**LAB8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

本实验主要实现了对TLB模块的结构设计和例外支持，并将其加入到流水线CPU中，从而为操作系统的存储管理提供硬件支持。

Exp17要求实现对单个TLB模块的结构设计，使其能够完成最基本的读、写和查找功能，并通过实验提供的测试程序。

Exp18要求在CPU中增添与TLB相关的控制寄存器，包括TLBIDX、TLBEHI、TLBELO0、TLBELO1、ASID、TLBRENTRY；添加对TLB相关指令的支持，包括TLBSRCH、TLBRD、TLBWR、TLBFILL、INVTLB；将TLB模块集成到CPU内，实现从CPU内对TLB进行基本的读、写和查找操作。

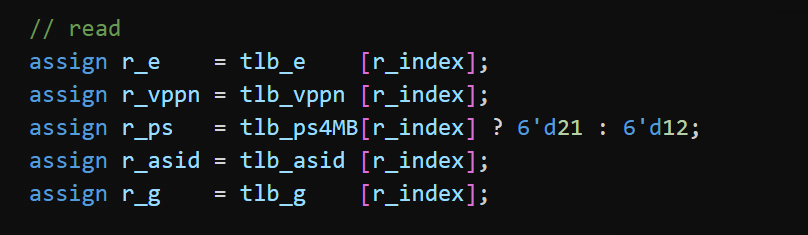
Exp19要求在CPU中添加对TLB例外的支持，包括TLB重填例外、load/store取指操作页无效例外、页修改例外、页特权等级不合规例外；在CPU中添加直接映射配置窗口控制寄存器DMW0和DMW1，并利用TLB查找功能实现CPU访存过程中的虚实地址转换

1. 实验设计
   1. 总体设计思路
2. Exp17

Exp17对单个TLB模块的设计可根据所需实现的功能分为读、写和查找三部分，下面将从读写功能和查找功能两方面阐述总体设计思路。

1. **读写功能设计**

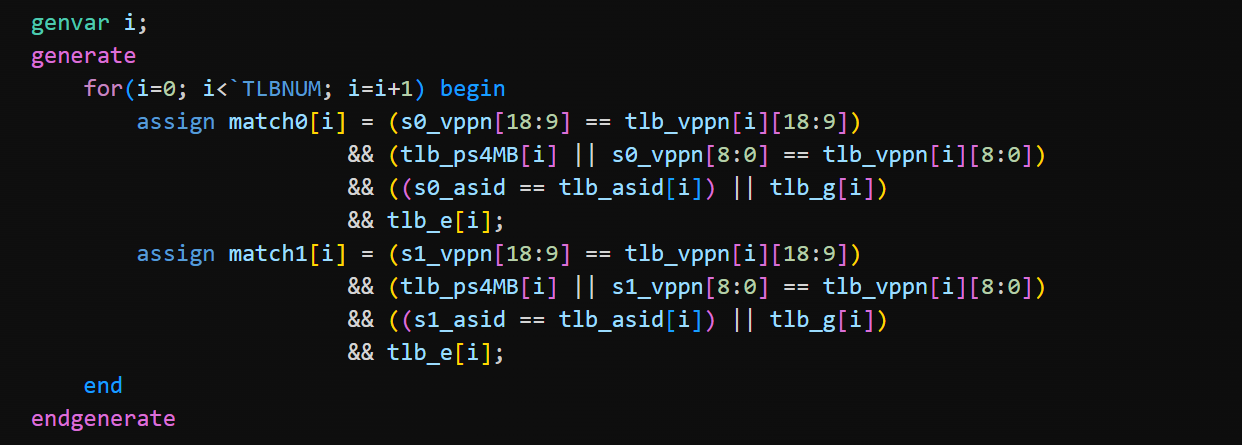
TLB模块的读写方式与寄存器堆的读写方式十分类似。读TLB的方式是根据传入的index将对应表项各个域的数据通过组合逻辑读出；写TLB的方式是根据传入的index和写使能信号，在下一拍将要写的各个域的数据写入对应表项相应位置。但在读写过程中需要注意的是，由于龙芯架构32位精简版只支持4KB和4MB两种页大小，因此尽管TLB表项中的PS域是6bit，但其只有0和1两种取值：0表示页大小为4KB（两个物理页大小均为4KB），1表示页大小为4MB（两个物理页大小均为2MB）。而TLB模块输入和输出中的PS端口标记了页的真实大小，取值分别是6’d21和6’d12，因此在对页表项进行读写操作时需要将两者进行转换。以读TLB为例：



图x. PS字段的含义及其赋值逻辑

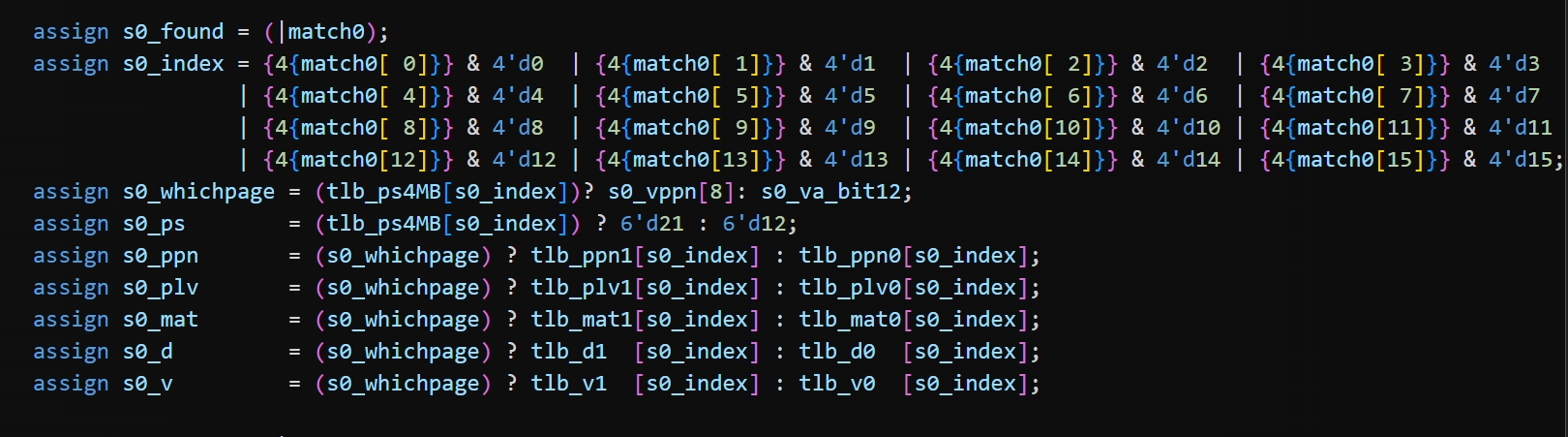
1. **查找功能设计**

TLB模块提供了两个通道用于查找，一个通道对应取指阶段的查找，另一个通道对应访存阶段的查找。查找的流程是并行化的，即同时对16个页表项的奇偶相邻页表的物理转换信息进行比较和查找，并用两个16位宽的match信号表示查找结果。在代码实现上，可以使用verilog语言中的for循环更方便地实现对页表项的同时匹配和查找过程，匹配和查找的逻辑是：对于每一个页表项，分别要比较和检查以下几点：一个是查看比较部分的存在位E，如果为1则表示页表项存在，若为0表示该页表项为空，查找失败；然后还需要查看比较部分的全局标志位G和地址空间表示位ASID，判断该页表项中的数据是否属于本进程，若属于不属于本进程则查找失败；除此之外，还需要查看比较部分的页大小PS位和虚双页号VPPN位，如果PS位为1，表示页大小为4MB（两个物理页大小均为2MB），此时只需要比较VPPN的高10位是否与s\_vppn的高10位相等，若相等则查找成功；如果PS位为0，表示页大小为4KB（两个物理页大小均为4KB），此时需要比较VPPN和s\_vppn的全部位数。如果以上全部满足则查找成功。



图x. TLB的查找判断逻辑

通过上述查找过程可以获得两个match信号，match0不为全0说明0号通道查找到了匹配的页表项（0号通道用于取指阶段的查找），match1不为全0说明1号通道查找到了匹配的页表项（1号通道用于访存阶段的查找）。match0或match1中哪一位为1则说明命中了对应位的页表项。然后再根据输入的s\_va\_bit12或者s\_vppn决定使用匹配到的TLB表项的偶页还是奇页的查找结果。查找的代码实现如下：



图x TLB的查找结果的输出

1. Exp18

本实验将exp17实现的TLB实例化，并集成到cpu中。为了实现对TLB的操作，本实验增加了TLBSRCH、TLBRD、TLBWR、TLBFILL、INVTLB这五条指令。此外，还需要在csr模块内增加TLB有关的寄存器TLBIDX、TLBEHI、TLBELO0、TLBELO1、ASID、TLBRENTRY，并将TLB的一部分端口与csr相连。如果沿用之前的设计，将csr模块放在wb级中，会使得wb级增加很多不必要的接口。为此，我们将csr移出wb级，成为独立的模块。

**（1）TLB维护指令**

在本实验新增的五条指令中，TLBSRCH、TLBRD、TLBWR、TLBFILL都会在TLB和CSR之间进行数据的传递。TLBSRCH在EXE级根据CSR中的寄存器查询TLB，并将查询结果从EXE级一路传到WB级，在WB级将它写入csr。这种安排使得它可以在EXE级复用访存的TLB查找通路，在WB级复用之前将数据写入csr的通路。TLBRD、TLBWR、TLBFILL均在WB级进行读写。csr寄存器中的相关数据通过端口直接传入WB级，cpu在WB级根据这些数据生成读写需要的数据，并和相关控制信号一起通过端口传给TLB。

不同于以上四条指令，INVTLB并不负责在TLB和CSR之间传递数据，它的数据来源也并非csr寄存器，而是通用寄存器。它会根据存放在寄存器中的ASID和VA，按照op对TLB进行无效操作。它也需要对TLB进行查找，因此我们让它和TLBSRCH使用相同的位于EXE级的数据通路进行查找。

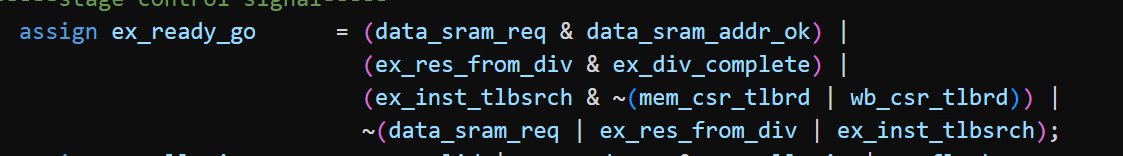
**（2）数据相关**

在增加TLB后，指令在取指和访存时都有可能查找TLB用于虚实地址转换，而TLBWR、TLBFILL、INVTLB都会更新TLB的内容，二者之间形成围绕TLB的写后读相关。此外，所有指令在取指和访存都可能读取CSR.ASID来查找TLB，而TLBRD会更新该寄存器，二者之间形成围绕CSR的写后读相关。为此，考虑到上述情况出现的频率极低，使用前递或者阻塞的操作，对性能提升很小但使流水线逻辑变的更加复杂，所以处于均衡的考虑，对于TLBWR、TLBFILL、TLBRD、INVTLB以及更改ASID或者CRMD寄存器的csr写指令，本组成员的设计是刷新流水级并重新取指；而刷新流水线的操作可以直接复用在之前实验的wb\_flush信号，在WB流水级发出，刷掉前面所有流水级，并把IF的下一个取值PC改为此时WB流水级的PC+4。



图x WB流水级对于TLB数据相关的处理

除此之外，还需要考虑一种写后读相关：当TLBSRCH位于EXE级时，它会用ASID或TLBEHI中的内容查询TLB。此时如果在MEM流水级或者WB流水级恰好有条修改ASID或TLBEHI的指令，或是TLBRD指令，就会引发围绕csr的数据相关。在这种情况下需要将TLBSRCH阻塞在EXE级，等前面的指令写入完成后再允许其流入下一级。



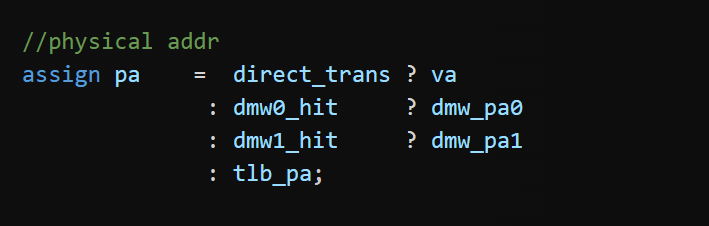
图x EXE流水级的阻塞逻辑

1. Exp19

对exp19设计思路主要从虚实地址转换的过程和TLB例外处理两方面来阐述。

1. **虚实地址转换过程**

本实验需要支持的虚实地址翻译模式有两种：直接地址翻译模式和映射地址翻译模式。控制寄存器CRMD中的DA域与PG域决定了采用何种地址翻译模式：当DA域为1，PG域为0时，MMU处于直接地址翻译模式，在这种映射模式下，物理地址直接等于虚拟地址；当DA域为1，PG域为0时，MMU处于映射地址翻译模式，此时则先查看配置窗口寄存器DMW0和DMW1，如果能命中且当前特权等级在该配置窗口中被允许，则采用直接映射模式，物理地址等于虚地址低29位拼接上被命中的配置窗口寄存器的高3位；如果两个配置窗口都不命中或特权等级不合规，则采用页表映射模式，需要根据虚地址的VPPN和当前的地址空间标识符ASID查找TLB，并根据查找TLB返回的PPN和虚地址的offset得出物理地址。通过多路选择器并根据当前的地址翻译模式可以得到最终的物理地址，然后将它作为inst\_sram\_addr或data\_sram\_addr传递给转接桥，进而传递到AXI总线进行地址访问。



图x 虚拟地址到物理地址的转换

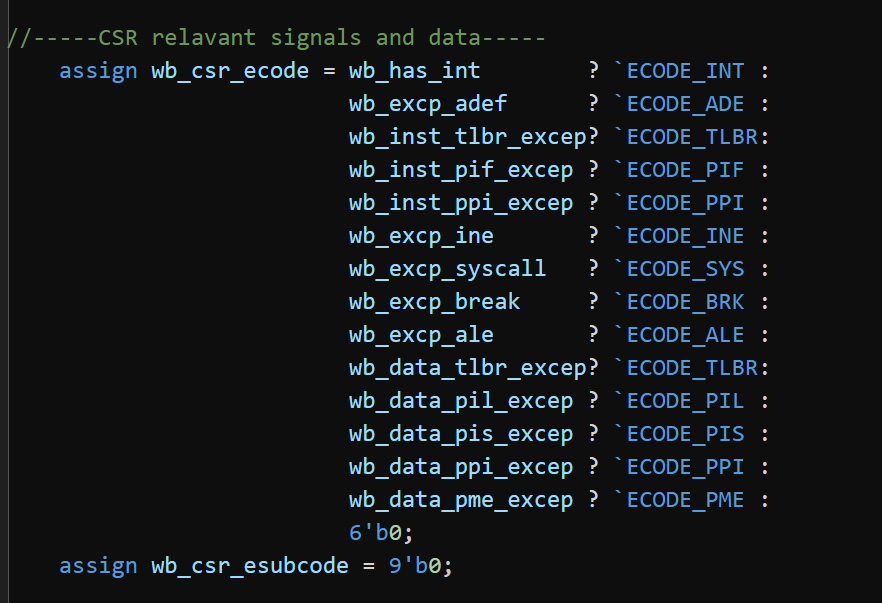
1. **TLB例外处理**

本实验首先要修改INE例外的逻辑，同时还要增添对PIF、PIL、PIS、PPI、PME和TLBR例外的处理。在之前的实验，INE取指地址错误仅仅处理译码得到的指令不是以及支持的指令，但在本实验还要考虑INVTLB且其op不合法的情况。如果是INVTLB且其op大于6，也要报出INE例外。

TLB重填例外是指在页表映射模式下没有命中TLB，**并且处理TLB重填例外的中断入口区别于其他例外**。取指操作页无效例外（PIF）需要在IF级进行判断，如果此时处于页表映射模式并且查找到的TLB表项无效，即s\_v为0，则报错。load操作页无效例外（PIL）和store操作页无效例外（PIS）需要在EXE级进行判断，判断例外的过程和PIF类似，只是还需要结合指令类型。页特权等级不合规例外需要比较命中TLB表项的PLV域和从CRMD寄存器中取出的当前PLV，如果前者小则报出该错。页修改例外也要在EXE级进行判断，如果此时执行的是store指令且处于页表映射模式，如果命中的TLB表项脏位为0，则报出该例外。

需要注意的是，在EXE级如果触发TLB相关例外，说明虚实地址翻译环节出现错误，没有得到正确的物理地址，因此此时不应该向总线发起请求，即不能拉高data\_sram\_req。此时不向总线发起请求，所以也不用等到总线发来的addr\_ok和data\_ok信号。

此外，在处理例外逻辑时，还要注意这些例外的优先级。根据龙芯架构的指令集手册的说明，IF阶段触发的例外优先级最高，ID阶段次之，EX级再次之。而在IF级触发的例外中，ADEF例外的优先级高于TLB相关例外；在EX流水级触发的例外中，ALE例外的优先级高于TLB的相关例外；在TLB相关例外中，TLBR的优先级最高，然后为PIF或者PIL或者PIS，再其次为PPI，最后为PME。在WB级中，需要按照上面逻辑，比较各个例外的优先级来设置ecode和esubcode：



图x 例外处理的优先级

* 1. 重要模块1设计：TLB模块
     1. 工作原理

TLB 采用全相联查找表的组织形式。在TLB模块中保存了TLB页表的内容，共16个页表项。并通过2个查找端口，1个读端口，1个写端口和一个INVTLB指令端口与外部交换数据。实现了对TLB页表的4种操作：

