

# LAB 6

小组成员（按姓氏排序）：林孟颖、谭茜、张梓堃

箱子号:79

## LAB 6

### 一.实验任务概览

### 二.实验设计

#### （一）总体设计思路

#### （二）重要模块1设计

##### 1. 工作原理

##### 2. 接口定义

##### 3. 功能描述

### 三.实验过程

#### （一）实验流水

#### （二）实践任务12：添加系统调用异常支持

错误1：数据通路未补充完整

错误2：信号为X

错误3：接口漏接

错误4：未取消系统调用时的前序流水级

错误5：忽略CSR寄存器同步读特性

错误6：数据通路未补充完整

错误7：错误简介命名

错误8：流水级取消后写使能信号处理不当

错误9：错误简介命名

错误10：流水级取消后写使能信号处理不当

错误11：csr写寄存器指令的阻塞未处理好

#### （三）实践任务13：添加其它异常支持

错误1：变量类型声明有误

错误2：信号未定义

错误3：例外ADEM判断有误

错误4：数组越界访问

错误5：例外ADEM实现依旧有误

错误6：ESTAT的IS域处理有误

错误7：wb\_ex处理不当

错误8：wb\_ecode类型未实现完整

错误9：写使能有效与例外的关系处理不当

错误10：wb\_ecode未考虑优先级关系

错误11：在EXE、MEM判断出ERTN后未拉低写使能信号

错误12：ALE与ADEM搞混

错误13：中断未复位ALU

错误14：es\_ex处理有误

## 一.实验任务概览

1. 理解Loongarch指令级的基本指令实现;
2. 分别完成5条、20条指令单周期CPU设计与验证;
3. 熟悉

## 二.实验设计

- (一) 总体设计思路
- (二) 重要模块1设计
  - 1. 工作原理
  - 2. 接口定义
  - 3. 功能描述

## 三.实验过程

- (一) 实验流水
- (二) 实践任务12：添加系统调用异常支持

错误1~10是我没有通读讲义时写的，当时未正确理解精确异常，将csr读写都放在id阶段执行（syscall和ertn等指令都可以在id阶段就判断出来），但后续实验13需支持的异常会出现在exe阶段，故又回过头来修改exp12。

错误11是修改时遇到的错误（期间还有一些信号未定义的低级错误，并未记录），此时需要将例外相关的指令类型信号都往后传递，一直到WB阶段才生效，此时需要注意的是csr写寄存器的指令一直到wb阶段才可以拿到操作数，故需要对ID做阻塞操作；同时在wb阶段判断出例外则需要取消当前流水级的所有指令（在下一时钟上升沿将ds\_valid, es\_valid, ms\_valid, ws\_valid置零）。

## 错误1：数据通路未补充完整

### (1) 错误现象

写回数据有误：

```
[1352477 ns] Error!!!
reference: PC = 0x1c04ef0c, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000008
mycpu      : PC = 0x1c04ef0c, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000000

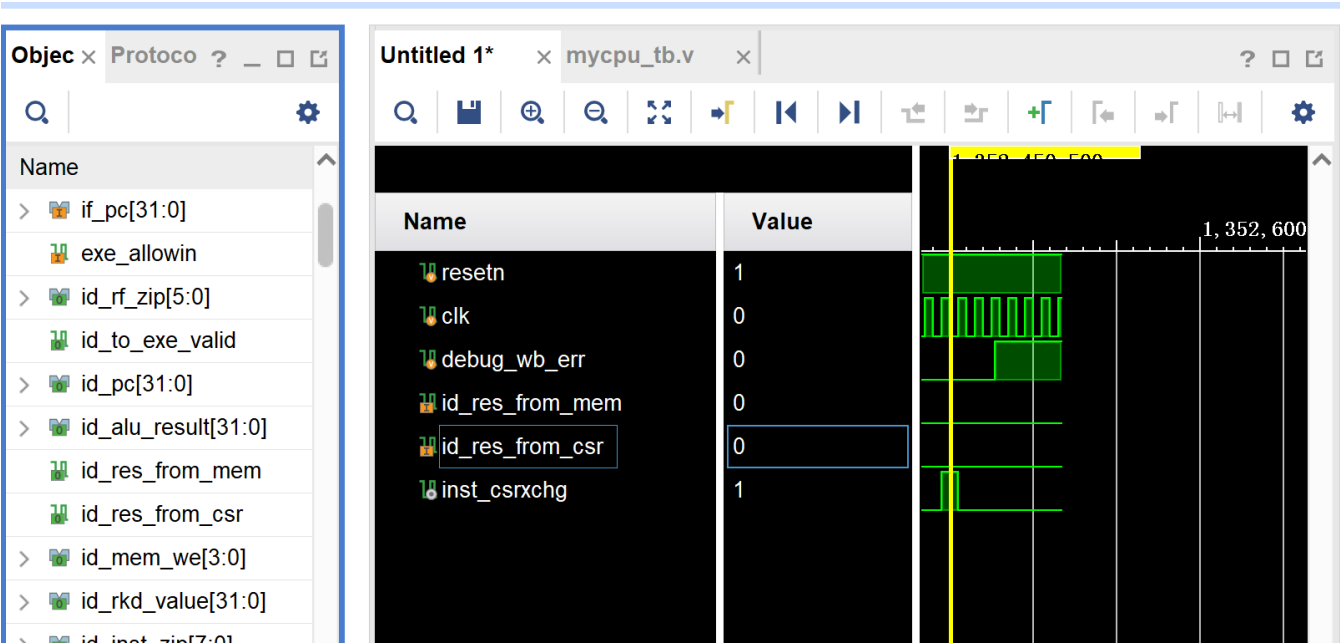
$finish called at time : 1352517 ns : File "D:/Desktop/cdb ede local/mvcpu env 12/soc verifv
```

### (2) 分析定位过程

查看对应汇编指令：

```
1c04ef08: 03801c0d    ori $r13,$r0,0x7
1c04ef0c: 040001ac    csrxchg $r12,$r13,0x0
1c04ef10: 1400021e    lu12i.w $r30,16(0x10)
```

得知是csrxchg指令。查看其写回数据来源的决定信号（res\_from\_mem和res\_from\_csr）：



发现res\_from\_csr未正常拉高，查看相关信息：

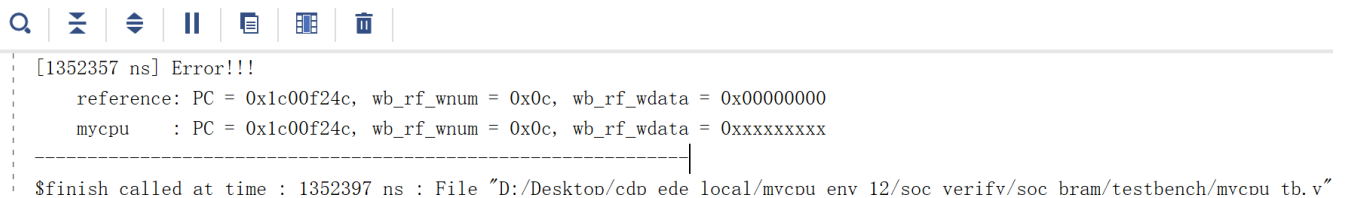
```
assign id_res_from_csr = inst_csrrd;
```

### (3) 错误原因

上述写回数据来源漏处理了 `inst_csrwr` 和 `inst_csrchg`，应改为：

```
assign id_res_from_csr = inst_csrrd | inst_csrwr | inst_csrchg;
```

### (4) 修正效果



```
[1352357 ns] Error!!!
reference: PC = 0x1c00f24c, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000000
mycpu      : PC = 0x1c00f24c, wb_rf_wnum = 0x0c, wb_rf_wdata = 0xxxxxxxxx
-----
$finish called at time : 1352397 ns : File "D:/Desktop/cdb ede local/mvcpu env 12/soc verifv/soc bram/testbench/mvcpu tb.v"
```

### (5) 归纳总结 (可选)

## - 错误2：信号为X

### (1) 错误现象

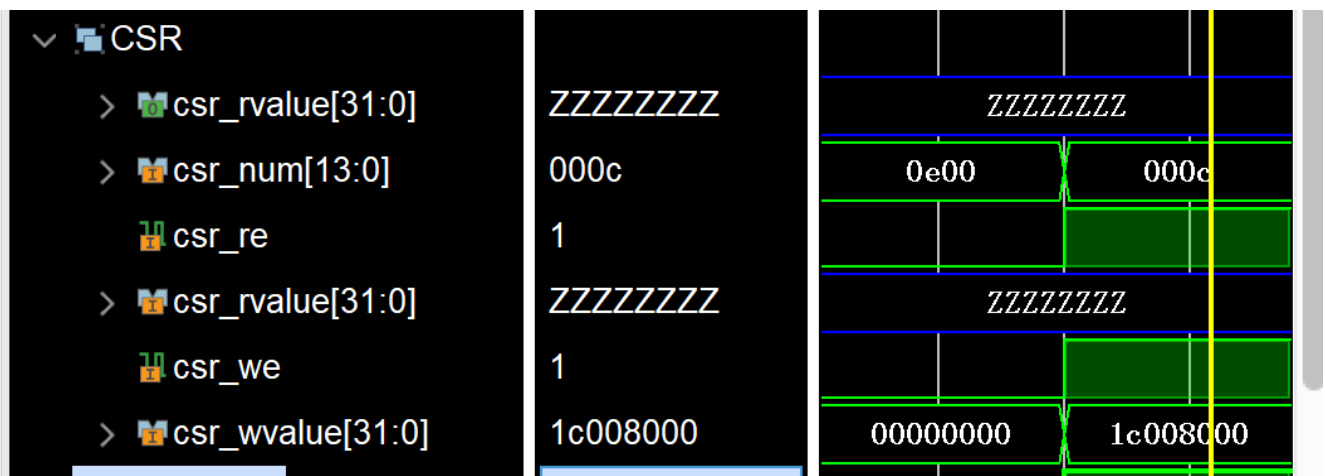
如1中的修改结果，写回数据未定义。

### (2) 分析定位过程

查看反汇编代码：

```
1c00f23c: 04000420    csrwr    $r0,0x1
1c00f240: 04001820    csrwr    $r0,0x6
1c00f244: 04003020    csrwr    $r0,0xc
1c00f248: 1438010c    lu12i.w  $r12,114696(0x1c008)
1c00f24c: 0400302c    csrwr    $r12,0xc
```

查看此时csr\_num：



查找loongarch指令集介绍得知是对ESTAT的寄存器进行写操作，查看读数据的实现：

```

// 例外状态
wire [31: 0] csr_estat_data;    // 保留位15:13
reg  [12: 0] csr_estat_is;      // 例外中断的状态位（8个硬件中断+1个定时器中断+1
// 个核间中断+2个软件中断）
reg  [ 5: 0] csr_estat_ecode;   // 例外类型一级编码
reg  [ 8: 0] csr_estat_esubcode; // 例外类型二级编码

assign csr_rvalue = {32{csr_num == `CSR_CRMD }} & csr_crmd_data
                    | {32{csr_num == `CSR_PRMD }} & csr_prmd_data
                    | {32{csr_num == `CSR_ECFG }} & csr_ecfg_data
                    | {32{csr_num == `CSR_ESTAT }} & csr_estat_data
                    | {32{csr_num == `CSR_ERA   }} & csr_era_data
                    | {32{csr_num == `CSR_EENTRY}} & csr_eentry_data
                    | {32{csr_num == `CSR_SAVE0 }} & csr_save0_data
                    | {32{csr_num == `CSR_SAVE1 }} & csr_save1_data
                    | {32{csr_num == `CSR_SAVE2 }} & csr_save2_data
                    | {32{csr_num == `CSR_SAVE3 }} & csr_save3_data;

```

### (3) 错误原因

想起忘记对 `csr_estat_data` 赋值，只在时序逻辑中给出其部分位的赋值逻辑。

### (4) 修正效果

加上如下代码：

```

assign csr_estat_data = { 1'b0, csr_estat_esubcode, csr_estat_ecode, 3'b0,
csr_estat_is};

```

来到后续错误点：

```

[1352557 ns] Error!!!
reference: PC = 0x1c008000, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x001d0000
mycpu      : PC = 0x1c04f034, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x2f000000
-----

```

### (5) 归纳总结（可选）

## - 错误3：接口漏接

### (1) 错误现象

如上，PC未能正常更新。

### (2) 分析定位过程

波形中查看pc的变化，结合反汇编代码得知PC在syscall处顺序执行下去：

id_pc[31:0]	1c04ef24	1c	1c04ef18	1c04ef1c	1c04ef20	1c04ef24	1c04ef28	1c04f034
-------------	----------	----	----------	----------	----------	----------	----------	----------

1c04ef20 <syscall\_pc1>:

```

1c04ef20: 002b0000    syscall 0x0
1c04ef24: 5c01133e    bne $r25,$r30,272(0x110) # 1c04f034 <inst_error>
1c04ef28: 14003a0c    lu12i.w $r12,464(0x1d0)
1c04ef2c: 03803c19    ori $r25,$r0,0xf

