**实验八报告**

学号 2017K8009929044

2017K8009929025

姓名 李昊宸 李颖彦

箱子号 33

一、实验任务（10%）

1. 加深对流水线结构的理解。

2. 了解 MIPS体系结构中的 CP0寄存器。

3. 学会在流水线 CPU中添加例外和中断的方法。

二、实验设计（40%）

（一）总体设计思路

在WB写回级模块添加CP0寄存器的设计（这一次只需要添加STATUS、CAUSE和EPC寄存器）。新增MTC0和MFC0指令实现对CP0寄存器的操作。新增SYSCALL系统调用例外，发生时将PC修改为0xbfc00380，执行中断处理程序，并将中断号0x08写入CAUSE寄存器的EXCODE位，并清除流水线缓存。新增ERET指令，

将PC修改为EPC寄存器内的值，清中断，清除流水线缓存。

（二）重要模块1设计：CP0寄存器模块

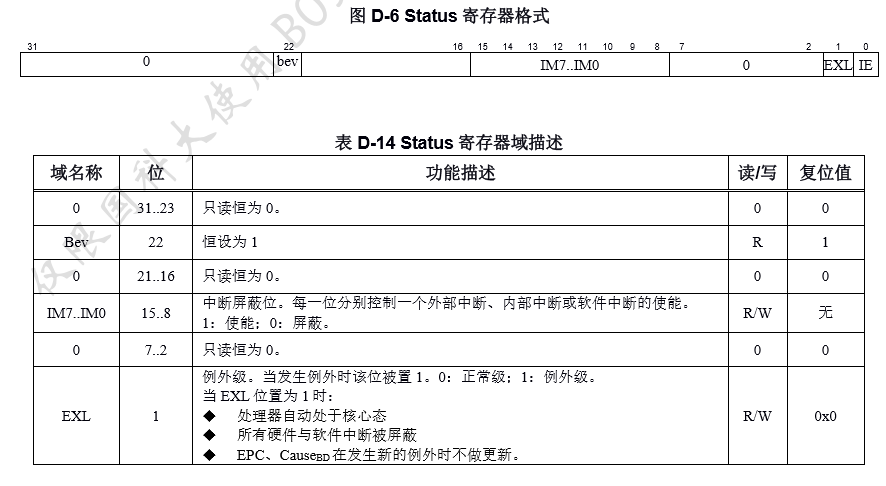
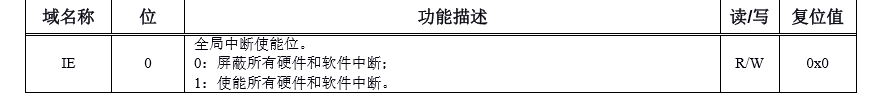
1. 工作原理

