

LAB 8

小组成员（按姓氏排序）：林孟颖、谭茜、张梓堃

箱子号:79

一.实验任务概览

1. 完成TLB模块的设计，实现该模块的全部功能；
2. 在 CPU 中增加 TLBSRCH、TLBRD、TLBWR、TLBFILL、INVTLB 指令。
3. 在 CPU 中增加 TLBIDX、TLBEHI、TLBELO0、TLBELO1、ASID、TLBRENTY、DMW 这些 CSR 寄存器。
4. 增加TLB相关的异常，以及添加TLB后才启用的异常：ADEM, TLBREFILL, PIF, PIL, PIS, PME, PPI.
5. 利用TLB，实现虚拟地址翻译为物理地址。

二.实验设计

• （一）总体设计思路

- （1）总述

这次实验的总体目的，其实是完成CPU中的内存地址虚实转换功能。

一般地，为了使得应用程序的内存地址不受具体环境下内存空间分配的影响，现代的计算机体系下，对于利用高级语言及汇编语言所编写的应用程序，其可见的内存地址，包括传给CPU的指令所直接涉及的内存地址，均为各个进程独立使用地址空间的虚地址。而操作系统及更底层的部分则有必要把虚地址转换为真实的物理地址，以同真实的物理内存进行交互。对于一个可用的计算机系统，这样的过程显然是必须借助硬件而实现的。这一部分硬件，就是CPU中的MMU，也即本次实验在CPU中的部分。

而现代的计算机系统中，内存的管理大多以页（Page）作为重要的单位。然而，程序中内存空间的使用，经常会在短时间频繁使用大量相近甚至相同的内存地址，其页地址一般是不变的。如果每次页地址都需要重复查询，势必会造成性能大量损失。

- **(二) 重要模块设计： ()**

- **1. 接口定义**

名称	方向	位宽	含义
	IN		

- **2. 工作原理**

三.实验过程

- **(一) 实验流水**

11.7-11.8 9:00-12:00; 13:30-18:00; 19:00-23:30 TLB模块本身的编写和简单debug

11.9 8:00-19:00 EXP18 规划设计

11.10 8:00-12:00 EXP18 规划设计

11.11 8:30-11:45 14:00-19:00 EXP18 规划设计

11.12 8:00-18:00 EXP18 修改macro和csr

11.13 9:00-21:00 EXP18 添加指令

11.14-11.15 9:00-21:00 EXP18 DEBUG (自己CPU的bug, 以及环境的bug)

11.16-11.18 8:30-23:30 EXP19 设计

11.19 8:00-10:00 添加DMW

10:20-11:40; 13:30-18:00 添加地址虚实转换

11.20 8:00-16:00 添加TLB相关异常

11.20 16:00-23:30; 11.21-23 8:30-23:00 EXP19 DEBUG

- **(二) EXP 17: TLB 模块**

这一实验总体上比较顺利, 在认真阅读实验讲义、指令手册以及理论课教材之后, 除了一开始因为实验环境的原因花了一点时间之外, 没有遇到什么bug。

- 错误: clk_pll这一IP核没有生成

(1) 错误现象

未生成IP核clk_pll:



查看Source, 有:



(2) 定位过程:

我首先查看了project的生成tcl文件, 发现导入IP核的部分在这一部分:

```
5  
6 # Add IPs  
7 add_files -quiet [glob -nocomplain ../rtl/xilinx_ip/**/*.xci]  
8
```

和真实路径比较, 我发现了问题之所在。

(3) 错误原因

IP核导入的路径出错了。

(4) 修正效果

我把正确的代码修改为下图, 删去了一个"/"

```
5  
6 # Add IPs  
7 add_files -quiet [glob -nocomplain ../rtl/xilinx_ip/*.xci]  
8
```

- 错误: TLB接口错误

(1) 错误现象:

仿真、编译出错

```
ERROR: [VRFC 10-2865] module 'tlb' ignored due to previous errors [C:/Users/Jason Zhang/Desktop/tlb_verify/rtl/tlb_module.v:1]
INFO: [USF-XSim-69] 'compile' step finished in '4' seconds
INFO: [USF-XSim-99] Step results log file:'C:/Users/Jason Zhang/Desktop/tlb_verify/run_vivado/project/loongson.sim/sim_1/behav/xsim/xvlog.log'
ERROR: [USF-XSim-62] 'compile' step failed with error(s). Please check the Tcl console output or 'C:/Users/Jason Zhang/Desktop/tlb_verify/run_vivado
ERROR: [Vivado 12-4473] Detected error while running simulation. Please correct the issue and retry this operation.
ERROR: [Common 17-39] 'launch_simulation' failed due to earlier errors.
ERROR: [Common 17-69] Command failed: ERROR: [Common 17-39] 'launch_simulation' failed due to earlier errors.
```

(2) 定位过程

报错提示可能和模块的输入输出有关，所以我进入tlb_top模块，查看TLB的实例化情况。

通过查看，发现s0_va_bit12, s0_plv, s1_va_bit12, s1_plv, w_ps这几个信号接口没有定义。

然而我是按照讲义写的。经过反复比较，我有一定把握是讲义出现问题了。

(3) 错误原因

讲义中的接口和实验中的环境不符。

(4) 尝试修改

增加s0_va_bit12, s0_plv, s1_va_bit12, s1_plv, w_ps这几个信号接口，并把一部分相关的生成逻辑重新修改为正确的。之后实验就通过了。

(5) 补充

笔者为此还在Piazza上提出了一个问题：



Q & A Resources Statistics Search Companies Zikun Zhang

misc 1 lab1:exp1 2 lab2:exp2~4 7 勘误 20 lab3:exp6 6 lab4:exp7~9 5 lab5:exp10~11 1 lab6:exp12~13 15 more