```

### (3) 错误原因

查看csr接口:

```

csr u_csr(
    .clk      (clk      ),
    .reset    (~resetn  ),
    .csr_re   (csr_re   ),
    .csr_num  (csr_num  ),
    .csr_rvalue (csr_rvalue),
    .csr_we   (csr_we   ),
    .csr_wmask (csr_wmask ),
    .csr_wvalue (csr_wvalue),
    .ex_entry (ex_entry ),
);

```

发现漏接了 `wb_ex` 和 `wb_pc` 的接口。

### (4) 修正效果

补充完后错误依旧如上，但pc已能跳转到正确入口，如图中蓝色光标处，已经正确取到了该指令：

> wb_ex_entry[31:0]	1c008000	1c008000				
> if_pc[31:0]	1c008008	1c04f034	1c04f038	1c008000	1c008004	1c008008
> id_pc[31:0]	1c008004	1c04ef28	1c04f034	1c04f038	1c008000	1c008004
> exe_pc[31:0]	1c008000	1c04ef24		1c04f034	1c04f038	1c008000
> wb_pc[31:0]	1c04f034	1c04ef1c	1c04ef20	1c04ef24		1c04f034
gr_we	1					

### (5) 归纳总结 (可选)

每次添加新端口时记得在模块连接处补充完整连线。

- 错误4：未取消系统调用时的前序流水级

(1) 错误现象

如上述，写回PC出错。

在WB阶段将例外信息发回IF阶段后，残存在流水线中的指令被继续执行。

(2) 分析定位过程

查看反汇编代码：

```
1c04ef20: 002b0000    syscall 0x0
1c04ef24: 5c01133e    bne $r25,$r30,272(0x110) # 1c04f034 <inst_error>
1c04ef28: 14003a0c    lu12i.w $r12,464(0x1d0)
1c04ef2c: 03803c19    ori $r25,$r0,0xf
1c04ef30: 29800199    st.w    $r25,$r12,0
1c04ef34: 002b0000    syscall 0x0
```

可见执行到 1c04ef28 后WB阶段又取到了 1c04ef24 的指令。

(3) 错误原因

在WB阶段收到例外的相关信息后应取消前序流水级的所有指令。此外，还有一种修改思路，在ID阶段获知其为系统调用后就将相关信息传回给IF级，同时在下一个时钟上升沿将id\_valid置为0，这样可减少流水线空泡时间，提高效率，本代码采取后一种修改方法。

(4) 修正效果

PC同步更新，如下：

```
[1352075 ns] Number 0 of functional test points passed.
-----
[1352607 ns] Error!!!
reference: PC = 0x1c008088, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x1c04ef20
mycpu      : PC = 0x1c008088, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000000
-----
```

(5) 归纳总结（可选）

- 错误5：忽略CSR寄存器同步读特性

(1) 错误现象

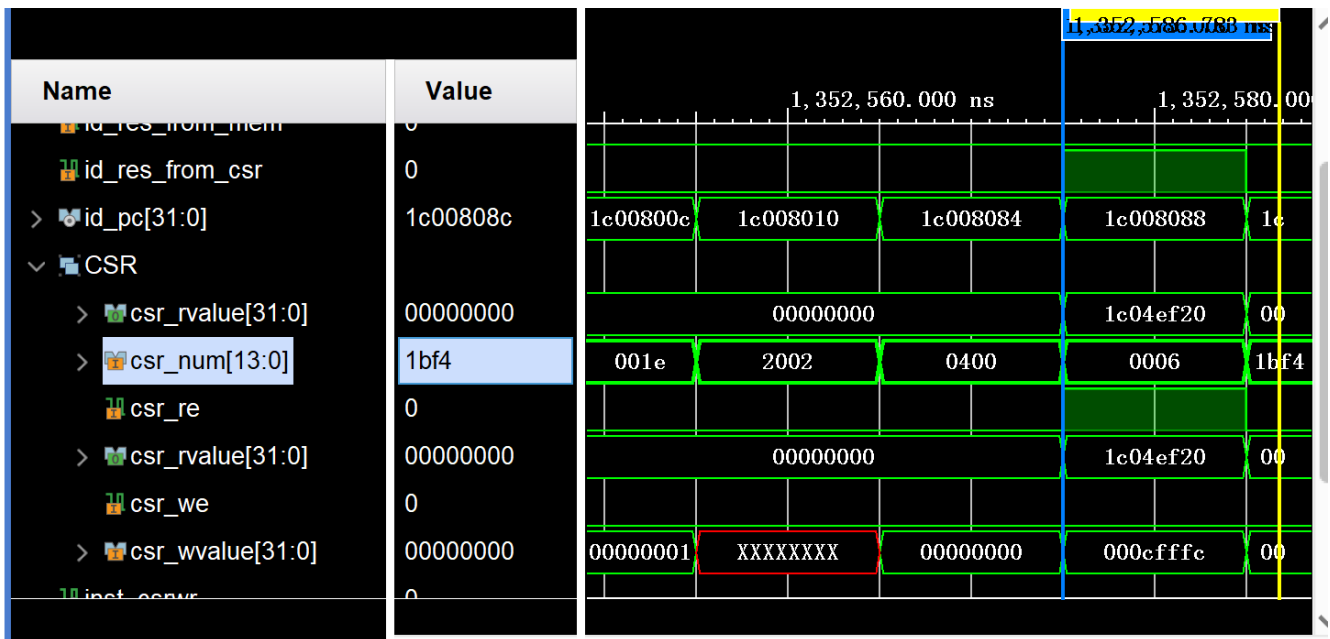
如上述，写回数据有误。

(2) 分析定位过程

查看指令类型：

```
1c008084: 00100019    add.w    $r25,$r0,$r0
1c008088: 0400180c    csrrd    $r12,0x6
1c00808c: 5c6fd36c    bne $r27,$r12,28624(0x6fd0) # 1c00f05c <ex_finish>
```

蓝色光标处即为出错位置：



查看写回的数据通路:

```
assign exe_rf_wdata      = exe_res_from_mem ? exe_mem_result :
                          exe_res_from_csr ? csr_rvalue : exe_alu_result;
```

其中csr\_value是id阶段直接读出的。

### (3) 错误原因

注意此处csr是同步读，故其在id阶段已读出，exe阶段应使用时序逻辑接受该值。即补充代码如下：

```
always @(posedge clk) begin
    if(~resetn) begin
        exe_csr_rvalue <= 32'b0;
    end
    else if(id_to_exe_valid & exe_allowin) begin
        exe_csr_rvalue <= id_csr_rvalue;
    end
end
```

#### (4) 修正效果

到下一个bug:



```
-----[1352075 ns] Number 8 d4b Functional test Point PASS!!!
-----
[1352357 ns] Error!!!
reference: PC = 0x1c00f24c, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000000
mycpu      : PC = 0x1c00f24c, wb_rf_wnum = 0x0c, wb_rf_wdata = 0xbfaaff080
-----
```

(5) 归纳总结 (可选)

- 错误6：数据通路未补充完整

(1) 错误现象

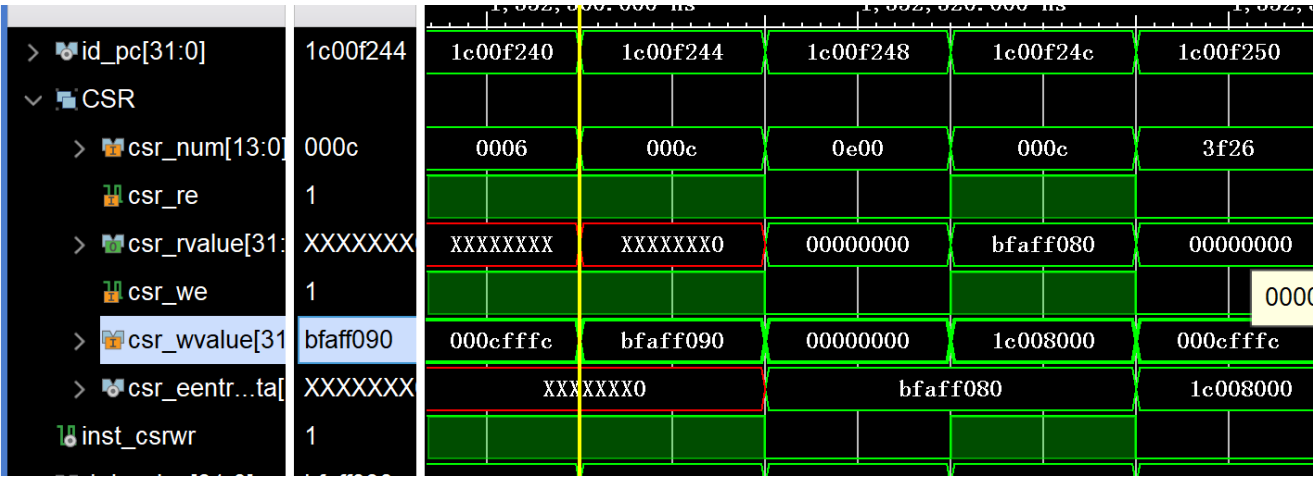
如上述，写回数据有误。

(2) 分析定位过程

查看指令：

```
1c00f23c: 04000420 csrwr $r0,0x1
1c00f240: 04001820 csrwr $r0,0x6
1c00f244: 04003020 csrwr $r0,0xc
1c00f248: 1438010c lu12i.w $r12,114696(0x1c008)
1c00f24c: 0400302c csrwr $r12,0xc
```

对应的是 `csrwr` 指令，同时注意到上述 `csrwr $r0,0xc` 一指令已将数据 `32'b0` 存入 `0xc` 对应的寄存器，猜测是前述写入时已经出错，再查看前一条指令写入时的值已经出错（光标指示处）：



而已知在0号寄存器读出的必为0，再查看指定寄存器来源的信号：

```
assign src_reg_is_rd = inst_beq | inst_bne | inst_blt | inst_bltu |
                      inst_bge | inst_bgeu | inst_st_w | inst_st_h |
                      inst_st_b;
```

### (3) 错误原因

忘记在写数据时指定 `src_reg_is_rd`。修改如下：

```
assign src_reg_is_rd = inst_beq | inst_bne | inst_bltn | inst_bltu |  
                      inst_bge | inst_bgeu | inst_st_w | inst_st_h |  
                      inst_st_b | inst_csrwr | inst_csrchg;
```

### (4) 修正效果

通关至下一个 bug。

```
----[1352075 ns] Number 8'd46 Functional Test Point PASS!!!  
-----  
[1352627 ns] Error!!!  
reference: PC = 0x1c008090, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x000b0000  
mycpu      : PC = 0x1c008090, wb_rf_wnum = 0x0c, wb_rf_wdata = 0xXxxxXxxX  
-----  
$finish called at time : 1352667 ns : File "D:/Desktop/cdp_edelocal/mycpu_env_12/soc_verify/soc_bram/te:  
run: Time (s): cpu = 00:00:46 ; elapsed = 00:01:17 . Memory (MB): peak = 1221.020 ; gain = 0.000
```

### (5) 归纳总结 (可选)

## - 错误7: 错误简介命名

### (1) 错误现象

注意到上述写回数据未定义。

### (2) 分析定位过程

查看手册可知对应的时ESTAT状态寄存器，查看其操作：

```
always @(posedge clk) begin  
    if (reset) begin  
        csr_estat_is[1:0] <= 2'b0;  
    end  
    else if (csr_we && (csr_num == `CSR_ESTAT)) begin  
        csr_estat_is[1:0] <= ( csr_wmask[`CSR_ESTAT_IS10] &  
csr_wvalue[`CSR_ESTAT_IS10])  
                                | (~csr_wmask[`CSR_ESTAT_IS10] &  
csr_estat_is[1:0]  
                                );  
    end  
  
    csr_estat_is[9:2] <= hw_int_in[7:0];  
    csr_estat_is[10] <= 1'b0;  
  
    if (timer_cnt[31:0] == 32'b0) begin  
        csr_estat_is[11] <= 1'b1;  
    end  
    else if (csr_we && csr_num == `CSR_TICLR && csr_wmask[`CSR_TICLR_CLR])
```

```

        && csr_wvalue[`CSR_TICLR_CLR])
    csr_estat_is[11] <= 1'b0;

    csr_estat_is[ 12] <= ipi_int_in;
end

```

### (3) 错误原因

当时看讲义时不求甚解地摘抄信号，但许多数据没有给出详细的定义，此处可先照着0x000b000倒推其应有的数值。

在piazza询问后得知，此处未给出的信号一律置为0即可。

### (4) 修正效果

```

----[1352075 ns] Number 8'd46 Functional Test Point PASS!!!
-----
[1352727 ns] Error!!!
    reference: PC = 0x1c0080b8, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000003
    mycpu      : PC = 0x1c0080b8, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000000
-----
$finish called at time : 1352767 ns : File "D:/Desktop/cdp_edelocal/mycpu_env_12/soc_veri
run: Time (s): cpu = 00:00:59 ; elapsed = 00:01:19 . Memory (MB): peak = 1019.758 ; gain =

```

## - 错误8：流水级取消后写使能信号处理不当

### (1) 错误现象

如上述错误7的修正后结果，写回数据有误。

### (2) 分析定位过程

查看此时反汇编指令：

```

1c0080b4: 5c6fa9ac    bne $r13,$r12,28584(0x6fa8) # 1c00f05c <ex_finish>
1c0080b8: 0400040c    csrrd  $r12,0x1
1c0080bc: 03801c0d    ori  $r13,$r0,0x7

```

由 `csr_num=0x1` 得知对应的寄存器是PRMD寄存器，查看其前序变化过程：

Name	Value
> id_pc[31:0]	1c008008
id_res...m_cs	0
CSR	
wb_ex	1
> wb_pc[31:0]	1c04ef20
> csr_num	2001
> csr_pr...	00000003
> csr_cr...	00000000
csr_re	0
> csr_rv...	00000000

The timing diagram shows the state of various CSR registers at two specific time points: 1,352,520.000 ns and 1,352,540.000 ns. The registers shown are id\_pc, id\_res...m\_cs, CSR, wb\_ex, wb\_pc, csr\_num, csr\_pr..., csr\_cr..., csr\_re, and csr\_rv.... At 1,352,520.000 ns, the values are as listed in the table above. At 1,352,540.000 ns, the values have changed: id\_pc is 1c008008, id\_res...m\_cs is 0, CSR is 1, wb\_ex is 1, wb\_pc is 1c04ef20, csr\_num is 2001, csr\_pr... is 00000003, csr\_cr... is 00000000, csr\_re is 0, and csr\_rv... is 00000000.

