**实验11报告**

蔡润泽、付轶凡

箱子号：45

一、实验任务（10%）

在 lab09 的实验环境的基础上，完成：

a) CPU 顶层修改为 AXI 接口。CPU 对外只有一个 AXI 接口，需内部完成取指和数据访问的仲裁。

b) 集成到 SoC\_AXI\_Lite 系统中。

c) 完成固定延迟和随机延迟的功能测试。  
二、实验设计（40%）

（一）总体设计思路

硬件设计图如下：

A screenshot of a video game

Description automatically generated

图1.1 硬件结构设计图

A screenshot of a cell phone

Description automatically generated

图1.2 流水示意图

如图1.2的流水示意图，在代码设计中，主要有9个部分，包括五级流水、ALU、寄存器堆，HI/LO协处理器（用来储存乘法指令的高32位、低32位，以及除法指令的余数、商），以及cp0寄存器。该设计使用了两个IP， mydiv和mydivu（分别计算有符号、无符号除法）。并且采用Tools模块进行译码。此外，WB模块与cp0寄存器相连形成了数据通路（cp0寄存器代码置于wb\_stage模块中）。CPU通过AXI BUS与指令ram和数据ram相连。

（二）重要模块1设计：算数逻辑单元（ALU）模块

1、工作原理

将CPU中的运算处理进行模块化，方便外界调用。同时模块化的ALU设计便于在其中增加新的运算功能，提高代码的扩展性。在本次实验中，加入了alu\_overflow的检测变量，并将结果输出，其判断代码如下：

assign alu\_overflow = (adder\_result[32]!=adder\_result[31]);【采用33位加法】

2、接口定义

input [15:0] alu\_op, //输入运算符

input [31:0] alu\_src1, //输入数据1

input [31:0] alu\_src2, //输入数据2

output [31:0] alu\_result, //输出结果

output [31:0] alu\_hi\_result , //输出存入hi寄存器的结果

output [31:0] alu\_lo\_result //输出存入lo寄存器的结果

output alu\_overflow

1. 功能描述

采用16位的独热码对ALU进行控制，根据独热码进行16项（相较之前的实验增添了有符号、无符号乘除法）不同的算数逻辑运算操作，并将结果传回给exe阶段。并输出overflow标志位。

（三）重要模块2设计：寄存器堆（Reg\_File）模块

1. 工作原理

将32个32位宽的寄存器堆模块化，以实现两读一写，同步读异步写的操作。

1. 接口定义

input clk,

// READ PORT 1

input [ 4:0] raddr1,

output [31:0] rdata1,

// READ PORT 2

input [ 4:0] raddr2,

output [31:0] rdata2,

// WRITE PORT

input we, //write enable, HIGH valid

input [ 4:0] waddr,

input [31:0] wdata

1. 功能描述

当写使能信号为1时，在写回阶段对寄存器堆进行写入。同时，对于两个读端口信号，进行异步读取，将输出结果传递给ID阶段。

（四）重要模块3设计：取指阶段（IF\_stage）模块

1. 工作原理

将取指操作模块化，IF通过AXI总线从axi\_ram中读出指令，并且在该周期模块之内处理PC的值，并且将指令传递给bus传递给ID模块，ID模块在下一周期再接收 。在IF阶段判断是否出现PC地址错误例外，若有，则将fs\_exc\_type例外类型标识进行修改。

1. 接口定义

表2.1 IF\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ds\_allowin | IN | 1 | ID模块允许接受IF传值 |
| br\_bus | IN | 33 | 输入是否跳转和branch的target |
| ws\_to\_fs\_bus | IN | 34 | 将WB阶段的ws\_ex,eret\_flush\_cp0\_epc传递给IF阶段 |
| fs\_to\_ds\_valid | OUT | 1 | IF模块可以向ID模块传值 |
| fs\_to\_ds\_bus | OUT | 68 | IF模块向ID模块传递数据（指令码和地址） |
| inst\_sram\_en | OUT | 1 | Inst\_sram读使能 |
| inst\_sram\_wen | OUT | 4 | Inst\_sram写使能，此处恒为0 |
| inst\_sram\_size | OUT | 2 | Inst\_sram写的size，此处恒为2 |
| inst\_sram\_addr | OUT | 32 | Inst\_sram目标地址 |
| inst\_sram\_wdata | OUT | 32 | Inst\_sram写数据 |
| inst\_sram\_rdata | IN | 32 | Inst\_sram读数据 |