查找：输入虚双页号vppn、虚地址12位va\_bit12与地址空间标识asid，得到对应物理页的内容。

写：输入页表项数index，并将输入的页表项内容写到对应的页表项里。

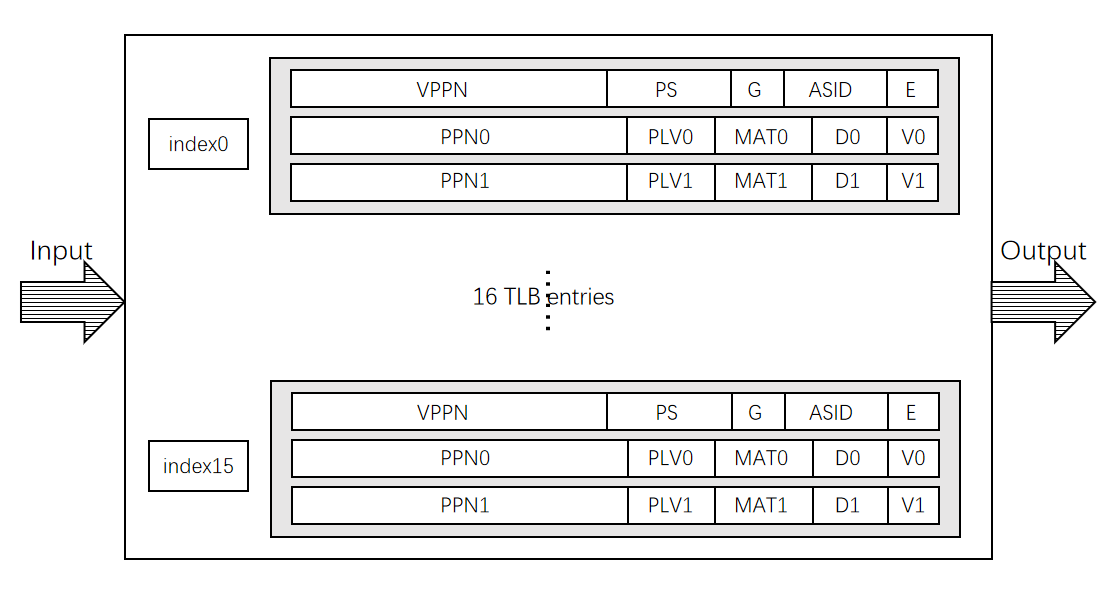
读：输入页表项数index，并得到对应的页表项中的内容。

无效指令INVTLB：使满足opcode对应条件的页表项无效化。

* + 1. 接口定义

表1 TLB模块接口信号

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| **Search port 0** | | | |
| s0\_vppn | IN | 19 | 查找虚双页号 |
| s0\_va\_bit12 | IN | 1 | 访存虚地址12位 |
| s0\_asid | IN | 10 | 地址空间标识 |
| s0\_found | OUT | 1 | 查找到信号 |
| s0\_index | OUT | 4 | 页表项数 |
| s0\_ppn | OUT | 20 | 物理页号 |
| s0\_ps | OUT | 6 | 页大小 |
| s0\_plv | OUT | 2 | 权限等级 |
| s0\_mat | OUT | 2 | 访存种类 |
| s0\_d | OUT | 1 | 脏页信号 |
| s0\_v | OUT | 1 | 页有效信号 |
| **Search port 1** | | | |
| s1\_vppn | IN | 19 | 查找虚双页号 |
| s1\_va\_bit12 | IN | 1 | 访存虚地址12位 |
| s1\_asid | IN | 10 | 地址空间标识 |
| s1\_found | OUT | 1 | 查找到信号 |
| s1\_index | OUT | 4 | 页表项数 |
| s1\_ppn | OUT | 20 | 物理页号 |
| s1\_ps | OUT | 6 | 页大小 |
| s1\_plv | OUT | 2 | 权限等级 |
| s1\_mat | OUT | 2 | 访存种类 |
| s1\_d | OUT | 1 | 脏页信号 |
| s1\_v | OUT | 1 | 页有效信号 |
| **INVTLB opcode** | | | |
| invtlb\_valid | IN | 1 | 无效指令有效信号 |
| invtlb\_op | IN | 5 | 无效指令操作码 |
| **Write port** | | | |
| we | IN | 1 | 写使能信号 |
| w\_index | IN | 4 | 写页表项数 |
| w\_e | IN | 0 | 写页表项有效信号 |
| w\_vppn | IN | 19 | 写虚拟页号 |
| w\_ps | IN | 6 | 写页大小 |
| w\_asid | IN | 10 | 写进程号 |
| w\_g | IN | 1 | 写全局信号 |
| w\_ppn0 | IN | 20 | 写物理页号0 |
| w\_plv0 | IN | 2 | 写权限等级0 |
| w\_mat0 | IN | 2 | 写访存种类0 |
| w\_d0 | IN | 1 | 写脏页信号0 |
| w\_v0 | IN | 1 | 写页有效信号0 |
| w\_ppn1 | IN | 20 | 写物理页号1 |
| w\_plv1 | IN | 2 | 写权限等级1 |
| w\_mat1 | IN | 2 | 写访存种类1 |
| w\_d1 | IN | 1 | 写脏页信号1 |
| w\_v1 | IN | 1 | 写页有效信号1 |
| **Read port** | | | |
| r\_index | IN | 4 | 读页表项数 |
| r\_e | OUT | 0 | 读页表项有效信号 |
| r\_vppn | OUT | 19 | 读虚拟页号 |
| r\_ps | OUT | 6 | 读页大小 |
| r\_asid | OUT | 10 | 读进程号 |
| r\_g | OUT | 1 | 读全局信号 |
| r\_ppn0 | OUT | 20 | 读物理页号0 |
| r\_plv0 | OUT | 2 | 读权限等级0 |
| r\_mat0 | OUT | 2 | 读访存种类0 |
| r\_d0 | OUT | 1 | 读脏页信号0 |
| r\_v0 | OUT | 1 | 读页有效信号0 |
| r\_ppn1 | OUT | 20 | 读物理页号1 |
| r\_plv1 | OUT | 2 | 读权限等级1 |
| r\_mat1 | OUT | 2 | 读访存种类1 |
| r\_d1 | OUT | 1 | 读脏页信号1 |
| r\_v1 | OUT | 1 | 读页有效信号1 |

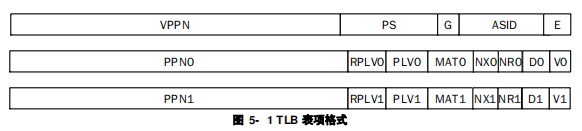


图x. TLB模块示意图

* + 1. 功能描述

1. TLB结构

TLB页表中每一个页表项为一个双页，包含比较部分和物理转换部分，结构如下图：



TLB表项比较部分包括：

VPPN：虚双页号，19位，虚页号最低位不保存在TLB中。

PS：页大小（2的幂次），6位，本次实验中有21（2MB页）和12（4KB页）两种情况。

G：全局标志位，1位，为1时该虚拟地址在所有进程间共享。

ASID：地址空间标识，10位，用于区分不同进程中同样的虚地址。

E：存在位，1位，表示该表项存在。

TLB表项物理转换部分保存一对奇偶相邻页表的物理转换信息，包括：

PPN：物理页号，20位。

PLV：特权等级，2位。

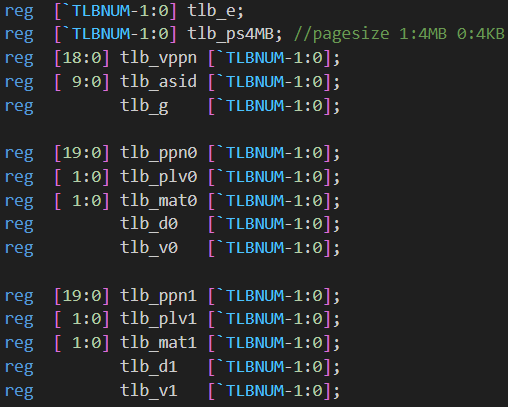
MAT：存储访问类型，2位。

D：脏位，1位，表示该页表项地址范围内有脏数据。

V：有效位，1位，表示该页有效。

RPLV、NX、NR在本次实验中未被定义。

本次实验中实现的TLB页表中包含16个页表项，能保存16对32个页。页大小有2MB与4KB两种，通过ps4MB信号指示。使用16组寄存器来保存页表内容：



（2）TLB操作

1）搜索

实现两个相同搜索端口的并行搜索，一个用于取指，一个用于内存读写指令。

输入为虚双页号vppn，访存虚地址12位va\_bit\_12与进程号asid。

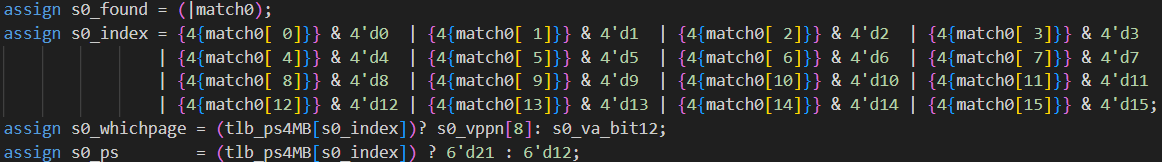
首先搜索有没有匹配的页表项，得到页表中每一项是否匹配的match信号（01表示两个搜索端口）：



当页大小为2MB时，匹配vppn的[18:9]位，页大小为4KB时，匹配vppn的所有位。

如果G位为1则不用考虑ASID，否则对ASID进行匹配。

然后得到found，index，whichpage，ps信号（端口0为例）：



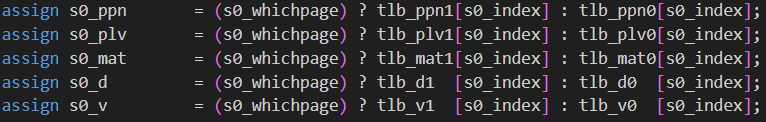
found：是否匹配到。

index：匹配到的页表项的项数。

whichpage：双页中的哪一页，页大小为2MB时通过vppn的第8位来判断，4KB时通过输入的va\_bit12来判断。

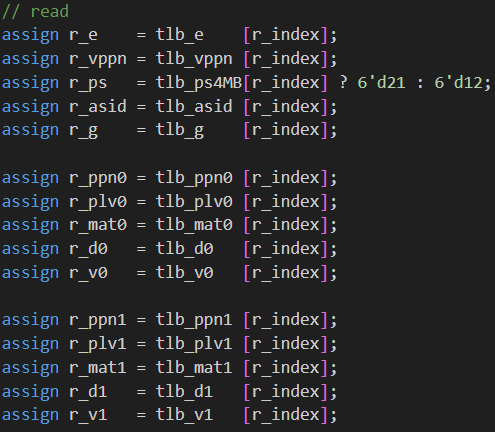
ps：页大小，2MB为21，4KB为12。

最后通过index和whichpage从页表中得到物理转换信息并输出：



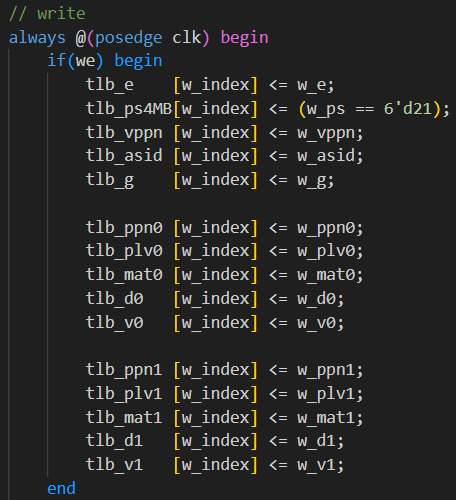
2）读

根据输入的项数r\_index，将对应页表项内的内容输出即可：



3）写

根据输入的写使能信号we与项数w\_index，将输入内容填入对应的页表项即可:

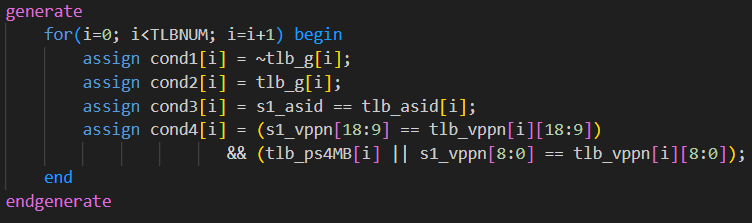


4）无效指令

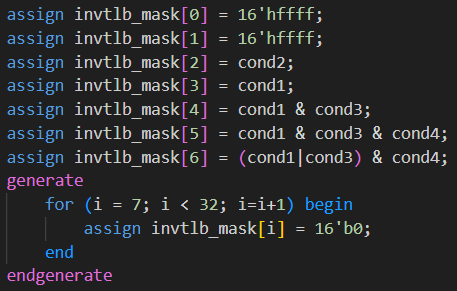
invtlb指令的opcode对应的操作如下表：

|  |  |
| --- | --- |
| opcode | 操作 |
| 0x0 | 清除所有页表项 |
| 0x1 | 清除所有页表项 |
| 0x2 | 清除所有G=1的页表项 |
| 0x3 | 清除所有G=0的页表项 |
| 0x4 | 清除所有G=0，且ASID等于寄存器指定ASID的页表项 |
| 0x5 | 清除所有G=0，且ASID等于寄存器指定ASID，且VA等于寄存器指定VA的页表项 |
| 0x6 | 清除所有G=1或ASID等于寄存器指定ASID，且VA等于寄存器指定VA的页表项 |
| 0x7~0x20 | 不进行清除操作 |

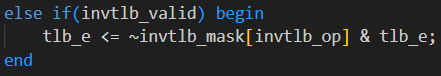
由此对每个页表项，定义4个条件cond1~cond4：



根据cond得到mask信号如下：



在invtlb指令有效时，根据输入的opcode选择mask，将mask对应位为1的页表项无效化（e值清零），为0的页表项不变：



* 1. 重要模块2设计：CSR模块
     1. 工作原理

新增用于虚实地址转换及TLB相关例外处理的寄存器

* + 1. 接口定义

表2 CSR模块接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | 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 |
| tlb\_ex\_entry | OUT | 32 | TLB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 |
| csr\_tlbehi\_vppn | OUT | 19 | CSR.TLBEHI.VPPN，TLB表项的虚页号 |
| csr\_tlbidx\_index | OUT | 4 | CSR.TLBIDX.INDEX，TLB表项索引 |
| tlbsrch\_we | IN | 1 | TLBSRCH指令的写使能 |
| tlbsrch\_hit | IN | 1 | TLBSRCH查询命中 |
| tlbrd\_we | IN | 1 | TLBRD指令的写使能 |
| tlbsrch\_hit\_index | IN | 4 | TLBSRCH查询命中的索引 |
| csr\_crmd\_rvalue | OUT | 32 | CSR.CRMD的值 |
| csr\_asid\_rvalue | OUT | 32 | CSR.ASID的值 |
| csr\_dmw0\_rvalue | OUT | 32 | CSR.DMW0的值 |
| csr\_dmw1\_rvalue | OUT | 32 | CSR.DMW1的值 |
| r\_tlb\_e | IN | 1 | 从TLB读出的e域信息 |
| r\_ tlb\_vppn | IN | 19 | 从TLB读出的vppn域信息 |
| r\_ tlb\_ps | IN | 6 | 从TLB读出的ps域信息 |
| r\_ tlb\_asid | IN | 10 | 从TLB读出的asid域信息 |
| r\_ tlb\_g | IN | 1 | 从TLB读出的g域信息 |
| r\_ tlb\_ppn0 | IN | 20 | 从TLB读出的偶页ppn域信息 |
| r\_ tlb\_plv0 | IN | 2 | 从TLB读出的偶页plv域信息 |
| r\_ tlb\_mat0 | IN | 2 | 从TLB读出的偶页mat域信息 |
| r\_ tlb\_d0 | IN | 1 | 从TLB读出的偶页d域信息 |
| r\_ tlb\_v0 | IN | 1 | 从TLB读出的偶页v域信息 |
| r\_ tlb\_ppn1 | IN | 20 | 从TLB读出的奇页ppn域信息 |
| r\_ tlb\_plv1 | IN | 2 | 从TLB读出的奇页plv域信息 |
| r\_ tlb\_mat1 | IN | 2 | 从TLB读出的奇页mat域信息 |
| r\_ tlb\_d1 | IN | 1 | 从TLB读出的奇页d域信息 |
| r\_ tlb\_v1 | IN | 1 | 从TLB读出的奇页v域信息 |
| w\_ tlb\_e | OUT | 1 | TLB写行为的e域信息 |
| w\_ tlb\_vppn | OUT | 19 | TLB写行为的vppn域信息 |
| w\_ tlb\_ps | OUT | 6 | TLB写行为的ps域信息 |
| w\_ tlb\_asid | OUT | 10 | TLB写行为的asid域信息 |
| w\_ tlb\_g | OUT | 1 | TLB写行为的g域信息 |
| w\_ tlb\_ppn0 | OUT | 20 | TLB写入偶页的ppn域信息 |
| w\_ tlb\_plv0 | OUT | 2 | TLB写入偶页的plv域信息 |
| w\_ tlb\_mat0 | OUT | 2 | TLB写入偶页的mat域信息 |
| w\_ tlb\_d0 | OUT | 1 | TLB写入偶页的d域信息 |
| w\_ tlb\_v0 | OUT | 1 | TLB写入偶页的v域信息 |
| w\_ tlb\_ppn1 | OUT | 20 | TLB写入奇页的ppn域信息 |
| w\_ tlb\_plv1 | OUT | 2 | TLB写入奇页的plv域信息 |
| w\_ tlb\_mat1 | OUT | 2 | TLB写入奇页的mat域信息 |
| w\_ tlb\_d1 | OUT | 1 | TLB写入奇页的d域信息 |
| w\_ tlb\_v1 | OUT | 1 | TLB写入奇页的v域信息 |

* + 1. 功能描述

在虚实地址转换时，地址翻译模式由CRMD中的DA和PG域控制，为此需要让这两个域能被csr指令更新。当DA=0，PG=1时，处理器核的MMU处于直接地址翻译模式。当DA=0，PG=1时，MMU处于映射地址翻译模式，这种模式又可分为直接映射地址翻译模式和页表映射地址翻译模式。

对于直接映射地址翻译模式，相关配置信息存储在DMW0、DMW1寄存器中，每个窗口除了地址范围信息外，还可以配置该窗口在哪些特权等级下可用，以及虚地址落在该窗口上的访存操作的存储访问类型。存储访问类型由CRMD的DATF和DATM域控制。这些域都只需支持csr指令的读写。

对于没有落在DMW0、DMW1设置的直接映射配置窗口中的地址，需要采用页表映射地址翻译模式，为此也需要在CSR模块中增加TLB相关的控制状态寄存器。TLBIDX包含TLB指令操作TLB时的索引值、PS以及是否有效等信息。TLBEHI包含TLB 表项高位的虚页号，TLBELO0、TLBELO1则包含表项低位的物页号、有效、脏等信息。ASID寄存器包含地址空间标识符等信息。TLBSRCH、TLBRD、TLBWR、TLBFILL指令会对这些寄存器进行读写。于是，这些寄存器更新的逻辑需要考虑两种情况：被TLB维护指令更新，或是被之前实验实现的csr写指令更新。TLBEHI在此基础上还需多考虑一种情况，即在发生例外时将触发例外的虚地址记录到其VPPN域中。根据这些TLB相关的控制状态寄存器，便可以产生传递给TLB模块的各种信号，其中需要注意的是，传递给TLB的w\_e信号在TLB充填例外时恒为1而不受TLBIDX的NE位的影响。

在发生TLB重填例外时，处理器核将通过更新DA和PG来进入直接地址翻译模式。TLB重填例外的入口地址储存在TLBENTRY中，该寄存器也只需支持csr指令的读写。在处理完例外后，处理器执行ETRN指令，此时需要让DA=0，PG=1，回到映射地址翻译模式。因此，DA和PG除了要支持csr指令的更新，还需考虑在发生TLB重填例外和执行ERTN时的更新。

* 1. 重要模块3设计：MMU模块
     1. 工作原理

MMU模块完成虚实地址的转换，包括判断映射模式以及查找TLB等操作，同时MMU模块还会输出一系列用于判断TLB相关异常的信号。

* + 1. 接口定义

表2 MMU模块接口定义

| **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sram请求信号 |
| inst\_sram\_wr | OUT | 1 | 指令sram读写控制信号 |
| inst\_sram\_size | OUT | 2 | 指令sram该次请求传输的字节数 |
| inst\_sram\_wstrb | OUT | 4 | 指令sram该次请求的字节写使能信号 |
| inst\_sram\_addr | OUT | 32 | 指令sram请求地址 |
| inst\_sram\_wdata | OUT | 32 | 指令sram该次写请求的写数据 |
| inst\_sram\_addr\_ok | IN | 1 | 指令sram该次请求的地址传输OK |
| inst\_sram\_data\_ok | IN | 1 | 指令sram该次请求的数据传输OK |
| inst\_sram\_rdata | IN | 32 | 指令sram该次请求返回的读数据 |
| axi\_arid | IN | 4 | AXI转接桥的读请求ID号 |
| 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 |

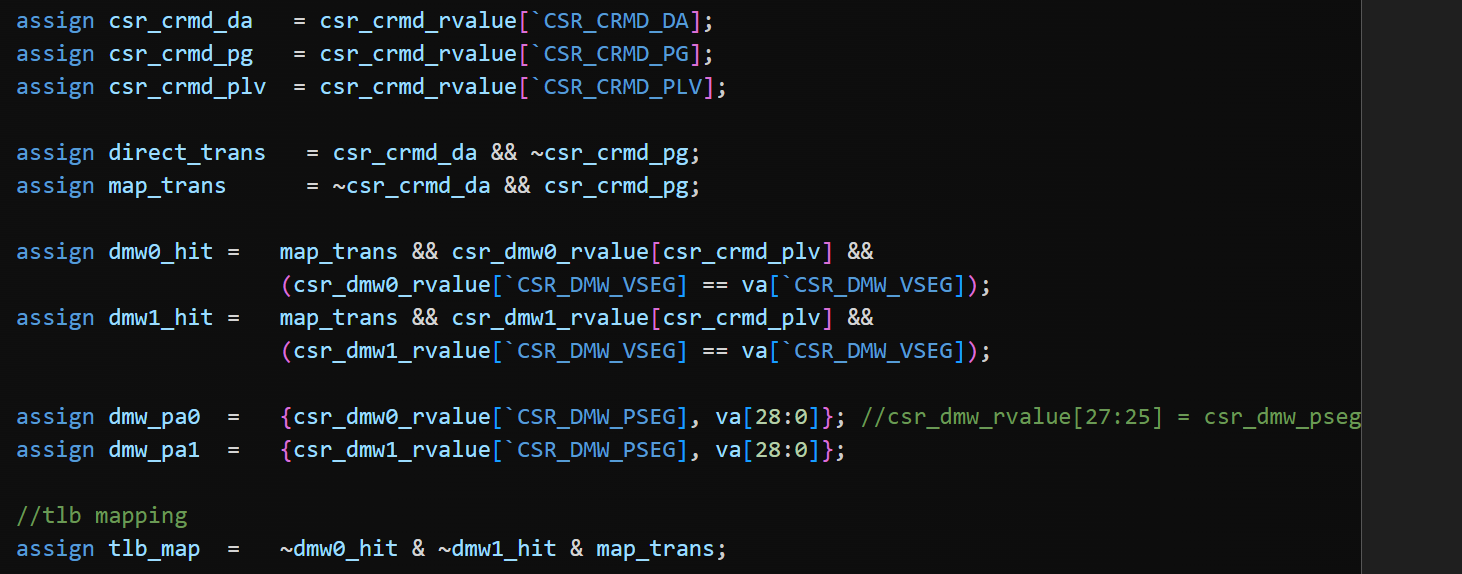
（需要修改。。。。。。。。）

* + 1. 功能描述

MMU模块完成虚实地址的转换，地址翻译模式由CRMD中的DA和PG域控制。当DA=0，PG=1时，处理器核的MMU处于直接地址翻译模式。当DA=0，PG=1时，MMU处于映射地址翻译模式，这种模式又可分为直接映射地址翻译模式和页表映射地址翻译模式。