### (3) 错误原因

由于syscall带来的流水线空泡使得wb\_ex在流水级中多滞留了一个周期，使得prmd的值被二度更新，故应该在将例外信号传回csr时设置如下（将wb\_ex项更改为wb\_ex & wb\_valid）：

```
assign wb_except_zip = {wb_ecode, wb_esubcode, wb_ex & wb_valid, wb_pc};
```

#### (4) 修正效果

```
[1352877 ns] Error!!!
```

```
reference: PC = 0x1c04ef28, wb rf wnum = 0x0c, wb rf wdata = 0x001d0000
```

```
mycpu      : PC = 0x1c00f07c, wb rf wnum = 0x04, wb rf wdata = 0xbfaaff00
```

(5) 归纳总结 (可选)

### 错误9：错误简介命名

### (1) 错误现象

如上述错误8的修正后结果，pc值未能正常更新。

## (2) 分析定位过程

查看该指令前序指令:

```

1c00f06c: 0280032d    addi.w $r13,$r25,0
1c00f070: 5c000b20    bne $r25,$r0,8(0x8) # 1c00f078 <ex_ret>
1c00f074: 141ffff9    lu12i.w $r25,65535(0xffff)

1c00f078 <ex_ret>:
ex_ret():
1c00f078: 06483800    ertn

1c00f07c <locate>:
locate():
1c00f07c: 157f5fe4    lu12i.w $r4,-263425(0xbfaff)

```

发现其遇到ertn指令未能正常跳转。

### (3) 错误原因

发现我直接漏写了ertn的数据通路.....同样仿照syscall的处理，在ID级就将ertn的信号传递给IF级，减少流水级取消带来的损失

### (4) 修正效果

```

-----[1352075 ns] Number 8'd46 Functional Test Point PASS!!!
-----
[1353517 ns] Error!!!
reference: PC = 0x1c008004, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x00000001
mycpu    : PC = 0x1c008004, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x1c04ef70
-----

```

### (5) 归纳总结 (可选)

## - 错误10：流水级取消后写使能信号处理不当

### (1) 错误现象

如上，写回数据有误。

### (2) 分析定位过程

查看指令，是ld.w指令，查看前序指令流：

DRAM					
> exe_pc[31:0]	1c04ef70	1	1c04ef70	1c008000	1c008004
> addra[15:0]	4000	0	4000		0000
> dina[31:0]	00000000	0	1c04ef70	00000000	5454f254
> douta[31:0]	00000001		000d0004	00000001	1c04ef70
> wea[3:0]	0	0	f	0	

发现在黄色光标处wea信号异常拉高，发现是在判断为syscall指令时取消了流水，但忘记取消mem\_we信号。

查看其原本实现：

```
assign mem_we[0] = inst_st_w | inst_st_h & ~id_alu_result[1] |
inst_st_b & ~id_alu_result[0] & ~id_alu_result[1];
assign mem_we[1] = inst_st_w | inst_st_h & ~id_alu_result[1] |
inst_st_b & id_alu_result[0] & ~id_alu_result[1];
assign mem_we[2] = inst_st_w | inst_st_h & id_alu_result[1] |
inst_st_b & ~id_alu_result[0] & id_alu_result[1];
assign mem_we[3] = inst_st_w | inst_st_h & id_alu_result[1] |
inst_st_b & id_alu_result[0] & id_alu_result[1];
```

### (3) 错误原因

mem\_we未随流水级取消而取消。应给每一项都加上valid信号的约束。

### (4) 修正效果

```
----[1310415 ns] Number 8' d44 Functional Test Point PASS!!!
[1312000 ns] Test is running, debug_wb_pc = 0x1c050eb0
[1322000 ns] Test is running, debug_wb_pc = 0x1c051e50
[1332000 ns] Test is running, debug_wb_pc = 0x1c052df0
----[1332195 ns] Number 8' d45 Functional Test Point PASS!!!
[1342000 ns] Test is running, debug_wb_pc = 0x1c02df68
[1352000 ns] Test is running, debug_wb_pc = 0x1c02ef08
----[1352075 ns] Number 8' d46 Functional Test Point PASS!!!
----[1356755 ns] Number 8' d47 Functional Test Point PASS!!!
=====
Test end!
----PASS!!!
$finish called at time : 1357195 ns : File "D:/Desktop/cdp_edc_local/mycpu_env_12/soc_verify/soc_bram/testbench
run: Time (s): cpu = 00:00:50 ; elapsed = 00:01:19 . Memory (MB): peak = 1085.754 ; gain = 0.000
```

### (5) 归纳总结 (可选)

流水级置为无效或者阻塞时，尤其需要注意写使能相关信号的处理。

## - 错误11: csr写寄存器指令的阻塞未处理好

### (1) 错误现象

pc跑飞了：

```
-----
[1362307 ns] Error!!!
reference: PC = 0x1c008090, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x000b0000
mycpu      : PC = 0x1c00f060, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x1c04ef20
-----
```

```
$finish called at time : 1362347 ns : File "D:/Desktop/cdp_edc_local/mycpu_env_12/soc_verify/soc_bram/testbench
```

### (2) 分析定位过程

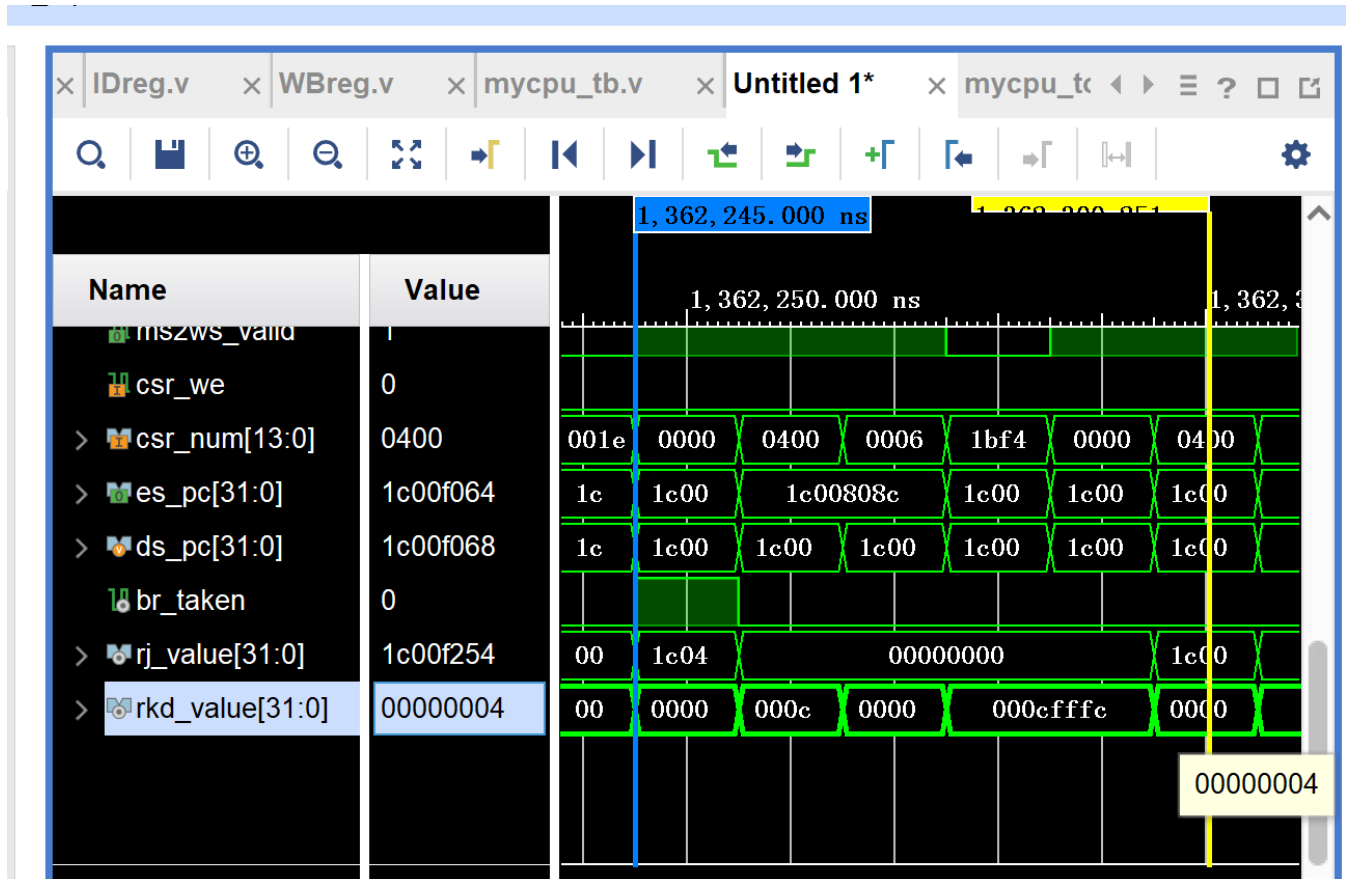
查看PC=0X1C008090时的前序指令：

```

1c008084 <syscall_ex>:
syscall_ex():
1c008084: 00100019    add.w    $r25,$r0,$r0
1c008088: 0400180c    csrrd    $r12,0x6
1c00808c: 5c6fd36c    bne $r27,$r12,28624(0x6fd0) # 1c00f05c <ex_finish>
1c008090: 0400140c    csrrd    $r12,0x5
1c008094: 14fffe0d    lu12i.w  $r13,524272(0x7fff0)

```

查看波形，发现在pc=1c00808c时判断为跳转：



意识到第二个操作数不对。

### (3) 错误原因

上一个指令是csrrd，此时还未从csr寄存器中读出数据，应该在csr读写指令到达wb阶段前将id阻塞。考虑将csr\_re打包进rf\_zip，回传至id阶段，用于判断是否阻塞。逻辑如下：

```

assign ds_stall      = (es_res_from_mem|es_csr_re) & (conflict_r1_exe &
need_r1|conflict_r2_exe & need_r2)|
                    ms_csr_re & (conflict_r1_mem |
conflict_r2_mem);

```

### (4) 修正效果

通过测试：

```

[1312000 ns] Test is running, debug_wb_pc = 0x1c020bb8
----[1315675 ns] Number 8'd44 Functional Test Point PASS!!!
[1322000 ns] Test is running, debug_wb_pc = 0x1c051534
[1332000 ns] Test is running, debug_wb_pc = 0x1c052354
----[1339735 ns] Number 8'd45 Functional Test Point PASS!!!
[1342000 ns] Test is running, debug_wb_pc = 0x1c02d354
[1352000 ns] Test is running, debug_wb_pc = 0x1c02e174
----[1361695 ns] Number 8'd46 Functional Test Point PASS!!!
[1362000 ns] Test is running, debug_wb_pc = 0x1c00f250
----[1367305 ns] Number 8'd47 Functional Test Point PASS!!!
=====
Test end!
----PASS!!!
$finish called at time : 1367755 ns : File "D:/Desktop/cdp_ede_local/mycpu_env_12/soc_verifv/s

```

## (5) 归纳总结 (可选)

# • (三) 实践任务13: 添加其它异常支持

## - 错误1: 变量类型声明有误

### (1) 错误现象

仿真失败, console报错定位至xvlog.log。

### (2) 分析定位过程

查看log:

```

INFO: [VRFC 10-311] analyzing module inst_ram
INFO: [VRFC 10-2263] Analyzing Verilog file
"D:/Desktop/cdp_ede_local/mycpu_env_13/myCPU/EXEreg.v" into library xil_defaultlib
INFO: [VRFC 10-311] analyzing module EXEreg
ERROR: [VRFC 10-3236] concurrent assignment to a non-net 'es_rf_result_tmp' is not
permitted [D:/Desktop/cdp_ede_local/mycpu_env_13/myCPU/EXEreg.v:128]
ERROR: [VRFC 10-2865] module 'EXEreg' ignored due to previous errors
[D:/Desktop/cdp_ede_local/mycpu_env_13/myCPU/EXEreg.v:2]