Question History:

? question @225 stop following 16 views Actions

tlb的接口定义，讲义和tlb_top模块里的不同

如题，讲义P218对于TLB的定义中，比tlb_top.v中的tlb接口多了resetrn，但是少了s0_va_bit12, s0_plv, s1_va_bit12, s1_plv, w_ps几个接口，请问应该以哪个为准？

lab8:exp17~19 勘误

~ An instructor (XingJinzhang) endorsed this question ~

Edit good question 1 Updated 4 hours ago by XingJinzhang and Anonymous Calc

根据助教回答，应该是讲义版本出了错。

• (三) EXP 18

– 错误：ASID寄存器写入错误

(1) 错误现象：

在地址为0x1c00f5cc的指令出错了。

```
[8152000 ns] Test is running, debug_wb_pc = 0x1c00f5b0
```

```
[8152467 ns] Error!!!
```

```
reference: PC = 0x1c00f5cc, wb_rf_wnum = 0x0e, wb_rf_wdata = 0x000a00aa
mycpu      : PC = 0x1c00f5cc, wb_rf_wnum = 0x0e, wb_rf_wdata = 0x000a0000
```

(2) 定位过程

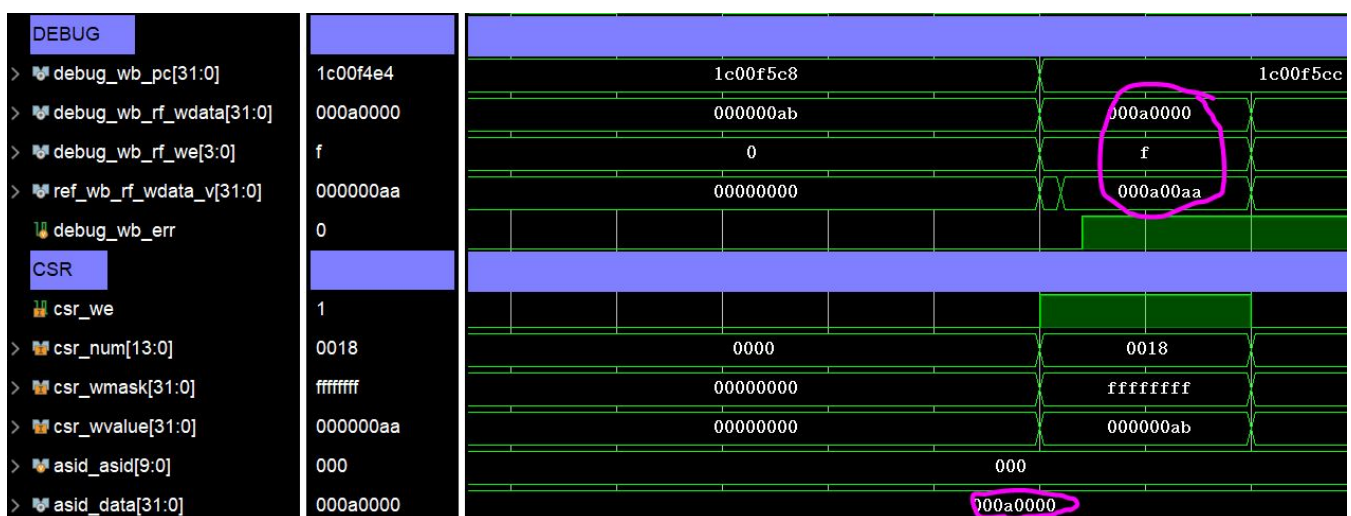
通过查看反汇编，我发现这条指令是CSRWR指令，这条指令向CSR的指定寄存器中写入内容，并把CSR寄存器的原值写入指定的GPR。

```
847 1c00f5c8: 0382ac0e ori $r14,$r0,0xab
848 1c00f5cc: 0400602e csrwr $r14,0x18
849 1c00f5d0: 14007150 lui2i.w $r16,906(0x38a)
```

查看CSR的csr_num定义，发现0x18对应CSR.ASID寄存器：

0x13	TLB 表项低位 1	TLBELO1
0x18	地址空间标识符	ASID
0x19	低半地址空间全局目录基址	PGDL

于是我去查看ASID相关的波形，结果如下：



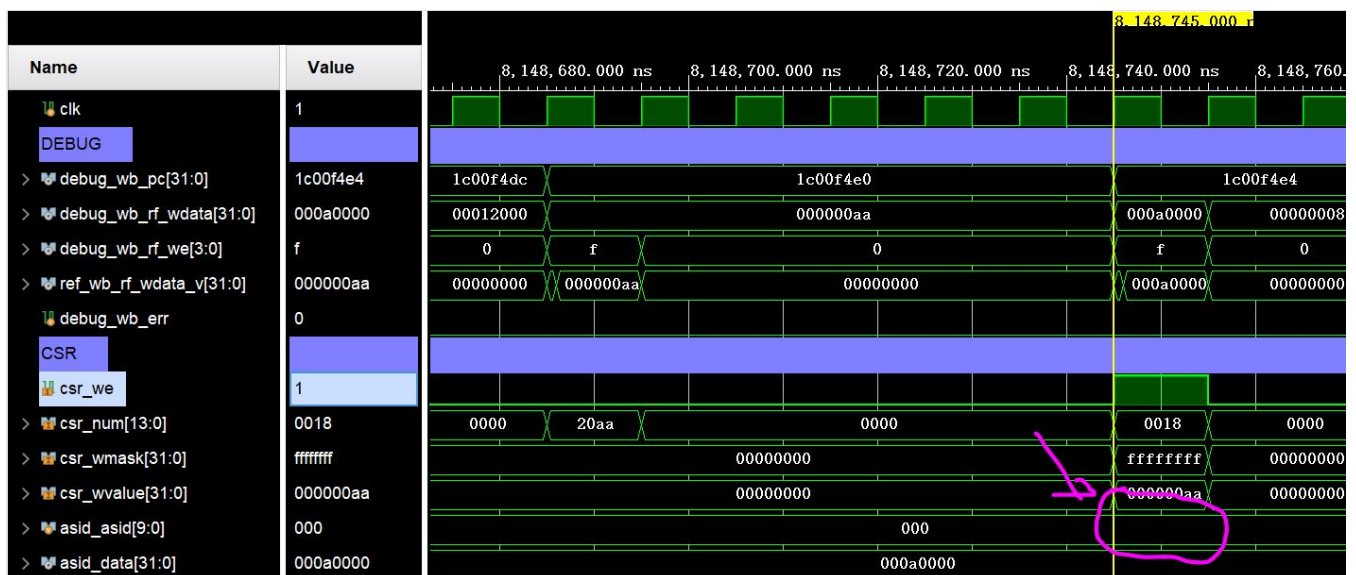
发现，对应的ASID.ASID应读出0x0aa，但实际上读出0x000。该寄存器里的值也是0x000。