1. 功能描述

PC在收到reset信号时设为偏移量32'hbfbffffc，并且在该周期模块之内处理PC的值，PC值的变化根据br\_bus取出来决定是否跳转还是加4。IF模块当inst\_sram\_en请求信号为1时，将处理后的next\_pc作为地址传递给inst\_sram, 在inst\_data\_ok置1时表示取到对应指令。IF模块将取出的指令和地址传递给ID模块，ID模块在下一周期再接收 。在IF阶段判断是否出现PC地址错误例外。fs\_pc的取指也根据WB阶段传回的信号，增加了两种选择的可能，即跳转到中断入口地址和处理eret。

（五）重要模块4设计：译码阶段（ID\_stage）模块

1. 工作原理

将从IF模块获取的指令进行译码，获得指令格式类型、ALU操作类型、是否需要加载、写回内存和参与运算数据的值、是否需要加载协处理器cp0的hi和lo寄存器、是否需要写回协处理器cp0的hi和lo寄存器、跳转的目标PC。判断PC是否需要跳转，并将结果返还给IF模块。将译码后的数据和控制信号传递给EXE模块,EXE模块在下一时钟周期接受。另外，写回阶段的数据也通过该模块传递给寄存器堆。

CPU数据通路增加旁路设计，来让前面的指令直接把已经生成出来的结果直接转给后面的指令。在本设计中，采用了“流水级组合逻辑的结果传递到译码级寄存器读出处”的方案。并通过后续阶段的valid信号和gr\_we信号来控制ID模块中，rs\_value和rt\_value的值。

另外对于ready\_go信号，当译码级的指令和处在执行级的LW指令相关时，需要设置成“0”。

在ID译码级需要标记三种类型的例外：无效指令、系统调用以及BREAK，并将其传给后续阶段。

1. 接口定义

表2.2 ID\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| es\_allowin | IN | 1 | EXE模块允许接受ID传值 |
| ds\_allowin | OUT | 1 | 允许IF模块向ID模块传递数据 |
| fs\_to\_ds\_valid | IN | 1 | IF模块可以向ID模块传值 |
| fs\_to\_ds\_bus | IN | 68 | IF模块向ID模块传递数据（指令码及地址） |
| ds\_to\_es\_valid | OUT | 1 | 允许ID模块向ES模块传递数据 |
| ds\_to\_es\_bus | OUT | 168 | ID模块向EXE模块传递数据 |
| br\_bus | OUT | 33 | 输出是否跳转和branch的target给IF模块 |
| ws\_to\_rf\_bus | IN | 43 | WB模块向ID模块传递的需要写回REG FILE的信息 |
| es\_to\_ms\_bus | IN | 154 | EXE模块向MEM模块传递数据 |
| ms\_to\_ws\_bus | IN | 118 | MEM模块向WB模块传递数据 |
| es\_to\_ms\_valid | IN | 1 | EXE模块可以向MEM模块传值 |
| ms\_to\_ws\_valid | IN | 1 | MEM模块可以向WB模块传值 |
| out\_es\_valid | IN | 1 | 接收es\_valid是否为1 |
| out\_ms\_valid | IN | 1 | 接收ms\_valid是否为1 |