```

### (3) 错误原因

由上可知, 误将 `es_rf_result_tmp` 声明为寄存器类型:

```
reg [31:0] es_rf_result_tmp;
```

### (4) 修正效果

来到新的报错, 见错误2。



(5) 归纳总结 (可选)

## - 错误2：信号未定义

(1) 错误现象 & 分析定位过程 & 错误原因

xvlog.log中指出信号未定义：

```
[D:/Desktop/cdp_eae_local/mycpu_env_13/myCPU/csr.v:202]
ERROR: [VRFC 10-2989] 'csr_tid_tid' is not declared
[D:/Desktop/cdp_eae_local/mycpu_env_13/myCPU/csr.v:217]
ERROR: [VRFC 10-2989] 'csr_tid_tid' is not declared
[D:/Desktop/cdp_eae_local/mycpu_env_13/myCPU/csr.v:220]
ERROR: [VRFC 10-2989] 'csr_tid_tid' is not declared
[D:/Desktop/cdp_eae_local/mycpu_env_13/myCPU/csr.v:221]
ERROR: [VRFC 10-3236] concurrent assignment to a non-net 'csr_tid_data' is not
permitted [D:/Desktop/cdp_eae_local/mycpu_env_13/myCPU/csr.v:271]
ERROR: [VRFC 10-2989] 'csr_tid_tid' is not declared
[D:/Desktop/cdp_eae_local/mycpu_env_13/myCPU/csr.v:271]
ERROR: [VRFC 10-2865] module 'csr' ignored due to previous errors
[D:/Desktop/cdp_eae_local/mycpu_env_13/myCPU/csr.v:2]
```

补充状态寄存器相关操作流时忘记补全信号定义。

(2) 修正效果

可进入仿真。

## - 错误3：例外ADEM判断有误

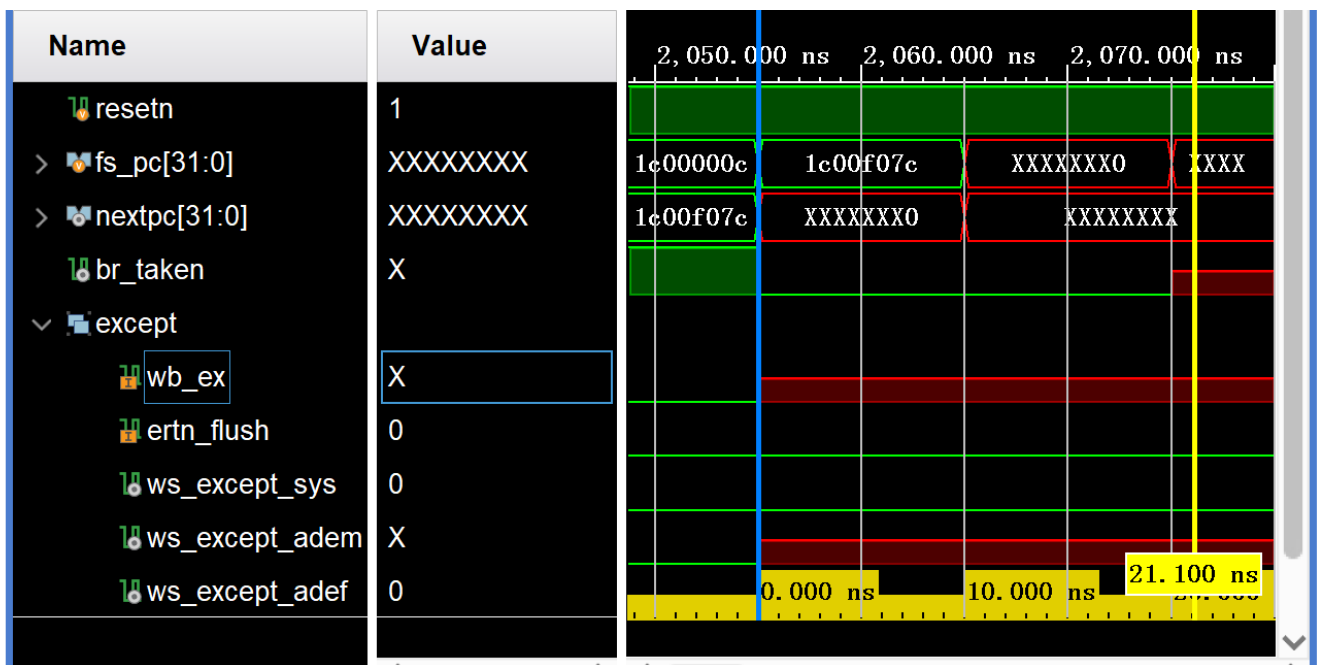
(1) 错误现象

PC不更新：

```
[1162000 ns] Test is running, debug_wb_pc = 0x1c00f07c
[1172000 ns] Test is running, debug_wb_pc = 0x1c00f07c
[1182000 ns] Test is running, debug_wb_pc = 0x1c00f07c
[1192000 ns] Test is running, debug_wb_pc = 0x1c00f07c
[1202000 ns] Test is running, debug_wb_pc = 0x1c00f07c
[1212000 ns] Test is running, debug_wb_pc = 0x1c00f07c
```

(2) 分析定位过程

查看if阶段PC更新状况：



如图所示，nextpc在光标指示处出错，查看nextpc实现：

```

assign nextpc          = wb_ex? ex_entry:
                        ertn_flush? ertn_entry:
                        br_taken ? br_target : seq_pc;

```

在原有基础上只是wb\_ex做了改动，查看WB阶段wb\_ex的实现：

```

assign wb_ex = ws_except_ade | // 用错误地址取指已经发生，
// 故不与ws_valid挂钩
ws_except_int | // 中断由状态寄存器中的计
// 时器产生，不与ws_valid挂钩
(ws_except_adem | ws_except_ine | ws_except_brk |
ws_except_sys) & ws_valid;

```

观察波形得知 ws\_except\_adem 未定义，查看EXE级其实现：

```

assign es_except_adem = (|es_alu_result[1:0]) & es_valid;

```

### (3) 错误原因

忘记将访存地址错误与指令类型挂钩，只有指令类型为ld或者st时会用alu\_result作为地址去访存。修改为：

```

assign es_except_adem = (|es_alu_result[1:0]) & es_valid & ((|es_ld_inst_zip) |
(|es_st_op_zip));

```

### (4) 修正效果

pc往后更新：

```
[ 252000 ns] Test is running, debug_wb_pc = 0x1c047cb0
[ 262000 ns] Test is running, debug_wb_pc = 0x1c048c50
[ 272000 ns] Test is running, debug_wb_pc = 0x1c051a8c
-----
[ 281347 ns] Error!!!
reference: PC = 0x1c06170c, wb_rf_wnum = 0x0a, wb_rf_wdata = 0xc822c7e8
mycpu      : PC = 0x1c06170c, wb_rf_wnum = 0x0a, wb_rf_wdata = 0x00000000
-----
```

(5) 归纳总结 (可选)

错误4：数组越界访问

(1) 错误现象

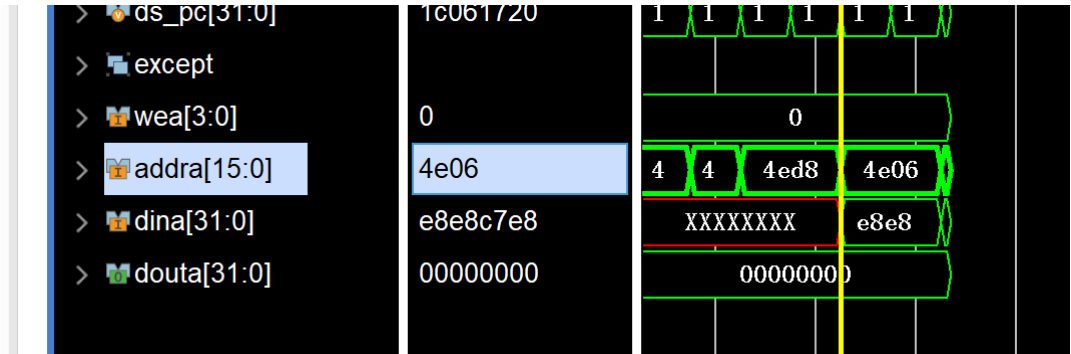
写回数据有误。

(2) 分析定位过程

查看指令类型：

85113	1c061704:	299aa084	st.w	\$r4,\$r4,1704(0x6a8)
85114	1c061708:	299aa0a5	st.w	\$r5,\$r5,1704(0x6a8)
85115	1c06170c:	289aa18a	ld.w	\$r10,\$r12,1704(0x6a8)
85116	1c061710:	289aa086	ld.w	\$r6,\$r4,1704(0x6a8)

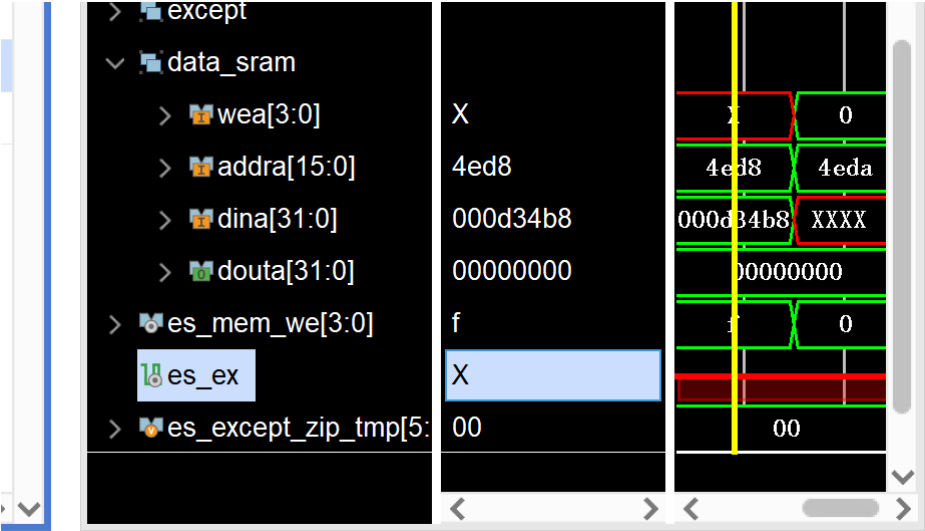
查看数据SRAM的前序读情况：



发现写使能信号始终拉低，查看信号实现：

```
assign data_sram_we = {4{es_valid & ~wb_ex & ~ms_ex & ~es_ex}} & es_mem_we;
```

查看相关信号，发现es\_ex为高阻态：



查看 es\_ex 的实现：

```
assign es_ex = |es_except_zip_tmp[6:1];
```

(3) 错误原因

es\_except\_zip\_tmp 只有6位.....下标越界了。

(4) 修正效果

pc往后更新：

```
----[1180835 ns] Number 8'd38 Functional Test Point PASS!!!
[1182000 ns] Test is running, debug_wb_pc = 0x1c0748ec
[1192000 ns] Test is running, debug_wb_pc = 0x1c075a08
[1202000 ns] Test is running, debug_wb_pc = 0x1c076acc
[1212000 ns] Test is running, debug_wb_pc = 0x1c077b94
----[1218665 ns] Number 8'd39 Functional Test Point PASS!!!
[1222000 ns] Test is running, debug_wb_pc = 0x1c021cd4
[1232000 ns] Test is running, debug_wb_pc = 0x1c022d80
[1242000 ns] Test is running, debug_wb_pc = 0x1c023ebc
[1252000 ns] Test is running, debug_wb_pc = 0x1c024ff0
----[1255725 ns] Number 8'd40 Functional Test Point PASS!!!
[1262000 ns] Test is running, debug_wb_pc = 0x1c02f8c8
[1272000 ns] Test is running, debug_wb_pc = 0x1c02f8c8
[1282000 ns] Test is running, debug_wb_pc = 0x1c02f8c8
[1292000 ns] Test is running, debug_wb_pc = 0x1c02f8c8
[1302000 ns] Test is running, debug_wb_pc = 0x1c02f8c8
```