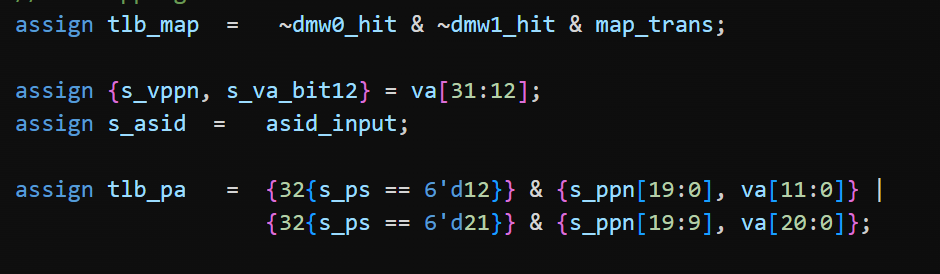
对于直接映射地址翻译模式，相关配置信息存储在DMW0、DMW1寄存器中。每个窗口除了地址范围信息外，还可以配置该窗口在哪些特权等级下可用，以及虚地址落在该窗口上的访存操作的存储访问类型。

当虚地址命中某个有效的直接映射配置窗口时，其物理地址直接等于虚地址的[28:0]位拼接上该映射窗口所配置的物理地址高位。命中的判断方式是：虚地址的最高 3 位（[31:29]位）与配置窗口寄存器中的[31:29]相等，且当前特权等级在该配置窗口中被允许。映射地址翻译模式下，除了落在直接映射配置窗口中的地址之外，其余所有合法地址都必须通过页表映射完成虚实地址转换。



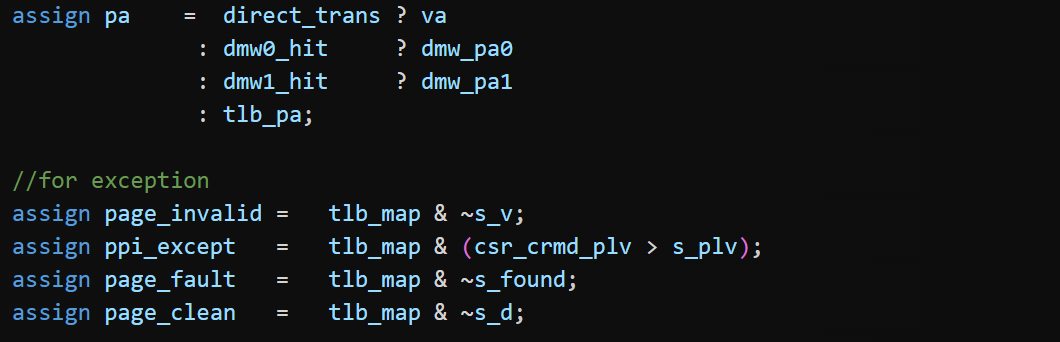
图x MMU模块对于地址翻译模式的判断

对于页表映射地址翻译模式，TLB 作为处理器中存放操作系统页表信息的一个临时缓存，用于加速映射地址翻译模式下的取指和 load/store 操作的虚实地址转换过程。所以只需要把虚拟地址的相关信息传给TLB模块，如果能匹配到页表项，则根据TLB模块返回的PS字段的值，把物理页表项s\_ppn拼接上对应的offset，便可以得到页表映射的物理地址。



图x MMU模块对于页表映射地址翻译模式下的物理地址的产生

如果没有匹配到页表项，则把输出信号page\_fault拉高，方便TLB重填例外的判断。除此之外，MMU模块还会根据TLB模块返回的页表项的V，D，PLV字段的值，对输出信号进行赋值，方便进行load/store取指操作页无效例外、页修改例外、页特权等级不合规例外的判断。

****

图x MMU模块物理地址的产生以及用于例外判断的信号

* 1. 重要模块4设计：IF流水级
     1. 工作原理

IF模块内包含一个伪流水级pre-IF，pre-IF级负责发送请求并等待地址握手，IF级负责接收指令。如果需要取消指令，则将IF级或pre-IF级的对于cancel信号拉高，并把if\_inst\_reg\_valid信号置为0，从而完成对后面返回指令的取消

* + 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sram请求信号 |
| inst\_sram\_wr | OUT | 1 | 指令sram读写控制信号 |
| inst\_sram\_size | OUT | 2 | 指令sram该次请求传输的字节数 |
| inst\_sram\_wstrb | OUT | 4 | 指令sram该次请求的字节写使能信号 |
| inst\_sram\_addr | OUT | 32 | 指令sram请求地址 |
| inst\_sram\_wdata | OUT | 32 | 指令sram该次写请求的写数据 |
| inst\_sram\_addr\_ok | IN | 1 | 指令sram该次请求的地址传输OK |
| inst\_sram\_data\_ok | IN | 1 | 指令sram该次请求的数据传输OK |
| inst\_sram\_rdata | IN | 32 | 指令sram该次请求返回的读数据 |
| axi\_arid | IN | 4 | AXI转接桥的读请求ID号 |
| 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 |

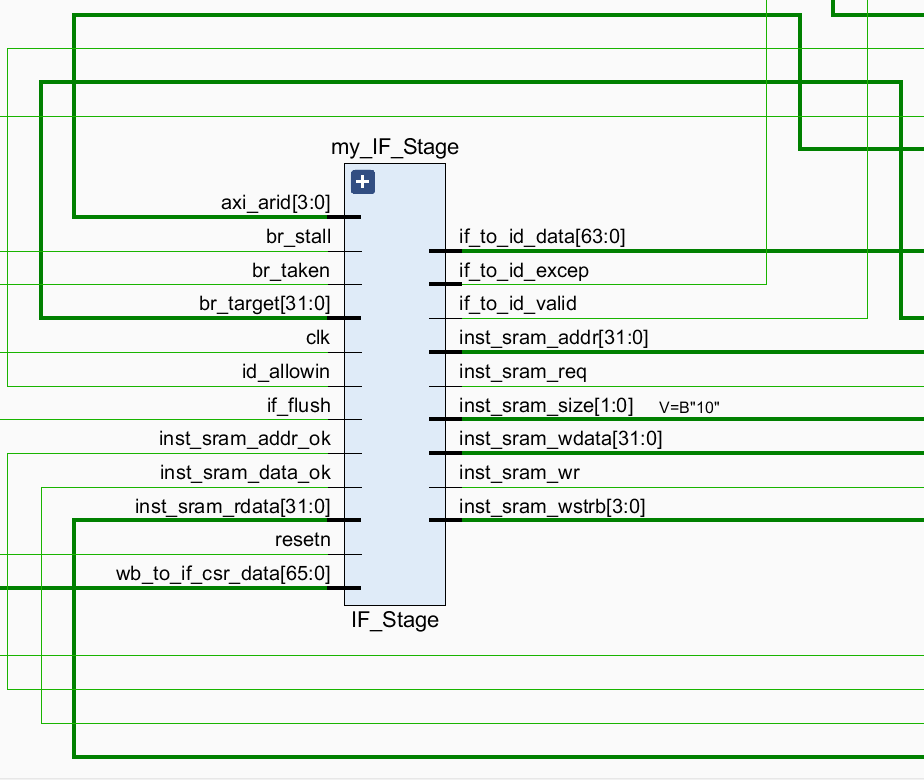
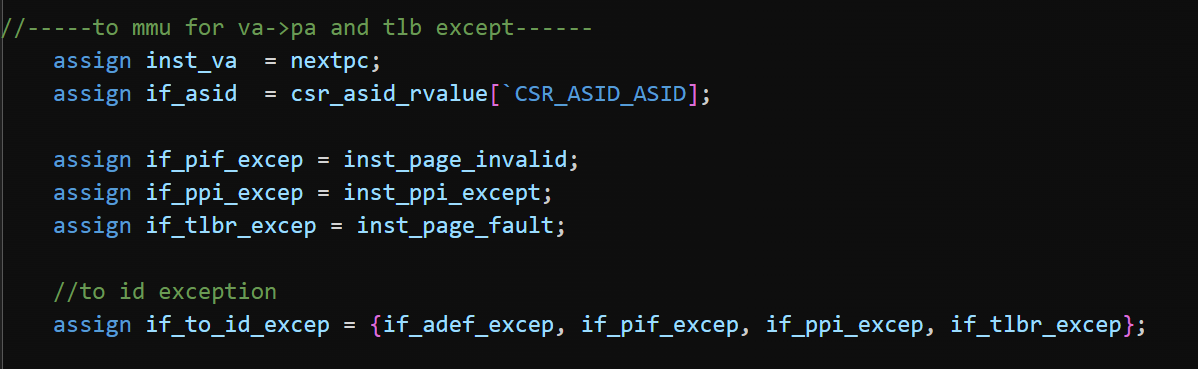


图8 IF模块接口

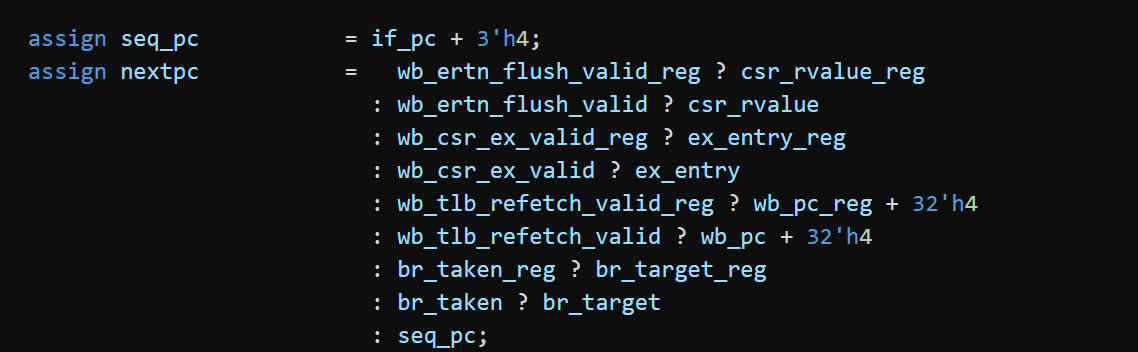
* + 1. 功能描述

在之前实验的基础上，本次实验在IF模块的伪流水级pre-IF中增加了对虚地址nextpc的地址转换功能。为了支持该功能，还需要在其中增添对TLB的查找操作并处理查找过程中出现的例外，然后将这些例外传递给ID级进行下一步操作。而虚实地址的转换以及例外的判断则主要通过MMU模块处理。



图x IF模块的虚实地址转换和TLB例外判断

除此之外，由于新添了流水线的回滚操作，所以将WB流水级的PC传到了IF流水级，用于对nextpc的选择，最终逻辑如下：

****

图x IF模块nextpc更新逻辑的修改

* 1. 重要模块5设计：ID流水级
     1. 工作原理

新增TLB指令的译码，TLB相关例外以及对INE例外的修改。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 |

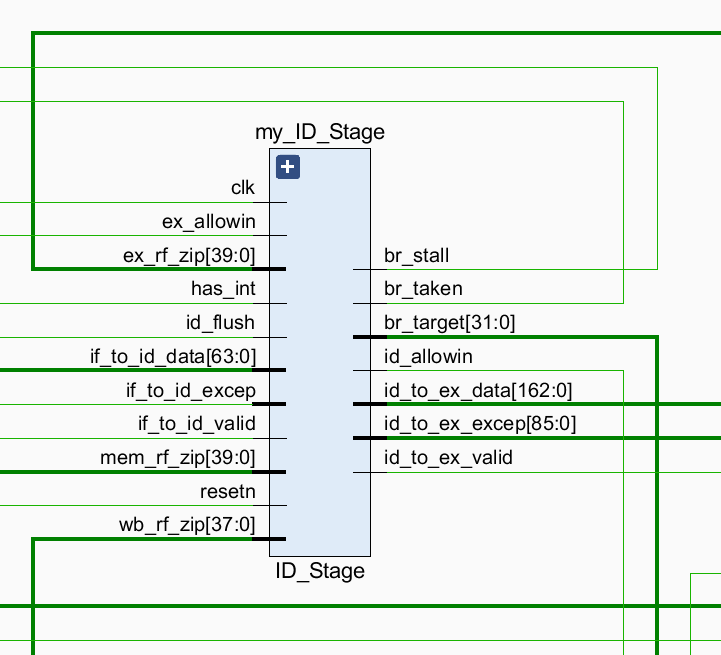


图15 ID流水级接口

* + 1. 功能描述

本实验新增的5条TLB指令在ID级译码，并将控制信号传入EXE级：

另外在之前的实验，INE取指地址错误仅仅处理译码得到的指令不是以及支持的指令，但在本实验还要考虑INVTLB且其op不合法的情况。如果是INVTLB且其op大于6，也要报出INE例外

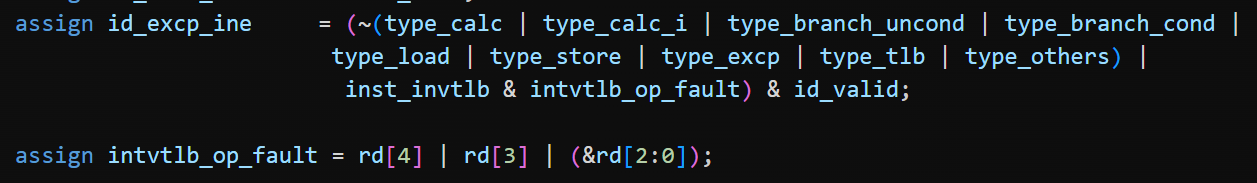


图16 INE例外判断逻辑的修改

* 1. 重要模块6设计：EX流水级
     1. 工作原理

新增虚实地址转换的过程以及TLB例外的判断。根据指令的需要，进行虚实地址转换后向数据RAM发起请求。此外，TLBSRCH在EX级查询TLB，INVTLB也在EX级进行处理。其余功能不变。

* + 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 |

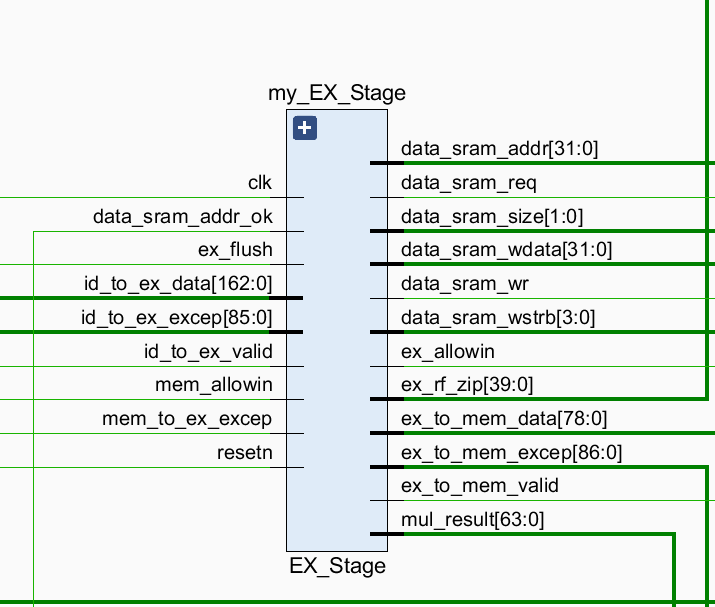


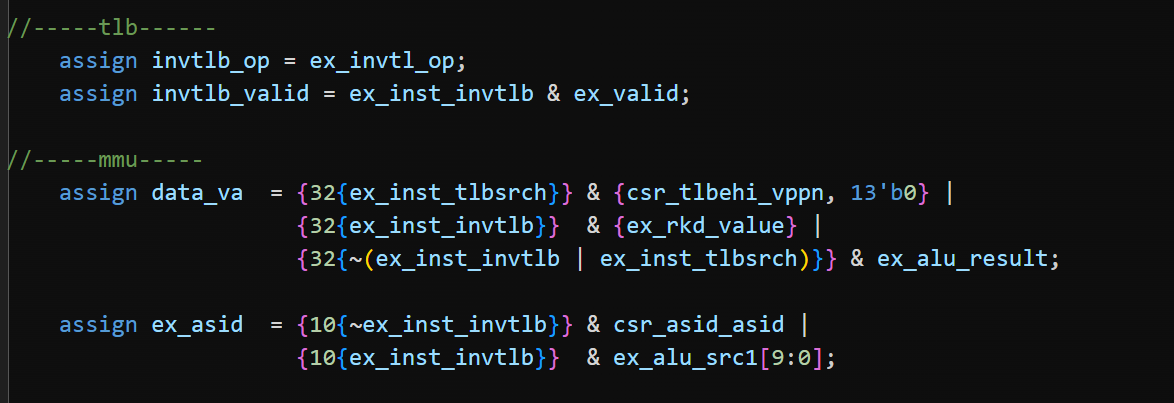
图17 EX流水级接口

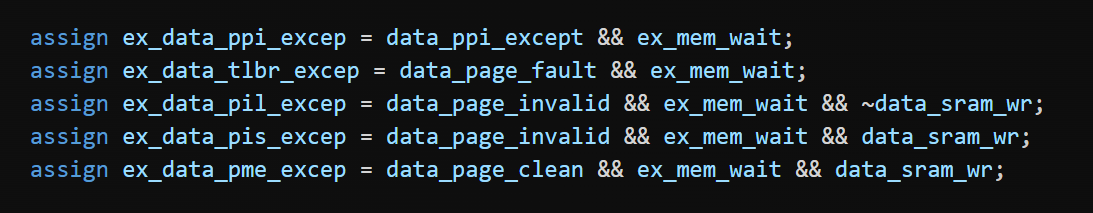
* + 1. 功能描述

在之前实验的基础上，本实验在EXE级中增加了对虚地址es\_alu\_result进行地址转换的功能。和IF级类似，虚实地址转换通过MMU模块进行。此外，在本流水级还需要处理TLB相关例外，也主要结合指令类型和MMU返回信号进行判断。此外，如果在load/store指令中出现了访存虚地址错误或者TLB相关例外，则取消访存，不向总线发送请求。

另外，在EX流水级进行了TLBSRCH的查找TLB操作以及完成对INVTLB操作的处理。对于TLBSRCH指令，会用ASID或TLBEHI中的内容查询TLB，并把返回的index和found信号传递给WB流水级进行进一步处理。

对于INVTLB指令，则把其op传递给TLB模块，并把invtlb\_valid拉高，让TLB模块进行处理。这两条指令均会复用EX流水级对TLB的虚实地址转换的通路：

图x EX流水级对TLB查找的通路的复用



图x EX流水级对TLB相关例外的判断

本实验对ex\_ready\_go也进行了相关修改，当TLBSRCH位于EXE级时，此时如果在MEM流水级或者WB流水级恰好有条修改ASID或TLBEHI的指令，或是TLBRD指令，就会引发围绕csr的数据相关。在这种情况下需要将TLBSRCH阻塞在EXE级，等前面的指令写入完成后再允许其流入MEM级。