在WB模块增加CP0寄存器模块，实现对协寄存器的操作。

1. 功能描述

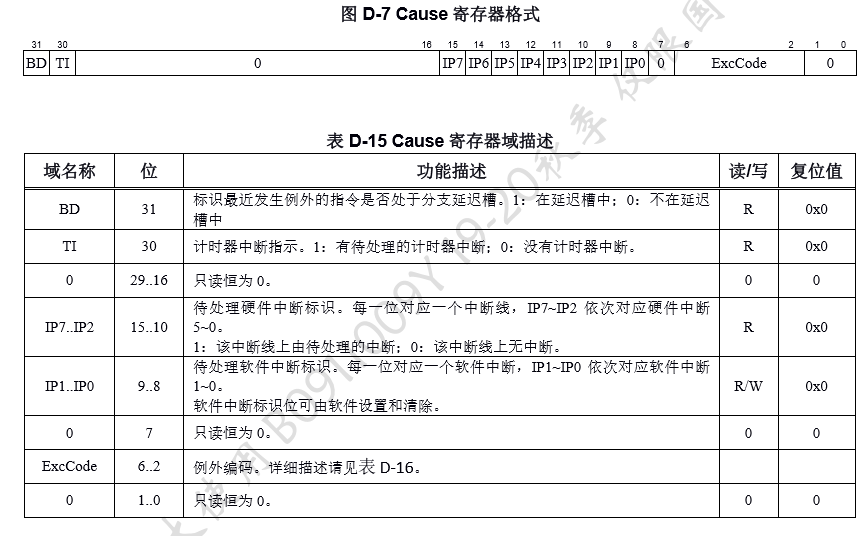
新增的寄存器有三个：STATUS、CAUSE和EPC。

Status寄存器是一个可读写寄存器，包含有处理器操作模式、中断使能以及处理器状态诊断信息。



表一 STATUS寄存器的域

Cause 寄存器主要用于描述最近一次例外的原因。除此之外还对软件中断进行了控制。除了IP1..IP0域外Cause寄存器的其它域对于软件均只读。



表二 CAUSE寄存器的域

EPC寄存器是一个 32位可读写寄存器，其包含例外处理完成后继续开始执行的指令的 PC。 在响应精确例外时，处理器向 EPC寄存器中写入：

(1) 直接触发例外的指令的 PC。

(2) 当直接触发例外的指令位于分支延迟槽时，记录该指令前一条分支或跳转指令的 PC，同时Cause.BD置为 1。

在响应非精确例外时，处理器向 EPC寄存器中写入例外处理完成后继续执行的指令的 PC。

当 Status寄存器的 EXL位为 1时，发生例外时不更新 EPC寄存器。



表三 EPC寄存器的域

1. 具体代码实现

模块封装：

*module CP0(*

*input clk,*

*input reset,*

*input mtc0\_we,*

*input [4:0] c0\_addr,*

*input [31:0] c0\_wdata, //CP0寄存器写入数据*

*input wb\_ex, //写回级例外信号*

*input eret\_flush, //eret信号*

*input wb\_bd, //延迟槽信号*

*input [5:0] ext\_int\_in, //硬件中断信号*

*input [4:0] wb\_excode, //例外编号*

*input [31:0] wb\_pc,*

*output[31:0] rdata //CP0寄存器读出数据*

*);*

STATUS寄存器：

*wire [31:0] c0\_status;*

*wire [8:0] c0\_status\_31\_23;*

*wire c0\_status\_bev;*

*wire [5:0] c0\_status\_21\_16;*

*reg [7:0] c0\_status\_im;*

*wire [5:0] c0\_status\_7\_2;*

*reg c0\_status\_exl;*

*reg c0\_status\_ie;*

*assign c0\_status = {*

*c0\_status\_31\_23,*

*c0\_status\_bev,*

*c0\_status\_21\_16,*

*c0\_status\_im,*

*c0\_status\_7\_2,*

*c0\_status\_exl,*

*c0\_status\_ie*

*};*

*assign c0\_status\_31\_23 = 9'b0;*

*assign c0\_status\_21\_16 = 6'b0;*

*assign c0\_status\_7\_2 = 6'b0;*

*assign c0\_status\_bev = 1'b1;*

*always @(posedge clk) begin*

*if(mtc0\_we && c0\_addr == `CR\_STATUS)*

*c0\_status\_im <= c0\_wdata[15:8];*

*end*

*always @(posedge clk) begin*

*if(reset)*

*c0\_status\_exl <= 1'b0;*

*else if(wb\_ex)*

*c0\_status\_exl <= 1'b1;*

*else if(eret\_flush)*

*c0\_status\_exl <= 1'b0;*

*else if(mtc0\_we && c0\_addr == `CR\_STATUS)*

*c0\_status\_exl <= c0\_wdata[1];*

*end*

*always @(posedge clk) begin*

*if(reset)*

*c0\_status\_ie <= 1'b0;*

*else if(mtc0\_we && c0\_addr == `CR\_STATUS)*

*c0\_status\_ie <= c0\_wdata[0];*

*end*

STATUS寄存器中可写的有im段、exl段和ie段。对这三个段的操作类似同步ram，有同步写、异步读的特性。其中要注意的是，exl值的来源有三个：MTC0写入、例外信号和eret信号。