(5) 归纳总结（可选）

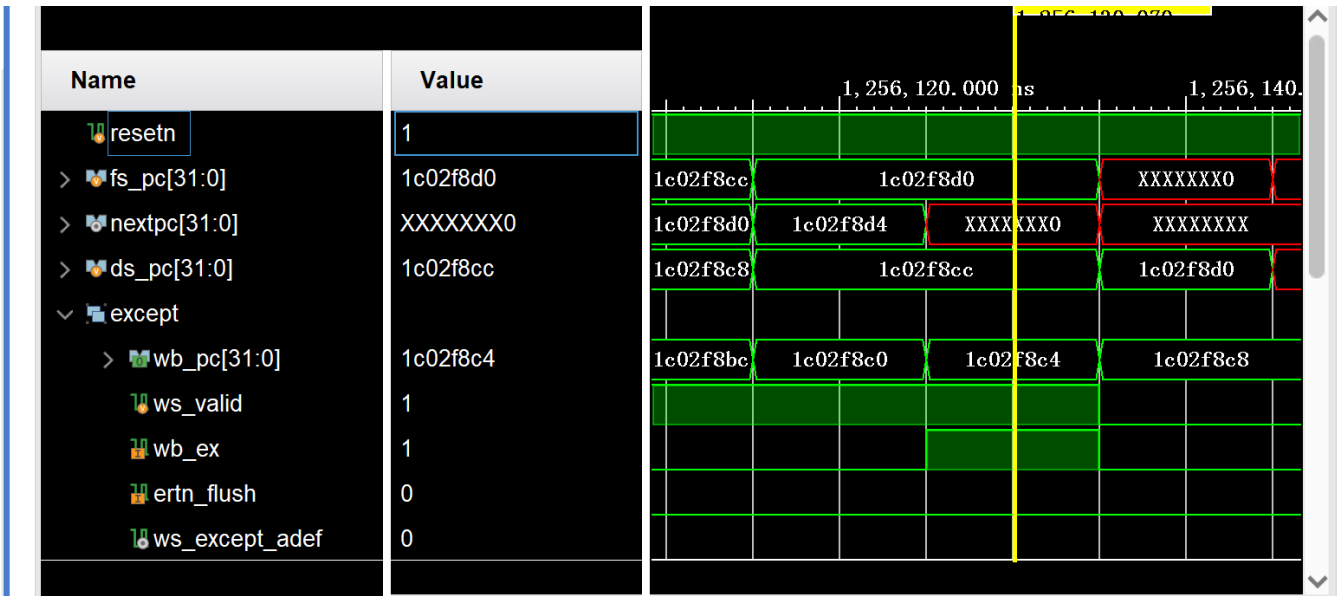
- 错误5：例外ADEM实现依旧有误

(1) 错误现象

pc在进入第40个测试点后卡死。

(2) 分析定位过程

查看nextpc失效的位置：



得知此时由于判断出地址未对齐导致阻塞，而此时还未进入csr测试点，故中断入口地址还未被置入，导致出错。而理论上前面的测试点是暂未出现地址未对齐的情况的，猜测是实现判断except\_ade 有误，当前实现为：

```
assign es_except_adem = ((|es_alu_result[1:0])) & es_valid & ((|es_ld_inst_zip) |
(|es_st_op_zip));
```

(3) 错误原因

上述判断过于粗粒度，alu低2位非0且当前为ld和st类指令未必有错，还需考虑当前指令读写的位数：

```
assign es_except_adem = (((|es_alu_result[1:0])) & (op_st_w | op_ld_w) |
es_alu_result[0] & (op_st_h|op_ld_hu|op_ld_h)) &
es_valid;
```

(4) 修正效果

pc往后更新：

```
-----[1377099 ns] Number 8 d40 functional test point PASS!!!
-----
[1378337 ns] Error!!!
reference: PC = 0x1c008090, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x000b0000
mycpu      : PC = 0x1c008090, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x000b0X00
-----
```

(5) 归纳总结（可选）

## - 错误6: ESTAT的IS域处理有误

### (1) 错误现象

写回数据部分无效，如上。

### (2) 分析定位过程

查看当前指令：

得知是读取ESTAT寄存器，查看其实现：

```
// ESTAT的IS域
assign hw_int_in = 8'b0;
assign ipi_int_in= 1'b0;
always @(posedge clk) begin
    if (reset) begin
        csr_estat_is[1:0] <= 2'b0;
    end
    else if (csr_we && (csr_num == `CSR_ESTAT)) begin
        csr_estat_is[1:0] <= ( csr_wmask[`CSR_ESTAT_IS10] &
csr_wvalue[`CSR_ESTAT_IS10])
                                | (~csr_wmask[`CSR_ESTAT_IS10] &
csr_estat_is[1:0]
                                );
    end

    csr_estat_is[9:2] <= hw_int_in[7:0]; //硬中断
    if (timer_cnt[31:0] == 32'b0) begin
        csr_estat_is[11] <= 1'b1;
    end
    else if (csr_we && csr_num == `CSR_TICLR && csr_wmask[`CSR_TICLR_CLR]
        && csr_wvalue[`CSR_TICLR_CLR])
        csr_estat_is[11] <= 1'b0;
    csr_estat_is[ 12] <= ipi_int_in;      // 核间中断
end
```

### (3) 错误原因

发现改代码的时候漏处理is域的第十位（起初是全部未写核间中断等，全部置零，后修改漏了一位）。

### (4) 修正效果

可通过测试点，见下。

### (5) 归纳总结（可选）

- 错误7: wb\_ex处理不当

(1) 错误现象

pc二度卡住：

Q

⏏

⏏

⏏

⏏

⏏

⏏

----[1383345 ns] Number 8'd47 Functional Test Point PASS!!!

----[1389185 ns] Number 8'd48 Functional Test Point PASS!!!

[1392000 ns] Test is running, debug\_wb\_pc = 0x1c0786e0

[1402000 ns] Test is running, debug\_wb\_pc = 0x1c0786e0

[1412000 ns] Test is running, debug\_wb\_pc = 0x1c0786e0

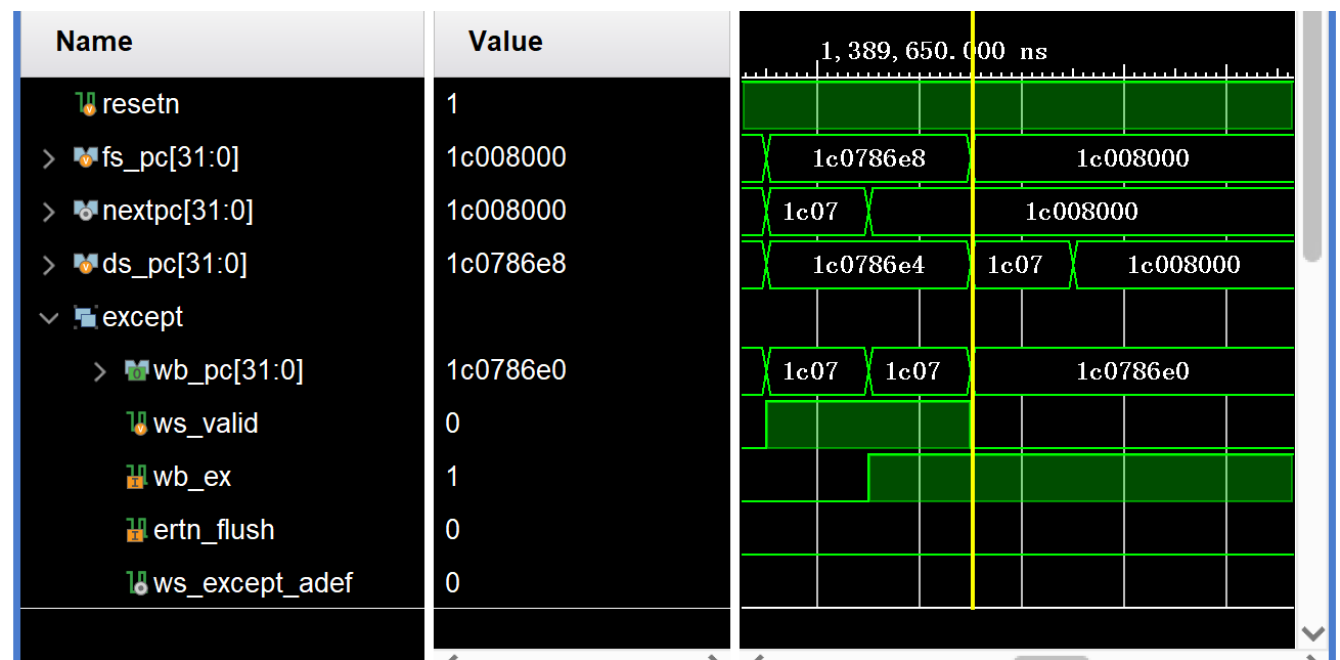
[1422000 ns] Test is running, debug\_wb\_pc = 0x1c0786e0

[1432000 ns] Test is running, debug\_wb\_pc = 0x1c0786e0

[1442000 ns] Test is running, debug wb pc = 0x1c0786e0

(2) 分析定位过程

查看pc更新过程：



发现例外信号拉高后未拉低，导致nextpc始终指向例外入口，查看相关信号得知是has\_int拉高，查看其实现：

```
assign has_int = (~|(csr_estat_is[11:0] & csr_ecfg_lie[11:0])) &
csr_crmd_ie;
```

(3) 错误原因

不知道为啥手抖加了个非.....

修改为：

```
assign has_int = (|(csr_estat_is[11:0] & csr_ecfg_lie[11:0])) & csr_crmd_ie;
```

同时意识到原本对wb\_ex的处理也有误：

```
assign wb_ex = ws_except_adev | // 用错误地址取指已经发生，
故不与ws_valid挂钩
ws_except_int | // 中断由状态寄存器中的计
时器产生，不与ws_valid挂钩
(ws_except_adem | ws_except_ine | ws_except_brk |
ws_except_sys) & ws_valid;
```

若ws\_except\_int和ws\_except\_adev不与ws\_valid挂钩，将导致中断生效后的下一个周期wb\_ex 仍旧拉高，导致流水级被阻塞，使得pc卡住。

#### (4) 修正效果

通过当前测试点：

```
[1902000 ns] Test is running, debug_wb_pc = 0x1c078b04
[1912000 ns] Test is running, debug_wb_pc = 0x1c078b04
[1922000 ns] Test is running, debug_wb_pc = 0x1c078b04
----[1927315 ns] Number 8'd49 Functional Test Point PASS!!!
-----
[1927907 ns] Error!!!
reference: PC = 0x1c008128, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x000d0000
mycpu      : PC = 0x1c008128, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00000000
-----
```

#### (5) 归纳总结（可选）

### - 错误8：wb\_ecode类型未实现完整

#### (1) 错误现象

写回数据有误。

#### (2) 分析定位过程

查看当前指令类型：

```
L18
L19 1c00811c <ine_ex>:
L20 ine_ex():
L21 1c00811c: 00100019 add.w $r25,$r0,$r0
L22 1c008120: 0400180c csrrd $r12,0x6
L23 1c008124: 5c6f3b6c bne $r27,$r12,28472(0x6f38) # 1c00f05c <ex_finish>
L24 1c008128: 0400140c csrrd $r12,0x5
L25 1c00812c: 14fffe0d lu12i.w $r13,524272(0x7fff0)
```

得知当前操作为读ESTAT寄存器，且在ine测试点。根据金标准反推得知查看其上一次例外ecode应该为 001101，但发现其为0：





查看wb\_ecode的实现:

```
assign wb_ecode = ws_except_int ? `ECODE_INT :
                  ws_except_sys? `ECODE_SYS:
                  ws_except_brk? `ECODE_BRK:
                  (ws_except_ade|ws_except_adem) ? `ECODE_ADE:
                  6'b0; // 未包含ALE和TLBR
```

### (3) 错误原因

漏处理int的情况。

### (4) 修正效果

通过当前测试:

[1928947 ns] Error!!!

```
reference: PC = 0x1c008000, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x001d0000
mycpu     : PC = 0x1c01fb14, wb_rf_wnum = 0x1f, wb_rf_wdata = 0x00000000
```

\$finish called at time : 1928987 ns : File "D:/Desktop/cdp\_edelocal/mycpu\_env\_13/soc\_v  
run: Time (s): cpu = 00:01:13 : elapsed = 00:02:00 . Memory (MB): peak = 1432.270 : gai

### (5) 归纳总结 (可选)

## - 错误9: 写使能有效与例外的关系处理不当

### (1) 错误现象

pc有误。

### (2) 分析定位过程

查看当前指令:

93	1c01fb0c:	29801273	st.w	\$r19,\$r19,4(0x4)
94	1c01fb10:	2980127b	st.w	\$r27,\$r19,4(0x4)
95	1c01fb14:	ffffffff	0xffffffff	
96	1c01fb18:	2980027b	st.w	\$r27,\$r19,0
97	1c01fb1c:	2880126d	ld.w	\$r13,\$r19,4(0x4)

### (3) 错误原因

例外时不应拉高任何写有效信号，故应该将写使能与例外判断挂钩：

```
assign ws_rf_we = ws_rf_we_tmp & ws_valid & ~wb_ex;
```

### (4) 修正效果

来到下一个测试点：

```
[1922000 ns] Test is running, debug_wb_pc = 0x1c078b04
----[1927315 ns] Number 8'd49 Functional Test Point PASS!!!
[1932000 ns] Test is running, debug_wb_pc = 0x1c01fb9c
----[1933155 ns] Number 8'd50 Functional Test Point PASS!!!
[1942000 ns] Test is running, debug_wb_pc = 0x1c031860
----[1944675 ns] Number 8'd51 Functional Test Point PASS!!!

-----
[1945367 ns] Error!!!
reference: PC = 0x1c0081ec, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00080000
mycpu    : PC = 0x1c0081ec, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x000d0000
-----
```

