实验 3 报告

2017K8009908018
蔡润泽
箱子号：45

一、实验任务（10%）

本次实验任务是了解无阻塞五级流水线 CPU 的设计思路，并调试给出的 CPU 设计代码，本次实验分为了两项子任务，其中：

子任务一:阅读代码并结合讲义，画出简单流水线 CPU 的结构设计框图和软件部分流程图。

子任务二: 结合仿真和上板测试，对给出的代码进行调试，找出其中的 7 处 bug。（实际找出 8 处 bug）

二、实验设计（40%）

（一）总体设计思路

硬件结构设计图如下：

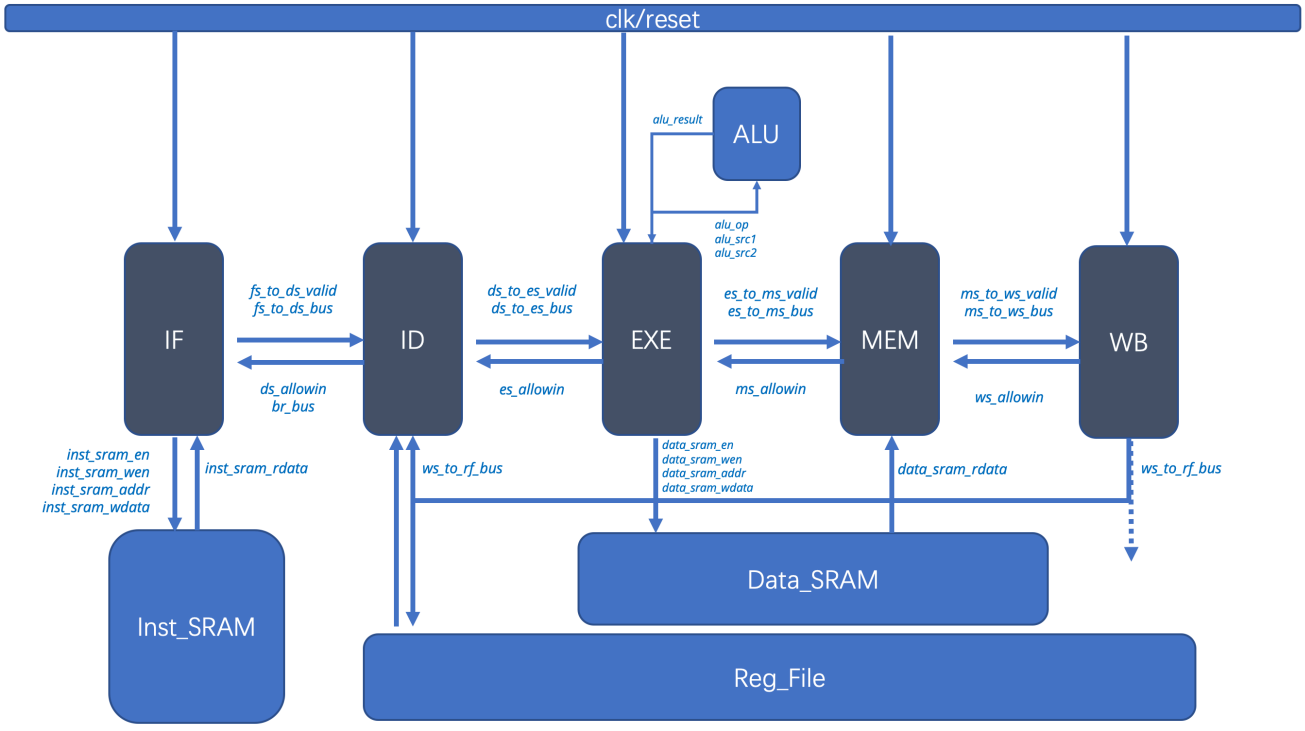

图 1.1 硬件结构设计图

代码设计中，主要有 7 个模块，包括五级流水、ALU 以及寄存器堆。该设计使用了两个 IP，Inst_RAM 和 Data_RAM。并且采用 Tools 模块进行译码。

（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。

2、接口定义

```
input  [11:0] 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
```