* 1. 重要模块7设计：MEM流水级
     1. 工作原理

新增传递给EX流水级的TLB阻塞信号。其余功能不变。

* + 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 |

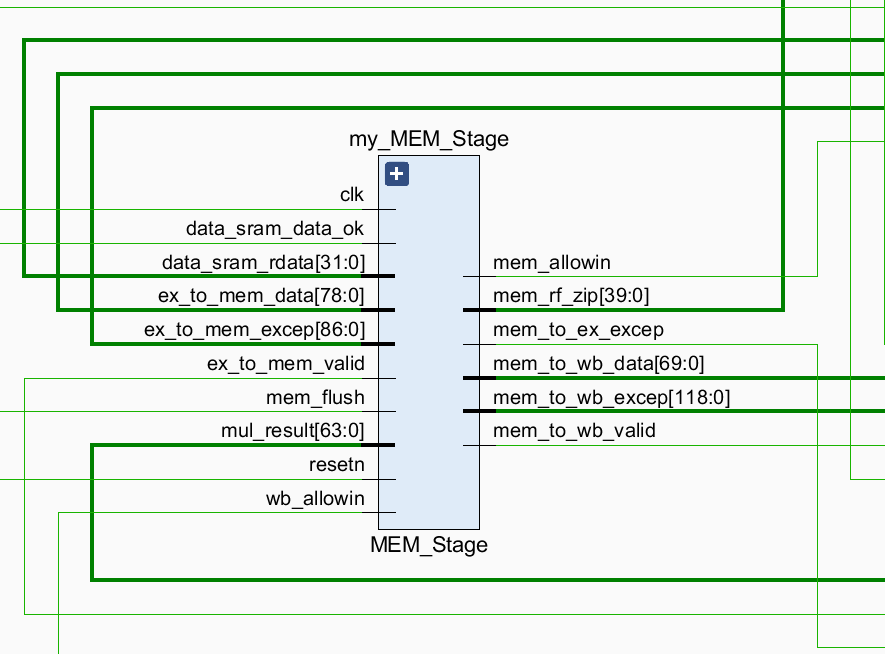


图20 MEM流水级接口

* + 1. 功能描述

新增传递给EX流水级的TLB阻塞信号，当TLBSRCH位于EXE级时，此时如果在MEM流水级恰好有条修改ASID或TLBEHI的指令，或是TLBRD指令，就会引发围绕csr的数据相关。在这种情况下需要将TLBSRCH阻塞在EXE级。

* 1. 重要模块8设计：WB流水级
     1. 工作原理

本次实验中，TLB指令在WB级完成对csr和TLB进行读写。此外，本次实验还新添了回滚操作，若需要回滚，把WB的PC传递给IF，并清空流水线。除此之外，还新增了传递给EX级的TLB阻塞信号。其余功能不变。

* + 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 |

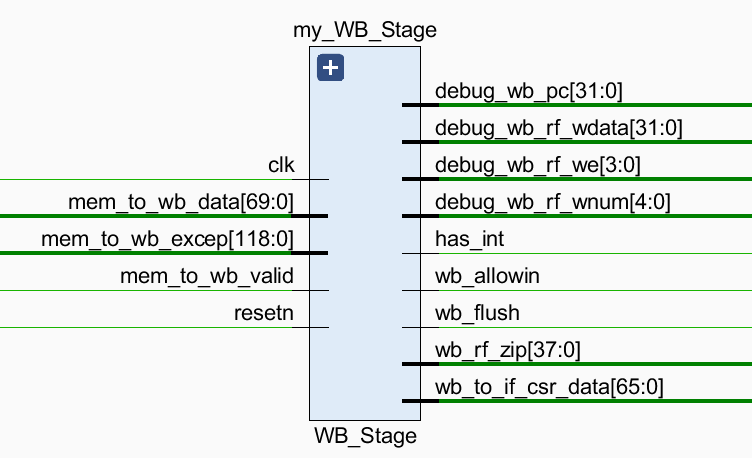
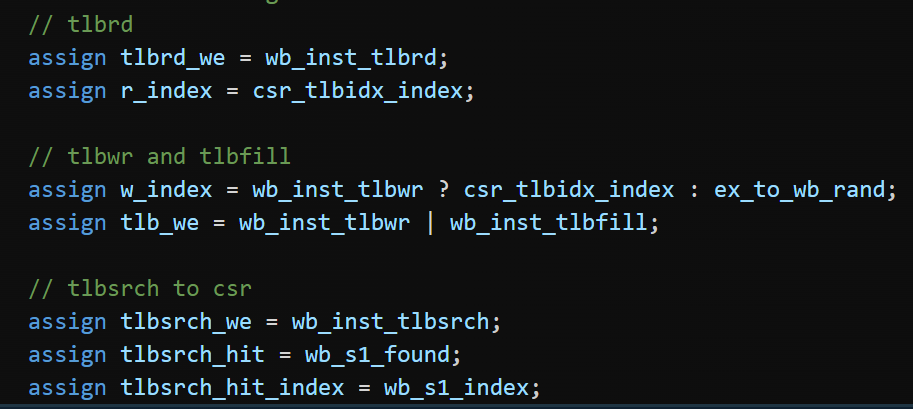


图14 WB流水级接口

* + 1. 功能描述

根据TLB指令的类型产生读写使能和索引，并传入TLB。如果是TLBSRCH指令，则将在EXE级获得的查询结果写入CSR。对于TLBFILL指令，其会随机选择一个TLB项进行写入，此时复用EX流水级的计数器的第四位作为写入的TLB项的索引，实现随机的效果。



图x WB级TLB指令对TLB和CSR的读写

根据从上一级流入的重取标志和例外相关的控制信号来进行相应处理。如果重取标志为1，则需要刷新流水级，为此将重取标志加入ws\_reflush。如果发生TLB重填例外，例外入口地址的来源是从csr中读出的ws\_tlb\_entry，其余情况下依然是ws\_entry。

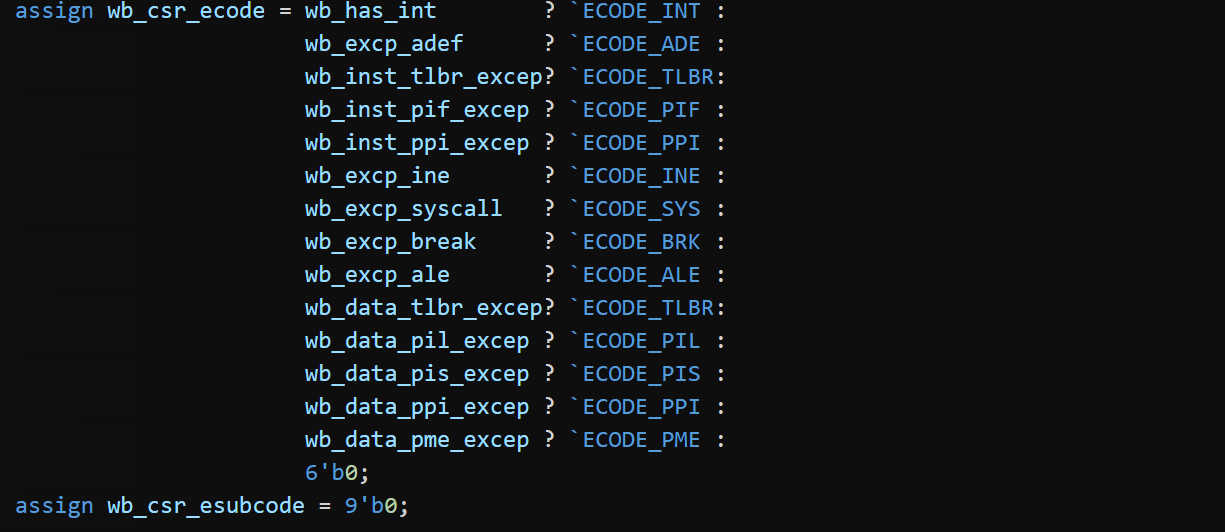
对于TLBWR、TLBFILL、TLBRD、INVTLB以及更改ASID或者CRMD寄存器的csr写指令，为了解决数据相关，本组成员的设计是刷新流水级并重新取指，直接复用在之前实验的wb\_flush信号即可。然后把wb\_pc传递给IF流水级，让的下一个取值PC改为此时WB流水级的PC+4。



图x WB级的回滚操作

除此之外，当TLBSRCH位于EXE级时，如果在WB流水级恰好有条修改ASID或TLBEHI的指令，或是TLBRD指令，就会引发围绕csr的数据相关。在这种情况下需要将TLBSRCH阻塞在EXE级。

WB模块还会根据前面模块传递过来的异常类型信息为csr\_ecode和csr\_esubcode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。而在IF级触发的例外中，ADEF例外的优先级高于TLB相关例外；在EX流水级触发的例外中，ALE例外的优先级高于TLB的相关例外；在TLB相关例外中，TLBR的优先级最高，然后为PIF或者PIL或者PIS，再其次为PPI，最后为PME。因此使用如下代码逻辑进行赋值：



图x csr\_ecode和csr\_esubcode赋值

1. 实验过程
   1. 实验流水账

李金明

2023.11.12 21：00-22：00 阅读讲义

2023.11.13 8：00-12：00 完善exp14代码

2023.11.13 12：20-15：00 18：00-21：00 完成exp14的debug

2023.11.17 14:00-15:00 阅读讲义

2023.11.17 15:00-22:00 完成exp15代码

2023.11.17 22:00-次日00:30 exp15的debug

2023.11.18 8:30-11:40 完成exp15的debug

贾城昊

2023.11.13 8:00-9:00 阅读讲义

2023.11.13 10:00-12:00 完善exp14代码

2023.11.13 15:00-21:00 完成exp14debug

2023.11.18 16:00-17:00 阅读讲义

2023.11.18 17：00-23：00 进行exp16的debug

2023.11.19 8：00-12：00 完成exp16的debug

牛浩宇

2023.11.9晚 阅读讲义&增加exp14接口并完成大致逻辑

2023.11.16晚 阅读讲义&增加exp15接口与转接桥。

* 1. 错误记录
     1. 错误1：exp14中跳转指令目标地址指令可能被跳过
        1. 错误现象

Console报错如下：

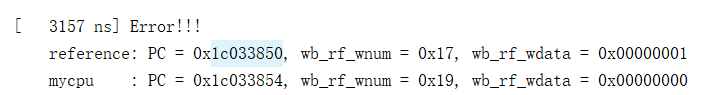


图23 错误1对应的Console报错

* + - 1. 分析定位过程

查看波形图及相关的汇编代码，可知错误发生在跳转指令目的地址的指令，其指令值没有被取出来就被跳过了，转而执行了目的地址的下一条指令。



图24 错误1对应的波形图



图25 错误1对应的汇编代码（1）



图26 错误1对应的汇编代码（2）

* + - 1. 错误原因

这里会出错是因为当有一条跳转指令时，跳转指令的下一条指令也会被送到inst\_sram并发出请求。虽然跳转指令此时已经在ID阶段，可以判断出存在跳转，需要取消下一条指令，但是需要被取消的那条指令以及发出请求并且可能在当拍返回了addr\_ok，此时当inst\_sram\_raddr切换到跳转指令目的地址后，可能会返回之前被取消指令地址读出的数据，且data\_ok拉高会让跳转指令目的地址的那条指令误以为取到了对应的指令而跳过。因此需要修改next\_pc的赋值。

* + - 1. 修正效果

增加代码：

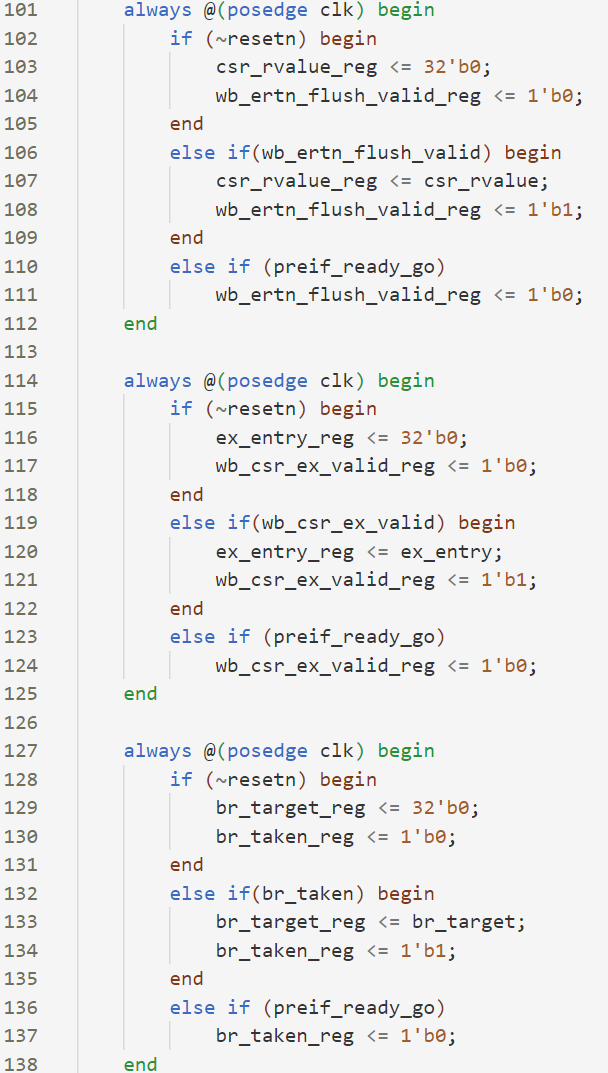


图27 错误1对应的修改代码（1）

修改next\_pc的赋值逻辑为：

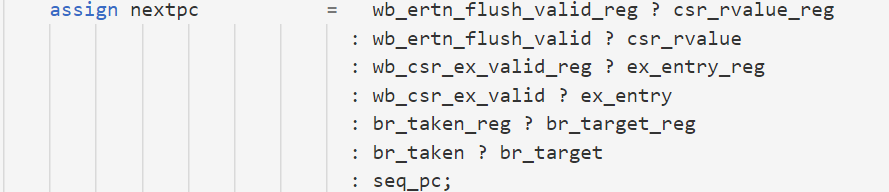


图28 错误1对应的修改代码（2）

此处逻辑为当有例外、ertn指令或跳转时，则将跳转地址存入一个寄存器并将有效位置为1，保证在addr\_ok前请求地址都是csr\_rvalue、ex\_entry或是br\_target，避免在指令地址握手成功前next\_pc就又被更新了。