### (5) 归纳总结（可选）

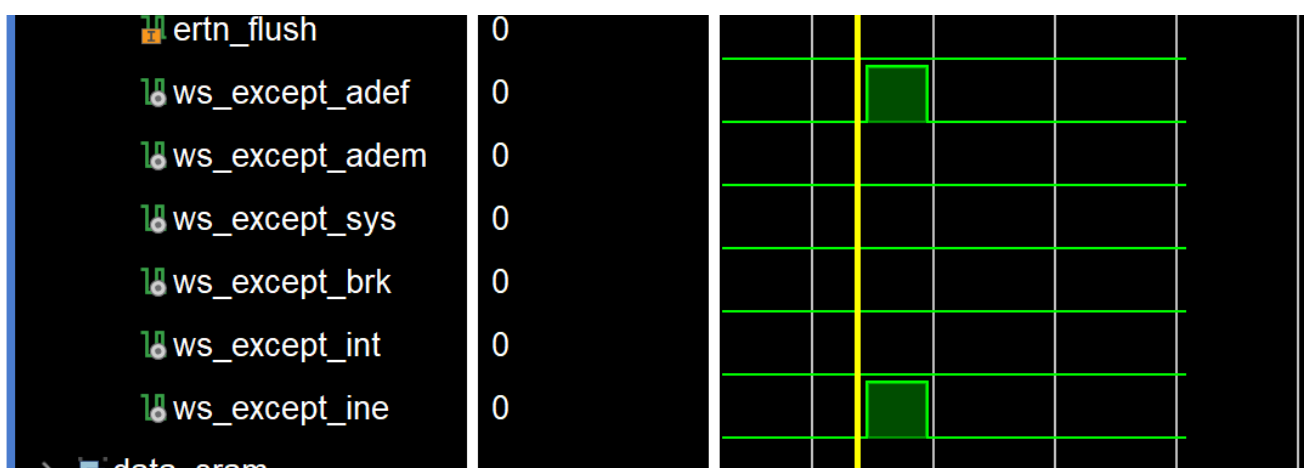
## - 错误10：wb\_ecode未考虑优先级关系

### (1) 错误现象

如上，写回数据有误。

### (2) 分析定位过程

查看当前异常状况：

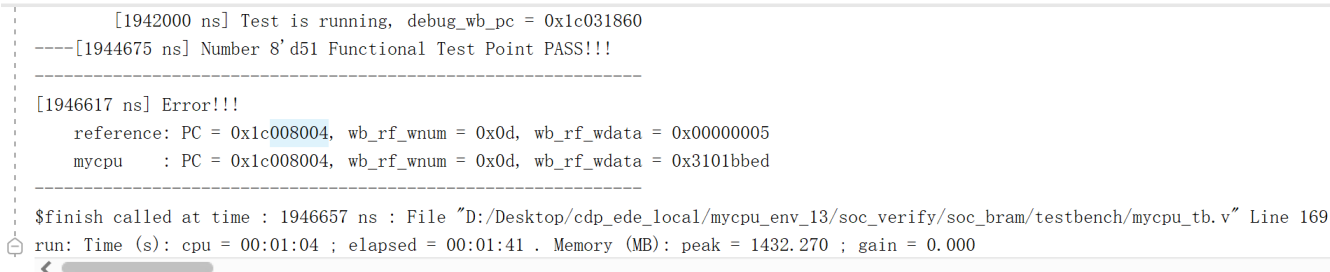


同时发生了两种例外，ecode选取错误。但注意到由于取指地址有误可能才进一步导致指令未定义，故前者优先级应该更高。

### (3) 错误原因

未区分好ecode的优先级。

### (4) 修正效果



**错误11: 在EXE、MEM判断出ERTN后未拉低写使能信号**

如上，写回数据有误。

## 查看当前指令类型:

猜测前序写内存时有误，查看对应地址最近一次被写入的时机：

Value	1,946,500.000 ns																1,946,550.000 ns																1,946,600.000 ns																1
1c008018	1c	1e07	1c07	1c07	1c07	1c074604				3101	3101	3101	3101	bbf9	1c00	1e00	1c00	1c00	1c00	1c00	1c00	1c																											
0	0	f		0	f																	0																											
0																																																	
0000	d17f	4001		0000	4000	4001				4000	c800	c860	77a0		4000	0000	6efb	0000	6efa	0000																													
XXXXXXXX	34	001d	3101	0000	3101	fcfcfffc				0000	fcfc	0000	80802180		00000000	7070	0101	XXXX	0202	XX																													
3101bbcd	00000005		001d																	3101bbcd																													
0																																																	
2002	0006	0000	2010	2004	120e	0000				2000	0000		000e	2000	2001	001e	2002																																
0																																																	
00080000					000b0000																				00080000																								
00					00																				00																								
000																					000																												

104581	1c0745f0:	0280439c	addi.w	\$r28,\$r28,16(0x10)	
104582	1c0745f4:	29801273	st.w	\$r19,\$r19,4(0x4)	
104583	1c0745f8:	2980127b	st.w	\$r27,\$r19,4(0x4)	
104584	1c0745fc:	06483800	ertn		
104585	1c074600:	2980027b	st.w	\$r27,\$r19,0	
104586	1c074604:	2880126d	ld.w	\$r13,\$r19,4(0x4)	
104587	1c074608:	5c015dbb	bne	\$r13,\$r27,348(0x15c) # 1c074764 <inst_error>	
104588	1c07460c:	5c015b3e	bne	\$r25,\$r30,344(0x158) # 1c074764 <inst_error>	
104589	1c074610:	04001c0c	csrrd	\$r12,0x7	

No. 27 / 36

### (3) 错误原因

中断返回后后续指令应该取消，故不应该拉高写使能，而讲义中提及让ertn在wb才修改状态寄存器，并不意味着在exe或者mem阶段就无需对之作判断：

么那个既（什么那个既忽尔有个安进行中断标记，个安修改取指+0）。最大的坑洞四难除了系统在第4种情况，因为流水线中最早只能在译码级才可以知晓写者和相关对象，但是此时取指级很可能已经有一条指令了，pre-IF级也很可能把不合适的PC取指请求发出去了<sup>8</sup>，也就说单纯靠阻塞是没办法彻底解决问题的。为此，我们引入一种特殊的解决方案：**ertn**指令直到写回级才修改CRMD，与此同时**清空流水线并更新取指PC**。这也就是前面提到的**ertn\_flush**信号的由来。对于流水级缓存来说，这个信号与异常信号**wb\_ex**的作用是一样的，所不同的是它不是一个软件可见的异常。为什么这种方案能解决情况4的问题呢？请读者思考一下。

原本exe阶段的例外判断信号（mem阶段类似）为：

```
assign es_ex = |es_except_zip_tmp[5:1];
```

其中该zip的第0位即为ertn的判断信号，故应该将最后一位包含进去，即将异常处理返回也当作是一种“例外”，这或许在定义上不太恰当，但在设计中**es\_ex**信号只参与判断写使能是否有效，而不传递给下一流水级，同理**ms\_ex**，故是合理的（ws\_ex判断例外是另外实现的，并未将ertn包含进去）。

### (4) 修正效果

通过当前测试点：

```
----[1944675 ns] Number 8'd51 Functional Test Point PASS!!!
----[1951695 ns] Number 8'd52 Functional Test Point PASS!!!
      [1952000 ns] Test is running, debug_wb_pc = 0x1c064cc4
-----
[1952447 ns] Error!!!
      reference: PC = 0x1c008238, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00090000
      mycpu      : PC = 0x1c008238, wb_rf_wnum = 0x0c, wb_rf_wdata = 0x00480000
-----
```

## - 错误12：ALE与ADEM搞混

### (1) 错误现象

如上，写回数据有误。

### (2) 分析定位过程

查看当前指令：

```
1c00822c <ale_ex>:
ale_ex():
1c00822c: 00100019 add.w $r25,$r0,$r0
1c008230: 0400180c csrrd $r12,0x6
1c008234: 5c6e2b6c bne $r27,$r12,28200(0x6e28) # 1c00f05c <ex_finish>
1c008238: 0400140c csrrd $r12,0x5
1c00823c: 14fffe0d lu12i.w $r13,524272(0x7fff0)
```

依旧是读ESTAT状态寄存器。

(3) 错误原因

注意到当前为ALE测试，发现未处理此类ecode。前期查看状态寄存器ecode表时：

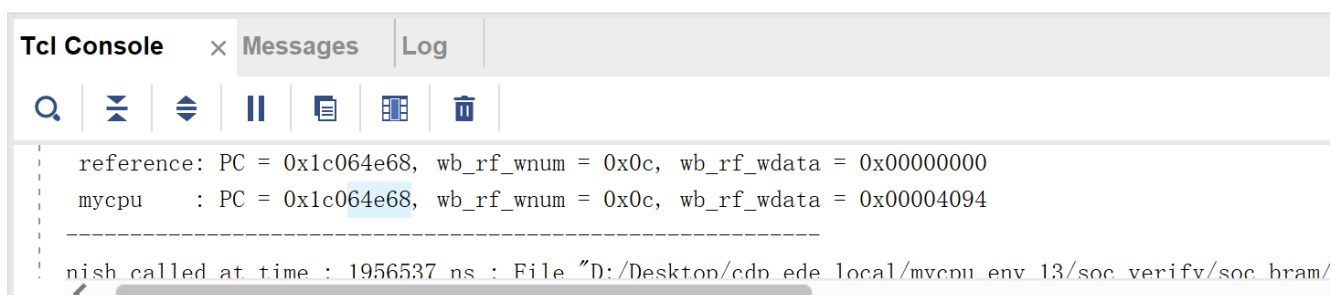
表 7-7 例外编码表

Ecode	EsubCode	例外代号	例外类型
0x0	0	INT	中断。
0x1	0	PIL	load 操作页无效例外
0x2	0	PIS	store 操作页无效例外
0x3	0	PIF	取指操作页无效例外
0x4	0	PME	页修改例外
0x7	0	PPI	页特权等级不合规例外
0x8	0	ADEF	取指地址错例外
	1	ADEM	访存指令地址错例外
0x9	0	ALE	地址非对齐例外
0xB	0	SYS	系统调用例外
0xC	0	BRK	断点例外
0xD	0	INE	指令不存在例外
0xE	0	IPE	指令特权等级错例外

误以为ld和st指令非对齐是处理为 ecode=0x8 , esubcode=9'b1 的情况，仔细阅读将以得知其对应的是ALE例外（目前暂不理解ADEM对应的是何种错误，查找资料也无果，暂且记下）。

(4) 修正效果

来到新的报错：



## - 错误13：中断未复位ALU

### (1) 错误现象

如上，写回数据有误。

### (2) 分析定位过程

查看当前指令类型：

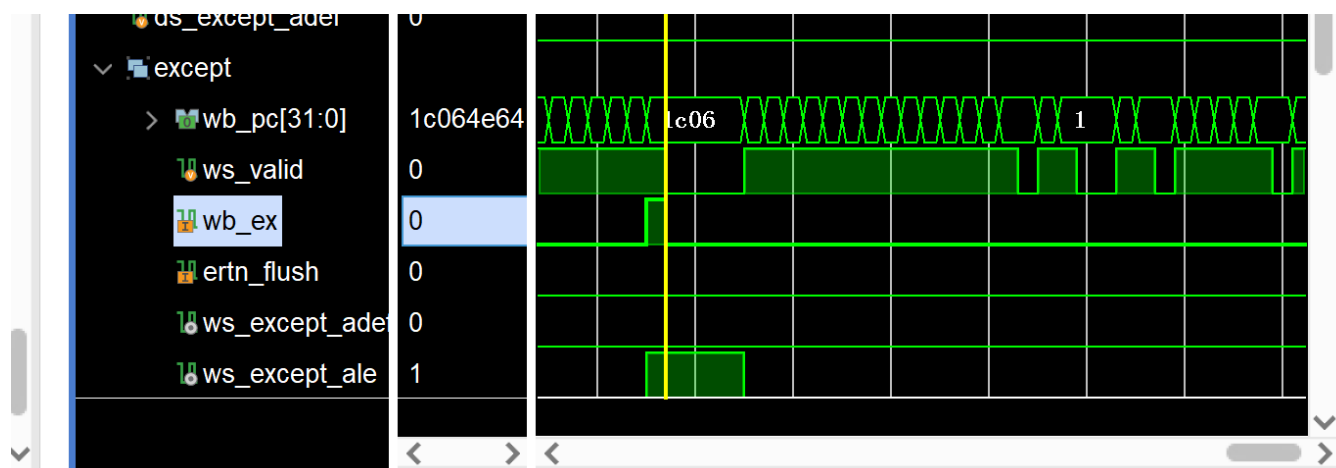
```

3  1c064e54: 0285cc87 addi.w $r7,$r4,371(0x173)
4  1c064e58: 2985c085 st.w $r5,$r4,368(0x170)
5  1c064e5c: 1c00001b pcaddu12i $r27,0
6  1c064e60: 0280237b addi.w $r27,$r27,8(0x8)
7  1c064e64: 2885cc8e ld.w $r14,$r4,371(0x173)
8  1c064e68: 0020418c div.w $r12,$r12,$r16
9  1c064e6c: 5c01173e bne $r25,$r30,276(0x114) # 1c064f80 <inst_error>

```

查看前序操作流：

发现恰好该div指令在wb阶段判断出异常：



### (3) 错误原因

异常发生时alu中乘法和除法器的计数器未被复位，导致后续操作出错。应该令alu的resetrn信号与上~wb\_ex。

### (4) 修正效果

通过当前测试点：

```

----[1966475 ns] Number 8'd54 Functional Test Point PASS!!!
      [1972000 ns] Test is running, debug_wb_pc = 0x1c00f05c
----[1973815 ns] Number 8'd55 Functional Test Point PASS!!!

-----
[1974917 ns] Error!!!
      reference: PC = 0x1c07b090, wb_rf_wnum = 0x0e, wb_rf_wdata = 0x8003602a
      mycpu      : PC = 0x1c07b090, wb_rf_wnum = 0x0e, wb_rf_wdata = 0xf6da602a
-----
$finish called at time : 1974957 ns : File "D:/Desktop/eda_eda_learn/mycpu_ony_12/acc_verify/acc_brom/