3、功能描述

采用 12 位的独热码对 ALU 进行控制，根据独热码进行 12 项不同的算数逻辑运算操作，并将结果传回给 exe 阶段。

（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。

2、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
```

3、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 从 inst_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，在下一周期再传递给 ID 模块 。

2、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
fs_to_ds_valid	OUT	1	IF 模块可以向 ID 模块传值
fs_to_ds_bus	OUT	64	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_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

3、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 读使能信号为 1 时，将处理后的 next_pc 作为地址传递给 inst_sram, 并从 inst_ram 中读出指令。IF 模块将取出的指令和地址传递给 bus，在下一周期再传递给 ID 模块。

（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，并处理 PC 是否需要跳转，将结果返还给 IF 模块。将译码后的数据和控制信号传通过数据总线在下一时钟周期递给 EXE 模块。另外，写回阶段的数据也通过该模块传递给寄存器堆。此模块需要调用寄存器堆模块和 decode 模块。

2、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	64	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	136	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	38	WB 模块向 ID 模块传递的需要写回 REG FILE 的信息

3、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、跳转的目标 PC。并处理 PC 是否需要跳转，将结果返还给 IF 模块。将译码后的数据和控制信号在下一时钟周期传递给 EXE 模块。另外，写回阶段的数据也通过该模块传递给寄存器堆。

（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。将执行后和前阶段传递的数据控制信号传通过数据总线在下一时钟周期递给 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。此阶段在进行算数逻辑运算时，需要调用 ALU 模块。

2、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	136	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	71	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

3、功能描述

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 传通过总线在下一时间周期更新给 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。此阶段在进行算数逻辑运算时，需要调用 ALU 模块。

（六）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。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递给 WB 模块。

2、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	71	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	70	MEM 模块向 WB 模块传递数据
data_sram_rdata	OUT	32	data_sram 读出的数据

3、功能描述

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递给 WB 模块。

（七）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。

2、接口定义

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	70	MEM 模块向 WB 模块传递数据
ws_to_rf_bus	OUT	38	WB 模块向寄存器堆模块传递数据
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

3、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。

三、实验过程（50%）

（一）实验流水账

9 月 14 日 20：00-22：00 阅读讲义

9 月 15 日 10：00-22：00 进行试验撰写实验报告

9 月 16 日 19：00-23：00 进行试验撰写实验报告

以下错误记录 也就是记录 子任务二 的完成过程。

（二）错误记录

1、错误 1：PC 跳转错误

（1）错误现象

运行仿真，比对 trace 发现 PC 的值发生跳转错误，如图 3.1。

（2）分析定位过程

如图 3.1，PC 出现错误，这说明跳转指令可能出现问题，对应查找 PC 的跳转相关指令，发现 br_bus 只有 32 位。通过找 br_bus 的定义和赋值语句发现，其定义语句中使用了宏定义，通过查看其宏定义，发现出现了“`define BR_BUS_WD 32”，这表明宏定义中的位宽出现了错误。

```
[   2107 ns] Error!!!
    reference: PC = 0xbfc0038c, wb_rf_wnum = 0x04, wb_rf_wdata = 0xbfb00000
    mycpu    : PC = 0xbfc00010, wb_rf_wnum = 0x08, wb_rf_wdata = 0x80000000
--------------------------------------------------------------
```

图 3.1.1 IF 模块里的 BR_BUS 信号出现 Z

（3）错误原因

br_bus 的宏定义语句出现了“`define BR_BUS_WD 32”，这表明宏定义中的位宽出现了错误。在实际使用过程中，br_bus 的最真实位宽应该是 33，这导致到最终 PC 跳转错误。

（4）修正效果

如图 3.2，通过将宏定义语句里的 32 改成 33，PC 跳转正常，此处 error 消失。

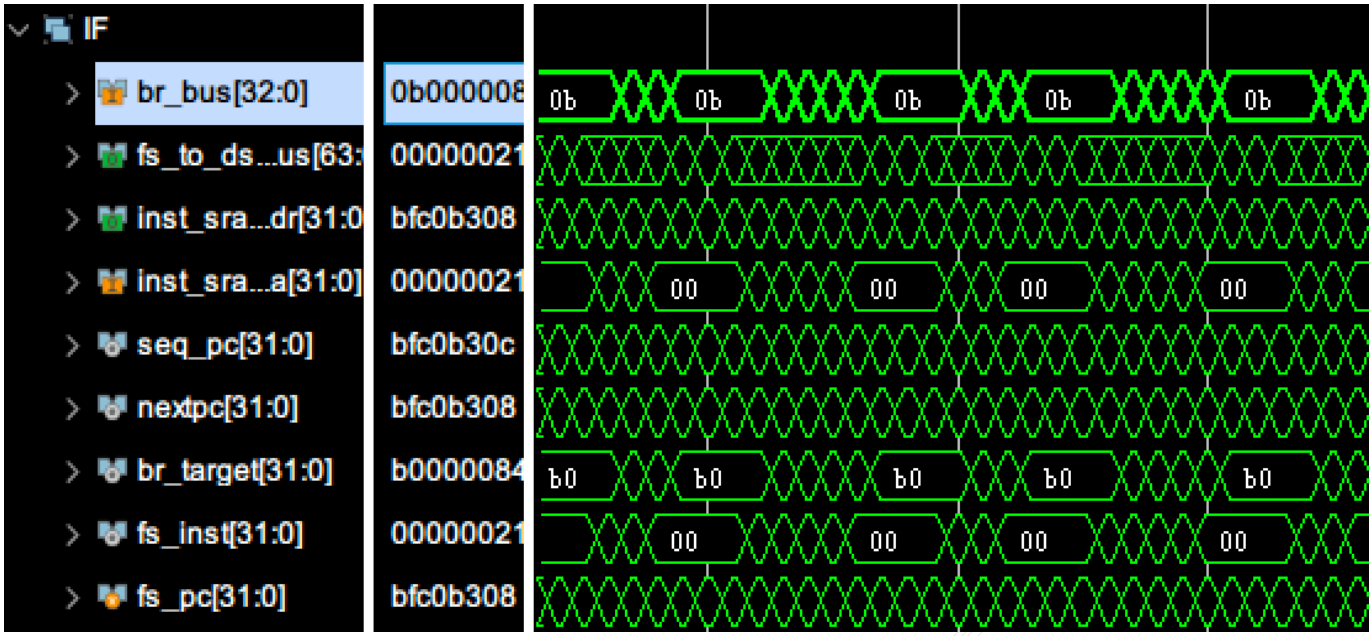

图 3.1.2 BR_BUS_WD 位宽改为 33

（5）归纳总结（可选）

采用宏定义来定义位宽有便于修改的好处。但在今后的代码书写过程中，要养成写开发文档的习惯，最后根据 br_bus 开发文档中的位宽，来对代码中的宏定义进行二次确认。

2、错误 2：信号“X”

（1）错误现象

ID 模块中，ds_valid 信号为 X。

（2）分析定位过程

如图 3.2，ds_valid 信号为 X，通过找 ds_valid 的定义和赋值语句发现，源代码并未给 ds_valid 变量进行赋值。

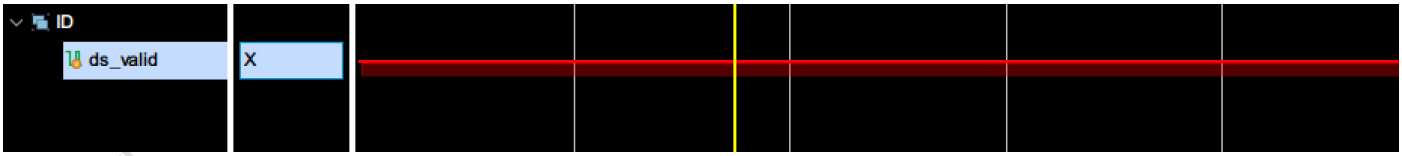

图 3.2 ID 模块里的 ds_valid 信号出现 X

（3）错误原因

ds_valid 信号为 X，其原因是源代码并未给 ds_valid 变量进行赋值。

（4）修正效果

在 always 语句中，增加对 ds_valid 的赋值语句：