CAUSE寄存器：

*wire [31:0] c0\_cause;*

*reg c0\_cause\_bd;*

*reg c0\_cause\_ti;*

*wire [13:0] c0\_cause\_29\_16;*

*reg [7 :0] c0\_cause\_ip;*

*wire c0\_cause\_7;*

*reg [4 :0] c0\_cause\_excode;*

*wire [1 :0] c0\_cause\_1\_0;*

*assign c0\_cause = {*

*c0\_cause\_bd,*

*c0\_cause\_ti,*

*c0\_cause\_29\_16,*

*c0\_cause\_ip,*

*c0\_cause\_7,*

*c0\_cause\_excode,*

*c0\_cause\_1\_0*

*};*

*assign c0\_cause\_29\_16 = 14'b0;*

*assign c0\_cause\_7 = 1'b0;*

*assign c0\_cause\_1\_0 = 2'b0;*

*always @(posedge clk) begin*

*if(reset)*

*c0\_cause\_bd <= 1'b0;*

*else if(wb\_ex && !c0\_status\_exl)*

*c0\_cause\_bd <= wb\_bd;*

*end*

*always @(posedge clk) begin*

*if(reset)*

*c0\_cause\_ti <= 1'b0;*

*else if(mtc0\_we && c0\_addr == `CR\_COMPARE)*

*c0\_cause\_ti <= 1'b0;*

*else if(count\_eq\_compare) c0\_cause\_ti <= 1'b1;*

*end*

*always @(posedge clk) begin*

*if(reset)*

*c0\_cause\_ip[7:2] <= 6'b0;*

*else begin*

*c0\_cause\_ip[7] <= ext\_int\_in[5] | c0\_cause\_ti;*

*c0\_cause\_ip[6:2] <= ext\_int\_in[4:0];*

*end*

*end*

*always @(posedge clk) begin*

*if(reset)*

*c0\_cause\_ip[1:0] <= 2'b0;*

*else if(mtc0\_we && c0\_addr == `CR\_CAUSE)*

*c0\_cause\_ip[1:0] <= c0\_wdata[9:8];*

*end*

*always @(posedge clk) begin*

*if(reset)*

*c0\_cause\_excode <= 1'b0;*

*else if(wb\_ex)*

*c0\_cause\_excode <= wb\_excode;*

*end*

CAUSE寄存器中可写的有bd段、ti段、ip段和excode段。对这四个段的操作类似同步ram，有同步写、异步读的特性。其中要注意的是，bd段仅当例外信号拉高且延迟槽信号为高时才被置为1；ti值的来源有两个：当计时器发出时钟中断，以及COMPARE寄存器被修改时它置为0；ip信号略微复杂，分为了第七位（响应时钟中断和外部中断）、第六到第二位（响应外部中断）、低两位（软中断）。

EPC寄存器：

*wire [31:0] c0\_epc;*

*reg [31:0] c0\_epc\_domain;*

*assign c0\_epc = c0\_epc\_domain;*

*always @(posedge clk) begin*

*if(wb\_ex && !c0\_status\_exl)*

*c0\_epc\_domain <= wb\_bd ? wb\_pc - 3'h4 : wb\_pc;*

*else if (mtc0\_we && c0\_addr == `CR\_EPC)*

*c0\_epc\_domain <= c0\_wdata;*

*end*

EPC寄存器存放例外发生的指令返回地址。如果在延迟槽中就记录上一条指令的地址，如果不在就记录当前的地址。

读寄存器端口：

*assign rdata = (c0\_addr == `CR\_STATUS ) ? c0\_status*

*:(c0\_addr == `CR\_CAUSE ) ? c0\_cause*

*:(c0\_addr == `CR\_EPC ) ? c0\_epc*

*:32'b0;*

（二）重要模块2设计：特权指令--MFC0、MTC0和ERET指令、SYSCALL指令

1. 工作原理

MFC0

汇编格式：MFC0 rt, rd, sel

功能描述：从协处理器 0的寄存器取值

操作定义：GPR[rt] ← CP0[rd, sel]

例外 ：无

MTC0

汇编格式：MTC0 rt, rd, sel

功能描述：向协处理器 0的寄存器存值

操作定义：CP0[rd, sel] ← GPR[rt]

例外 ：无

ERET

汇编格式：ERET

功能描述：从中断、例外处理返回。

操作定义：PC ← EPC，Status.EXL ← 0，刷新流水线。

例外 ：无

附 ：ERET指令没有延迟槽

SYSCALL

汇编格式：SYSCALL

功能描述：触发系统调用例外。

操作定义：SignalException(SystemCall)

例外 ：系统调用例外

1. 功能描述

特权指令的实现与普通指令的实现类似，需要在译码级译码相应的数据通路。考虑到MTC0和MFC0指令都需要rt的值，需要修改原先的数据通路，将rt也传递至WB级。

ERET在译码级需要将eret\_flush信号拉高，当执行到WB级的时候需要清空流水线，并将取值级nextpc的值更新为EPC的值。为读取EPC的值，需要在译码级将dest设置为`CR\_EPC。

SYSCALL与ERET类似，在WB也需要清空流水线。不过它拉高的是ex例外信号，更新nextpc的值为例外入口32’hbfc00380，并将6’h08写入excode段。