```

## - 错误14: es\_ex处理有误

### (1) 错误现象

如上，写回数据有误。

### (2) 分析定位过程

查看当前指令类型：

```

1429  1c07b080: 1c00001b    pcaddu12i   $r27,0
1430  1c07b084: 0280237b    addi.w     $r27,$r27,8(0x8)
1431  1c07b088: 2960d485    st.h       $r5,$r4,-1995(0x835)
1432  1c07b08c: 5c02873e    bne $r25,$r30,644(0x284) # 1c07b310 <inst_error>
1433  1c07b090: 28a0c88e    ld.w       $r14,$r4,-1998(0x832)
1434  1c07b094: 5c027dcf    bne $r14,$r15,636(0x27c) # 1c07b310 <inst_error>
1435  1c07b098: 14003a0c    lu12i.w   $r12,464(0x1d0)
1436  1c07b09c: 03803c19    ori $r25,$r0,0xf

```

查看当前内存地址，为 5617：

data_sram	1c07b090	1c00	1	1	1	1	1c00	1	1	1	1c00	1	1	1	1c00	1	1c07b090	1	1	1			
> es_pc[31:0]	0	0																		f			
> wea[3:0]	5617	0001	0000	0001	0000	8	0000	e	0	8000	0	f	f	fc00	0	5617	4	0	4				
> addra[15:0]	XXXXXXX	5f5f	0	0000	9	5f5f	0	0	X	3737	0	fcfcfffc	2	8	00000000	2	3	0	3838	0	XXXXXXXX	2a2a	0
> dina[31:0]	00000006	00000006																		f6da602a			
> douta[31:0]																							
csr																							

查看最近一次写入该地址的时刻：

data_sram	1c07b088	1c07b080	1c07b084	1c07b088	1c07b08c	1c07b310	1c00
> es_pc[31:0]	c	0	c			0	
> wea[3:0]	5617	ec20	ec22	5617	8001	0000	
> addra[15:0]	f6daf6da	00000000	XXXXXXXX	f6daf6da	00000000	5050f050	2a2a
> dina[31:0]	8003602a		8003602a			f6da602a	
> csr							
> wb_icode[5:0]	00			00	09		
> wb_esubcode[8:0]	000					000	
> es_pc[31:0]	1c07b088	1c07b080	1c07b084	1c07b088	1c07b08c	1c07b310	1c00

给出的掩码是 1100，而此时同时判断出了地址未对齐例外：

Name	Value	1, 974, 300.000 ns				1, 97
>  es_pc[31:0]	1c07b088	1c07b080	1c07b084	1c07b088	1c07b08c	
>  es_except_ale	1					
>  wea[3:0]	c		0	c	0	
>  addra[15:0]	5617	ec20	ec22	5617	8001	
>  dina[31:0]	f6daf6da	00000000	XXXXXXXX	f6daf6da	00000000	
>  douta[31:0]	8003602a		8003602a		f6da602a	

查看使能信号实现：

```
assign es_ex          = |es_except_zip_tmp;
assign es_except_zip = {es_except_ale, es_except_zip_tmp};
assign data_sram_we   = {4{es_valid & ~wb_ex & ~ms_ex & ~es_ex}} & es_mem_we;
```

### (3) 错误原因

es\_ex的判断应该包含ale，即 `assign es_ex = |es_except_zip`。

### (4) 修正效果

通过测试。

```
-----[1973815 ns] Number 8' d55 Functional Test Point PASS!!!
-----[1981295 ns] Number 8' d56 Functional Test Point PASS!!!
      [1982000 ns] Test is running, debug_wb_pc = 0x1c008034
-----[1988775 ns] Number 8' d57 Functional Test Point PASS!!!
      [1992000 ns] Test is running, debug_wb_pc = 0x1c042f78
-----[1992935 ns] Number 8' d58 Functional Test Point PASS!!!
=====
Test end!
---PASS!!!
$finish called at time : 1993385 ns : File "D:/Desktop/cdp_edelocal/mycpu_env_13/soc_verify/soc_b
run: Time (s): cpu = 00:01:00 : elapsed = 00:01:52   Memory (MR): peak = 1228.582 : gain = 0.000
```

## 四.实验总结

## 五.附录（Lab5验收后的思考）

后续验收后老师建议改变流水级切分方式。

起初是在华莱士树的level 3处做的切分，综合时序尚可：



Design Timing Summary					
<div>General Information</div> <div>Timer Settings</div> <div>Design Timing Summary</div> <div>Clock Summary (4)</div> <div>Check Timing (86)</div> <div>Intra-Clock Paths</div> <div>Inter-Clock Paths</div> <div>Other Path Groups</div>	Setup		Hold	Pulse Width	
	Worst Negative Slack (WNS): 4.409 ns		Worst Hold Slack (WHS): 0.053 ns	Worst Pulse Width Slack (WPWS): 3.000 ns	
	Total Negative Slack (TNS): 0.000 ns		Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns	
	Number of Failing Endpoints: 0		Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	
	Total Number of Endpoints: 8959		Total Number of Endpoints: 8959	Total Number of Endpoints: 2314	
	All user specified timing constraints are met.				

但是exp11布局布线后就没法看了：

Design Timing Summary

General Information

Timer Settings

Design Timing Summary

Clock Summary (4)

Check Timing (86)

Intra-Clock Paths

Inter-Clock Paths

Other Path Groups

Setup

Worst Negative Slack (WNS): -1.751 ns

Total Negative Slack (TNS): -421.653 ns

Number of Failing Endpoints: 851

Total Number of Endpoints: 9115

Hold

Worst Hold Slack (WHS): 0.044 ns

Total Hold Slack (THS): 0.000 ns

Number of Failing Endpoints: 0

Total Number of Endpoints: 9115

Pulse Width

Worst Pulse Width Slack (WPWS): 3.000 ns

Total Pulse Width Negative Slack (TPWS): 0.000 ns

Number of Failing Endpoints: 0

Total Number of Endpoints: 2366

Timing constraints are not met.

查看关键路径，意识到此时重点优化对象已不再是乘法器流水级切分，而是讲义中所提的分支判断的路径：

Delay Type	Incr (ns)	Path ...	Location	Netlist Resource(s)
<a href="#">RAMB36E1 (Prop_ramb3...CLKARDCLK_DOADO[7])</a>	(r) 2.454	0.657	Site: RAMB36_X0Y1	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[20]
net (fo=1, routed)	3.204	3.861		data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
			Site: SLICE_X51Y77	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
<a href="#">LUT6 (Prop_lut6_I5_O)</a>	(r) 0.124	3.985	Site: SLICE_X51Y77	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
net (fo=1, routed)	0.000	3.985		data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
			Site: SLICE_X51Y77	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
<a href="#">MUXF7 (Prop_muxf7_I1_O)</a>	(r) 0.245	4.230	Site: SLICE_X51Y77	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
net (fo=1, routed)	0.000	4.230		data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
			Site: SLICE_X51Y77	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
<a href="#">MUXF8 (Prop_muxf8_I0_O)</a>	(r) 0.104	4.334	Site: SLICE_X51Y77	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
net (fo=1, routed)	0.802	5.136		bridge_1x2/douta[15]
			Site: SLICE_X45Y71	bridge_1x2/es_rkd_value[31]_i_33/12
<a href="#">LUT6 (Prop_lut6_I2_O)</a>	(r) 0.316	5.452	Site: SLICE_X45Y71	bridge_1x2/es_rkd_value[31]_i_33/O
net (fo=4, routed)	0.617	6.069		bridge_1x2/ms_alu_result_reg[1]_6
			Site: SLICE_X45Y70	bridge_1x2/es_rkd_value[25]_i_9/I0
<a href="#">LUT3 (Prop_lut3_I0_O)</a>	(r) 0.124	6.193	Site: SLICE_X45Y70	bridge_1x2/es_rkd_value[25]_i_9/O
net (fo=10, routed)	0.233	6.426		cpu/my_memReg/ws_rf_wdata_reg[7]
			Site: SLICE_X45Y70	cpu/my_memReg/es_rkd_value[31]_i_16/I1
<a href="#">LUT2 (Prop_lut2_I1_O)</a>	(r) 0.124	6.550	Site: SLICE_X45Y70	cpu/my_memReg/es_rkd_value[31]_i_16/O
net (fo=32, routed)	0.998	7.548		cpu/my_memReg/ms_mem_result10_out[7]
			Site: SLICE_X44Y64	cpu/my_memReg/es_rkd_value[17]_i_2/I0
<a href="#">LUT4 (Prop_lut4_I0_O)</a>	(r) 0.124	7.672	Site: SLICE_X44Y64	cpu/my_memReg/es_rkd_value[17]_i_2/O
net (fo=2, routed)	0.323	7.996		cpu/my_idReg/ms_rf_zip[16]
			Site: SLICE_X44Y63	cpu/my_idReg/es_rkd_value[17]_i_1/I2
<a href="#">LUT5 (Prop_lut5_I2_O)</a>	(r) 0.124	8.120	Site: SLICE_X44Y63	cpu/my_idReg/es_rkd_value[17]_i_1/O
net (fo=7, routed)	1.051	9.171		cpu/my_idReg/u_regfile/rj_eq_rd_carry_0_0
			Site: SLICE_X46Y64	cpu/my_idReg/u_regfile/rj_ge_rd_u_carry_1_i_8/I3
<a href="#">LUT4 (Prop_lut4_I3_O)</a>	(r) 0.124	9.295	Site: SLICE_X46Y64	cpu/my_idReg/u_regfile/rj_ge_rd_u_carry_1_i_8/O
net (fo=1, routed)	0.000	9.295		cpu/my_idReg/u_regfile_n_69
			Site: SLICE_X46Y64	cpu/my_idReg/rj_ge_rd_u_carry_1/S[0]
<a href="#">CARRY4 (Prop_carry4_S[0]_CO[3])</a>	(r) 0.513	9.808	Site: SLICE_X46Y64	cpu/my_idReg/rj_ge_rd_u_carry_1/CO[3]
net (fo=1, routed)	0.000	9.808		cpu/my_idReg/rj_ge_rd_u_carry_1_n_0
			Site: SLICE_X46Y65	cpu/my_idReg/rj_ge_rd_u_carry_2/CI
<a href="#">CARRY4 (Prop_carry4_CI_CO[3])</a>	(r) 0.117	9.925	Site: SLICE_X46Y65	cpu/my_idReg/rj_ge_rd_u_carry_2/CO[3]
net (fo=5, routed)	1.090	11.015		cpu/my_idReg/u_regfile/fs_pc[0]_i_3_0[0]
			Site: SLICE_X48Y68	cpu/my_idReg/u_regfile/fs_pc[0]_i_7/I0
<a href="#">LUT6 (Prop_lut6_I0_O)</a>	(r) 0.124	11.139	Site: SLICE_X48Y68	cpu/my_idReg/u_regfile/fs_pc[0]_i_7/O
net (fo=3, routed)	1.194	12.332		cpu/my_idReg/u_regfile/fs_pc[0]_i_7_n_0
			Site: SLICE_X56Y84	cpu/my_idReg/u_regfile/fs_pc[0]_i_2_comp_6/I1
<a href="#">LUT6 (Prop_lut6_I1_O)</a>	(r) 0.124	12.456	Site: SLICE_X56Y84	cpu/my_idReg/u_regfile/fs_pc[0]_i_2_comp_6/O
net (fo=50, routed)	0.535	12.992		inst_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[128]
			Site: SLICE_X57Y87	inst_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[128]
<a href="#">LUT6 (Prop_lut6_I5_O)</a>	(r) 0.124	13.116	Site: SLICE_X57Y87	inst_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[128]
net (fo=16, routed)	0.875	13.991		inst_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[132]
			Site: SLICE_X60Y90	inst_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[132]
<a href="#">LUT5 (Prop_lut5_I4_O)</a>	(r) 0.124	14.115	Site: SLICE_X60Y90	inst_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[132]
net (fo=4, routed)	4.882	18.997		inst_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[4].r
<a href="#">RAMB36E1</a>			Site: RAMB36_X7Y43	inst_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[4].r