```
if(reset)begin
```

```
ds_valid<=0;
end
else if (ds_allowin) begin
    ds_valid <= fs_to_ds_valid;
end
```

修改完毕后，X 信号消失，如图 3.3。

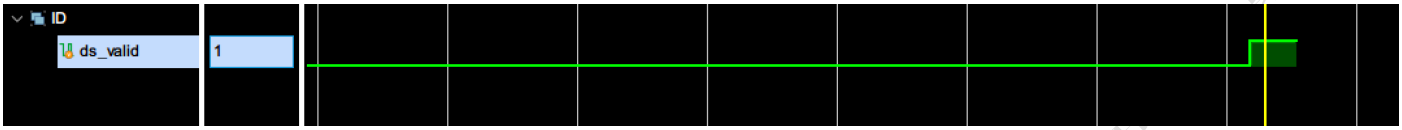

图 3.3 ID 模块里的 ds_valid 信号中 X 消失

（5）归纳总结（可选）

对于模式相近的代码块，可以采取复制粘贴框架的模式，以免一些重要组成部分忘记赋值。

3、错误 3：信号“Z”

（1）错误现象

ID 模块中，ds_to_es_bus 最高位信号为 Z。

（2）分析定位过程

如图 3.4，ds_to_es_bus 的 123 位(第 124 位)出现 Z 信号，通过找 ds_to_es_bus 的定义和赋值语句发现，该 bus 组成部分中的 load_op 并没有被赋值。

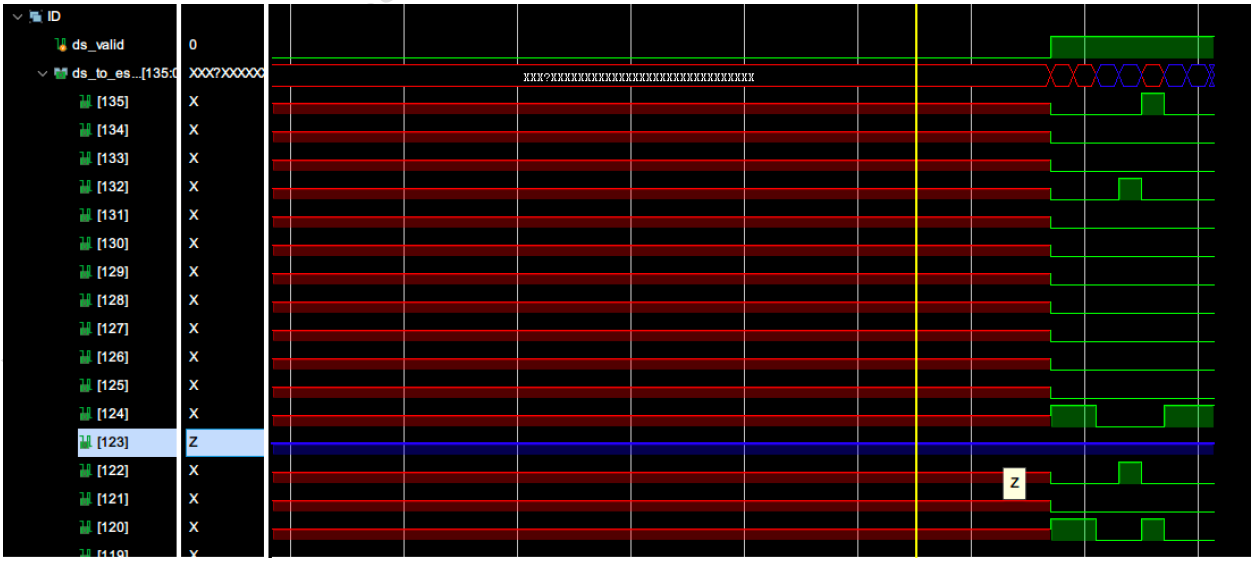

图 3.4 ID 模块里的 ds_to_es_bus 信号出现 Z

（3）错误原因

ds_to_es_bus 组成部分中，load_op 变量并没有被赋值，因此在 ds_to_es_bus 中出现了“Z”信号。

（4）修正效果

通过将 ds_to_es_bus 中的 load_op 进行赋值，代码中增加“assign load_op=inst_lw;”语句，修改的的波形如图 3.5, ID 模块里的 ds_to_es_bus 信号中 Z 消失。

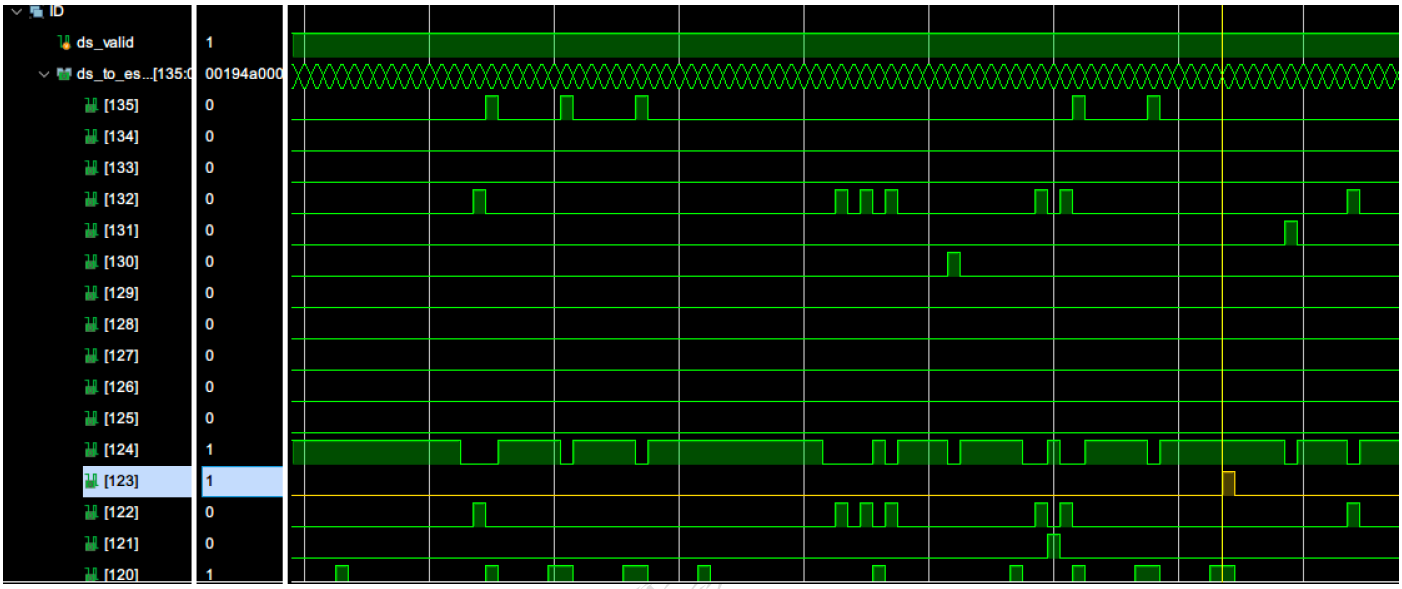

图 3.5 ID 模块里的 ds_to_es_bus 信号中 Z 消失

（5）归纳总结（可选）

变量忘记赋值可能容易发生，因此我们需要通过仿真查看波形来确定自己的代码中的变量是否都准确进行了赋值。

4、错误 4：模块对应接口连接错误

（1）错误现象

在 trace 对比中 wb_rf_wdata 出错。

（2）分析定位过程

仿真进行 get_trace 比对，发现下述 error:

```
Test begin!
----------------------------------------------------------------
[  2067 ns] Error!!!
    reference: PC = 0xbfc00000, wb_rf_wnum = 0x08, wb_rf_wdata = 0xffffffff
    mycpu    : PC = 0xbfc00000, wb_rf_wnum = 0x08, wb_rf_wdata = 0xfffffffe
----------------------------------------------------------------
```

图 3.6 get_trace 比对，发现 wb_rf_wdata 错误

这意味着写回寄存器堆的值出现错误，通过查找 ALU 模块，如图 3.7，发现 alu_src1 和 alu_src2 的值恒为一样，通过查找 ALU 的模块调用，发现 EXE 模块与 ALU 模块接口连接错误。出现了“.alu_src1 (es_alu_src2)”。

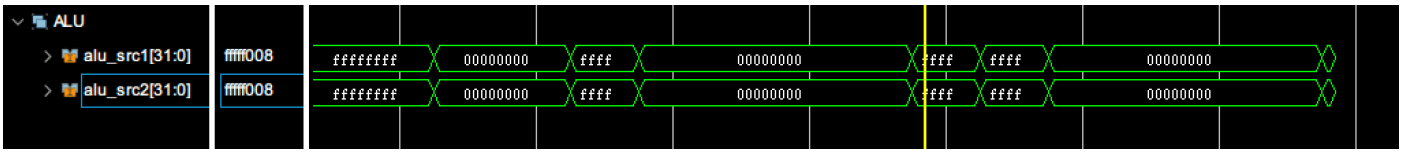

图 3.7 ALU 模块中 alu_src1 和 alu_src2 的值恒为一样

（3）错误原因

EXE 模块与 ALU 模块接口连接错误。出现了“.alu_src1 (es_alu_src2)”，导致 alu_src1 和 alu_src2 的值恒为一样，最终使得 wb_rf_wdata 的值错误。

（4）修正效果