1. 功能描述

将从IF模块获取的指令进行译码，获得指令格式类型、ALU操作类型、是否需要加载、写回内存和参与运算数据的值、是否需要加载协处理器cp0的hi和lo寄存器、是否需要写回协处理器cp0的hi和lo寄存器、跳转的目标PC。判断PC是否需要跳转，并将结果返还给IF模块。将译码后的数据和控制信号在传递给EXE模块,EXE模块在下一时钟周期接受。写回阶段的数据也通过该模块传递给寄存器堆。并且采用前递的方式来减少CPU的阻塞，缩短运行时间。在ID译码级需要标记三种类型的例外：无效指令、系统调用以及BREAK，并将其传给后续阶段。

（六）重要模块5设计：执行阶段（EXE\_stage）模块

1. 工作原理

将从ID模块获取的指令相应的执行。将执行后的ALU结果和前阶段传递的通用寄存器写使能、写地址控制信号、PC传通过总线传递给MEM模块,MEM模块在下一周期接收新值。EXE级如果是ram读写指令则发出data\_sram\_en请求，当data\_addr\_ok置1时，表示读写请求的地址已接收到，对于读ram指令，它的data\_data\_ok返回在MEM级，表示已从data\_sram读出对应的数。

此阶段在进行算数逻辑运算时，需要调用ALU模块。

除了对ALU模块需要新增有符号、无符号乘法外，还需要在此阶段调用我们生成的mydiv和mydivu两个除法器IP完成新增的有符号、无符号除法运算，以及对除法器中的控制信号进行相应的设置。

EXE阶段需要标记OVERFLOW例外和内存写地址错误例外，并将例外标记传到后续阶段。

1. 接口定义

表2.3 EXE\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ms\_allowin | IN | 1 | MEM模块允许接受EXE传值 |
| es\_allowin | OUT | 1 | EXE模块允许接受ID传值 |
| ds\_to\_es\_valid | IN | 1 | ID模块可以向EXE模块传值 |
| ds\_to\_es\_bus | IN | 168 | ID模块向EXE模块传递数据 |
| es\_to\_ms\_valid | OUT | 1 | EXE模块可以向MEM模块传值 |
| es\_to\_ms\_bus | OUT | 155 | EXE模块向MEM模块传递数据 |
| data\_sram\_en | OUT | 1 | data\_sram读使能 |
| data\_sram\_wen | OUT | 4 | data\_sram写使能 |
| data\_sram\_addr | OUT | 32 | data\_sram目标地址 |
| data\_sram\_wdata | OUT | 32 | data\_sram写数据 |
| out\_es\_valid | OUT | 1 | 将es\_valid的值传递给ID模块 |
| ws\_ex | IN | 1 | WB阶段有例外标记 |
| ms\_exc\_type | IN | 4 | MEM阶段例外类型 |
| ws\_exc\_type | IN | 4 | WB阶段里外类型 |
| ms\_eret | IN | 1 | MEM阶段有eret指令 |
| eret\_flush | IN | 1 | eret刷新流水线信号 |

mydiv和mydivu除法器IP数据通道定义如下：

s\_axis\_divisor\_tdata[31:0]; //除数

s\_axis\_divisor\_tready; //除数应答信号

s\_axis\_divisor\_tvalid; //除数请求信号

s\_axis\_dividend\_tdata[31:0]; //被除数

s\_axis\_dividend\_tready; //被除数应答信号

s\_axis\_dividend\_tvalid; //被除数请求信号

m\_axis\_dout\_tdata[63:0]; //商和余数，其中高32位为余数、低32位为商

m\_axis\_dout\_tvalid; //结果有效信号，表示除法已经算完，可以取得结果

aclk; //时钟信号

在除法指令处于执行流水级且没有对除法器成功输入数据的时候，同时将 s\_axis\_dividend\_tvalid 和s\_axis\_divisor\_tvalid 置为 1，向除法器IP发送调用除法器的请求。当发现s\_axis\_dividend\_tready 和 s\_axis\_divisor\_tready 反馈为 1 后，tvalid和tready成功握手，并需要在下一拍将s\_axis\_dividend\_tvalid和s\_axis\_di vior\_tvalid清为0，以此保证一个除法操作只调用一次除法器IP，避免除法器给CPU送多个结果从而导致出错。成功握手后，数据传入除法器的各个数据通路，8拍后，除法器产生结果并将m\_axis\_dout\_tvalid信号置1，表示可以取得除法结果。