因此，我猜测可能是之前赋值的位置出了错。

往前翻找反汇编，所有的tlb指令都是TLBWR，不会修改ASID，经过不懈努力，我终于发现了一条可能修改ASID的指令：

```
789 1c00f4e0: 0382a80e ori $r14,$r0,0xaa
790 1c00f4e4: 0400602e csrwr $r14,0x18
791 1c00f4e8: 14006950 lui2i.w $r16,842(0x34a)
```

这条指令修改了ASID。



注意到，如上图粉色位置所示，ASID理应在此时变为0x0aa，然而它并没有改变。

我同时检查波形，发现并没有什么明显的问题。同时，相关的控制信号也是正常的。

我再次重看相关的代码，发现了相关的问题。

(3) 错误原因

```

else if(csr_we && csr_num == `CSR_ASID) begin
    asid_asid <= csr_wmask[`CSR_ASID] & csr_wvalue[`CSR_ASID]
    | ~csr_wmask[`CSR_ASID] & asid_asid;
end

```

A pink arrow points to the macro definition ``CSR_ASID` in the code snippet.

在进行赋值时，位数的宏定义错误地用成了CSR_ASID，也即ASID的csr_num。

(4) 尝试修改

修改为正确的宏定义CSR_ASID_ASID，这个bug解决了。

- 错误：TLB漏接线

(1) 错误现象：

在地址为0x1c070c40的指令出错了。

```
[8162000 ns] Test is running, debug_wb_pc = 0x1c00f81c
```

```
[8165247 ns] Error!!!
```

```

reference: PC = 0x1c070c40, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x00000000
mycpu      : PC = 0x1c070c40, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x00012000

```

(2) 定位过程

与之前类似，我查看反汇编：

```

100933 1c070c38: 0400402c csrwr $r12,0x10
100934 1c070c3c: 06482c00 tlbrd
100935 1c070c40: 0400440d csrrd $r13,0x11
100936 1c070c44: 5c00eda0 bne $r13,$r0,236(0xec) # 1c070d30 <inst_error>
100937 1c070c48: 0400480d csrrd $r13,0x12

```

这条指令是csrrd，与上一个bug类似，读取的是TLBEHI的值。

往前看，应该是在这一条指令修改的TLBEHI：








```

100933 1c070c38: 0400402c csrwr $r12,0x10
100934 1c070c3c: 06482c00 tlbrd
100935 1c070c40: 0400440d csrrd $r13,0x11

```

查看这条指令的写入，需要查看TLB模块。

打开TLB模块，发现很多信号显示为Z。

>  w_index[3:0]	Z	Array
>  r_index[3:0]	Z	Array
 s0_va_bit12	Z	Logic
>  write_index[3:0]	Z	Array
>  s0_asid[9:0]	ZZZ	Array
>  s0_vppn[18:0]	ZZZZZ	Array
>  rand_num[7:0]	bb	Array

(3) 错误原因

TLB中的很多重要信号忘了接线。

(4) 尝试修改

把忘记接线的信号加上之后，问题得到了解决。

- 错误：TLB写入条件错

(1) 错误现象：

读取CSR.TLBEHI的值出错。

```

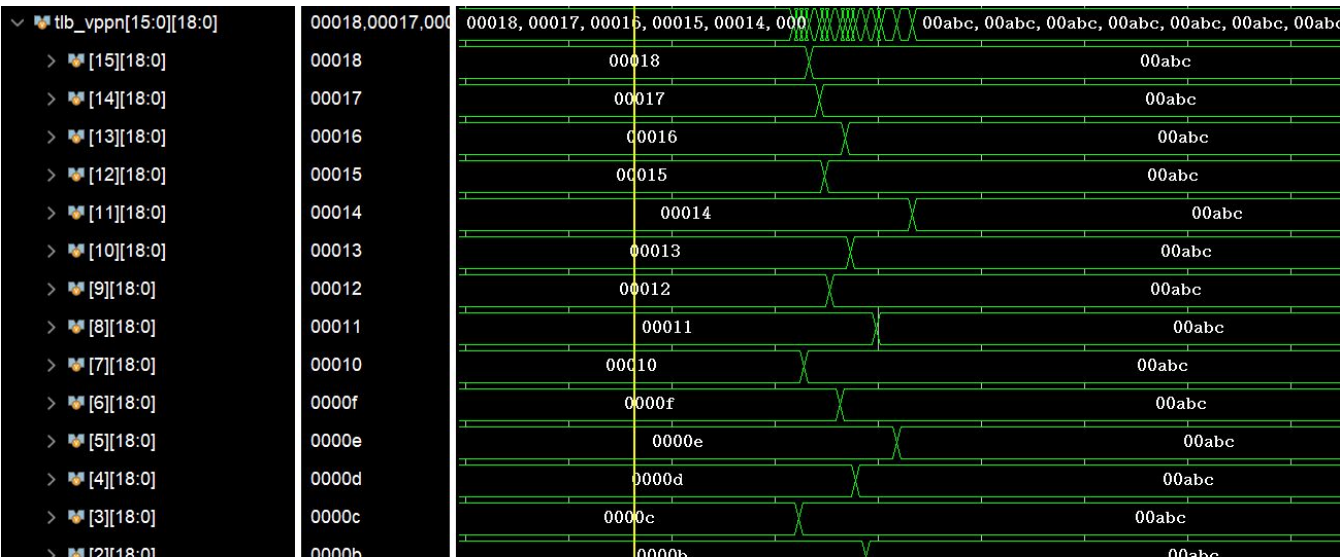
[8182000 ns] test is running, debug_wb_pc = 0x1c000190
-----
[8189667 ns] Error!!!
reference: PC = 0x1c05b4dc, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x00016000
mycpu    : PC = 0x1c05b4dc, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x01578000
-----