通过将 EXE 模块和 ALU 模块接口进行更改，改为“.alu_src1 (es_alu_src1)”，修改的的波形如图 3.8, ALU 模块中 alu_src1 和 alu_src2 的值不恒为一样，且为正确传递的值，此处 error 消失。

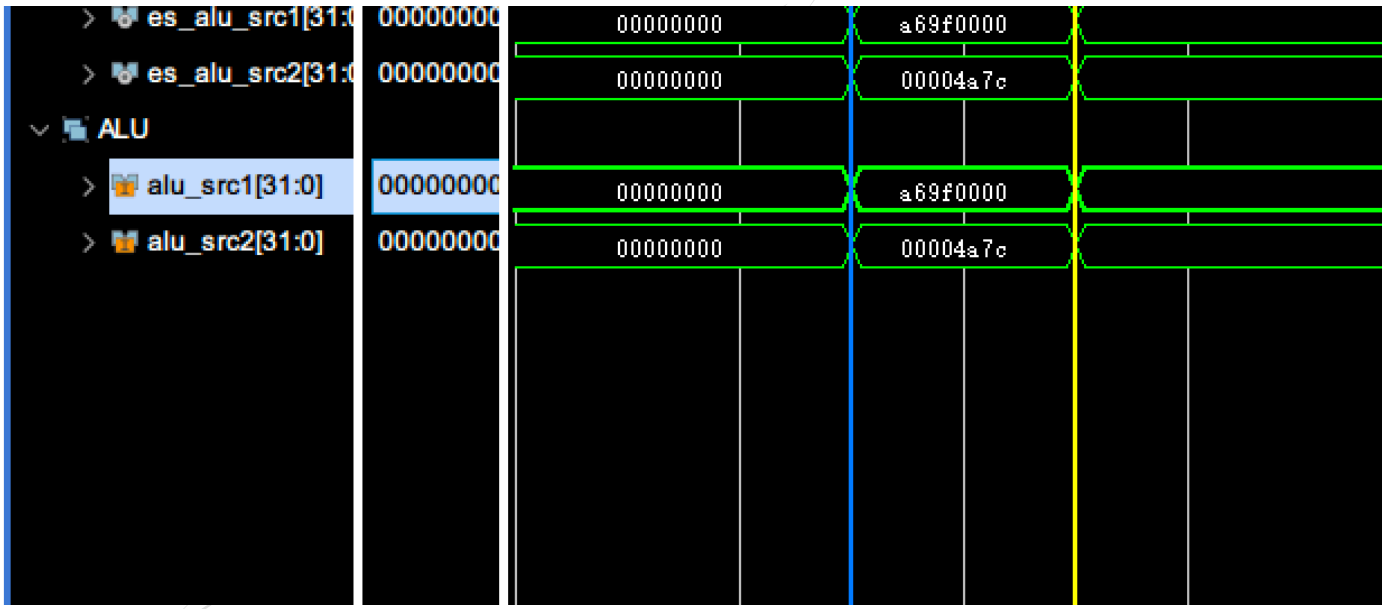

图 3.8 ID ALU 模块中 alu_src1 和 alu_src2 的值恒为一样

（5）归纳总结（可选）

模块间接口的连接也应该根据设计文档进行检查，以免出现连接错误。

5、错误 5：组合环

（1）错误现象

运行仿真，发现波形停止在某一时刻，如图 3.9。

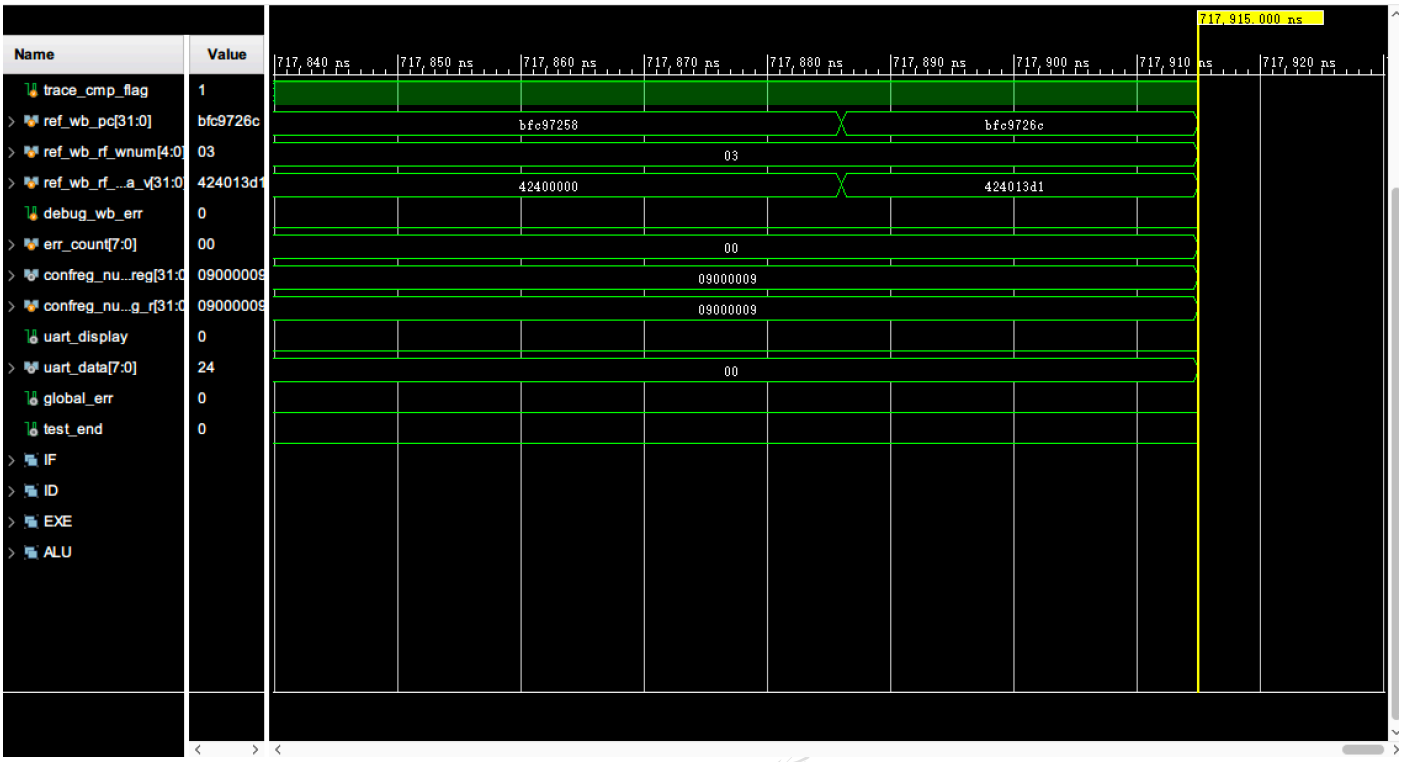

图 3.9 仿真波形停止在某一时刻

（2）分析定位过程

经排查（vivado 高亮指示）发现 ALU 模块的代码中出现了“assign or_result = alu_src1 | alu_src2 | alu_result;”而 alu_result 中可以被 or 赋值，这出现了组合环。

（3）错误原因

ALU 模块中出现了“assign or_result = alu_src1 | alu_src2 | alu_result;”而 alu_result 中可以被 or 赋值，这形成了一个组合环。

（4）修正效果

通过分析代码逻辑，将原代码更改为“assign or_result = alu_src1 | alu_src2 ;”，此 error 修复，进入到后续状态，但这时出现了新 error，如图 3.10。