该方法有效，来到下一个bug。

* + 1. 错误2：exp14中当其他流水级阻塞拍数过多时除法操作没有及时停止
       1. 错误现象

Console报错如下

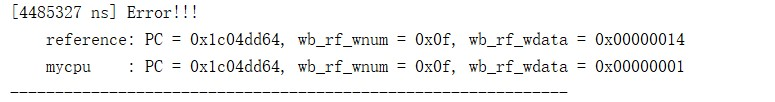


图29 错误2对应的Console报错

* + - 1. 分析定位过程

查阅汇编代码可知，出错的是一条除法指令。查看执行该除法指令的过程，波形图如下：

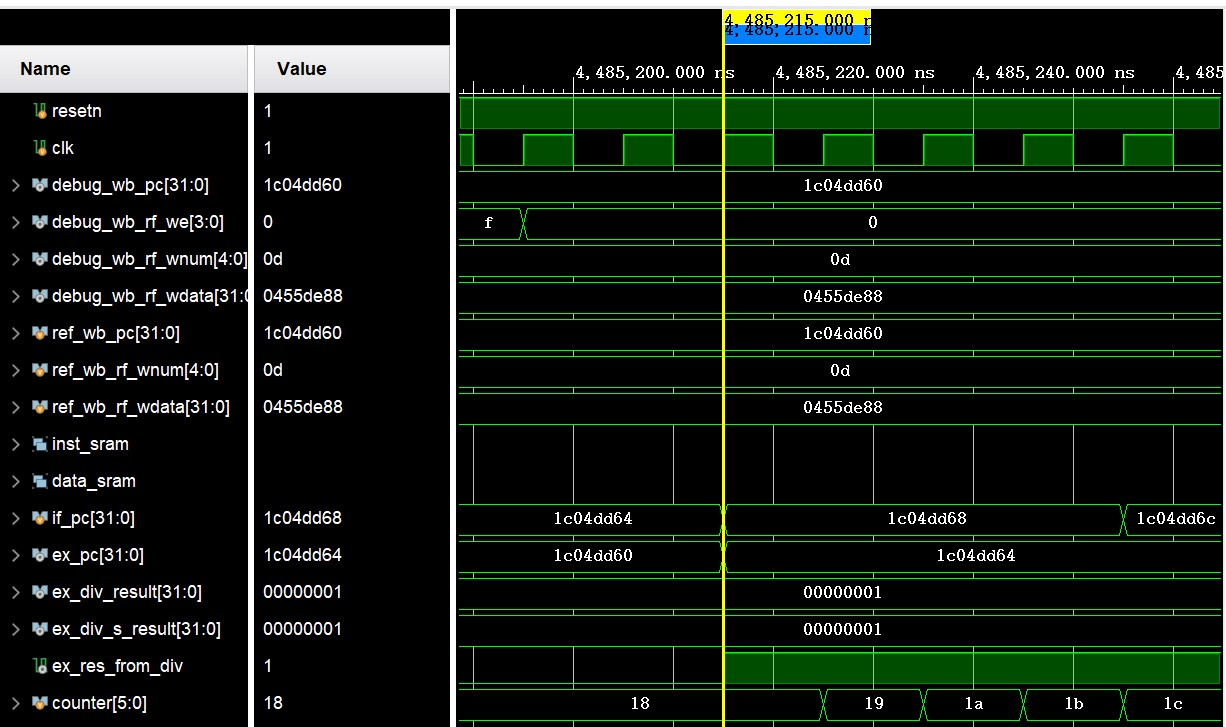


图30 错误2对应的波形图（1）

可以发现除法器对执行时钟周期计数的计数器初值不为0，导致除法器错误。查看上一次执行除法指令相关的波形图：

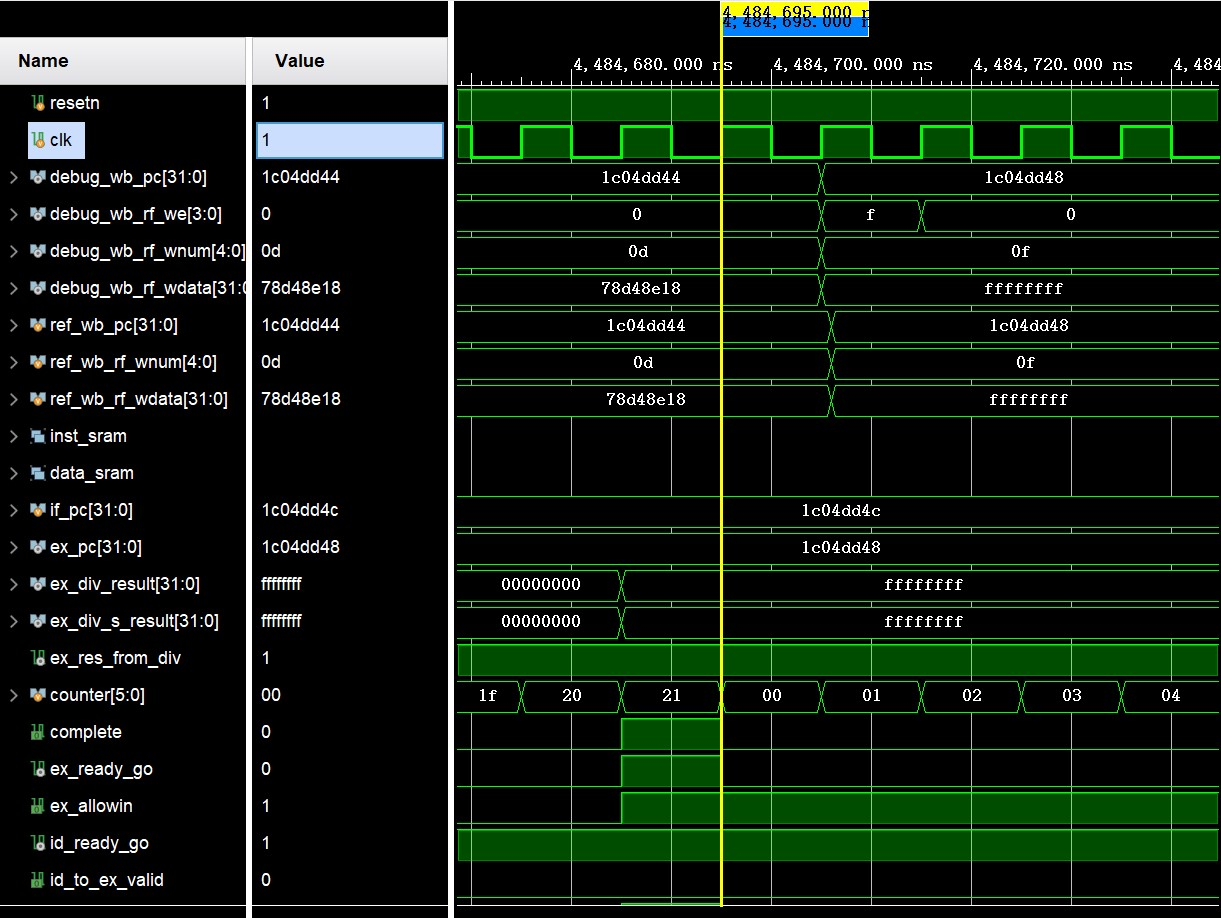


图31 错误2对应的波形图（2）

可以发现，当counter达到最大值0x21时，本应说明此时除法已经执行完毕，但是counter又从0开始加，没有结束。

* + - 1. 错误原因

除法器没有正常结束工作是因为某个流水级出现了多于33拍的阻塞，此时由于被阻塞，EX模块中传入除法器的div信号还是拉高的，导致除法器继续工作。

* + - 1. 修正效果

使用一个寄存器作为除法指令已经完成的标志，当除法器工作完成后，通过该标志将传入除法器的div信号拉低，避免除法器过度工作。修改代码如下：

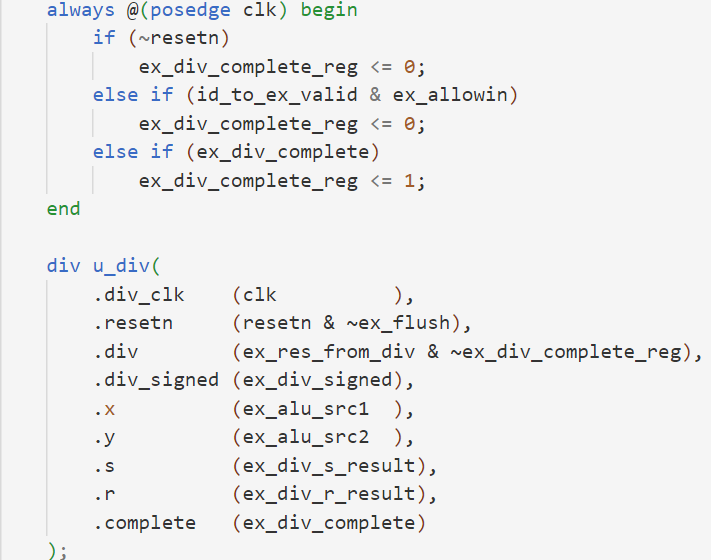


图32 错误2修改代码

该方法有效，来到下一个bug。

* + 1. 错误3：exp14中data\_sram\_size传入位宽出错
       1. 错误现象

Console报错如下

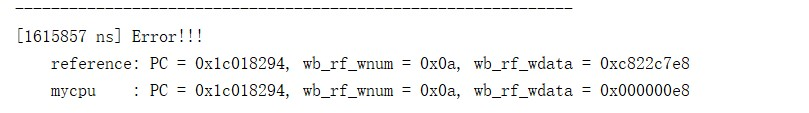


图33 错误3对应的Console报错

* + - 1. 分析定位过程

通过查看汇编代码，可以知道出错的是一条load指令，因此查看该条指令在EX阶段向data\_sram传送的请求信号，相关波形图如下：



图34 错误3对应的波形图

可以发现，出错的是一条ld.w指令，向sram传输的data\_sram\_size的值本应为2，但是此处为0。查看data\_sram\_size赋值相关代码：

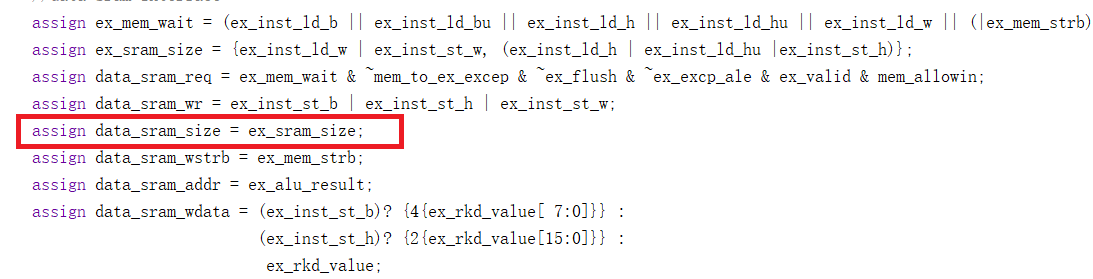


图35 错误3对应的错误代码（1）

可知出错的是EX模块中定义的ex\_sram\_size。查看这个信号赋值及定义，发现其定义位宽出错：

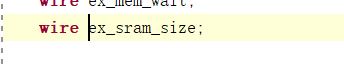


图36 错误3对应的错误代码（2）

* + - 1. 错误原因

信号data\_sram\_size的位宽为2，但是新定义了一个为其赋值的信号ex\_sram\_size的位宽为1，导致data\_sram\_size本应为2，却被赋值成了0

* + - 1. 修正效果

将ex\_sram\_size的位宽改为2，该方法有效，来到下一处bug。

* + 1. 错误4：exp14中inst\_cancel的更新逻辑有误
       1. 错误现象

运行仿真后，console报错如下：

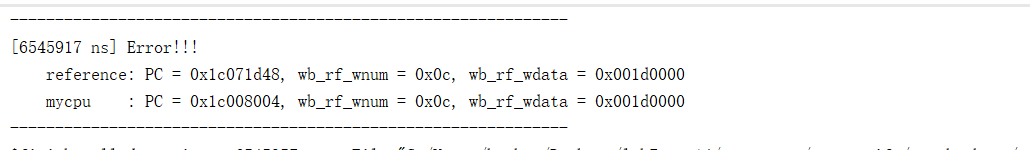


图37 错误4对应的Console报错

* + - 1. 分析定位过程

可以看出，PC发生了错误，很容易想到可能是inst\_cancel的问题，但是首先查看波形图，如下所示：



图38 错误4中断异常到来时inst\_cancel未正常拉高

可以看到，此时if\_flush信号拉高，代表进入了中断异常，此时请求握手成功但是数据还未返回，需要对取指进行取消，但是此时inst\_cancel却没有正常拉高，导致后面传给ID流水级的PC与返回的指令不对应而出错。查看inst\_cancel的的更新逻辑(此时本组还没有加inst\_cancel\_num信号)，如下所示：

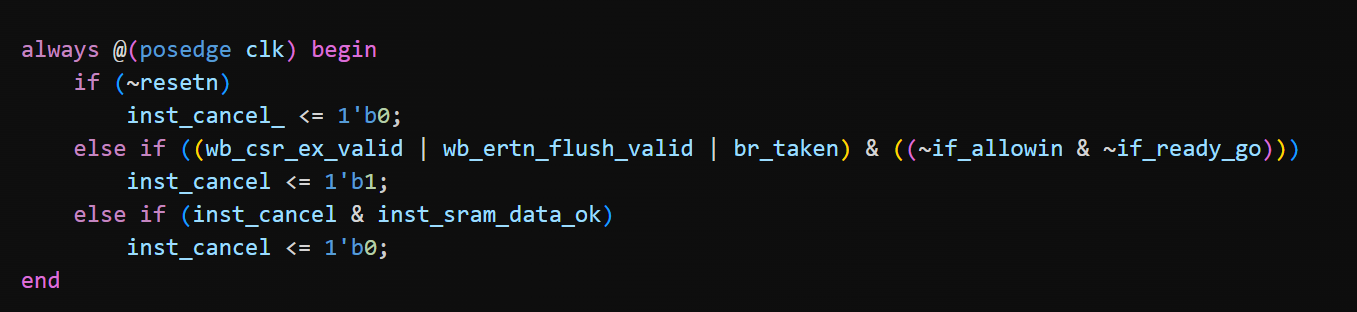


图39 错误4对应的错误代码（inst\_cancel更新逻辑）

可见此时inst\_cancel未拉高的原因是if\_allowin拉高，这是因为本组设计的是if\_allowin在if\_flush为1时拉高，代表此时IF流水级的工作已经无效，允许preIF把nextpc送过来。所以if\_allowin的逻辑是相对正确的，这时候，可以考虑到inst\_cancel之所以加上~if\_allowin，最初的本意是为了兼容if\_valid拉低的情况，所以直接把~if\_allowin改为if\_valid，便可以解决这个问题。

* + - 1. 错误原因

inst\_cancel的更新逻辑有误，判断逻辑之间存在相互冲突的地方，导致进入中断异常时，错误的取指的返回值没有被取消掉，进而导致传递给ID的PC与返回的指令不对应，出现问题。

* + - 1. 修正效果

修改inst\_cancel的更新逻辑，用if\_valid信号替换~if\_allowin，使得进入中断异常且需要取消取指的情况下，if\_cancel能正常拉高，如下所示：

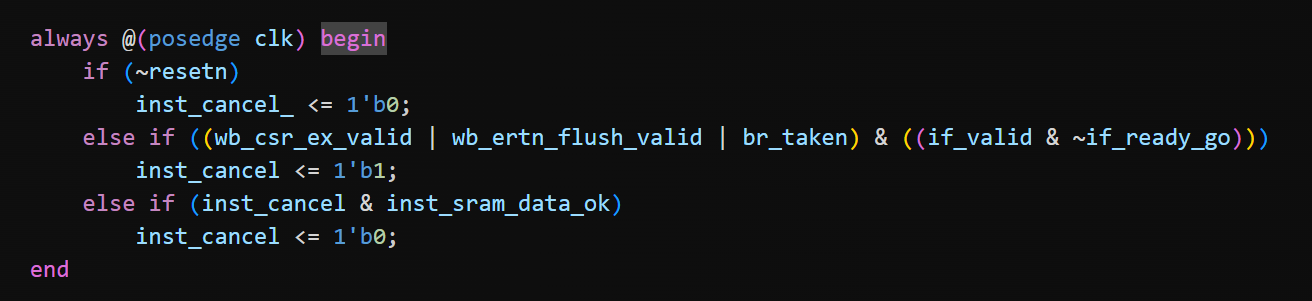


图40 错误4修改代码

该方法有效，来到下一个bug。

* + 1. 错误5：exp14中mem\_ready\_go信号逻辑出错
       1. 错误现象

运行仿真后，console报错如下：

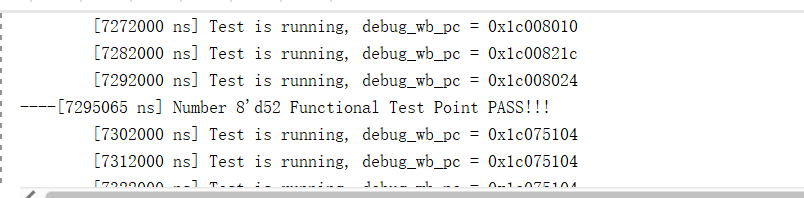


图41 错误5对应的Console报错

* + - 1. 分析定位过程

可见此时PC一直没有得到更新，导致CPU进入死循环，只能查看波形图观察是哪一个流水级的阻塞出现了问题，如下所示：

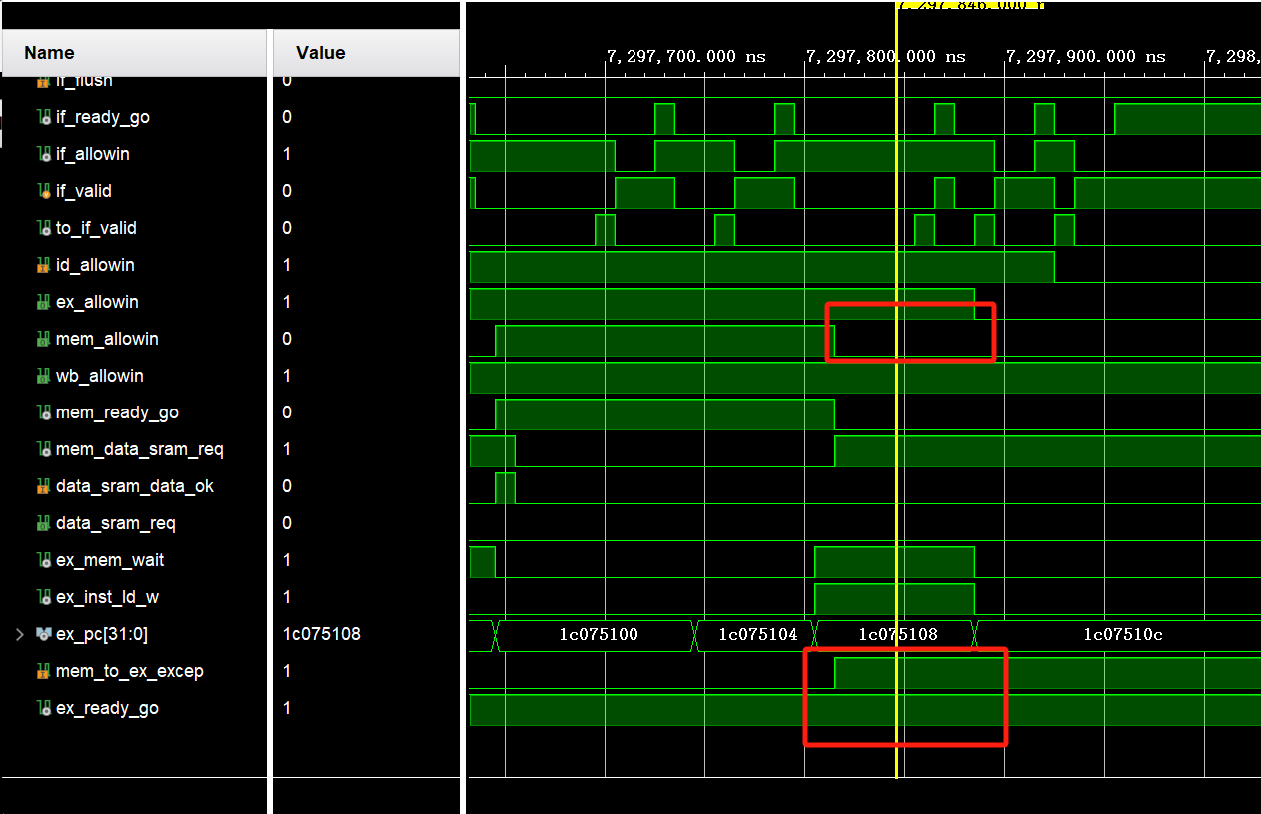


图42 错误5中MEM流水级错误阻塞

可以看到，mem\_allowin一直拉低，但ex\_ready\_go拉高，说明是MEM流水级的阻塞逻辑出现了问题，接着观察波形图，查看为啥阻塞出现问题，如下所示：

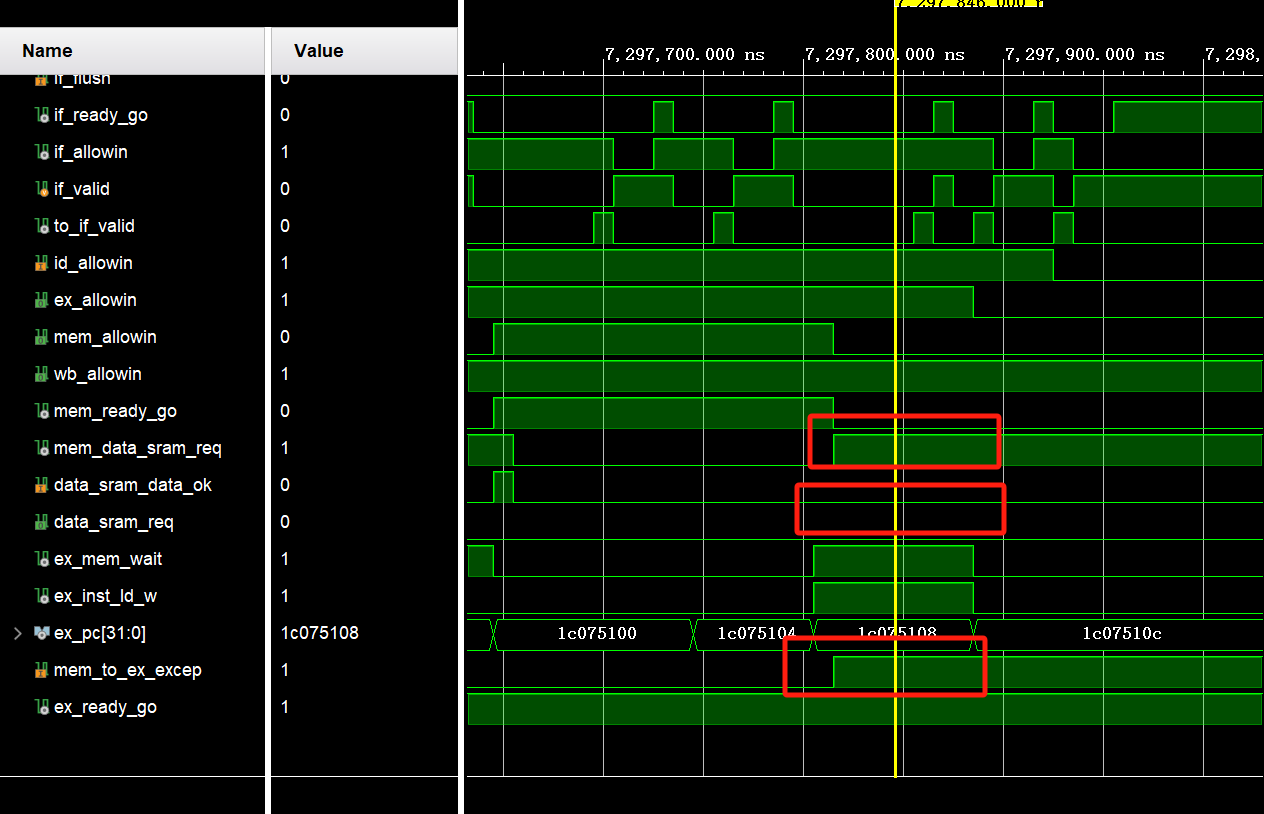


图43 异常到来EX未发请求，但MEM流水级错误阻塞等待数据返回

我们可以观察到，此时mem\_to\_ex\_excep信号拉高，代表着上一条指令发生了中断异常，所以此时EX流水级的data\_sram\_req不会发送请求，没有拉高，但是MEM从EX接受到的mem\_data\_sram\_req却拉高了，查看代码，这是因为mem\_data\_sram\_req对应的信号是EX的ex\_mem\_wait，其赋值逻辑如下：

（注：此时，本小组的信号命名还有些不规范，这也是debug环节对信号含义不断修改所导致的）

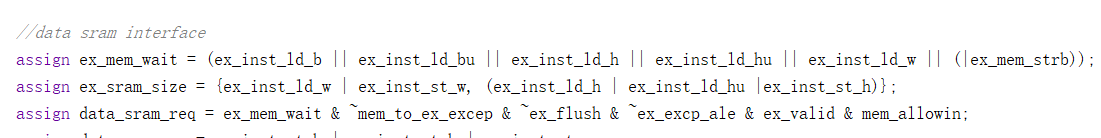


图44 错误5中ex\_mem\_wait信号未考虑中断异常

可以看到，ex\_mem\_wait没有考虑到异常中断或是流水级的刷新，所以此时EX流水级没有发req信号，但MEM错误的认为需要阻塞等待数据返回。

* + - 1. 错误原因

MEM流水级错误的阻塞，这是因为EX流水级传递给MEM流水级用于阻塞的ex\_mem\_wait信号没有考虑到异常中断或是流水级的刷新，所以此时EX流水级没有发req信号，但MEM错误的认为需要阻塞等待数据返回。