```

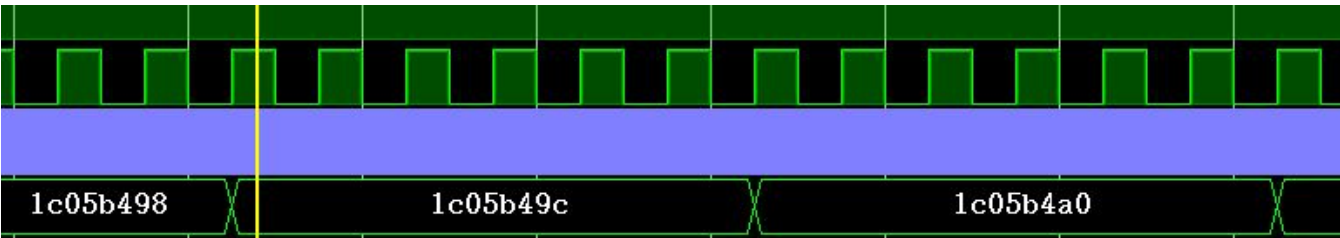
(2) 定位过程

翻看指令波形PC，发现前期出现了TLBRD指令，会修改CSR.TLBEHI的值。

继续检查TLB内部TLBEHI的信号，发现结果如图所示：

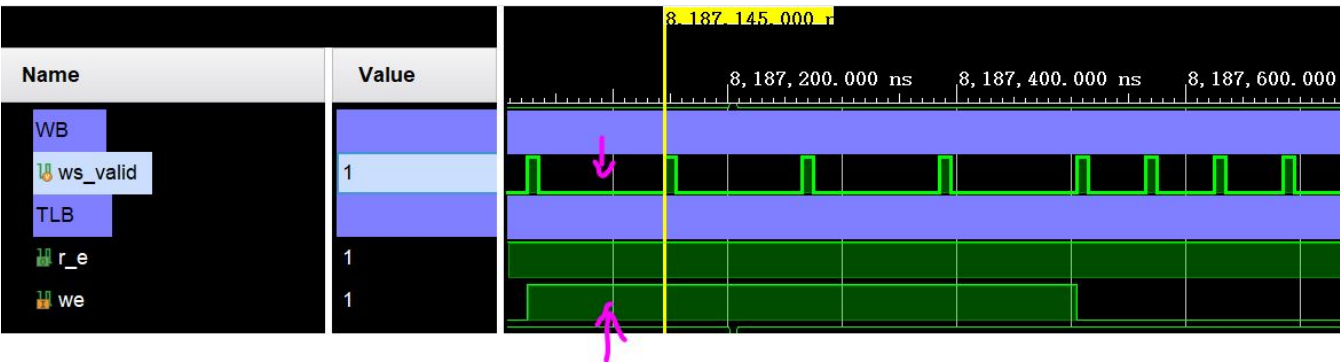


可以看到，几乎所有的寄存器都被覆盖了。这肯定是有问题的。



查看对应的指令，发现这里的指令是TLBFILL，对TLB中的寄存器进行随机覆盖写入。

所以我怀疑是写回的部分出现问题了。通过调出信号排查的方式，我证实了这一猜测。



可以看到，写回级指令无效的时候，发给TLB的写信号依然有效，导致了错误的写入。

(3) 错误原因

TLB写入有效信号存在问题，未考虑到写回级指令无效的情况。

```
assign tlbwe = inst_wb_tlbwr || inst_wb_tlbfill;
```

(4) 尝试修改

在写回级指令无效时，TLB写信号也无效。


```
assign tlbwe = (inst_wb_tlbwr || inst_wb_tlbfill) && ws_valid;
```