```
[ 888537 ns] Error!!!
    reference: PC = 0xbfc88c90, wb_rf_wnum = 0x02, wb_rf_wdata = 0xadff20c0
    mycpu    : PC = 0xbfc88c90, wb_rf_wnum = 0x02, wb_rf_wdata = 0x2dff20c0
```

图 3.10 修改为组合环后出现新 error

6、错误 6：ALU SRL 移位错误

（1）错误现象

如图 3.10，运行仿真，发现 trace 报错，wb_rf_wdata 出现不匹配。

图 3.9 仿真波形停止在某一时刻

（2）分析定位过程

通过查找对应 PC 的指令，如图 3.11，发现这是一个 srl 指令，通过查找 ALU 中的 srl 发现，其 srl 的结果位宽只有 31 位，少了一位。

```
143460    bfc88c84:    00000021    move    zero,zero
143461    bfc88c88:    00000021    move    zero,zero
143462    bfc88c8c:    00000021    move    zero,zero
143463    bfc88c90:    00081002    srl v0,t0,0x0
143464    bfc88c94:    00000021    move    zero,zero
143465    bfc88c98:    00000021    move    zero,zero
143466    bfc88c9c:    00000021    move    zero,zero
143467    bfc88ca0:    00000021    move    zero,zero
```

图 3.11 对应 PC 的指令是 srl

（3）错误原因

ALU 模块中出现“assign sr_result = sr64_result[30:0];”，即输出 srl 指令的位宽只有 31 位，缺少了一位，导致最终 wb_rf_wdata 的值错误。

（4）修正效果

通过分析代码逻辑，将原代码更改为“assign sr_result = sr64_result[31:0];”此 error 修复，仿真运行显示全部 PASS，如图 3.12。