为了避免除法指令与之后的指令产生“写后读”相关，考虑到除法器IP需要8拍才能拿到结果，所以如果执行流水级这一拍执行的是除法指令，需要将执行级阻塞住（即修改es\_ready\_go信号），拿到除法结果后再释放，因为通常程序中的除法指令较少，这样的阻塞设计不会太影响整体的性能。

1. 功能描述

将从ID模块获取的指令相应的执行。将执行后和前阶段传递的数据控制信号通过数据总线在下一时钟周期更新传给MEM模块。另外，load指令的发出读信号处理也在EXE阶段完成，EXE模块将数据传递给MEM模块，在下一周期进行写回。输出数据RAM的写信号和数据。EXE阶段需要标记OVERFLOW例外和内存写地址错误例外，并将例外标记传到后续阶段。

（七）重要模块6设计：访存阶段（MEM\_stage）模块

1. 工作原理

将从EXE模块获取的访存指令相应的执行。根据es\_to\_ms\_bs中的是否数据来自数据RAM信号确定是否有访存取出的数据，并将相应指令的最终结果和前阶段传递的通用寄存器写使能、写地址控制信号、PC传通过总线在下一时钟周期更新给WB模块。MEM阶段需要标记是否有LOAD地址错例外，并将结果传递给WS阶段。

1. 接口定义

表2.4 MEM\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ws\_allowin | IN | 1 | WB模块允许接受MEM传值 |
| ms\_allowin | OUT | 1 | MEM模块允许接受EXE传值 |
| es\_to\_ms\_valid | IN | 1 | EXE模块可以向MEM模块传值 |
| es\_to\_ms\_bus | IN | 155 | EXE模块向MEM模块传递数据 |
| ms\_to\_ws\_valid | OUT | 1 | MEM模块可以向EXE模块传值 |
| ms\_to\_ws\_bus | OUT | 118 | MEM模块向WB模块传递数据 |
| data\_sram\_rdata | OUT | 32 | data\_sram读出的数据 |
| out\_ms\_valid | OUT | 1 | 将ms\_valid的值传递给ID模块 |
| ws\_ex | IN | 1 | WB阶段有例外标记 |
| ms\_exc\_type | OUT | 4 | MEM阶段例外类型 |
| ms\_eret | OUT | 1 | MEM阶段有eret指令 |
| eret\_flush | IN | 1 | eret刷新流水线信号 |

1. 功能描述

将从EXE模块获取的访存指令相应的执行。确定是否有访存指令，并将相应指令的数据和前阶段传递的数据控制信号传通过总线在下一时钟周期更新给WB模块。

MEM阶段需要标记是否有LOAD地址错例外，并将结果传递给WS阶段。

（八）重要模块7设计：访存阶段（WB\_stage）模块

1. 工作原理

将从MEM模块获取的写回指令相应的执行。确定是否有写回指令，并进行相应的操作。同时，该模块将PC、寄存器堆写使能、地址、和写回结果传递给debug模块，用于调试CPU的正确性。在本次设计中，cp0寄存器位于WB文件下，即在WB级后处理例外。遇到例外时，因向之前的流水级发出eret\_flush信号来刷新流水线。相应流水级据此控制相关PC和写使能。