1. 具体代码实现

译码级：

*assign inst\_mfc0 = op\_d[6'h10] & rs\_d[5'h00] & (ds\_inst[10:3] == 8'h00);*

*assign inst\_mtc0 = op\_d[6'h10] & rs\_d[5'h04] & (ds\_inst[10:3] == 8'h00);*

*assign syscall = op\_d[6'h00] & func\_d[6'h0c];*

*assign inst\_eret = op\_d[6'h10] & (ds\_inst[25] == 1'b1) & (ds\_inst[24:6] == 19'b0) &func\_d[6'h18];*

*assign gr\_we = （之前的省略）& ~inst\_mtc0 & ~inst\_eret & ~syscall;*

*assign dest = inst\_eret ? `CR\_EPC :*

*dst\_is\_r31 ? 5'd31 :*

*dst\_is\_rt ? rt :*

*rd;*

*wire ds\_ex;*

*assign ds\_ex = (syscall)? 1'b1 : 1'b0;*

*wire [4:0] ds\_excode;*

*assign ds\_excode = (syscall)? 5'h08 :5'h0;*

写回级：

*wire [5:0] ext\_int\_in; //外部硬件中断*

*assign ext\_int\_in = 6'b0;*

*wire [31:0] cp0\_rdata;*

*wire wb\_ex;*

*assign wb\_ex = ex & ws\_valid;*

*wire eret\_flush;*

*assign eret\_flush = inst\_eret;*

*wire mtc0\_we;*

*assign mtc0\_we = ws\_valid && inst\_mtc0 && !wb\_ex;*

*CP0 cp0\_reg(*

*.clk (clk),*

*.reset (reset),*

*.mtc0\_we (inst\_mtc0),*

*.c0\_addr (ws\_dest), //dest 当Ì¡Àmfc0时º¡À为ard*

*.c0\_wdata (ws\_rt\_value),*

*.wb\_ex (wb\_ex), //写回级例外信号*

*.eret\_flush (eret\_flush),//清中断，清流水线*

*.wb\_bd (wb\_bd), //延迟槽信号*

*.ext\_int\_in (ext\_int\_in), //硬件中断信号*

*.wb\_excode (wb\_excode), //例外编号*

*.wb\_pc (ws\_pc),*

*.rdata (cp0\_rdata)*

*);*

*wire eret;*

*assign eret = inst\_eret & ws\_valid;*

*assign ws\_to\_fs\_bus = {*

*wb\_ex,*

*eret,*

*cp0\_rdata*

*};*

*assign rf\_we = ws\_gr\_we&&ws\_valid;*

*assign rf\_waddr = (inst\_mfc0)?rt*

*:ws\_dest;*

*assign rf\_wdata = (inst\_mfc0)?cp0\_rdata*

*:ws\_final\_result;*

取值级：

*assign nextpc = wb\_ex? 32'hbfc00380 :*

*inst\_eret? cp0\_rdata :*

*br\_taken ? br\_target : seq\_pc;*

（三）重要模块3设计：清空流水线

1. 工作原理

在出现例外和eret返回时，已经进入流水线的指令都是无效指令，需要将他们清除。也就是说，我们将判定例外的处理逻辑放置在WB级，当例外到达WB级时，IF、ID、EXE、MEM级的指令需要全部作废，并且将跳转地址通过总线传递给IF模块。另外，如果例外处于MEM级，而位于EXE级的指令是乘除法或者store指令的话，也需要将EXE级的指令清空。