```
================================================================
Test end!
----PASS!!!
$finish called at time : 1728025 ns : File "C:/Users/VincentTsai/Desktop/UCAS_CDE/mycpu_verify/testbench/mycpu_tb.v" Line 261
run: Time (s): cpu = 00:00:34 ; elapsed = 00:00:32 . Memory (MB): peak = 916.363 ; gain = 37.211
```

图 3.12 ALL PASS

（5）归纳总结（可选）

最后采用宏定义来定义位宽，同时要根据开发文档中的位宽来对代码中的宏定义进行二次确认。

7、错误 7：越沿采样

（1）错误现象

仿真无错误现象，该错误对本次实验的结果显示无影响。

（2）分析定位过程

当上述问题被解决后，trace 比对会全部通过，而此时还有 1 处 bug 未被发现。通过每个文件的一一查找，发

现了此次实验的 bug 不止 7 处。第七处 bug 为 always 语句中采用了阻塞赋值。MEM 模块中的 always 语句里出现了“es_to_ms_bus_r = es_to_ms_bus;”

（3）错误原因

MEM 模块中的 always 语句里出现了“es_to_ms_bus_r = es_to_ms_bus;”。而 always 语句中应该采用非阻塞赋值。

（4）修正效果

通过分析代码逻辑，将原代码更改为“es_to_ms_bus_r <= es_to_ms_bus;”，此错误修复。

（5）归纳总结（可选）

always 里必须采用非阻塞赋值，这样的错误不应当出现。

8、错误 8：信号“Z”

（1）错误现象

Tools 模块里的 decode 出现错误，信号中出现“Z”，如图 3.13。