表2.5 WB\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ws\_allowin | OUT | 1 | WB模块允许接受MEM传值 |
| ms\_to\_ws\_valid | IN | 1 | MEM模块可以向WB模块传值 |
| ms\_to\_ws\_bus | IN | 84 | MEM模块向WB模块传递数据 |
| ws\_to\_rf\_bus | OUT | 73 | WB模块向寄存器堆模块（通过ID模块）传递数据（包括ws\_gr\_we、ws\_valid等） |
| debug\_wb\_pc | OUT | 32 | debug显示PC |
| Debug\_wb\_rf\_wen | OUT | 4 | debug显示寄存器堆写使能 |
| Debug\_wb\_rf\_wnum | OUT | 5 | debug显示寄存器堆写地址 |
| Debug\_wb\_rf\_wdata | OUT | 32 | debug显示寄存器堆写数据 |
| ws\_ex | OUT | 1 | WB阶段有例外标记 |
| ws\_exc\_type | OUT | 4 | WS阶段例外类型 |
| eret\_flush | OUT | 1 | eret刷新流水线信号 |

1. 功能描述

将从MEM模块获取的写回指令相应的执行。确定是否有写回指令，并进行相应的操作。同时，该模块将PC、寄存器堆写使能、地址、和写回结果传递给debug模块，用于调试CPU的正确性。在WB流水级之后处理例外，并更新相应cp0寄存器的值。遇到例外时，因向之前的流水级发出eret\_flush信号来刷新流水线。相应流水级据此控制相关PC和写使能。

（九）重要模块8设计：HI/LO寄存器模块

1、工作原理

记录乘法指令生成的完整的64位乘积，以及除法指令生成的各32位的商和余数。

表2.6 hi/lo寄存器模块接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| rd\_hi | OUT | 32 | 从cp0模块中读取hi寄存器的值 |
| rd\_lo | OUT | 32 | 从cp0模块中读取lo寄存器的值 |
| hi\_we | IN | 1 | 写hi寄存器使能 |
| lo\_we | IN | 1 | 写lo寄存器使能 |
| wd\_hi | IN | 32 | 写hi寄存器数据，本实验中为乘法指令生成结果的高32位以及除法指令生成结果的余数 |
| wd\_lo | IN | 32 | 写lo寄存器数据，本实验中为乘法指令生成结果的低32位以及除法指令生成结果的商 |

2、功能描述

类似于寄存器堆regfile模块，实现对hi、lo寄存器的同步写和异步读。

（十）重要模块9设计：指令类SRAM(inst sram-like)

1. 工作原理

inst\_addr\_ok 信号用来和 req 信号一起完成读写请求的握手。只有在 clk 的上升沿同时看到 inst\_req 和 inst\_addr\_ok 都为 1 的时候才是一次成功的请求握手。inst\_data\_ok对应读事务的时候，它是数据返回的有效信号;对应写事务的时候，它写响应有效信号。当握手成功时，传递数据。

1. 接口定义

input inst\_req,

input inst\_wr,

input [ 1:0] inst\_size,

input [31:0] inst\_addr,

input [31:0] inst\_wdata,

output [31:0] inst\_rdata,

output inst\_addr\_ok,

output inst\_data\_ok

1. 功能描述

接收指令的读写请求，并在握手成功时传递数据。

（十一）重要模块10设计：数据类SRAM(data sram-like)

1. 工作原理

data\_addr\_ok 信号用来和 req 信号一起完成读写请求的握手。只有在 clk 的上升沿同时看到 data\_req 和 data\_addr\_ok 都为 1 的时候才是一次成功的请求握手。 data\_data\_ok对应读事务的时候，它是数据返回的有效信号;对应写事务的时候，它写响应有效信号。当握手成功时，传递数据。