* + - 1. 修正效果

把EX流水级传递给MEM流水级用于阻塞的ex\_mem\_wait信号加上对异常中断或是流水级的刷新的考虑即可，如下所示：

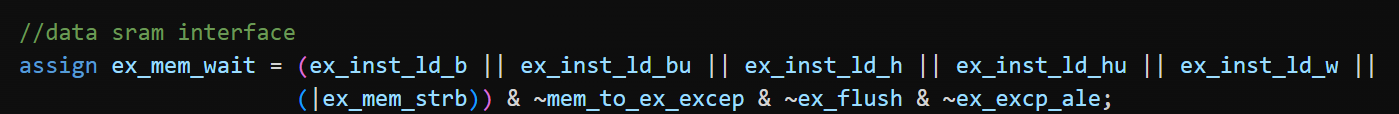


图45 错误5修改代码

该方法有效，来到下一个bug。

* + 1. 错误6：exp14中未设置inst\_cancel\_num而无法取消多条指令
       1. 错误现象

在解决前面的错误后，此时仿真已经能够通过了，但是上板后，修改种子进行测试时，发现有的种子无法通过，于是修改为对应的种子，再次进行仿真，console报错信息如下：

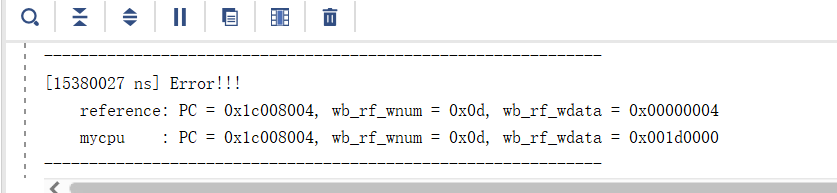


图46 错误6对应的Console报错

* + - 1. 分析定位过程

可以看到此时rf\_wdata出错，鉴于之前的经验，可能是指令取消出错，导致传递给ID的PC与指令不对应，查看波形图如下所示：

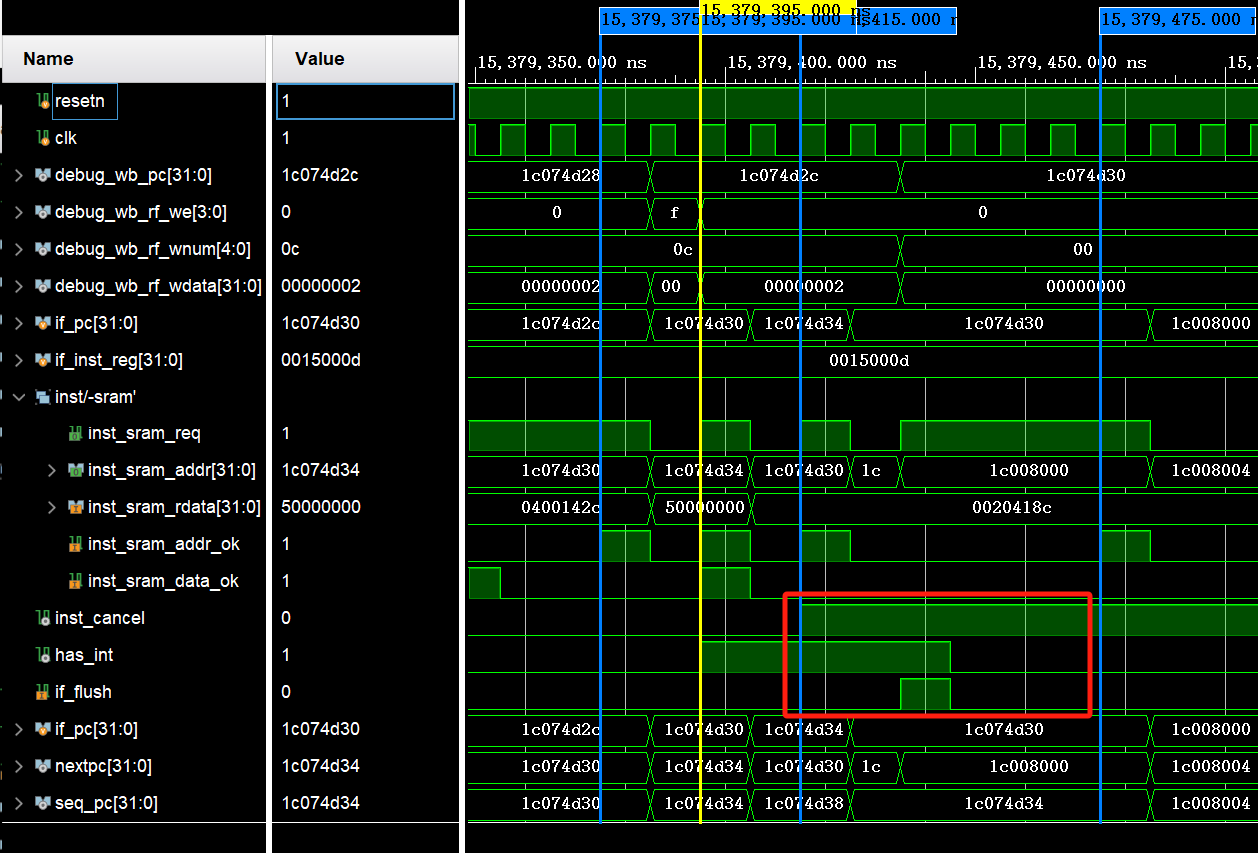


图47 错误6 中inst\_cancel无法取消多个指令返回

在仔细对应data\_ok与addr\_ok的关系后，发现果然是指令取消出现了问题，如上图所示，在if\_flush信号拉高时，inst\_cancel已经拉高了。这时inst\_cancel拉高的原因是前面出现了br\_taken。所以从逻辑上来说，此时有两个指令返回需要取消，但由于inst\_cancel只是一个1位寄存器信号，其一次最多只能取消一条指令的返回，所以才会导致后面指令没有被正确取消，进而导致传递给ID的PC和指令不对应。

* + - 1. 错误原因

未设置inst\_cancel\_num信号来取消多条指令，此时inst\_cancel只是一个1位寄存器信号，其一次最多只能取消一条指令的返回，导致后面指令没有被正确取消，传递给ID的PC和指令不对应。

* + - 1. 修正效果

新增inst\_cancel\_num寄存器，用来存储需要取消指令的数量，inst\_cancel信号则根据inst\_cancel\_num生成，代码修改如下：

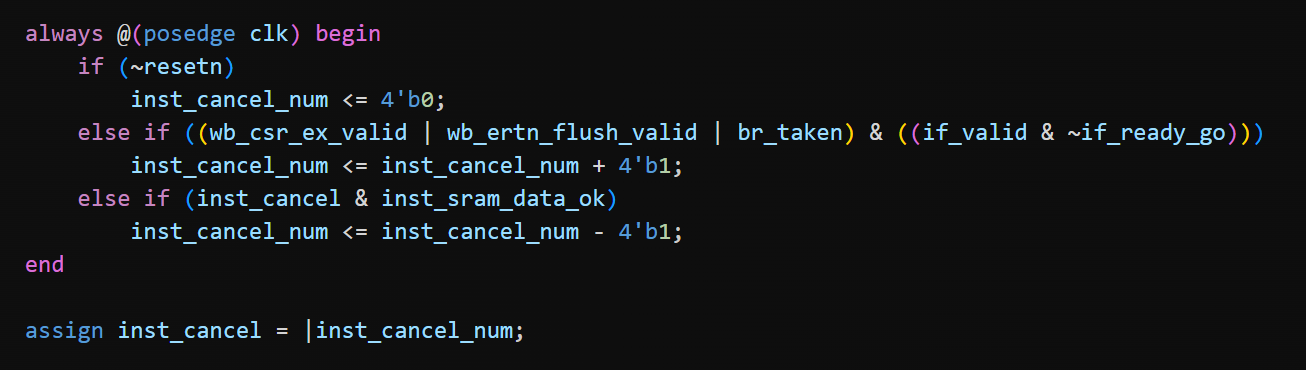


图48 错误6修改后代码(新增inst\_cancel\_num)

该方法有效，exp14仿真和上板均通过。

* + 1. 错误7：exp15中读响应状态机状态转移条件出错
       1. 错误现象

传递的PC值一直是0xxxxx\_xxxx

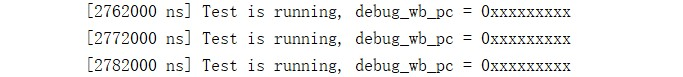


图49 错误7对应的Console界面

* + - 1. 分析定位过程

查看波形图可知第一条指令一直卡在了IF阶段，因此查看AXI转接桥的工作状态：

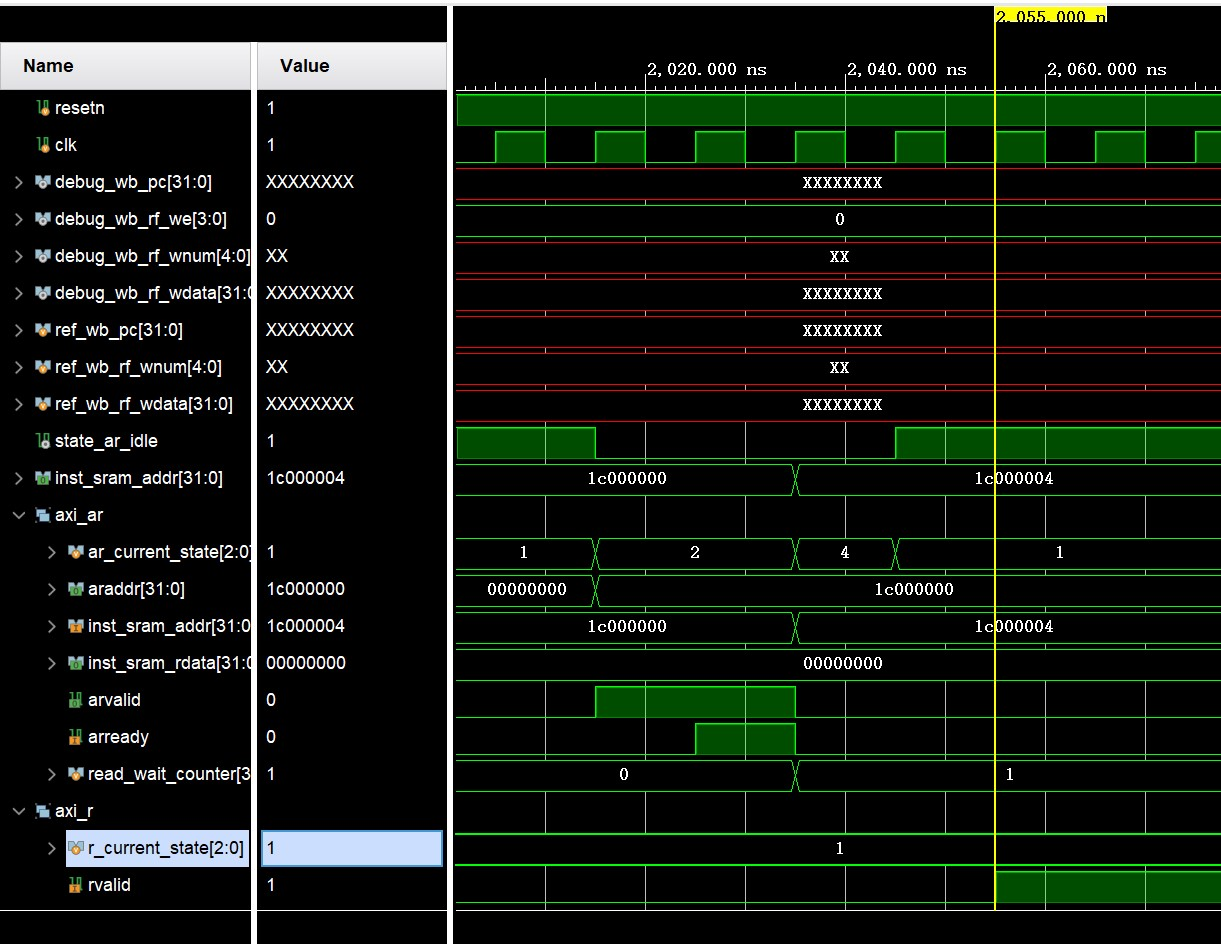


图50 错误7对应读请求通道和读响应通道的波形图

通过波形图可以看出读请求状态机正常工作了，但是读响应状态机一直停在IDLE状态。因此查看读响应状态机的状态转移逻辑：

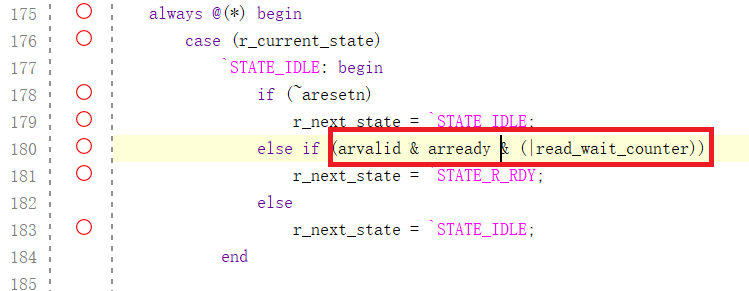


图51 错误7读响应状态机状态转移条件

* + - 1. 错误原因

读响应状态机从IDLE状态转移到R\_RDY状态的条件应是读请求地址握手成功**或**有读请求地址已握手成功但还未响应的读操作。因此状态转移条件应是arvalid & arready | (|read\_wait\_counter)。另外，原代码不能正常工作的原因还有read\_wait\_counter是一个寄存器，其会在arvalid和arready同时拉高的下一个周期加1，因此这个条件一直不会满足。

* + - 1. 修正效果

将读响应状态机在IDLE状态的状态转移机制修改如下：

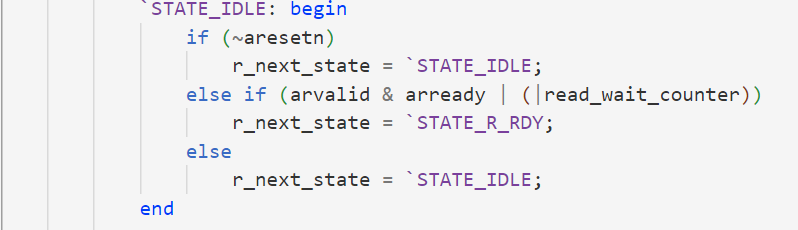


图52 错误7修改代码

该方法有效，来到下一个bug。

* + 1. 错误8：exp15中跳转指令后一条指令未取消
       1. 错误现象

Console报错如下：

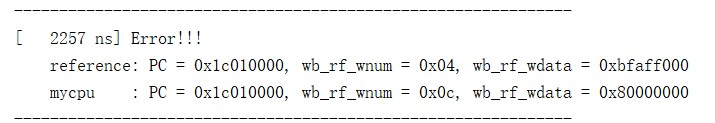


图53 错误8对应的Console报错

* + - 1. 分析定位过程

查看相关汇编代码得知错误出现在一条跳转指令后，相关波形图如下：

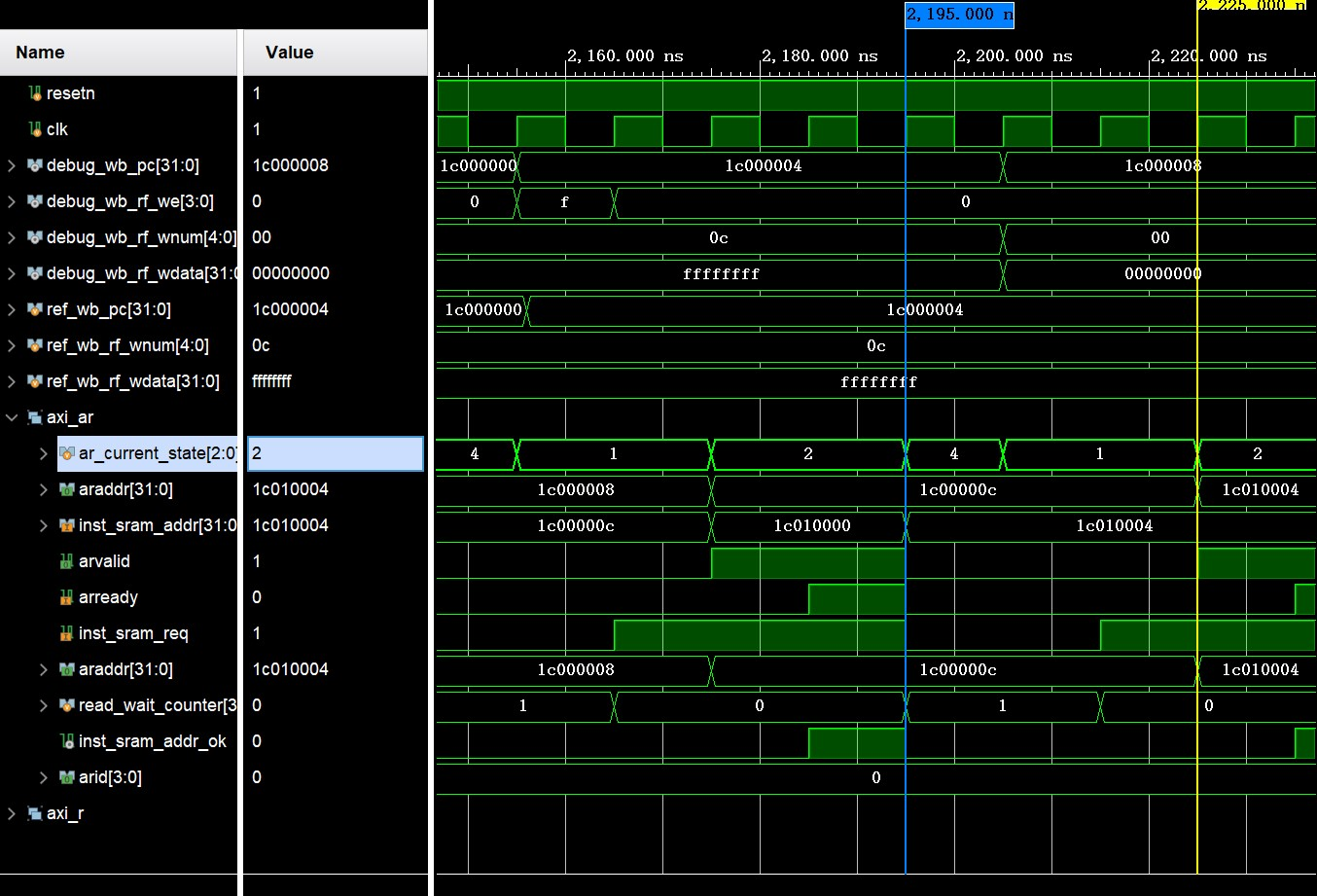


图54 错误8对应的波形图

可以发现CPU提交了对跳转指令下一条指令对应地址的请求，且此时读请求通道处于空闲状态，其接受了请求。由于跳转指令，inst\_sram\_addr发生了改变，但没有提交给读请求通道。

* + - 1. 错误原因

在程序中，0x1c000008是一个跳转到0x1c010000的指令，在这条指令译码时，它的下一条指令会发送对指令地址的请求，由于读请求通道此时处于空闲状态，会接受请求。而由于AXI总线“发送请求后不能更改请求地址”的特性，由于跳转变化的inst\_sram\_addr没有发送请求。当AXI总线返回addr\_ok时，实质上是对0x1c00000c这个地址的确认，而CPU误认为是对0x1c010000这个地址的确认，导致出错。

* + - 1. 修正效果

对跳转等情况加以讨论。若此时出现跳转、异常、ertn指令，且此时有指令已经发出了地址请求，则需要取消这条指令。在exp14中可以在发送请求的过程中更改请求地址，因此没有考虑指令在pre-IF阶段的取消，现在需要加上这种情况，修改代码如下：

