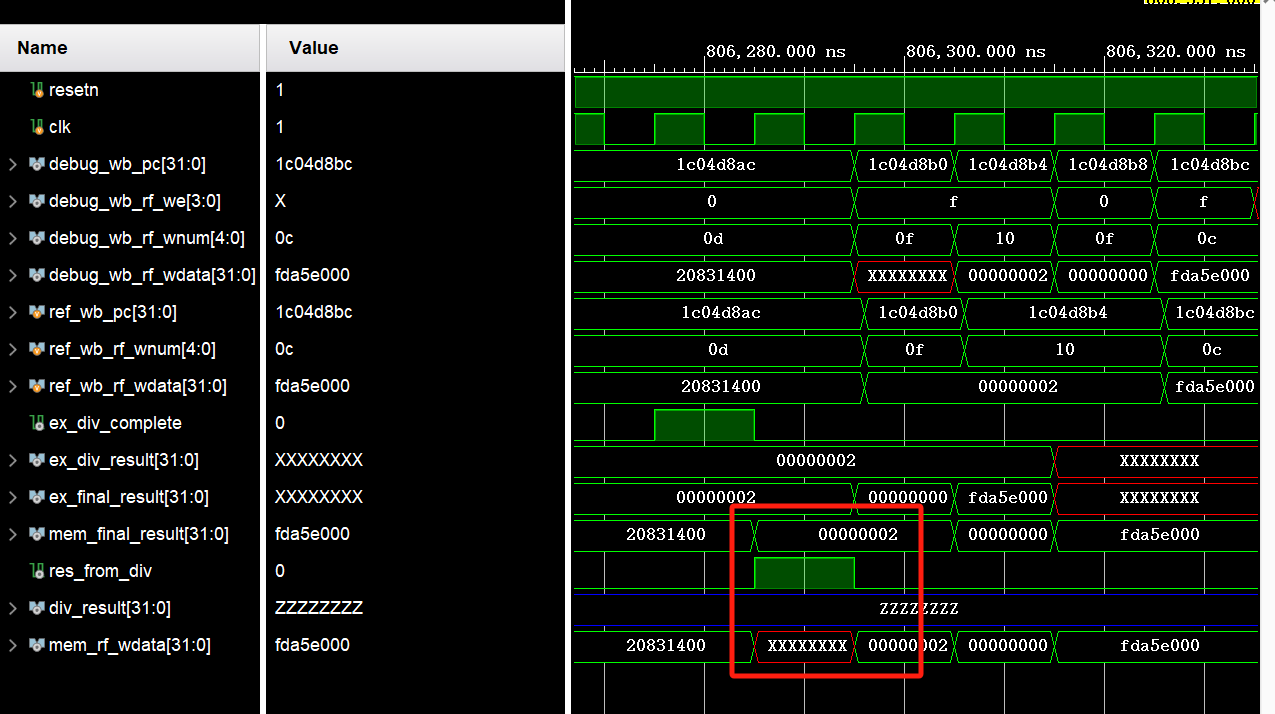


图29 错误4中alu\_src全为高阻态

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EXE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：

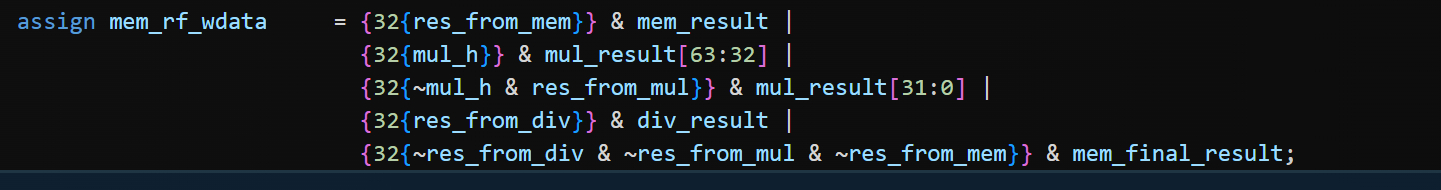


图30 错误4修改前代码

* + - 1. 错误原因

本次实验对数据通路进行了优化，在EXE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：

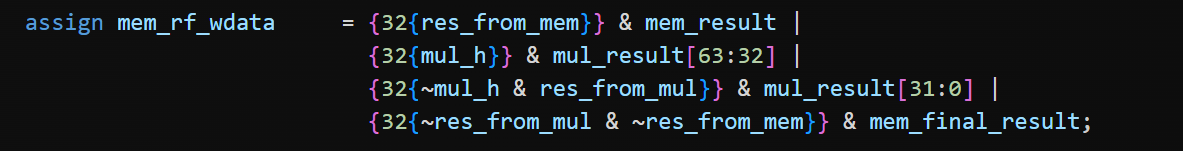


图31 错误4改后代码

* + 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：

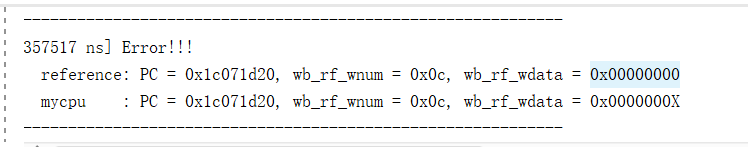


图32 错误5对应的Console报错

* + - 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33 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34 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35 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：