1. 接口定义

input data\_req,

input data\_wr,

input [ 1:0] data\_size,

input [31:0] data\_addr,

input [31:0] data\_wdata,

output [31:0] data\_rdata,

output data\_addr\_ok,

output data\_data\_ok

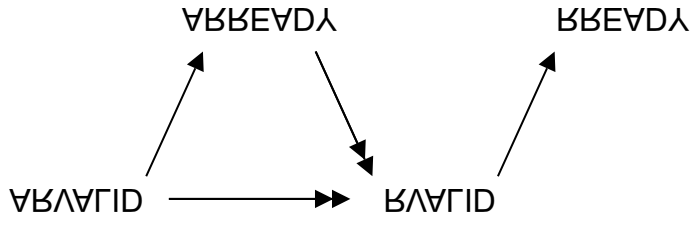
1. 功能描述

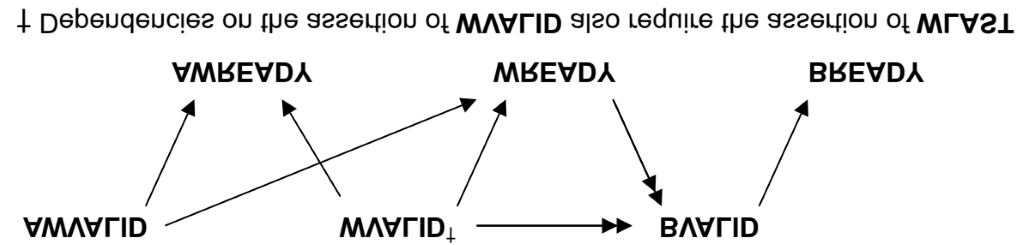
接收数据的读写请求，并在握手成功时传递数据。

（十二）重要模块11设计：AXI模块

1. 工作原理

总线的两端可以分为主方master和从方slave，对于读操作来说，主方提出请求，从方接收请求返回数据；对于写操作来说，主方提出请求并发出写数据，从方接收请求和数据，AXI 总线协议采用握手机制完成主方与从方的交互，AXI总线有五个通道，写地址请求、写数据请求、写响应、读地址请求、读数据请求，每一个通道都有一对valid和ready信号用来实现握手。这些通道的valid和ready依赖关系如下图：





1. 接口定义
2. //ar
3. output  [ 3:0]  arid,
4. output  [31:0]  araddr,
5. output  [ 7:0]  arlen,
6. output  [ 2:0]  arsize,
7. output  [ 1:0]  arburst,
8. output  [ 1:0]  arlock,
9. output  [ 3:0]  arcache,
10. output  [ 2:0]  arprot,
11. output          arvalid,
12. input           arready,
13. //r
14. input   [ 3:0]  rid,
15. input   [31:0]  rdata,
16. input   [ 1:0]  rresp,
17. input           rlast,
18. input           rvalid,
19. output          rready,
20. //aw
21. output  [ 3:0]  awid,
22. output  [31:0]  awaddr,
23. output  [ 7:0]  awlen,
24. output  [ 2:0]  awsize,
25. output  [ 1:0]  awburst,
26. output  [ 1:0]  awlock,
27. output  [ 3:0]  awcache,
28. output  [ 2:0]  awprot,
29. output          awvalid,
30. input           awready,
31. //w
32. output  [ 3:0]  wid,
33. output  [31:0]  wdata,
34. output  reg[ 3:0]   wstrb,
35. output          wlast,
36. output          wvalid,
37. input           wready,
38. //b
39. input   [ 3:0]  bid,
40. input   [ 1:0]  bresp,
41. input           bvalid,
42. output          bready
43. 功能描述