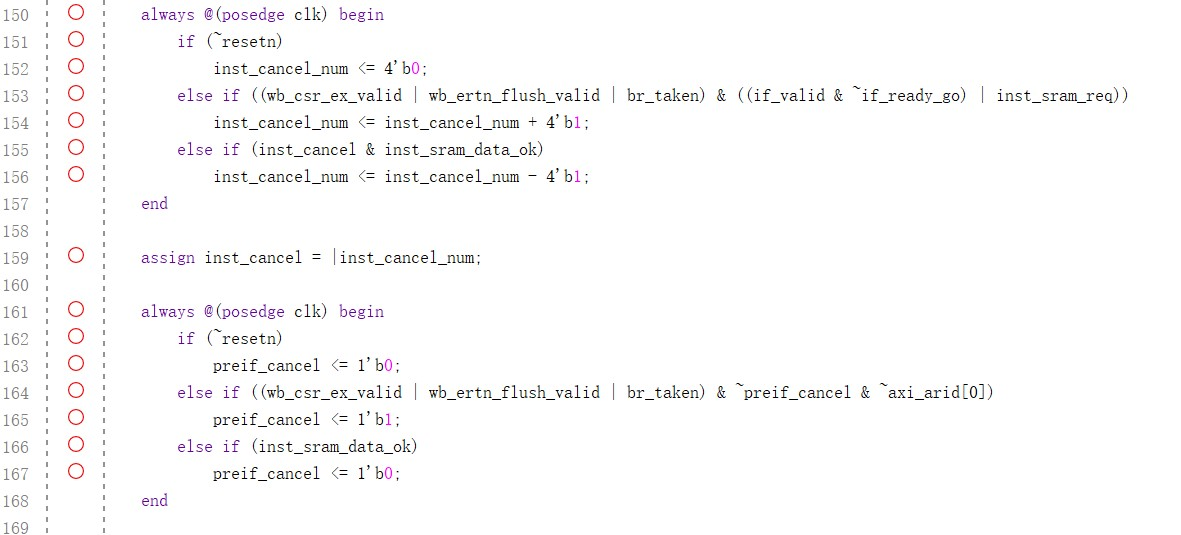


图55 错误8修改代码：增加pre-IF信号



图56 错误8修改代码：修改to\_if\_valid信号复制条件

该方法有效，来到下一处bug。

* + 1. 错误9：exp15中写响应状态机状态转移
       1. 错误现象

程序不会停下，Console界面如下：

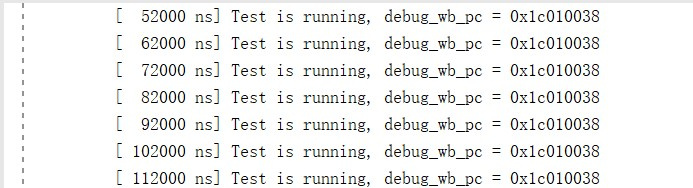


图57 错误9对应的Console界面

* + - 1. 分析定位过程

查看汇编代码，得知出错的指令是store指令。首先查看处理这条store指令时读请求和读数据通道、读响应通道状态机工作的波形图：

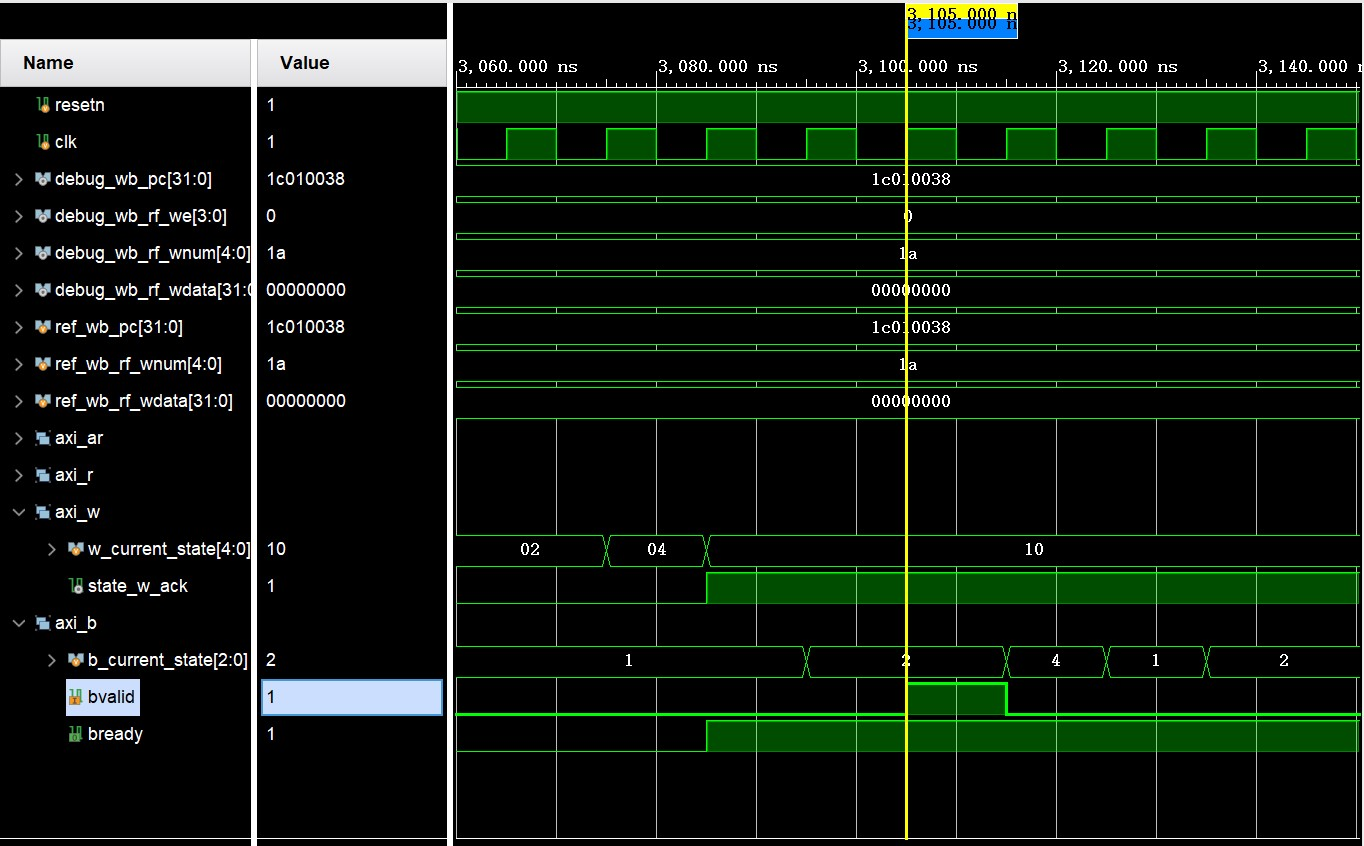


图58 错误9对应写请求和写数据通道、写响应通道状态机的波形图

可以发现写请求和写数据通道状态机一直卡在了W\_ACK状态，查看相关代码：

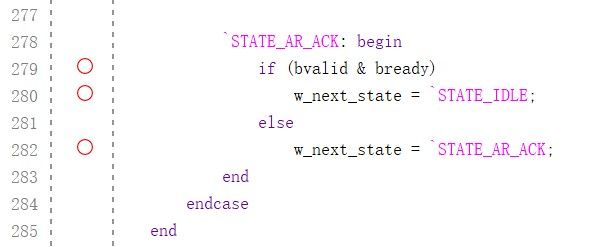


图59 错误9对应写请求和写数据通道状态机状态转移代码

* + - 1. 错误原因

可以发现在写请求和写数据状态机中，我们对状态判断出错：与写请求和写数据状态机对应的应当是W\_ACK，而不是AR\_ACK

* + - 1. 修正效果

将图中代码的AR\_ACK改为W\_ACK，修改如下：

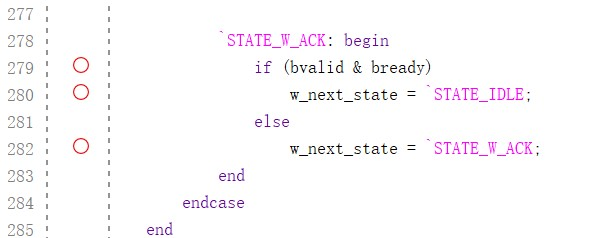


图60 错误9修改后代码

此方法有效，exp15通过。

* + 1. 错误10：exp16中preif\_cancel与inst\_cancel信号的复位逻辑出错
       1. 错误现象

在exp15通过后，上板发现只有无延迟的种子能过通过，其它种子均会出错，于是修改种子，再次运行仿真后，console报错如下所示：

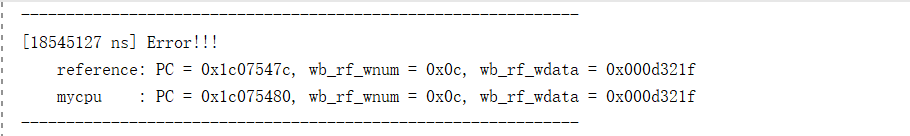


图 61 错误10对应的Console报错

* + - 1. 分析定位过程

可以看到此时PC出错，鉴于之前的经验，可能是指令取消出错，导致传递给ID的PC与指令不对应，查看波形图如下所示，具体波形图如下所示：

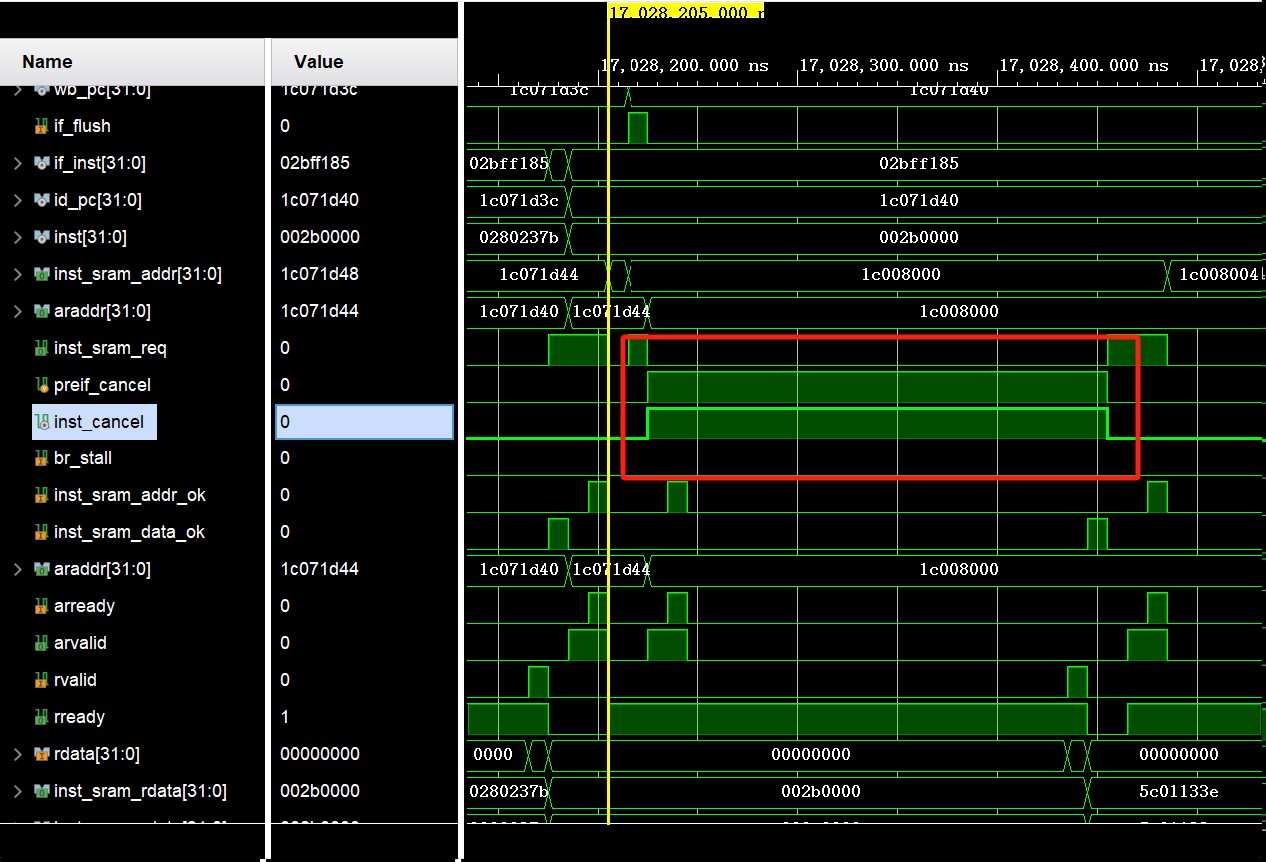


图62 错误10中preif\_cancel与inst\_cancel同时复位

可以看到，这里preif\_cancel与inst\_cancel信号同时复位为0，导致指令没有被正确取消，查看对应的代码，发现其均是在data\_ok返回后更新为0，所以会导致同时复位为0，出错前代码如下：

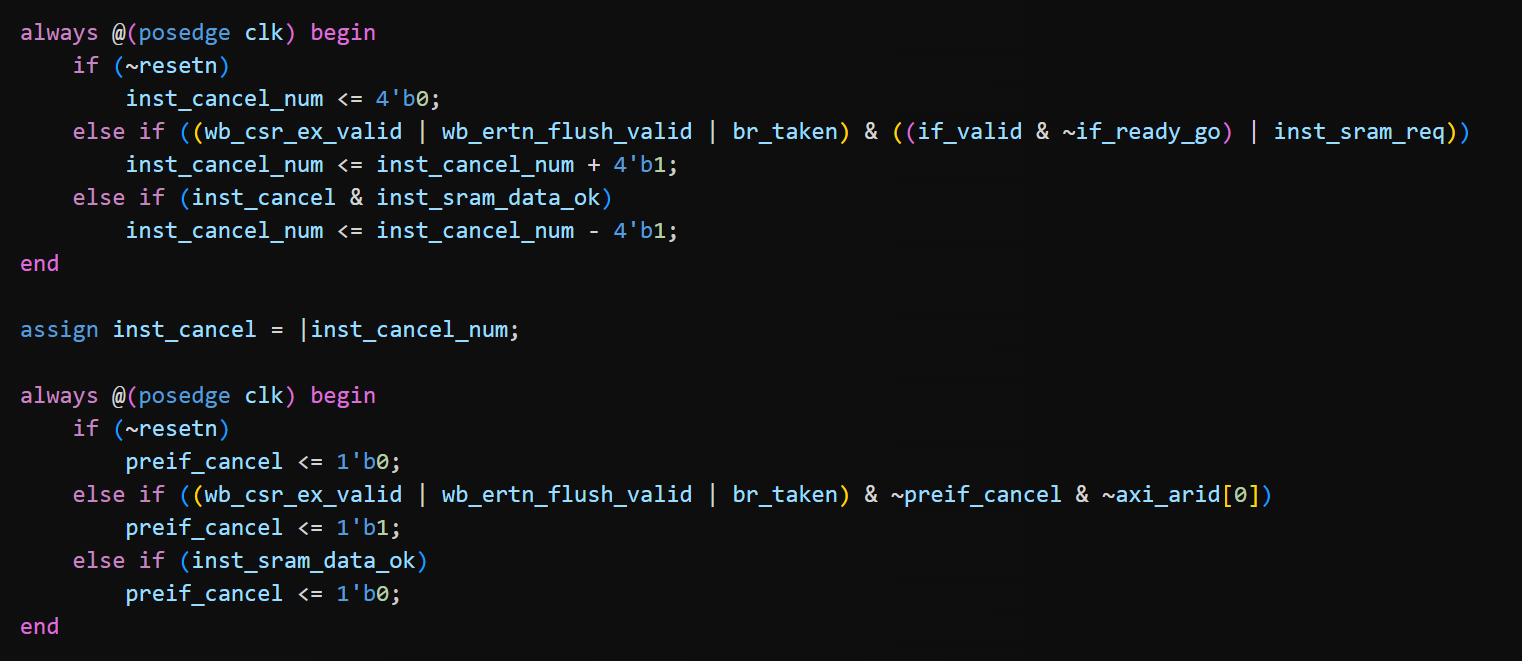


图63 错误10preif\_cancel与inst\_cancel修改前复位逻辑

* + - 1. 错误原因

preif\_cancel与inst\_cancel复位没有考虑到彼此，从而导致data\_ok来的时候，这两个信号同时复位为0，导致指令没有被正确取消。

* + - 1. 修正效果

对preif\_cancel信号的复位逻辑进行修改即可，修改后代码如下：

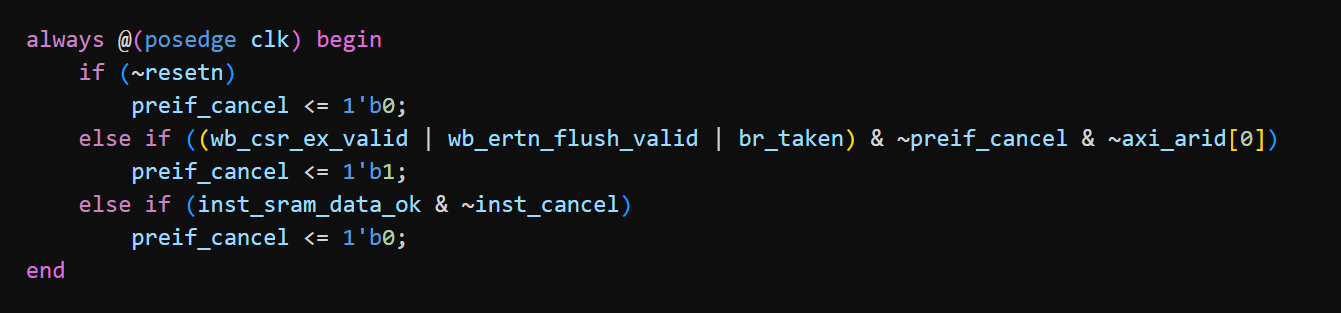


图64 错误4改后代码

这样修改后产生了新的错误，并没有让CPU运行到更后面的测试点，但这是exp15设计中的重大错误之一。

* + 1. 错误11：exp16中preif\_cancel和inst\_cancel的功能区分不清楚
       1. 错误现象

在修改了上一个错误后，console报错如下：

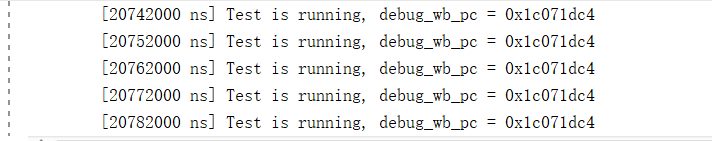


图65 错误11对应的Console报错

* + - 1. 分析定位过程

可见CPU陷入了停滞，于是只能查看波形图，观察是什么地方出现了错误，波形图如下所示：

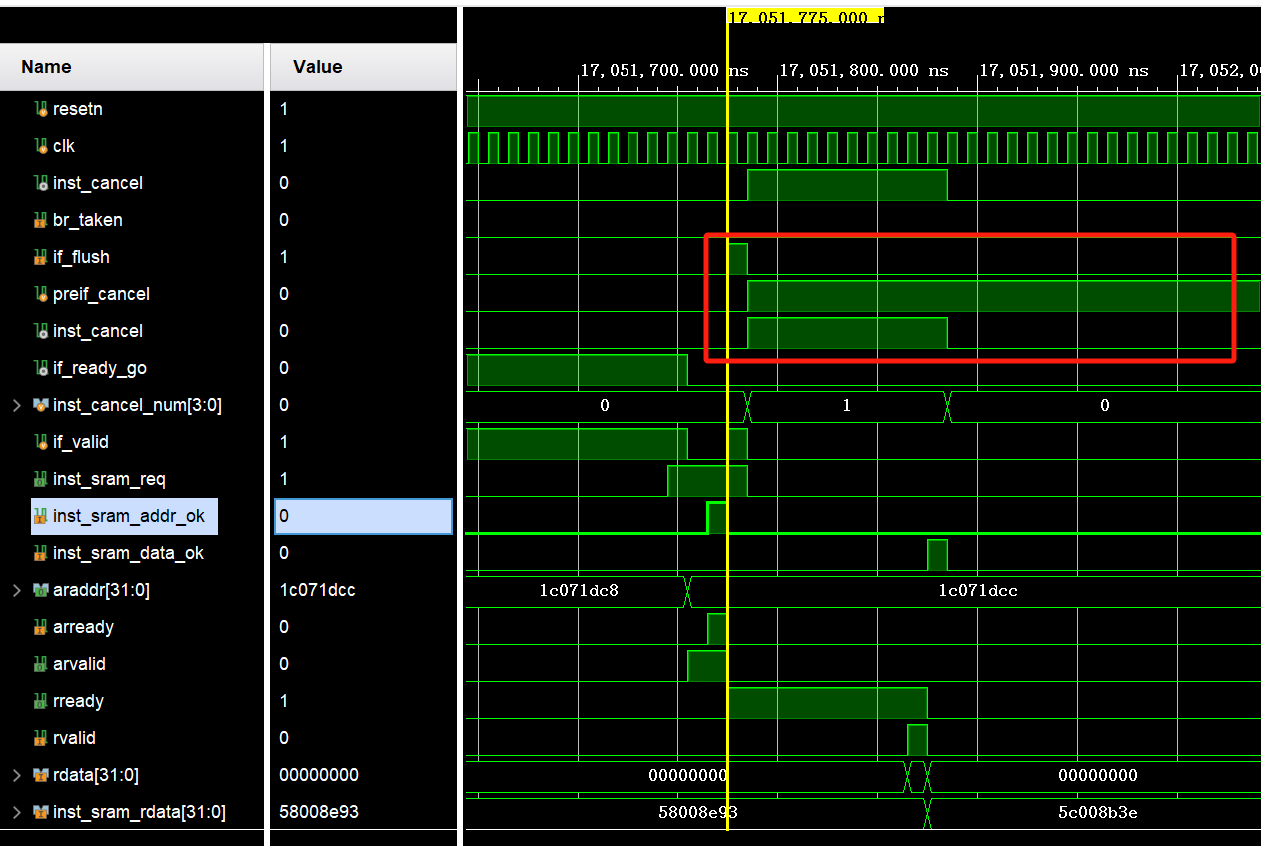


图66 错误11 的preif\_cancel与inst\_cancel拉高异常

可以看到，在if\_flush到来后，两个cancel信号同时拉高，但此时只需要取消一次取指。而由于修改了上一个bug，所以此时只有一个cancel信号拉低，从而导致出错。

