实验 4 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流水线冲突产生的原因、设计阻塞五级流水线 CPU，并在上次给出的无阻塞 CPU 设计代码的基础上，判断译码阶段的寄存器读操作和后续进行的寄存器写操作是否有冲突，并以此作为修改 ds_ready_go 置“1”的条件，以实现阻塞的五级流水 CPU。

二、实验设计（40%）

（一）总体设计思路

硬件结构设计图如下：

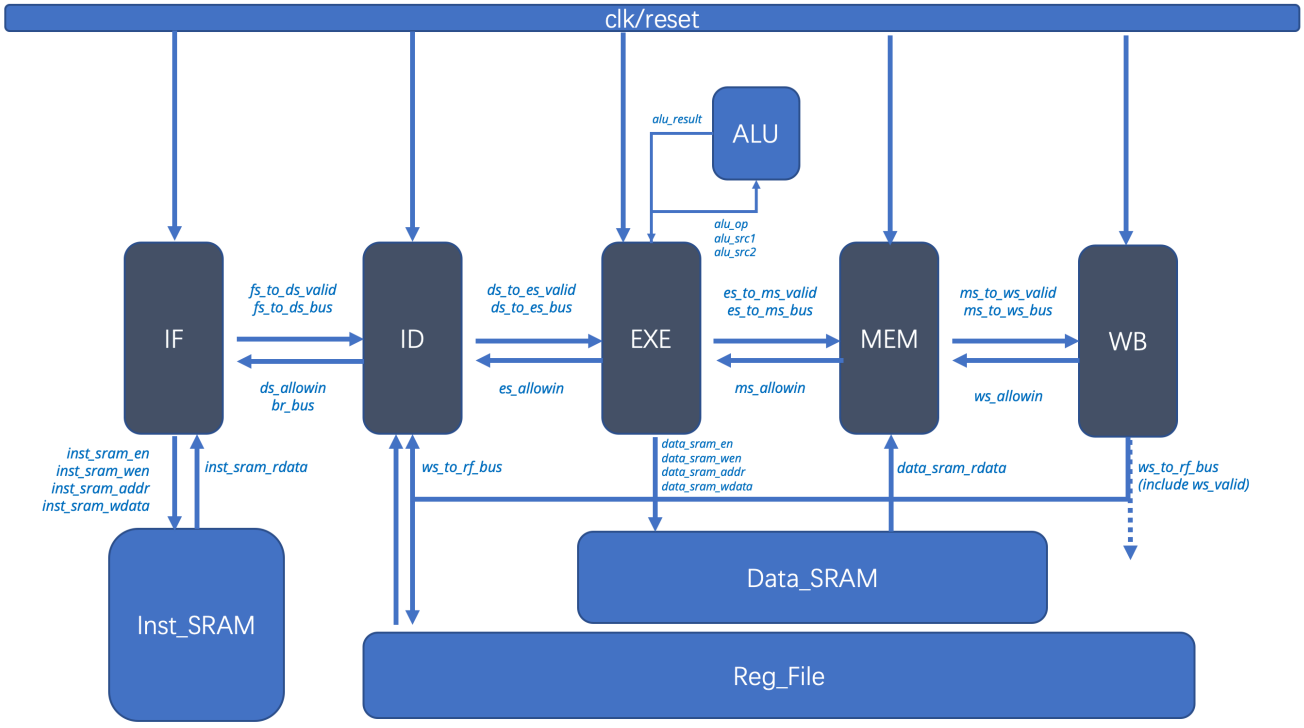

图 1.1 硬件结构设计图

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。在上次框架设计的基础上，该框架增加了四个接口连线，分别是将 es_to_ms_valid、es_to_ms_bus、ms_to_ws_valid、ms_to_ws_bus连接到了ID模块，其值将影响流水线阻塞的判断。

（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 传递给 ID 模块，ID 模块在下一周期再接收 。

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 ID 模块，ID 模块在下一周期再接收 。

（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、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。另外，写回阶段的数据也通过该模块传递给寄存器堆。

相较于之前的无阻塞五级流水设计，新的 ID 模块会利用 EXE、MEM、WB 传回的数据，比较三个后续状态里是否有与 ID 模块里寄存器读地址相同的寄存器，判断是否有“写后读”的行为，以此判断是否需要在该阶段进行阻塞。后阶段写寄存器的地址通过新接入的 es_to_ms_bus、ms_to_ws_bus 两条总线传输到 ID 模块（WB 阶段的写寄存器地址通过 ws_to_rf_bus 传递给 ID 模块）。