接收CPU的读、写指令和数据请求，并将请求传递给类SRAM，当相应的指令和数据准备好时。完成对应的读事务和写事务操作。

三、实验过程（50%）

（一）实验流水账

11月21日 20：00-20：30 阅读讲义

11月21日 20：30-22：00 设计代码

11月23日、24日 10：00-22：30 调试bug

11月25日 18：00-3：30 调试bug

11月26日9：00-10：30 撰写实验报告

（二）错误记录

1、错误1：cpu端data\_sram\_addr赋值错误。

（1）错误现象

运行仿真，在PC为0xbfc53300时，wb\_rf\_wdata值有误：

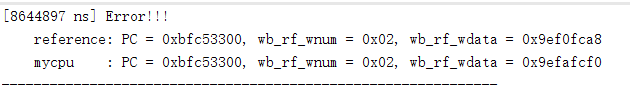


图3.1 read data 1和read data 2 测试错误

（2）分析定位过程

在test.S文件中定位相关指令，发现应该是SB指令没有写到正确的位置，波形图如下：

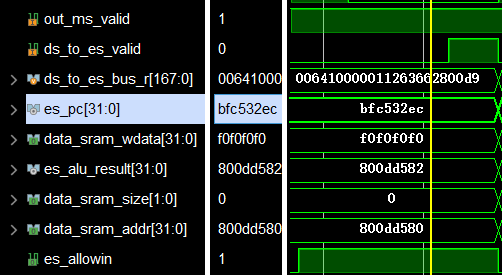


图3.2 相关波形

（3）错误原因

造成这样错误的原因是data\_sram\_addr与alu算得的地址不相同，我们在写data\_sram\_addr的时候把它地址的低两位置为了0。

（4）修正效果

直接令data\_sram\_addr等于es\_alu\_result，重新仿真，这个错误被解决。



图3.3 修改后的data\_sram\_addr

2、错误2：data\_sram\_size赋值错误。

（1）错误现象

运行仿真，在PC为0xbfc06950时，wb\_rf\_wdata值有误：

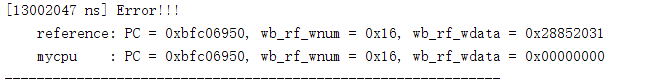


图3.4 read data 1和read data 2 测试错误

（2）分析定位过程

在test.S文件中定位相关指令，发现应该是div指令写hi/lo寄存器时没有正确写，相关波形图如下：

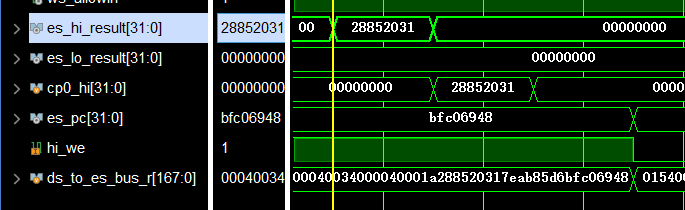


图3.5 相关波形

（3）错误原因

造成这样错误的原因是hi\_we整个除法过程中拉高，导致它一直可以写hi/lo寄存器，应该让他在es\_to\_ms\_valid条件下才允许写。

（4）修正效果

令hi/lo寄存器的写使能与上es\_to\_ms\_valid，重新仿真，这个错误被解决，测试通过。

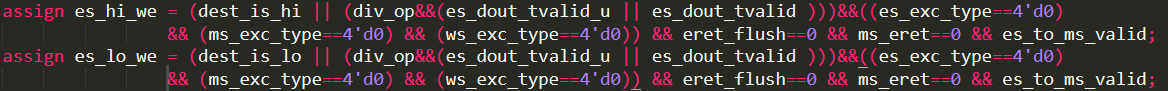


图3.6 hi/lo寄存器写使能相关代码

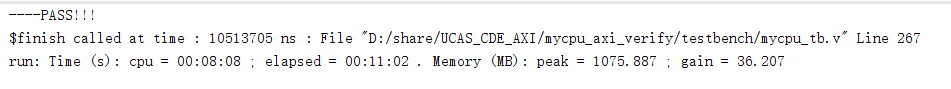


图3.7 TEST PASS

四、实验总结（可选）

本次实验调试难度突然增大，因为其涉及到CPU和总线两个部分的组合使用，因此设计的时候需要将各种valid、ready\_go以及其他的握手信号考虑清楚。在代码调整过程中需要对照波形以及设计一点点的修改错误的设计，因此需要花费较多的时间。下次实验开始写代码时，应该先仔细考虑清楚设计的原理和细节，在全部弄清楚以后在开始写代码能大大缩短调试的时间。