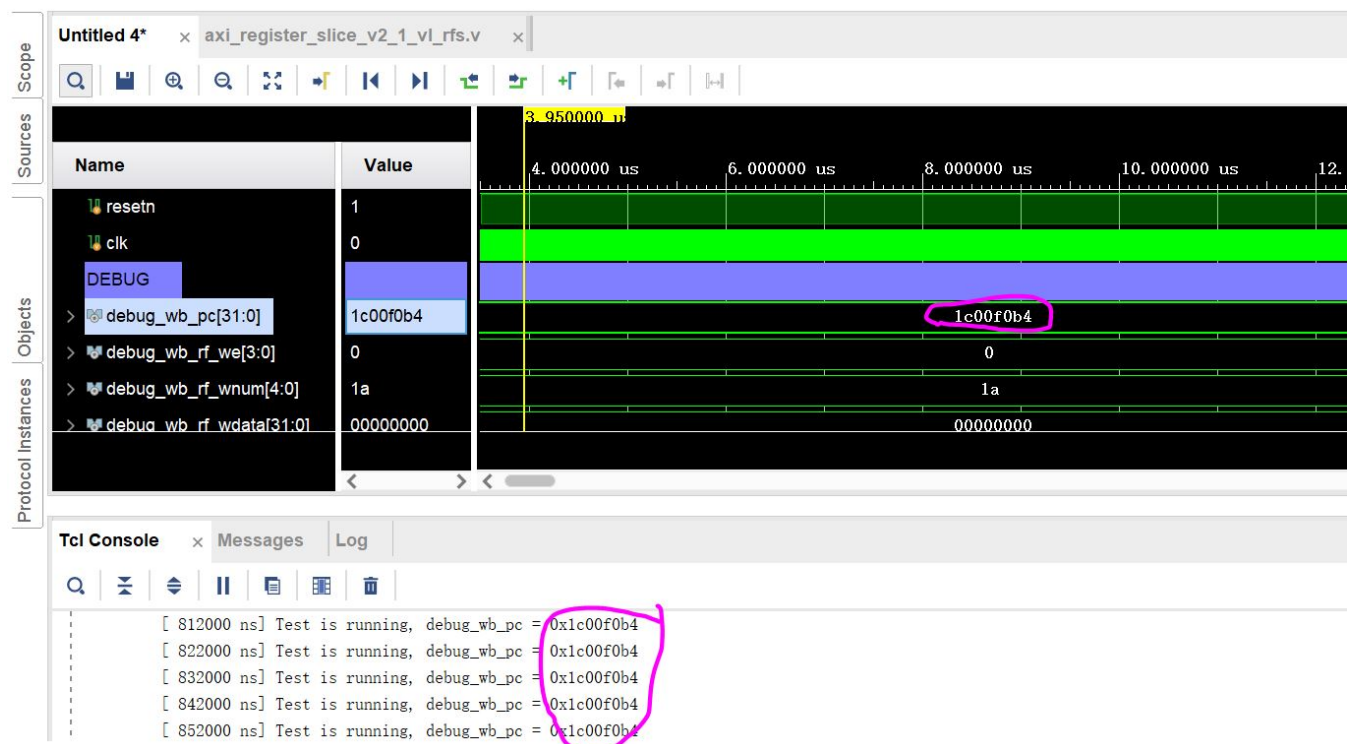
这一改动成功解决了问题。

• (四) EXP 19

- 错误：地址翻译忘了接线

(1) 错误现象：

仿真时，我发现pc停在一个固定的位置上不动了：



(2) 定位过程

我通过翻找反汇编，确定出现问题的虚拟地址对应的指令：

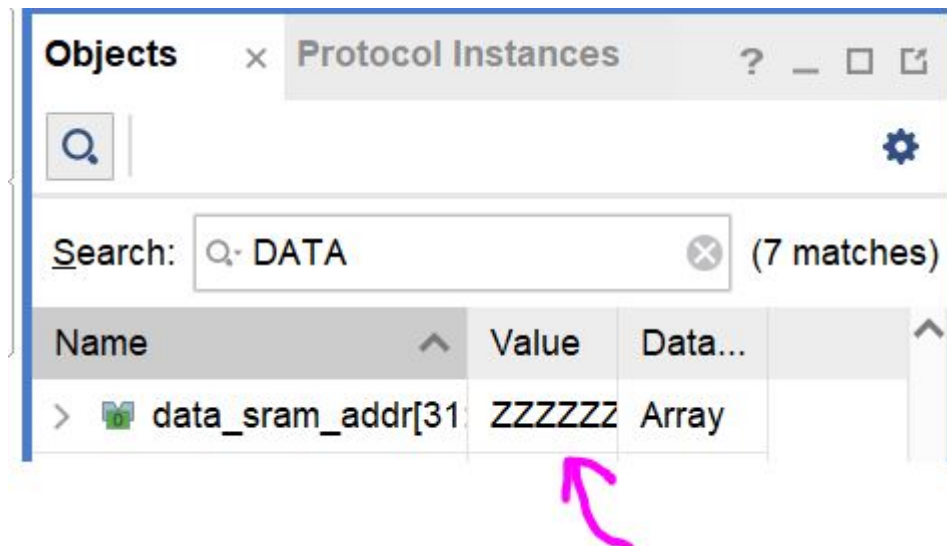
532	1c00f0b0:	02bfffdef	addi.w	\$r15,\$r15,-1(0xffff)
533	1c00f0b4:	1400001a	lu12i.w	\$r26,0
534	1c00f0b8:	2980008d	st.w	\$r13,\$r4,0
535	1c00f0bc:	298000ae	st.w	\$r14,\$r5,0

通过敏锐的直觉，我怀疑是接下来紧接着的一条访存指令出现了问题。毕竟这次实验处理的就是访存和取指中的问题，那么bug在这一部分的可能性就是极大的。

所以，我立即调出了EX阶段的相关信号，发现EX指令始终不能ready_go：



再查看相关的访存地址信号，还没打开波形就已经发现了问题：



data_sram_addr 始终处于高阻态。

(3) 错误原因

修改了物理地址翻译之后，对应的 data_sram_addr 信号忘记了接线。

(4) 尝试修改

加上接线：

```
assign data_sram_addr = phy_addr;
```

之后这个问题就解决了！

- 错误：异常判定条件未考虑指令类型

(1) 错误现象：

PC与ref完全不一致：

```
-----
[7964337 ns] Error!!!
reference: PC = 0x1c075b08, wb_rf_wnum = 0x05, wb_rf_wdata = 0xb27f9000
mycpu    : PC = 0x1c008000, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x001d0000
-----
```