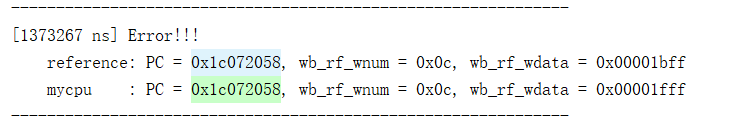


图34 错误6对应的Console报错

* + - 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35 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：

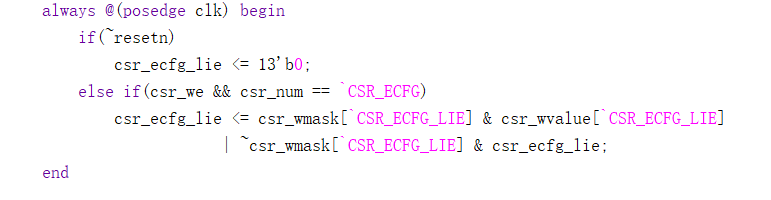


图36 ECFG寄存器复赋值代码

* + - 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。

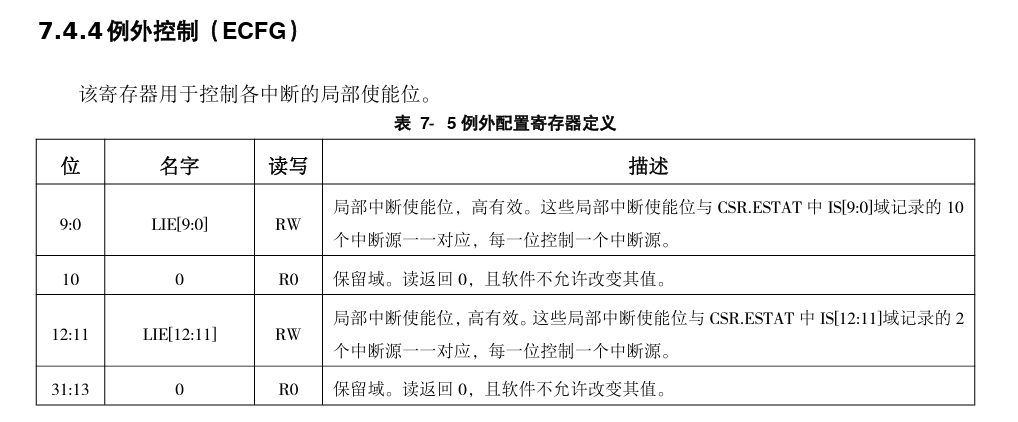


图36 指令集手册关于ECFG寄存器的介绍

* + - 1. 修正效果

将ECFG赋值逻辑进行如下修改即可：

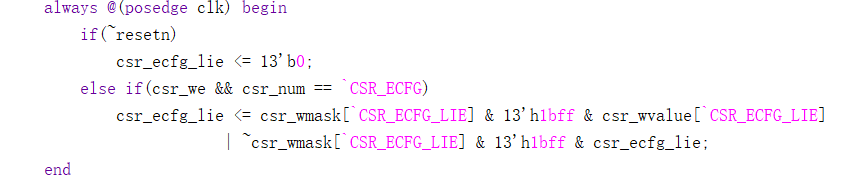


图37 错误6修改后代码

* + 1. 错误7：exp10中乘法器结果未进行高位和低位选择
       1. 错误现象

Console报错如下：

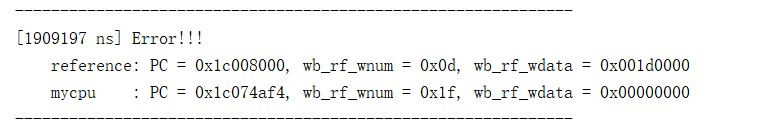


图 38 错误7对应的Console报错

* + - 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：