1. 功能描述

为实现功能，需要引入以下几条总线：

WB级到IF级的总线

WB级到ID级的总线

WB级到EXE级的总线

WB级到MEM级的总线

MEM级到EXE级的总线

另外，为处理指令是否在延迟槽中的情况，需要补充ID级到IF级的总线，用于传输ID级的指令是否为跳转指令的信号。

1. 具体代码实现

写回级：

*wire wb\_ex;*

*assign wb\_ex = ex & ws\_valid;*

*wire eret;*

*assign eret = inst\_eret & ws\_valid;*

*assign ws\_to\_fs\_bus = {*

*wb\_ex,*

*eret,*

*cp0\_rdata*

*};*

*assign ws\_to\_ds\_bus = wb\_ex | eret;*

*assign ws\_to\_es\_bus = wb\_ex | eret;*

*assign ws\_to\_ms\_bus = wb\_ex | eret;*

访存级：

*assign {*

*ex, //125*

*inst\_eret, //124*

*ms\_bd, //123*

*ms\_excode, //122:118*

*rt, //117:113*

*inst\_mfc0, //112*

*inst\_mtc0, //111*

*ms\_rt\_value, //110:79*

*inst\_lwl, //78*

*inst\_lwr, //77*

*load\_choice, //76:75*

*inst\_lb, //74*

*inst\_lbu, //73*

*inst\_lh, //72*

*inst\_lhu, //71*

*ms\_res\_from\_mem, //70:70*

*ms\_gr\_we , //69:69*

*ms\_dest , //68:64 //mfc0时为rd*

*ms\_alu\_result , //63:32*

*ms\_pc //31:0*

*} = (!ws\_to\_ms\_bus ) ? es\_to\_ms\_bus\_r : 0; //清空流水线缓存*

*assign ms\_ex = ex & ms\_valid;*

*assign ms\_to\_es\_bus = ms\_ex | inst\_eret &ms\_valid;*

执行级：

*assign {*

*ex, //171*

*inst\_eret, //170*

*es\_bd, //169*

*es\_excode, //168:164*

*rt, //163:159*

*inst\_mfc0, //158*

*inst\_mtc0, //157*

*inst\_sb, //156*

*inst\_sh, //155*

*inst\_swl, //154*

*inst\_swr, //153*

*inst\_lwl, //152*

*inst\_lwr, //151*

*load\_choice, //150:149*

*inst\_lb, //148*

*inst\_lbu, //147*

*inst\_lh, //146*

*inst\_lhu, //145*

*inst\_mflo, //144*

*inst\_mfhi, //143*

*inst\_mthi, //142*

*inst\_mtlo, //141*

*div\_or\_mul , //140:137*

*es\_src2\_is\_zimm, //136*

*es\_alu\_op , //135:124*

*es\_load\_op , //123:123*

*es\_src1\_is\_sa , //122:122*

*es\_src1\_is\_pc , //121:121*

*es\_src2\_is\_imm , //120:120*

*es\_src2\_is\_8 , //119:119*

*es\_gr\_we , //118:118 //写回寄存器*

*es\_mem\_we , //117:117*

*es\_dest , //116:112 //对于mfc0，dest是rd*

*es\_imm , //111:96*

*es\_rs\_value , //95 :64*

*es\_rt\_value , //63 :32*

*es\_pc //31 :0*

*} = (!ws\_to\_es\_bus & !ms\_to\_es\_bus ) ? ds\_to\_es\_bus\_r : 0; //这里对于执行级，只要访存和写回有例外或eret，都清零*