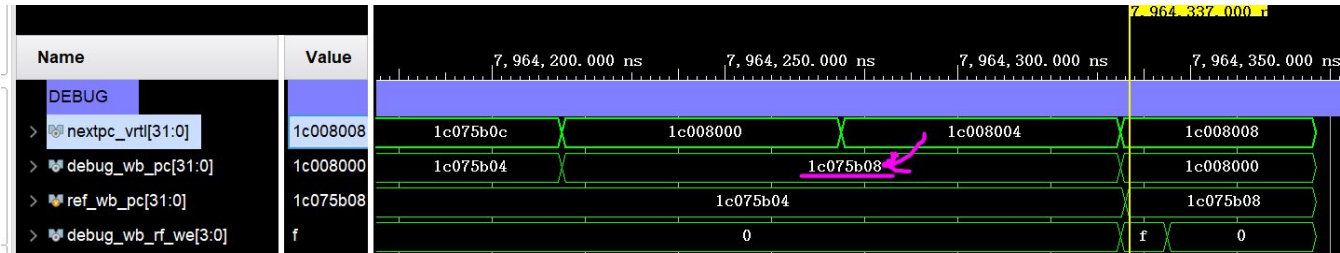
(2) 定位过程

我认真看这个错误的PC地址0x1c008000, 我意识到这应该不是一个普通的地址，而是异常处理的入口。

查看反汇编，的确如此：

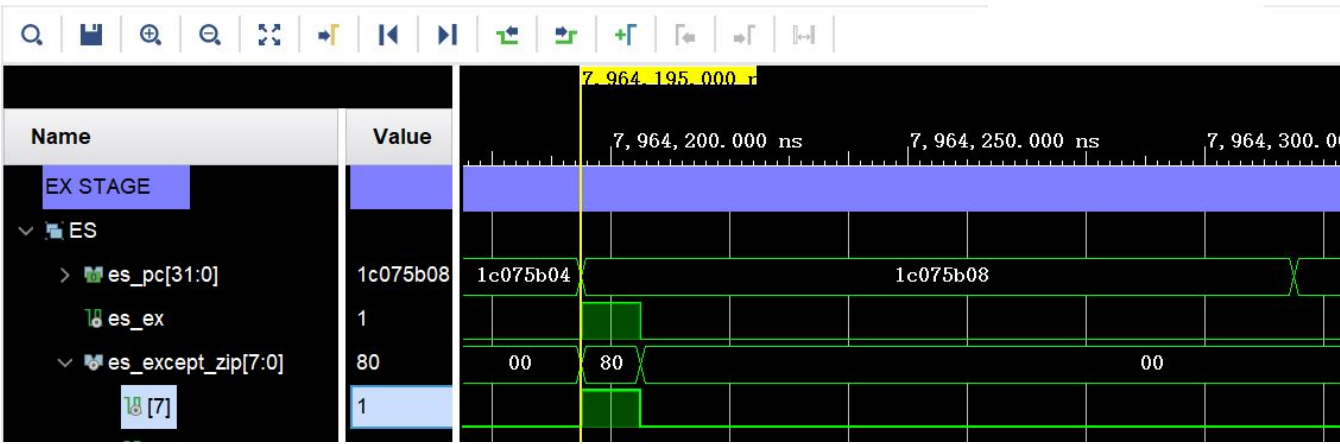
```
38 1c000140: 00104a2f add.w $r15,$r17,$r18
39 1c000144: 28800190 ld.w $r16,$r12,0
40 ...
41 1c008000: 14003a0d lu12i.w $r13,464(0x1d0)
42 1c008004: 288001ad ld.w $r13,$r13,0
43 1c008008: 0380040c ori $r12,$r0,0x1
44 1c00800c: 580079ac beq $r13,$r12,120(0x78) # 1c008084 <syscall_ex>
45 1c008010: 0380080c ori $r12,$r0,0x2
46 1c008014: 5800c1ac beq $r13,$r12,192(0xc0) # 1c0080d4 <brk_ex>
```

回看波形中的上一个地址：



细看发现这根本不是一个能够触发异常的指令，除非TLB出现问题：

于是我回到EX Stage，查看相关的异常：



查看对应的信号，发现是触发了ADEM异常。而这条指令是LU12I，并非访存。因此，应该是误触发。

(3) 错误原因

ADEM异常的判定，没有加入必须访存的先决条件。

(4) 尝试修改

把判定条件改成，只有在访存时，才可以判定成功。

尝试修改后，这个bug解决了。

- 错误：CSR.CRMD的部分域未考虑CSR指令手动写入

(1) 错误现象：

写入寄存器的值与ref不符，并且仅仅相差1 bit：

```
-----
[8520107 ns] Error!!!
reference: PC = 0x1c07c7c8, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000088
mycpu      : PC = 0x1c07c7c8, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000008
-----
```