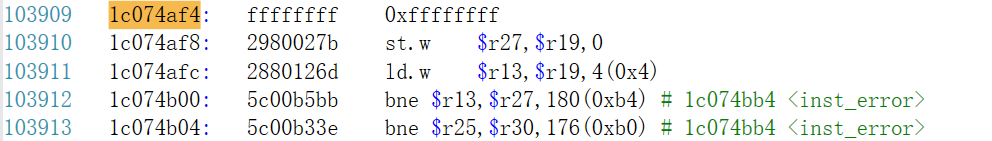


图 38 错误7对应的指令

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：

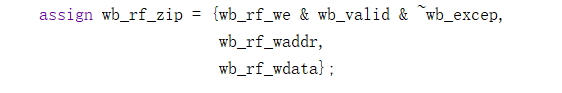


图39 错误7修改后代码（1）

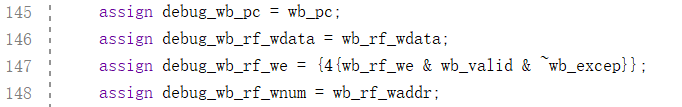


图40 错误7修改后代码（2）

* + 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：

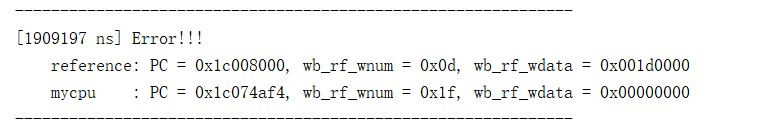


图 41 错误8对应的Console报错

* + - 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：

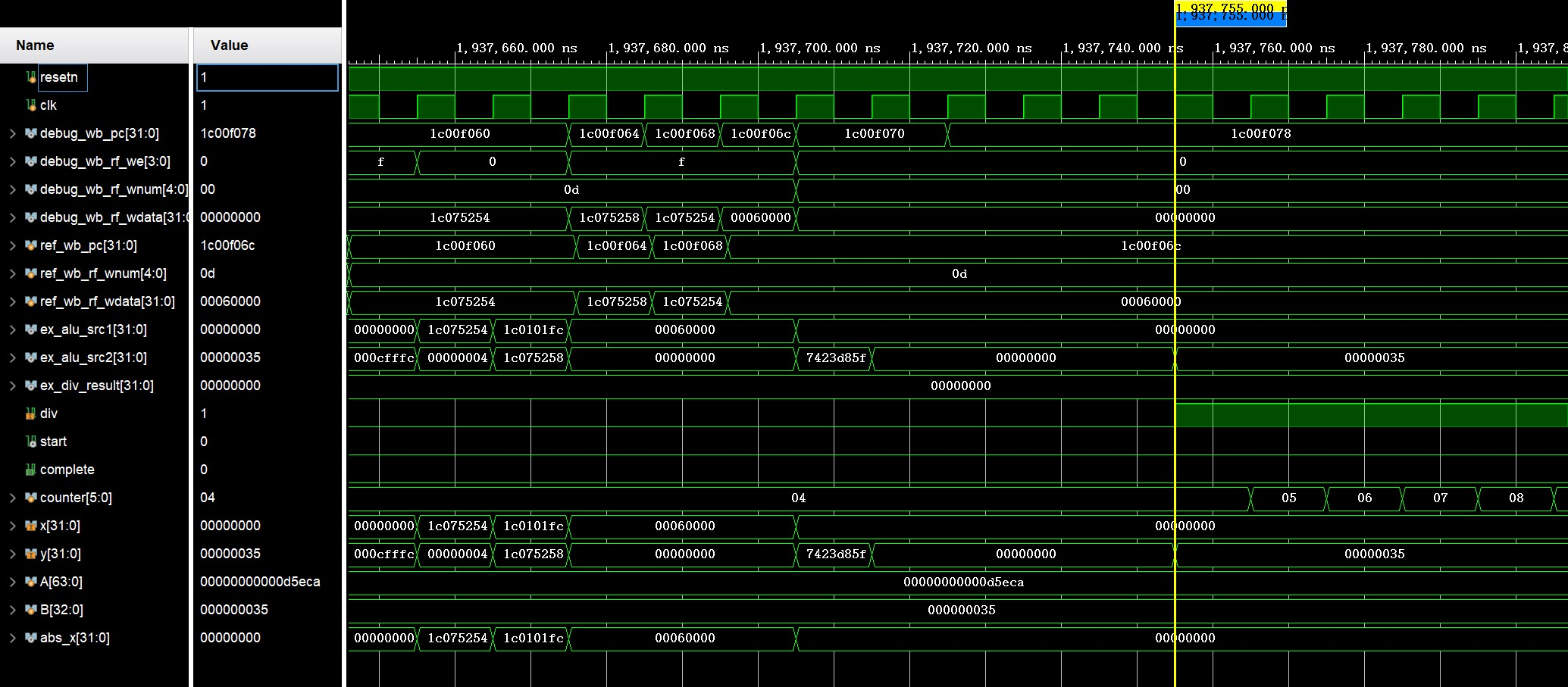


图 42 错误8处理除法时的波形图

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 43 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：