但若在EXE级再进行分支判断，需要cancel掉两个流水级，相应地耗费更多的cycle，在exp11时选择不再进行修改。

但是在exp12时在进行布局布线后就发现关键路径变为了乘法器：

Path 1

Delay Type	Incr (ns)	Path ...	Location	Netlist Resource(s)
FDRE (Prop fdre_C_Q)	(r) 0.456	-1.846	Site: SLICE_X57Y137	cpu/my_exeReg/u_alu/u_mul/level_3_r_reg[0][37]/Q
net (fo=3, routed)	1.049	-0.797		cpu/my_exeReg/u_alu/u_mul/level_3_r_reg_n_0[0][37]
			Site: SLICE_X53Y147	cpu/my_exeReg/u_alu/u_mul/data_ram_i_481/I0
LUT3 (Prop lut3_I0_O)	(r) 0.124	-0.673	Site: SLICE_X53Y147	cpu/my_exeReg/u_alu/u_mul/data_ram_i_481/O
net (fo=2, routed)	0.670	-0.003		cpu/my_exeReg/u_alu/u_mul/level_4[0]_22[38]
			Site: SLICE_X52Y147	cpu/my_exeReg/u_alu/u_mul/data_ram_i_421/I0
LUT5 (Prop lut5_I0_O)	(r) 0.124	0.121	Site: SLICE_X52Y147	cpu/my_exeReg/u_alu/u_mul/data_ram_i_421/O
net (fo=2, routed)	0.821	0.942		cpu/my_exeReg/u_alu/u_mul/level_5[0]_26[39]
			Site: SLICE_X49Y147	cpu/my_exeReg/u_alu/u_mul/data_ram_i_348/I0
LUT5 (Prop lut5_I0_O)	(r) 0.124	1.066	Site: SLICE_X49Y147	cpu/my_exeReg/u_alu/u_mul/data_ram_i_348/O
net (fo=2, routed)	0.680	1.746		cpu/my_exeReg/u_alu/u_mul/level_6[0]_28[40]
			Site: SLICE_X48Y147	cpu/my_exeReg/u_alu/u_mul/data_ram_i_352/I0
LUT6 (Prop lut6_I0_O)	(r) 0.124	1.870	Site: SLICE_X48Y147	cpu/my_exeReg/u_alu/u_mul/data_ram_i_352/O
net (fo=1, routed)	0.000	1.870		cpu/my_exeReg/u_alu/u_mul/data_ram_i_352_n_0
			Site: SLICE_X48Y147	cpu/my_exeReg/u_alu/u_mul/data_ram_i_248/S[3]
CARRY4 (Prop carry4_S[3]_CO[3])	(r) 0.376	2.246	Site: SLICE_X48Y147	cpu/my_exeReg/u_alu/u_mul/data_ram_i_248/CO[3]
net (fo=1, routed)	0.000	2.246		cpu/my_exeReg/u_alu/u_mul/data_ram_i_248_n_0
			Site: SLICE_X48Y148	cpu/my_exeReg/u_alu/u_mul/data_ram_i_227/C1
CARRY4 (Prop carry4_CI_O[2])	(r) 0.239	2.485	Site: SLICE_X48Y148	cpu/my_exeReg/u_alu/u_mul/data_ram_i_227/O[2]
net (fo=1, routed)	0.536	3.021		cpu/my_exeReg/u_alu_n_323
			Site: SLICE_X49Y148	cpu/my_exeReg/data_ram_i_137/I2
LUT6 (Prop lut6_I2_O)	(r) 0.301	3.322	Site: SLICE_X49Y148	cpu/my_exeReg/data_ram_i_137/O

考虑到最后一个加法器的权重很大，在level6处做流水级切分：

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): -1.403 ns	Worst Hold Slack (WHS): 0.122 ns	Worst Pulse Width Slack (WPS): 0.000 ns
Total Negative Slack (TNS): -260.530 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 582	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 9854	Total Number of Endpoints: 9854	Total Number of Endpoints: 9854

Timing constraints are not met.

虽然还是违约，但是违约时长已有所缩短，最长path的逻辑级数也降下来了：

Intra-Clock Paths - cpu\_clk\_clk\_pll - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	F
Path 1	-1.403	18	35	data_...DCLK	inst_...RDEN	20.554	5.385	15.169	
Path 2	-1.368	16	67	data_...DCLK	inst_...RDEN	20.329	5.137	15.192	
Path 3	-1.360	18	35	data_...DCLK	inst_...RDEN	20.515	5.385	15.130	
Path 4	-1.295	18	35	data_...DCLK	inst_...RDEN	20.259	5.385	14.874	
Path 5	-1.263	18	35	data_...DCLK	inst_...RDEN	20.219	5.385	14.834	

关键路径再度变回生成取指PC的逻辑：

Path 1 - soc_lite_top_timing_summary_routed				
Net Name	Net Delay (ns)	Net Length (mm)	Location	Resource
RAMB36E1 (Prop_ramb36_CLKARDCLK_DOADO7)	(r) 2.454	0.657	Site: RAMB36_X0Y1	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[5]
net (fo=1, routed)	2.274	2.931		data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
LUT6 (Prop_lut6_I3_O)	(r) 0.124	3.055	Site: SLICE_X25Y46	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
net (fo=1, routed)	0.000	3.055		data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
MUXF7 (Prop_muxf7_I1_O)	(r) 0.245	3.300	Site: SLICE_X25Y46	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
net (fo=1, routed)	0.000	3.300		data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
MUXF8 (Prop_muxf8_I0_O)	(r) 0.104	3.404	Site: SLICE_X25Y46	data_ram/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/has_mux_a
net (fo=1, routed)	0.793	4.197		bridge_1x2/douta[7]
LUT6 (Prop_lut6_I2_O)	(r) 0.316	4.513	Site: SLICE_X33Y51	bridge_1x2/ws_rf_wdata_tmp[7]_i_4/O
net (fo=2, routed)	0.626	5.139		bridge_1x2/ms_alu_result_reg[1]_6
LUT4 (Prop_lut4_I2_O)	(r) 0.124	5.263	Site: SLICE_X39Y55	bridge_1x2/ws_rf_wdata_tmp[31]_i_3/O
net (fo=26, routed)	0.837	6.100		cpu/my_memReg/ms_mem_result10_out[0]
LUT4 (Prop_lut4_I0_O)	(r) 0.124	6.224	Site: SLICE_X43Y60	cpu/my_memReg/ws_rf_wdata_tmp[11]_i_2/O
net (fo=3, routed)	0.464	6.688		cpu/my_idReg/ms_rf_zip[10]
LUT6 (Prop_lut6_I5_O)	(r) 0.124	6.812	Site: SLICE_X44Y60	cpu/my_idReg/es_rkd_value[11]_i_2/O
net (fo=1, routed)	0.576	7.388		cpu/my_idReg/u_regfile/es_rkd_value_reg[11]
LUT6 (Prop_lut6_I0_O)	(r) 0.124	7.512	Site: SLICE_X45Y60	cpu/my_idReg/u_regfile/es_rkd_value[11]_i_1/O
net (fo=7, routed)	0.751	8.262		cpu/my_idReg/u_regfile/ds_pc_reg[31][43]
LUT4 (Prop_lut4_I3_O)	(r) 0.124	8.386	Site: SLICE_X45Y64	cpu/my_idReg/u_regfile/fs_pc[5]_i_80/O
net (fo=1, routed)	0.000	8.386		cpu/my_idReg/u_regfile/fs_pc[5]_i_80_n_0
CARRY4 (Prop_carry4_S11_CO[3])	(r) 0.550	8.936	Site: SLICE_X45Y64	cpu/my_idReg/u_regfile/fs_pc_reg[5]_i_50/CO[3]
net (fo=1, routed)	0.000	8.936		cpu/my_idReg/u_regfile/fs_pc_reg[5]_i_50_n_0
CARRY4 (Prop_carry4_CI_CO[3])	(r) 0.114	9.050	Site: SLICE_X45Y65	cpu/my_idReg/u_regfile/fs_pc_reg[5]_i_28/CO[3]
net (fo=1, routed)	0.000	9.050		cpu/my_idReg/u_regfile/fs_pc_reg[5]_i_28_n_0
CARRY4 (Prop_carry4_CI_CO[3])	(r) 0.114	9.164	Site: SLICE_X45Y66	cpu/my_idReg/u_regfile/fs_pc_reg[5]_i_12/CO[3]
net (fo=2, routed)	1.537	10.701		cpu/my_idReg/u_regfile/rj_ge_rd_u

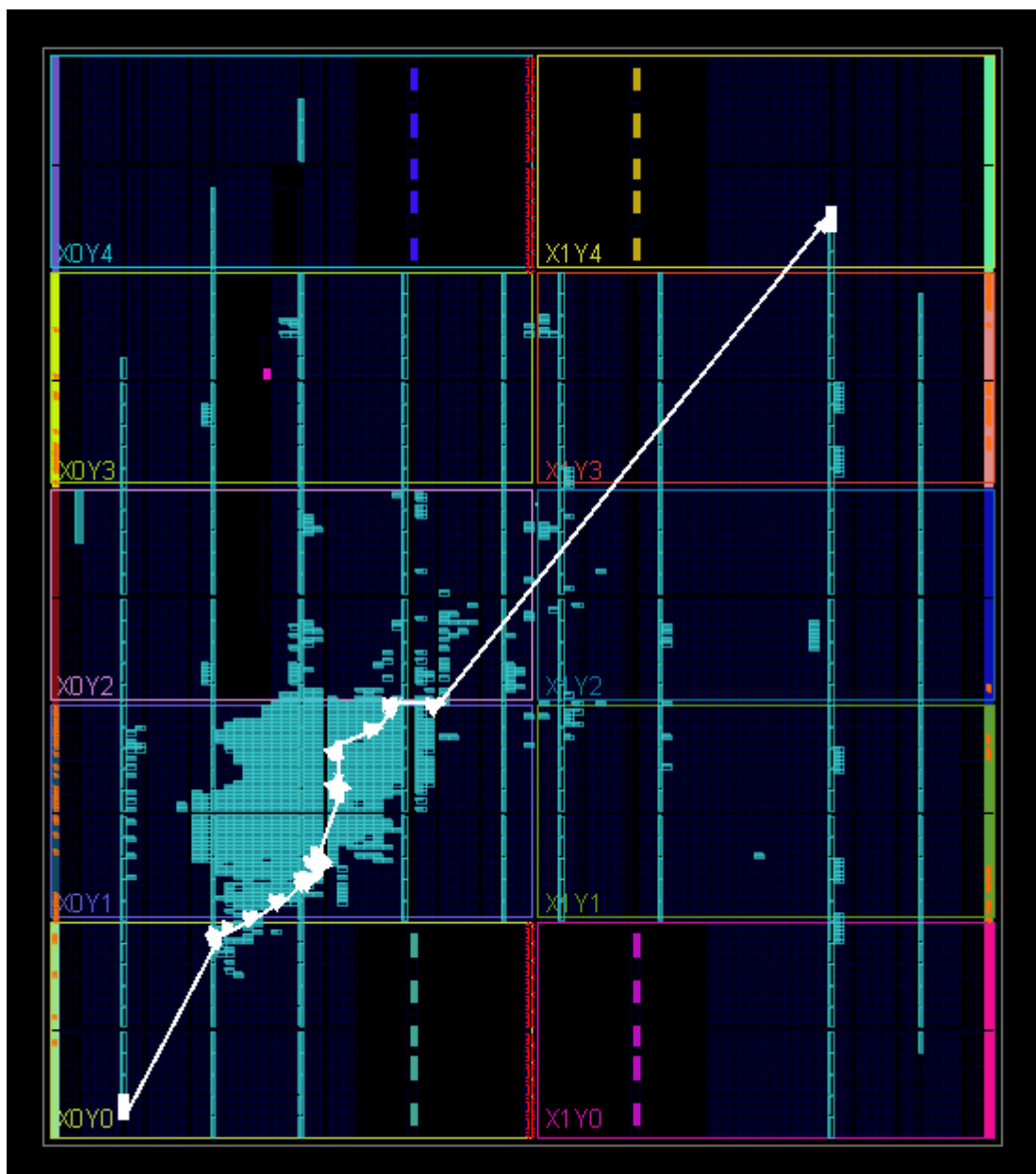
同时注意到此时占大头的还是线延时而不是逻辑延时，查找资料得知可能是某些高扇出信号导致布线过长，用如下指令查看高扇出的信号：

```
report_high_fanout_nets -fanout_greater_than 50 -max_nets 100
```

发现扇出较高的使用的都是LUT资源（据说高扇出的信号最好改成寄存输出），但占大头的在顶层模块，不考虑修改。

Net Name	Fanout	Driver Type
u_confreg/p_0_in	1074	LUT1
cpu/my_idReg/u_regfile/D[2]	257	LUT6
cpu/my_idReg/u_regfile/D[3]	257	LUT6
cpu/my_idReg/u_regfile/D[4]	257	LUT6
cpu/my_idReg/u_regfile/D[5]	257	LUT6
cpu/my_wbReg/D[0]	257	LUT3
cpu/my_wbReg/D[1]	257	LUT3
cpu/my_wbReg/D[2]	257	LUT3
cpu/my_wbReg/D[3]	257	LUT3
cpu/my_wbReg/D[4]	257	LUT3
cpu/my_wbReg/D[5]	257	LUT3
cpu/my_wbReg/D[6]	257	LUT3
cpu/my_wbReg/D[7]	257	LUT3
cpu/my_idReg/es_except_zip0	214	LUT6
cpu/my_exeReg/u_alu/u_div/es_valid_reg_1	159	LUT6
cpu/my_memReg/ms_valid_reg_0	158	FDRS
cpu/my_exeReg/u_alu/u_div/cpu_resetrn_reg_0[0]	157	LUT2
cpu/my_exeReg/es_alu_src2[0]	153	FDRS

考虑到线延迟还有可能是因为拥塞引起的走线不正常，故查看最长path的走线：



基本还是走的最短路径，线延迟长是因为本身物理位置较远，而为了实现数据前递技术，走线跨度不得不大.....（似乎无力挣扎了）