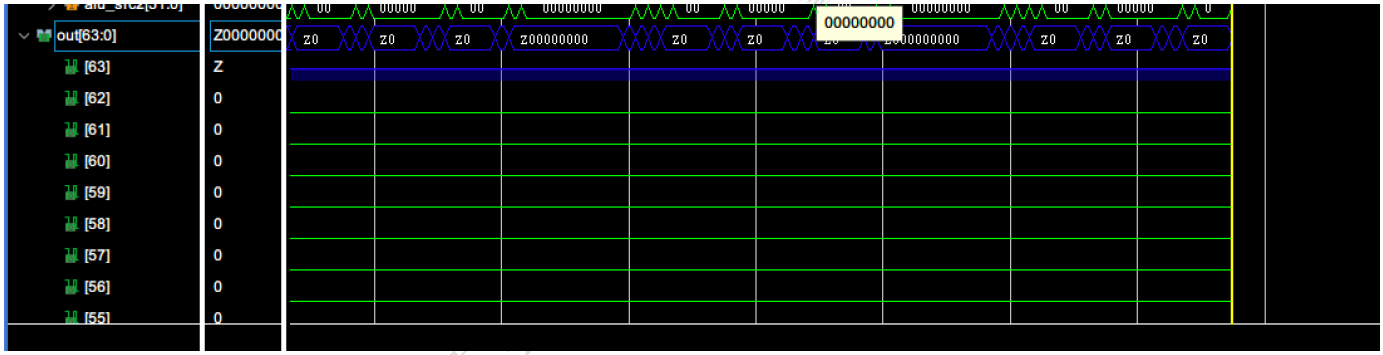

图 3.13 decode 的 OUT 信号最高位为’Z’

（2）分析定位过程

通过每个文件的一一查找，发现了此次实验的 bug 不止 7 处。第 8 处 bug 为 for 语句中的 i 的取值范围出现错误。导致上述信号最高位为 Z，通过找 tools 模块，定位到 64 译码 for 循环的 i 只赋值到了第 63 位。

（3）错误原因

Tools 模块中进行解码的 for 语句中的 i 的取值范围出现错误。代码中为“generate for (i=0; i<63; i=i+1)”，而上限应该是 i=63。

（4）修正效果

通过分析代码逻辑，将原代码更改为“generate for (i=0; i<64; i=i+1)”，此错误修复。波形恢复正常，如图 3.14。

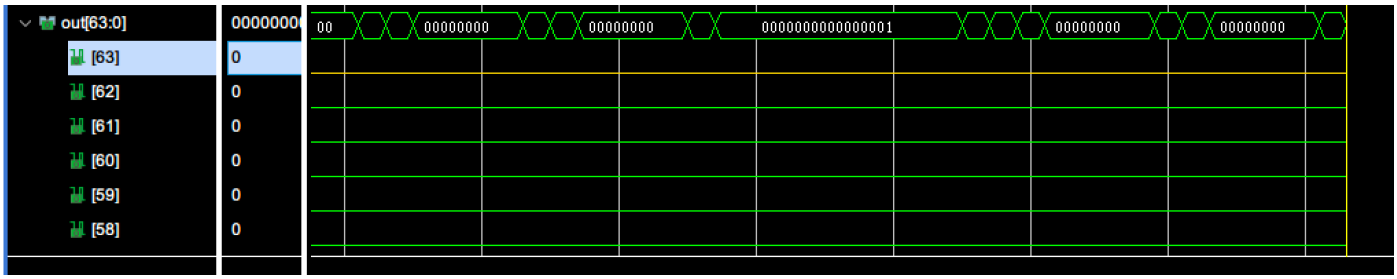

图 3.14 decode 的 OUT 信号最高位为恢复正常

（5）归纳总结（可选）

For 循环的语句的 i 的范围的确容易出错，写之前应当三思。

四、实验总结（可选）

本次试验由于中间增加了中秋小长假，导致做实验的时间所有缩短，短时间内要阅读实验文档、完成实验、撰写实验报告是一件工作量不小的事。第一次在 vivado 软件内部运用 trace 进行比对，一开始有一些不适应，习惯在 vivado 里遇到问题就查波形。但实验做着做着就发现，比对工作和上个学期的组成原理实验的比对流程很类似，于是就迅速习惯了这种方式。但是实验中除了 6 处明显能通过波形和比对 trace 能找出的 bug，另外两处 bug 都需要用细心一点去查看。实验指导里提到有 7 处 bug，但最终找出了 8 处 bug，不知道老师是刻意安排，还是也不小心笔误了。通过代码的调试，最终上板效果如下，表明调试成功。

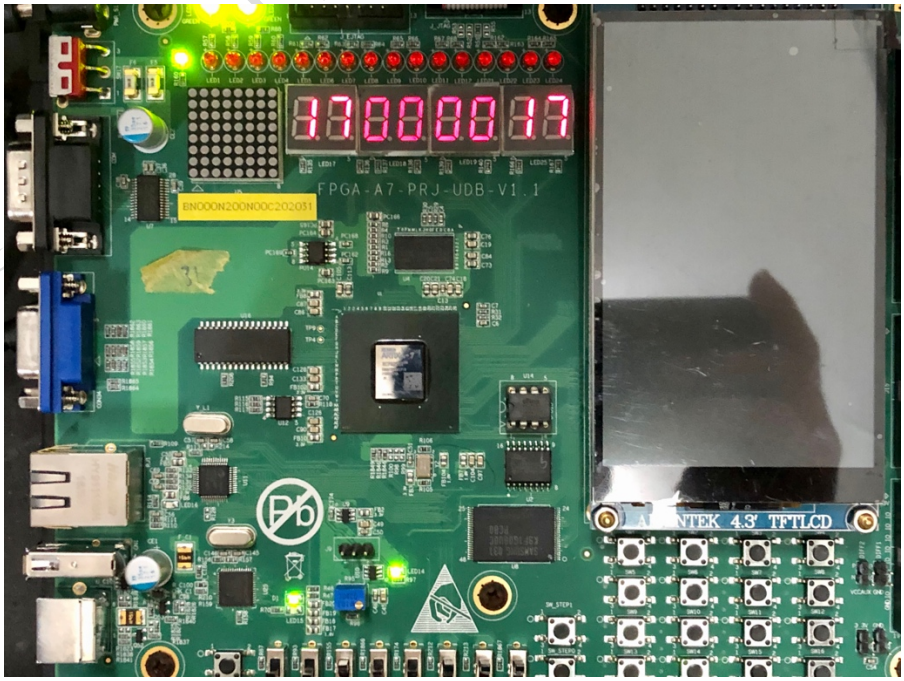

图 4.1 上板成功运行