译码级：

*assign fs\_bd = (inst\_beq|| inst\_bne || inst\_jal*

*|| inst\_jr|| inst\_bgez|| inst\_bgtz*

*|| inst\_blez|| inst\_bltz|| inst\_j*

*|| inst\_bltzal|| inst\_bgezal|| inst\_jalr) & ds\_valid;*

*assign br\_bus = {fs\_bd,br\_taken,br\_target};*

*wire ds\_bd;*

*assign {*

*ds\_bd,*

*ds\_inst,*

*ds\_pc } = (!ws\_to\_ds\_bus) ? fs\_to\_ds\_bus\_r : 0;*

取值级：

*assign fs\_inst = (~wb\_ex & ~inst\_eret )? inst\_sram\_rdata : 32'b0;*

*/////对于例外和eret的处理，不修改fs\_pc的值，而是将fs\_inst指令变为0，就可以起到清除的效果*

三、实验过程（50%）

（一）实验流水账

2019/10/24 20:00 – 24:00 加入CP0寄存器

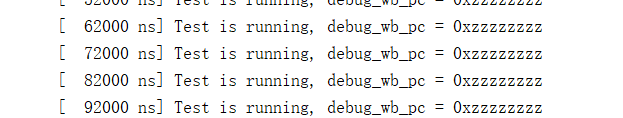
2019/10/25 0:00 – 8:00 加入特权指令

2019/10/25 17:00 – 20:00 书写实验报告

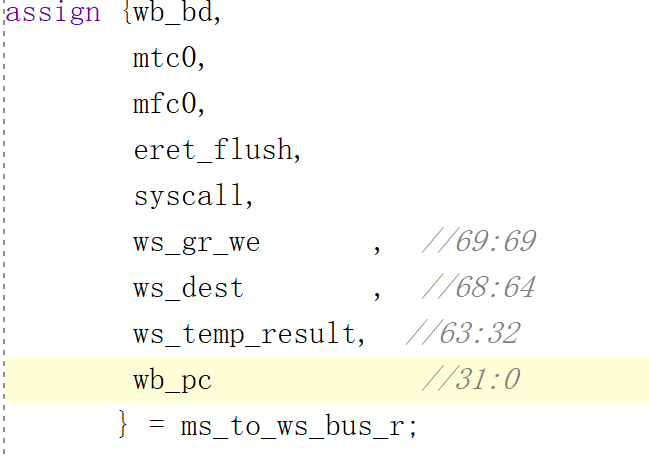
（二）错误记录

1、错误1：全部为高阻值

（1）错误现象



（2）分析定位过程



（3）错误原因

原来是自己为了调用省事把cp0需要的ws\_pc改成了wb\_pc,但原有代码需要ws\_pc;