根据分析发现，对于发生阻塞的原因可以根据读写地址分成两大部分:寄存器堆读地址 1 和写地址冲突、读地址 2 和写地址冲突。同时，根据 MIPS 指令的特点发现，addu、subu、slt、sltu、and、or、xor、nor、bne、beq 可能与两个源操作数有关。其中 beq 和 bne 虽然没有对应寄存器写操作，但在译码阶段，跳转指令涉及的寄存器读数时，如果不加阻塞，就会被前面其他指令还未写回的状态所干扰，从而读出错误的值。addiu、jr

只与源操作数 1 有关，sll、srl、sra、sw 只与源操作数 2 有关。

根据上述分析，该设计设置了 wire 类型三个变量 hazard，sr1_hazard，sr2_hazard。其中 sr1_hazard，sr2_hazard 是分别判断译码阶段的寄存器读地址与 EXE、MEM、WB 阶段的寄存器地址是否有冲突；而 hazard 变量则是结合 MIPS 不同指令产生源操作数冲突的不同，判断是否会出现写后读现象。若 hazard 置为 1，则 ds_ready_go 置为 0；

在设置完 hazard 阻塞控制变量后，另外一个需要解决的问题是 CPU 完成阻塞后，如何进行恢复。解决这个问题主要有一下两种思路：

第一种思路是判断 EXE、MEM 和 WB 三个阶段模块内的 PC 值是否相等。若相等，则说明可以在时钟的下一拍将 ID 阶段的阻塞信号恢复。然而这一种做法虽然可以通过原有的两个新接入的 bus 获得 EXE 和 MEM 两阶段的 PC 值，但对于 WB 阶段的 PC 值需要增加 ws_to_rf_bus 的位宽，将 WB 阶段的 PC 值通过 ws_to_rf_bus 一同传递过来。

第二种思路是判断 EXE、MEM 和 WB 阶段的 valid 信号，而这三个信号的值分别由 es_to_ms_valid, ms_to_ws_valid 和 ws_to_rf 中新增的 ws_valid 信号来反映。当流水线发生阻塞后，上述信号会依次从 1 变为 0，而此处在 es_to_ms_valid, ms_to_ws_valid 和 ws_valid 信号变为 0 进行阻塞复位,wire 类型变量 hazard_reset 置为 1。当 hazard_reset 置为 1 时，hazard 信号置为 0，即停止阻塞流水。

在上述两种方案的选择中，本设计选择了第二种方案，因为其增加 ws_to_rf_bus 的位宽更少。

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
es_to_ms_bus	IN	71	EXE 模块向 MEM 模块传递数据
ms_to_ws_bus	IN	70	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

3、功能描述

将 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。此模块需要调用寄存器堆模块和 decode 模块。此外，该模块会利用 EXE、MEM、WB 传回的数据，比较三个后续状态里是否有 ID 模块里寄存器读地址相同的寄存器，判断是否有“写后读”的行为，以此判断是否要在该阶段进行阻塞。

（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。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 WB 阶段进行写回。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。

（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。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。

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。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。

（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 21 日 09：00-09：30 阅读讲义

9 月 21 日 09：30-11：30 进行五级阻塞流水线 CPU 设计，调试 bug

9 月 16 日 19：30-21：00 撰写实验报告

以下设计阻塞信号时的错误记录

（二）错误记录

1、错误 1：仿真停止在某一时刻

（1）错误现象

运行仿真，发现仿真运行到某一时刻不能继续进行，PC 值一直保持不变，如图 3.1.1。

```
[9952000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[9962000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[9972000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[9982000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[9992000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10002000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10012000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10022000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10032000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10042000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10052000 ns] Test is running, debug_wb_pc = 0xbfc41af8
```

图 3.1.1 仿真停止

（2）分析定位过程

如图 3.1.2，仿真时 PC 值一直保持不变，这说明流水线可能出现了一直阻塞的现象，对应查找 hazard 变量对应波形可以发现：hazard 的值在发生阻塞后，恒为 1。因此，此处需要设置一个 hazard_reset 阻塞复位信号。