到这可以说明，本小组exp15的通过有很大的偶然因素，其中很多地方尤其是cancel信号的更新复位逻辑并未搞的很清楚。查看preif\_cancel和inst\_cancel的逻辑如下：

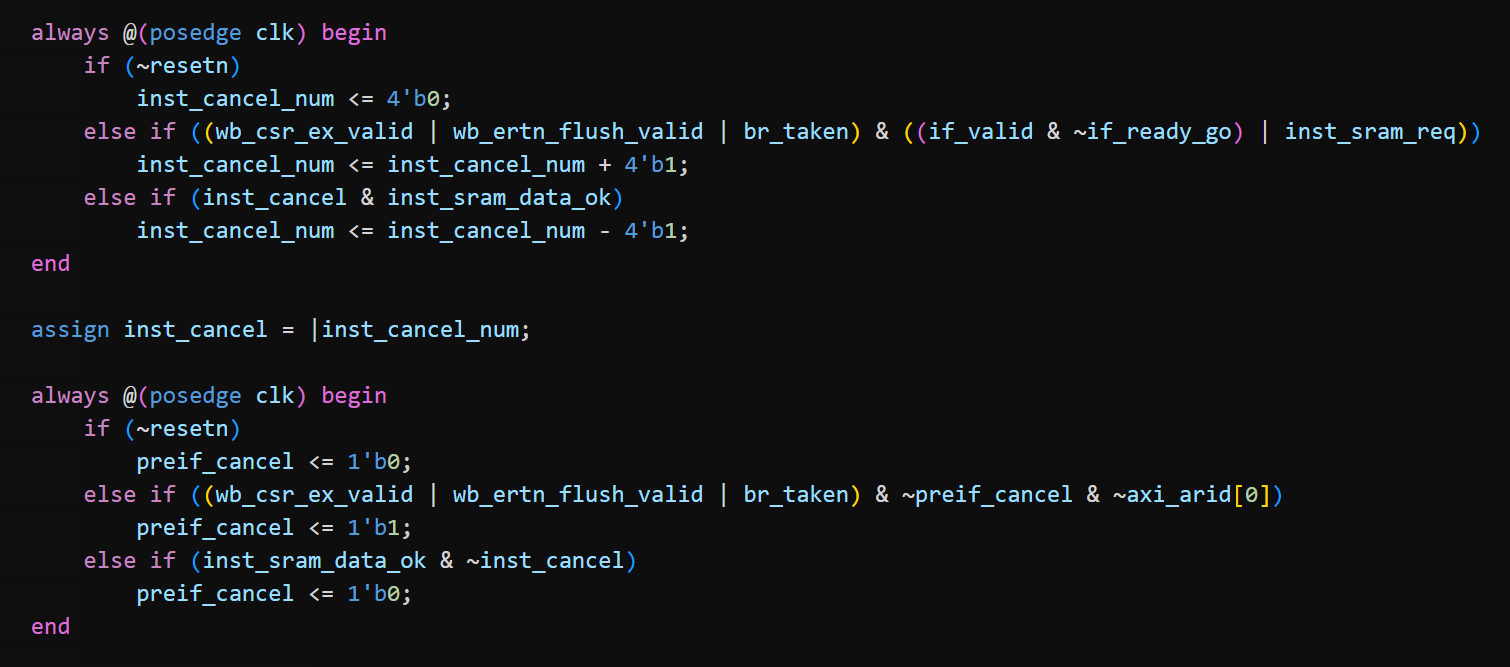


图67 错误11 的inst\_cancel与preif\_cancel修改前更新逻辑

可以看到，设计这两个信号的本意是区分不同情况下的取消指令操作。但是在该逻辑下，inst\_cancel\_num加一的逻辑中加上了“| inst\_sram\_req”，这个本来是preif\_cancel拉高的判断逻辑的一部分，所以这样写导致了在很多本应该只用preif\_cancel拉高的情况下inst\_cancel也拉高了。除此之外，preif\_cancel信号的更新逻辑也极其不完善，首先其根本没有考虑inst\_sram\_req，没有区分当跳转分支异常，中断等信号到来时，请求是否已经握手完成，导致其与inst\_cancel也混淆在一起。

出现上面这种错误的原因是本小组在定好每个信号的含义后，写exp15的同学没有清楚的区分这两个信号的作用，而仅仅根据exp15仿真中出现的错误对相关的逻辑进行强硬的修改，导致最终偶然条件下才能通过无延迟的种子，其它种子均无法通过。

* + - 1. 错误原因

exp16中preif\_cancel和inst\_cancel的功能区分不清楚，导致这两个信号均会存在本只应该对方拉高，但自己也拉高的情况，从而导致指令取消出错，CPU陷入停滞。

* + - 1. 修正效果

最后本组中写exp16的同学对这两个信号的更新逻辑进行大幅度的修改，其中也肯定不可避免的出现了许多的bug，但是在实验报告中就不全部进行展示了。对于这两个信号的修改，其中一个阶段性的结果如下：

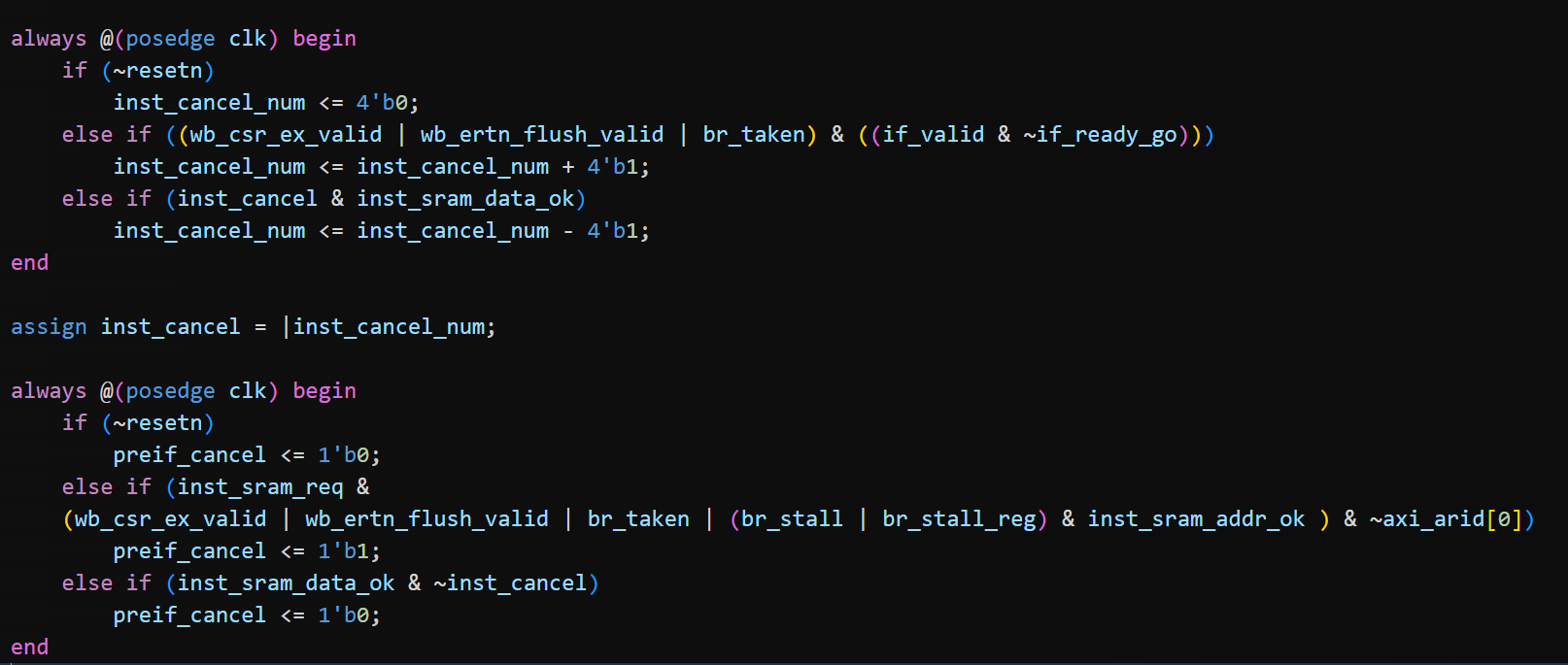


图68 错误11修改后代码

* + 1. 错误12：exp16中preif\_cancel信号未对br\_stall进行充分考虑出错
       1. 错误现象

在修改完上面的bug后，运行仿真，Console报错如下：

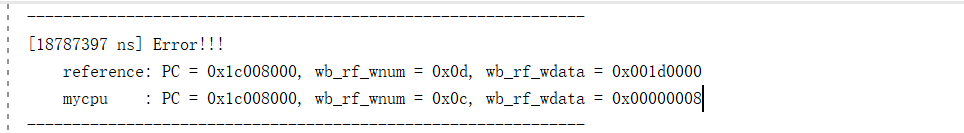


图69 错误12对应的Console报错

* + - 1. 分析定位过程

可以看到此rf\_wada出错，鉴于之前的经验，可能是指令取消出错，导致传递给ID的PC与指令不对应，查看波形图如下所示，具体波形图如下所示：：

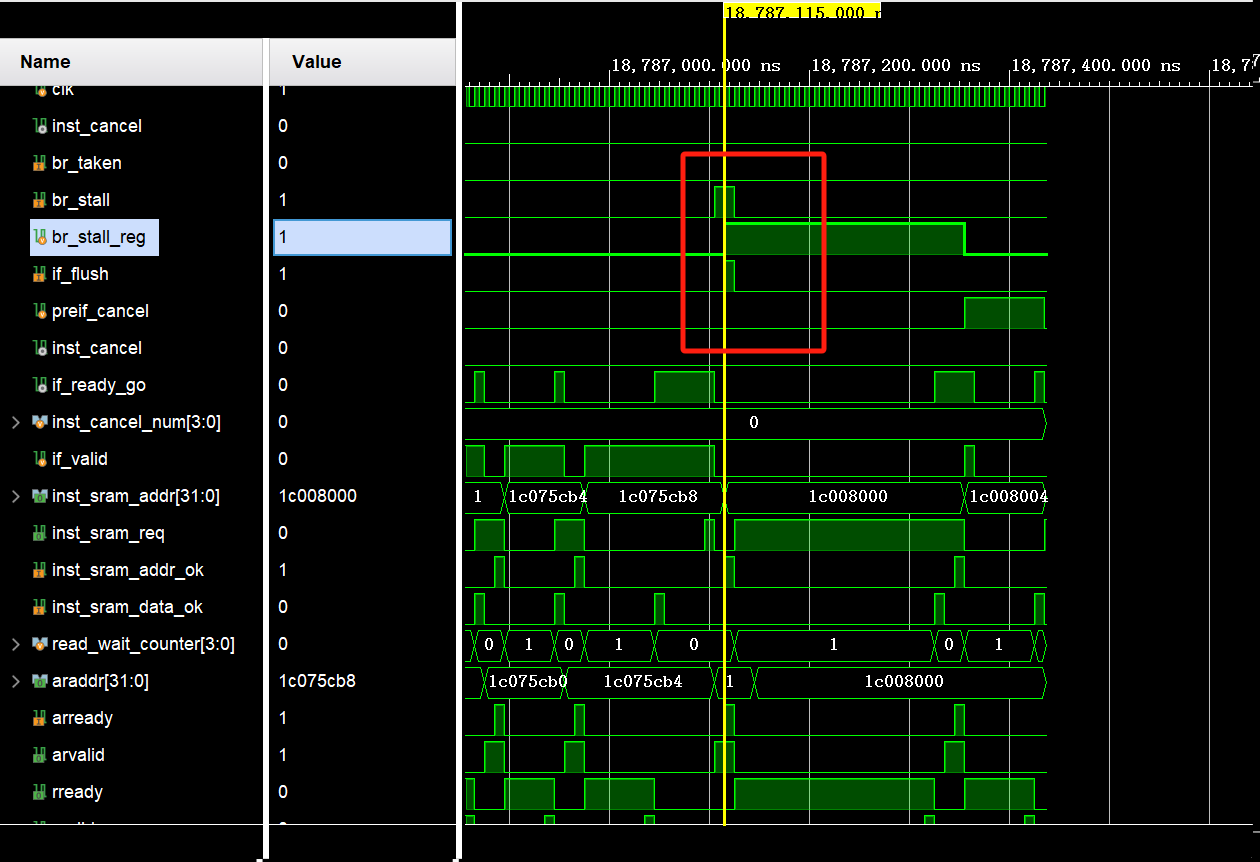


图70 错误12中因为br\_stall导致preif\_cancel未正常拉高

可以看到，此时if\_flush信号到来，且此时inst\_sram\_req发送了但addr\_ok还未返回，理论上应该preif\_cancel信号需要拉高，但是此时它没有正常拉高，查看preif\_cancel信号的更新逻辑，如下：

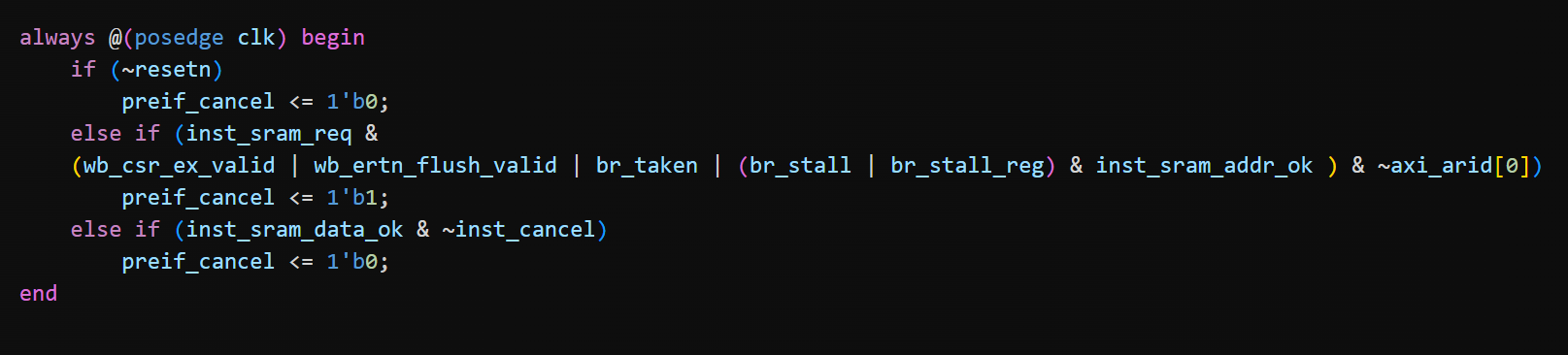
****

图71 错误12修改前preif\_cancel赋值代码(未对br\_stall充分考虑)

可以看到，此时preif\_cancel没有正常拉高的原因是在if\_flush到来时，br\_stall信号拉高了，所以此时inst\_sram\_req信号不会拉高（br\_stall时不发请求，等待ID数据冲突解决了再进行请求的发送），但是在指令进入ID级、拉高br\_stall前，它的后一条指令已经开始发送取指请求，因此不能仅仅通过在br\_stall为1时拉低req来处理br\_stall，还需要拉高pre\_cancel信号，等待先发完再取消。

* + - 1. 错误原因

preif\_cancel信号未对br\_stall进行充分考虑，导致取指没有被正确取消。在指令进入ID级、拉高br\_stall前，它的后一条指令已经开始发送取指请求，因此不能仅仅通过在br\_stall为1时拉低req来处理br\_stall，还需要拉高pre\_cancel信号，等待先发完再取消。

* + - 1. 修正效果

修改preif\_cancel的逻辑，添加对br\_stall的考虑，最终代码如下所示：

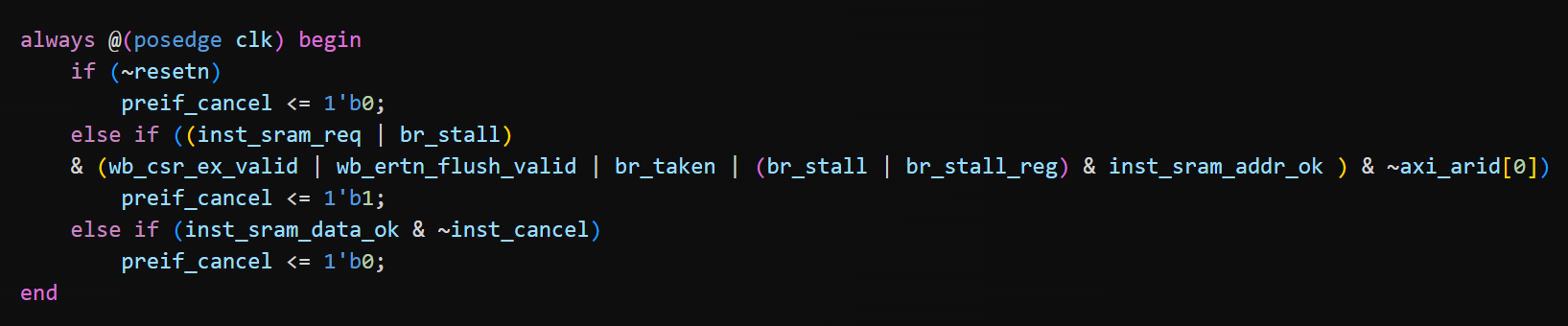


图72 错误6修改后代码

修改完这个bug后，在该种子下的仿真便能正常通过了。同时上板后，发现其它种子也能正常通过。至此，对exp16终于结束。

1. 实验总结

本实验为CPU增加了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。。

类SARM总线和AXI总线的引入使得流水级变化的情形更加复杂，尤其是IF流水级，逻辑较之前的实验复杂了很多，同时随机种子也为debug增加了一些难度，exp14和exp16均出现了对应种子能过，但其它种子不能通过的情况。在debug的过程中，我们需要遍历所有可能的情况，认真分析CPU的设计，这提高了我们对流水线CPU、总线协议等各方面的认识，也积累了不少debug的经验，并且通过本次实验，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。

在这三个实验中，我们的时序有了很大的改善，exp16的时序见下图。本组成员认为这是主要因为在这三次实验中，CPU对外连接的变成了AXI总线而不是原来的SRAM接口，这样可能导致组合逻辑链变得比原来更加简单。除此之外，原先的组合逻辑有不少都被换成了时序逻辑，比如原先的nextpc是由一个多路选择器在多个组合逻辑信号的基础上生成的，而现在这些信号中的一部分被换成了时序逻辑，组合逻辑链变得更加简单，时序也有了很大的提升。

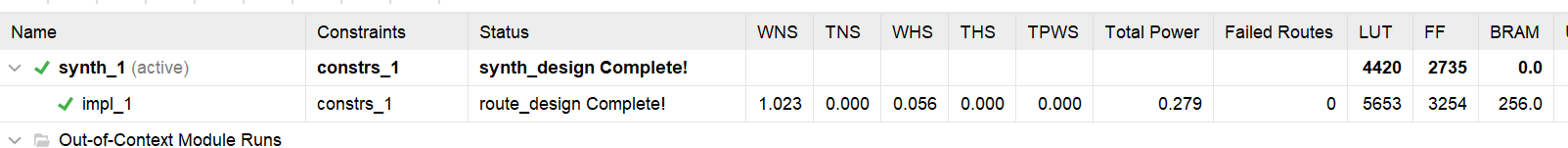


图73 exp16的时序结果