（4）修正效果



2、错误2： pc启动异常

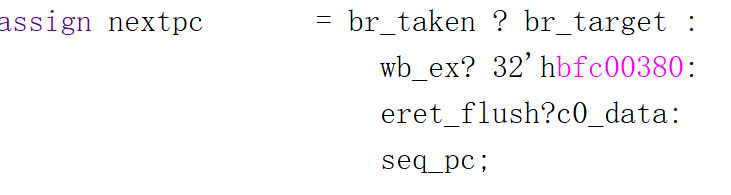
（1）错误现象



黄线部位可见，pc在第一次赋值后变x

（2）分析定位过程

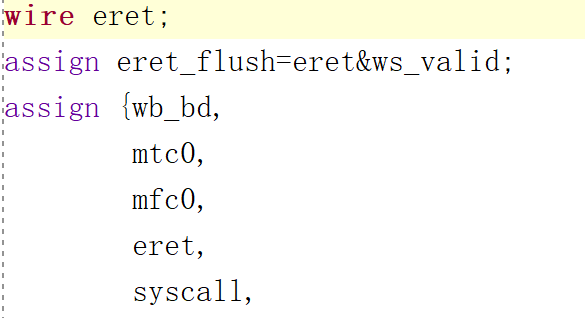
有赋值却无跳转，是nextpc的问题。



（3）错误原因

在这里我用eret译码信号赋值的eret\_flush，那么它在初始时刻为x，导致nextpc为x。

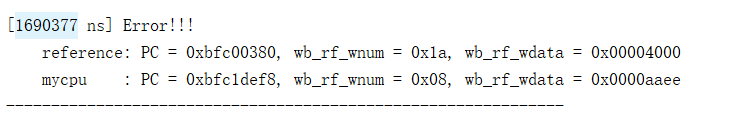
（4）修正效果



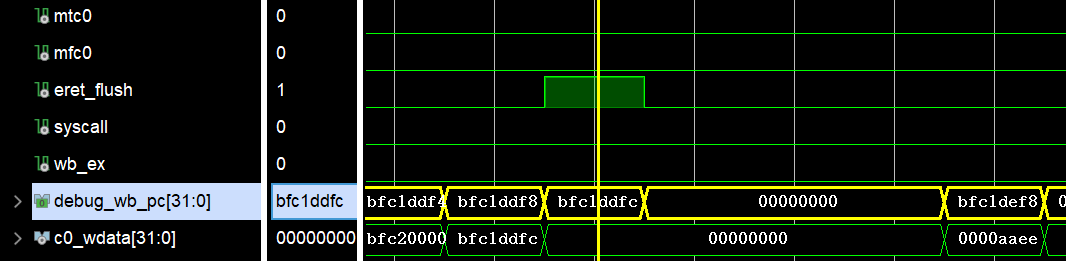
让eret信号与上ws\_valid来实现初始化

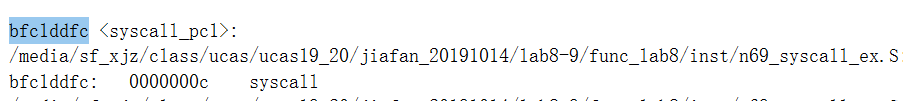
3、错误3：syscall未发生跳转

（1）错误现象

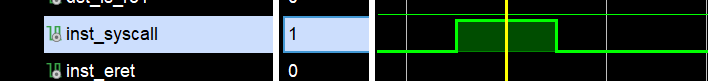


（2）分析定位过程



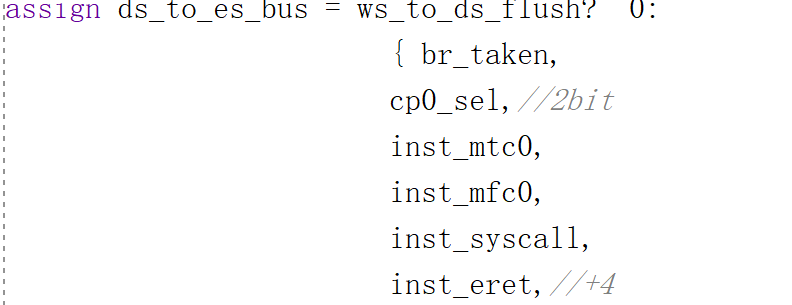


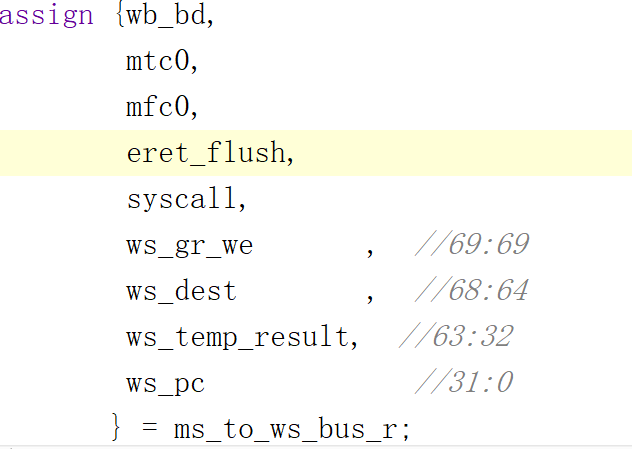
然而指令在运输途中发生了变化





（3）错误原因





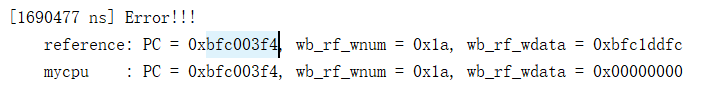
传递过程中eret和syscall 的顺序互换，导致逻辑错误

（4）修正效果

将两者放入正确位置即可

4、错误4： mfc0写入值为0

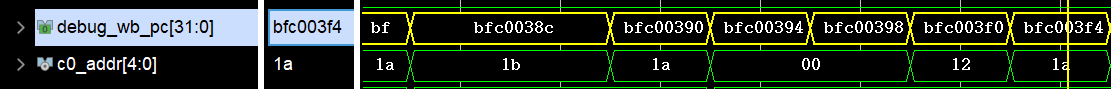
（1）错误现象



（2）分析定位过程



这条指令是mfc0指令，但



十分迷惑，地址怎么会是26?

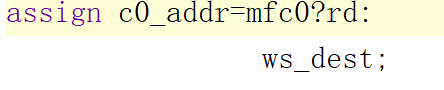
按理说应该是0xe,即epc的地址

（3）错误原因

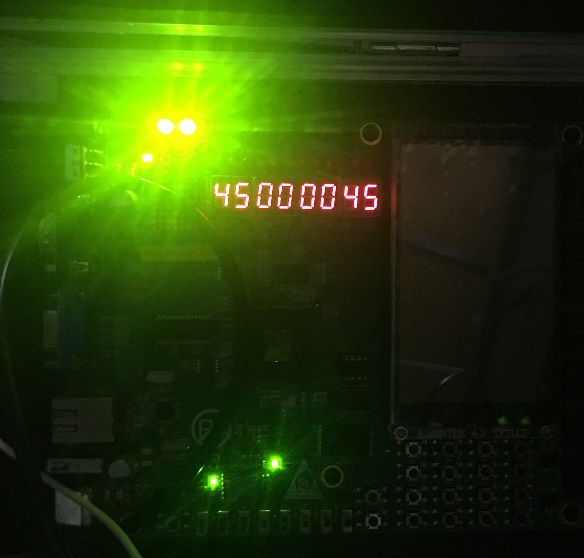


直接赋值成了dest，但是注意mfc0的dest是rt而非rd。

（4）修正效果



四、实验总结（可选）



图一 上板成功拍照留念

爱是一道光，如此美妙~~~~