(2) 定位过程

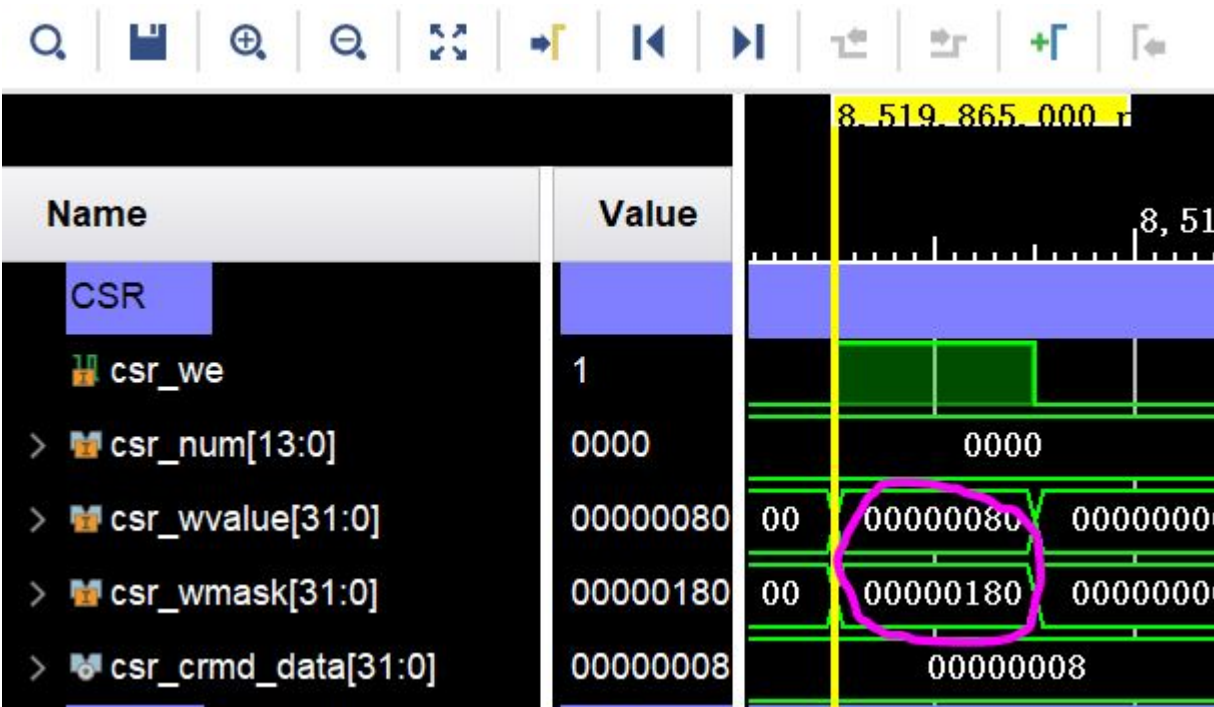
我首先定位到出错的指令，发现是一条csrcxchg指令，写入的值是CSR.CRMD的旧值：

```
113015 1c07c7b4: 0382000c ori $r12,$r0,0x80
113016 1c07c7b8: 0386000d ori $r13,$r0,0x180
113017 1c07c7bc: 040001ac csrcxchg $r12,$r13,0x0
113018 1c07c7c0: 03804c0c ori $r12,$r0,0x13
113019 1c07c7c4: 03807c0d ori $r13,$r0,0x1f
113020 1c07c7c8: 040001ac csrcxchg $r12,$r13,0x0
113021 1c07c7cc: 03800c11 ori $r17,$r0,0x3
```

既然如此，那肯定是上一条csrcxchg指令出了问题。

然而这里的每一条指令都需要写入GPR，那说明上几条指令中，写入GPR的过程绝对都没有问题。由此，我怀疑是csrcxchg指令在写入CSR这一问题上的问题，而非读取出现的问题。

所以，我调出相对应的信号：



按理说，根据粉色圈出的信号，CRMD的值应该变为0x88的，结果并没有。所以，我更加坚信，问题出在了写入的过程。

之后，我重新审查了另一位组员写的关于CRMD写入部分的代码，发现问题确实在此。

(3) 错误原因

CSR.CRMD的部分高位字段没有设置根据csr_num写入。

(4) 尝试修改

我参照其他字段的写入，添加了根据csr_num写入的代码，这个问题就解决了。

- 错误：访存阶段出现组合逻辑环

(1) 错误现象：

仿真阶段报错，并且波形没有办法再走下去了：

```
----[8498775 ns] Number 8'd70 Functional Test Point PASS!!!  
      [8502000 ns] Test is running, debug_wb_pc = 0x1c044d6c  
      [8512000 ns] Test is running, debug_wb_pc = 0x1c044fe0  
FATAL_ERROR: Iteration limit 10000 is reached. Possible zero delay oscillation detected where simulation time can not  
Time: 8520505 ns Iteration: 10000  
run: Time (s): cpu = 00:02:35 ; elapsed = 00:02:24 . Memory (MB): peak = 1572.387 ; gain = 0.000
```

(2) 定位过程

十分显而易见地，这是一个Combinational Loop的bug。

根据这个bug的特性，我直接调用了综合工具进行处理，找出Loop：

```
CRITICAL WARNING: [Synth 8-295] found timing loop. [c:/Users/Jason Zhang/Desktop/Exp19/soc_verify/soc_axi/rtl/soc_lite_top.v:67]  
Inferred a: "set_disable_timing -from I1 -to 0 i_4071"  
Found timing loop:  
0: i_4073/0 (LUT2)  
   [c:/Users/Jason Zhang/Desktop/Exp19/myCPU/EXEreg.v:135]  
1: i_4073/I1 (LUT2)  
2: i_4072/0 (LUT2)  
   [c:/Users/Jason Zhang/Desktop/Exp19/myCPU/EXEreg.v:135]  
3: i_4072/I1 (LUT2)  
4: i_4071/0 (LUT2)  
   [c:/Users/Jason Zhang/Desktop/Exp19/myCPU/EXEreg.v:135]  
5: i_4071/0 (LUT2)
```

根据Critical Warning的提示，我找到了对应的代码位置，并发现形成了：es_ex依赖es_tlb_exc，es_tlb_exc依赖es_mem_req，es_mem_req依赖es_ex，形成环路。