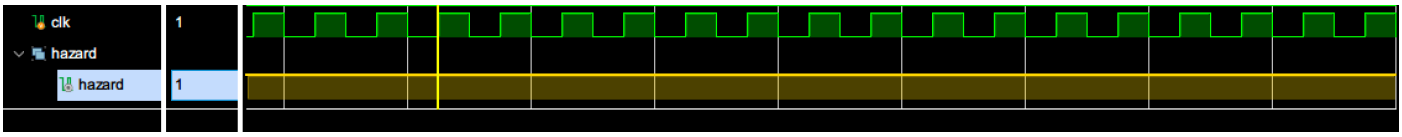

图 3.1.2 发生阻塞后 hazard 值恒为 1

（3）错误原因

hazard 变量由于未设置阻塞复位信号，导致发生阻塞后，值恒为 1。因此，此处需要设置一个 hazard_reset 阻塞复位信号。

（4）修正效果

设置一个 reg 类型的 hazard_reset 阻塞复位信号，当时钟上升沿时，倘若(!es_to_ms_valid&&!ms_to_bs_valid&&!ws_valid)为真，则 hazard_reset 信号置为“1”，其他状态置为“0”。如此修改后，波形不再停止，但出现新的错误,如图 3.1.3。

```
Test begin!
--------------------------------------------------------------
[  21985 ns] Error(  0)!!! Occurred in number 8'd240 Functional Test Point!
--------------------------------------------------------------
       [  22000 ns] Test is running, debug_wb_pc = 0xbfc41b04
--------------------------------------------------------------
[  22197 ns] Error!!!
    reference: PC = 0xbfc0069c, wb_rf_wnum = 0x0b, wb_rf_wdata = 0x00000000
    mycpu    : PC = 0xbfc0069c, wb_rf_wnum = 0x0b, wb_rf_wdata = 0x01555400
--------------------------------------------------------------
```

图 3.1.3 波形不再停止，但出现新的错误

（5）归纳总结（可选）

在设置判断信号时，不仅要想其置为 1 的情况，也要考虑其置为 0 的情况。

2、错误 2：数据错误

（1）错误现象

仿真时，trace 反映 wb_rf_wdata 错误，如图 3.1.3。

（2）分析定位过程

通过查看 trace 对比波形，如图 3.2.1，发现在 22,204,827ns 时刻，实际数据和对比标准数据不同。通过查看此刻的 valid 波形和 hazard_reset 波形发现，hazard_reset 在本应置为 1 的后一个时钟周期的上升沿才置为 1，导致寄存器读数读了之后写入的数 0x01555400 而非 0x00000000，如图 3.2.2。