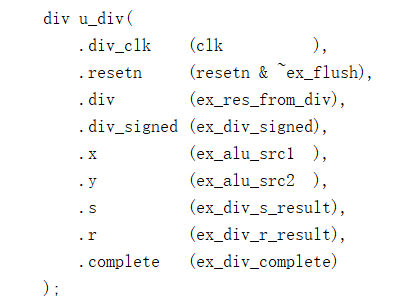


图 44 错误9修改后

修改有效，来到下一个bug。

* + 1. 错误9：exp13中
       1. 错误现象

Console报错如下：

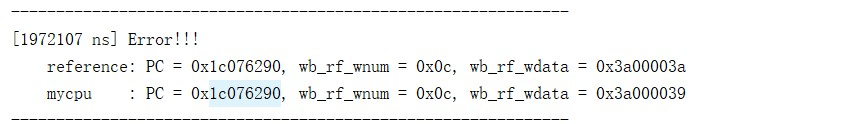


图 45 错误8对应的Console报错

* + - 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：

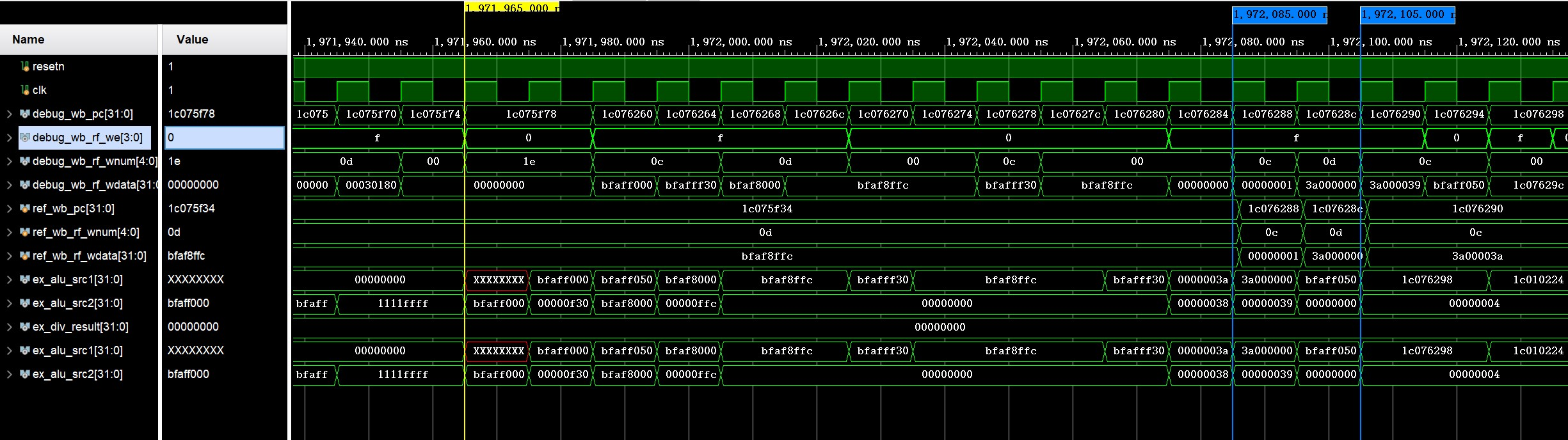


图 46 错误9处理除法时的波形图

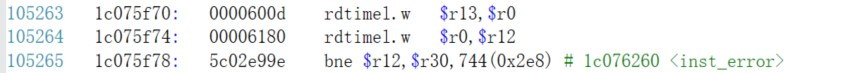


图 47 计时器相关指令部分的汇编指令

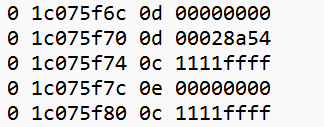


图 46 计时器相关指令部分的金标准

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 47 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，用当从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 48 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：

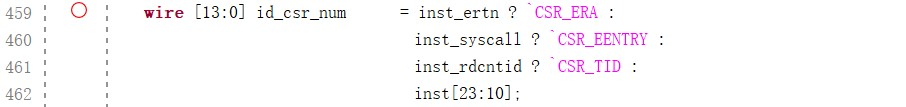


图 49 修改后ID模块为csr寄存器号的代码



图 50 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，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。这一过程是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。