(3) 错误原因

上述电路赋值时，不小心生成了逻辑闭环。

(4) 尝试修改

我认为，这一问题在于es_tlb_exc并不完全依赖es_mem_req。相反，可以用更简略、涉及更少信号、不生成环路的方式实现。

照此修改过后，这个问题解决了，新的综合log中没有Critical Warning了。


```

Finished Writing Synthesis Report : Time (s): cpu = 00:05:21 ; elapsed = 00:05:27 . Memory (MB): peak = 1653.809 ; gain = 1048.023

Synthesis finished with 0 errors, 0 critical warnings and 101 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:05:02 ; elapsed = 00:05:20 . Memory (MB): peak = 1653.809 ; gain = 887.328
Synthesis Optimization Complete : Time (s): cpu = 00:05:21 ; elapsed = 00:05:28 . Memory (MB): peak = 1653.809 ; gain = 1048.023

```

- 错误：Load/Store指令标志生成错

(1) 错误现象：

如图所示，PC指向的位置错误：

```

[8535287 ns] Error!!!
reference: PC = 0x1c008000, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x001d0000
mycpu      : PC = 0x1c07cc48, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x47000000

```

(2) 定位过程

首先，出现问题的是一条Store指令：

```

113090 1c07c8e0: 1c00001b pcaddu12i $r27,0
113091 1c07c8e4: 0280237b addi.w $r27,$r27,8(0x8)
113092 1c07c8e8: 298003ec st.w $r12,$r31,0
113093 1c07c8ec: 5c035f3e bne $r25,$r30,860(0x35c) # 1c07cc48 <inst_error>
113094 1c07c8f0: 14003a0c lu12i.w $r12,464(0x1d0)

```

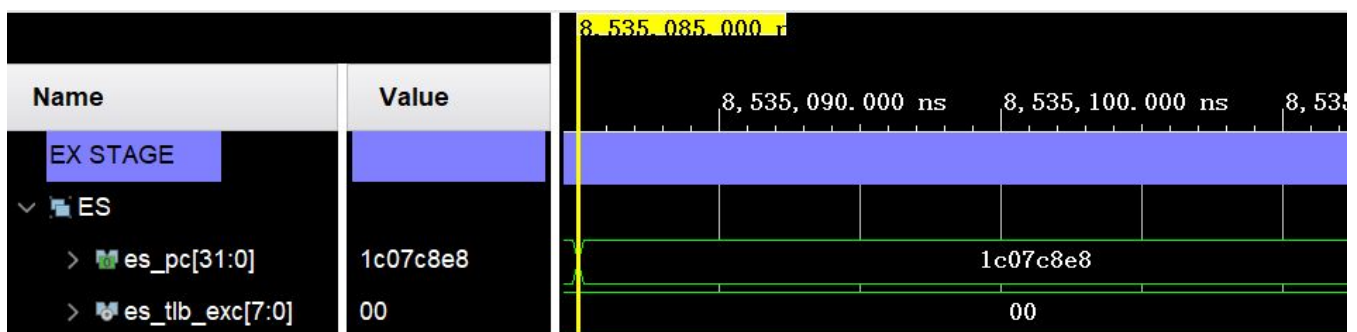
根据我丰富的debug经验和敏锐的直觉，我高度怀疑ref pc指向的是一个例外入口。通过查看反汇编，我确认了这一点，并根据golden trace，意识到它将会来到PIS例外的入口：

```

285
286 1c008370 <st_page_invalid>:
287 st_page_invalid():
288 1c008370: 00100019 add.w $r25,$r0,$r0

```

然而显而易见地，我的CPU并没有给出对应的Exception判定信号，因此导致出现问题：



随后，我开始排查。由于在我的设计中，PIS例外信号必须在 isStore 信号为1时才可能为1：

```

assign es_tlb_exc[`EARRAY_PIS] = es_valid & tlb_used & isStore &
!es_tlb_exc[`EARRAY_TLBR_MEM] & !s1_v;

```

而我的 isStore 信号与 res_from_mem 不小心挂了钩，这显然是十分错误的：

```
assign isStore = es_res_from_mem & es_valid & |es_mem_we;
```

(3) 错误原因

isStore 的生成不小心写错了。

(4) 尝试修改

把它改对：

```
assign isStore = |es_mem_we;
```

之后就没有问题了。

- 错误：把PPI误判为PME

(1) 错误现象：

如图所示，报错寄存器写入错误：

```
-----  
[8550807 ns] Error!!!  
reference: PC = 0x1c0083dc, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00070000  
mycpu      : PC = 0x1c0083dc, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00040000  
-----
```

(2) 定位过程

我找到对应的指令，发现实为CSR.ESTAT出错，也即异常代码错误：

```
316 1c0083d4: 0400180c csrrd $r12,0x6  
317 1c0083d8: 5c6c876c bne $r27,$r12,27780(0x6c84) # 1c00f05c <ex_finish>  
318 1c0083dc: 0400140c csrrd $r12,0x5  
319 1c0083e0: 0044c18c srli.w $r12,$r12,0x10  
320 1c0083e4: 0380fc0d ori $r13,$r0,0x3f
```

这说明应该是异常类型判断出错了。

查找异常代码，我发现是我的CPU把PPI例外判定为了PME例外，或者是在判定高优先级的PPI时，未能成功生成例外信号。所以我回到EX阶段的代码处，重新查看。

我一眼就发现，与之前类似地，我错用了 res_from_mem 信号作为“属于访存指令”的判定标准。

(3) 错误原因

PPI Exception的判定信号中，使用的条件信号用错了。

(4) 尝试修改

我把它改成正确的标志“当前指令是访存指令”的信号，就完成了修改。重新仿真，这个问题得到了解决。