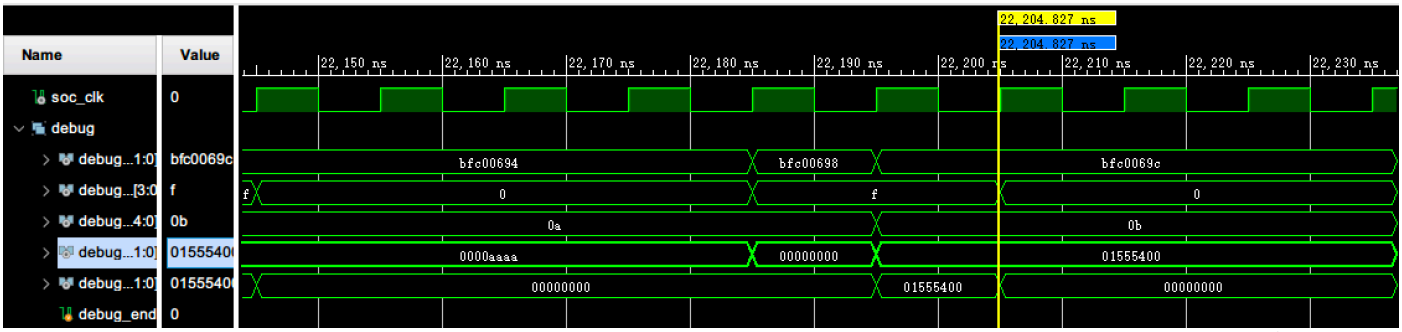

图 3.2.1 debug 模块波形对比

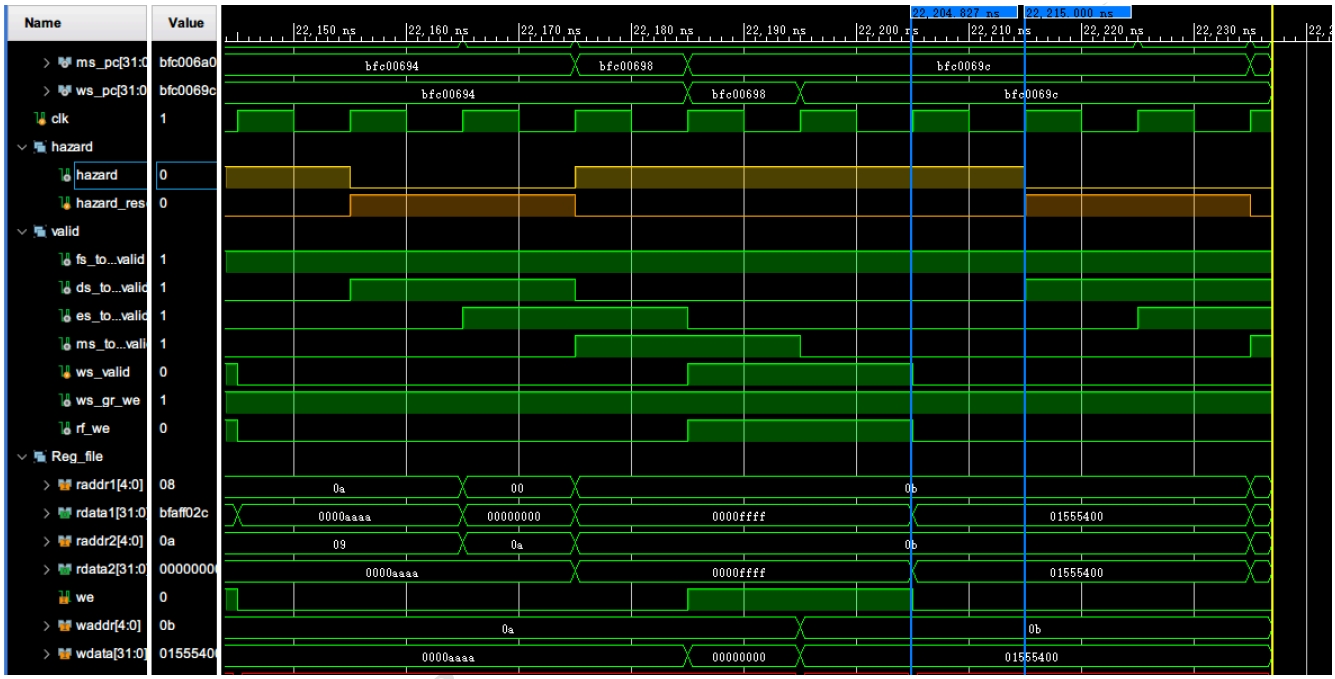

图 3.2.2 debug 模块波形对比

（3）错误原因

hazard_reset 在本应置为 1 的后一个时钟周期的上升沿才置为 1，导致寄存器读数读了之后写入的数 0x01555400 而非 0x00000000,因此，应该把 hazard_reset 信号更改为 wire 类型的变量。

（4）修正效果

hazard_reset 信号更改为 wire 类型的变量后。通过 assign hazard_reset = (!es_to_ms_valid&&!ms_to_bs_valid&&!ws_valid); 修改完毕后，该问题消失，全部测试点通过，如图 3.2.3。

```
================================================================
Test end!
----PASS!!!
$finish called at time : 1312485 ns : File "C:/Users/VincentTsai/Desktop/UCAS_CDE/mycpu_verify/testbench/mycpu_tb.v" Line 261
```

图 3.2.3 所有测试点通过

（5）归纳总结（可选）

对于时序逻辑电路部分，要弄清处在时钟上升沿时不同变量之间的关系，要思考清楚变量到底应该采用同步逻辑还是异步逻辑。

四、实验总结（可选）

本次试验开始需要自己在已有代码的基础上设计一些信号传递方式和数据通路。这是第一次需要独立设计一个小功能实现方式的模式，相较于之前做过的实验，有很多自主性，同时，在无形中也增加了一些难度。自主设计意味着需要权衡利弊，例如本次试验关于阻塞信号复位的判断就想出了两种方案，再仔细思考过后，我发现两种方案看似不同，实际上有相同之处，而只增加一个 ws_valid 信号的方案在数据传输带宽上要求更低，因此最终采用了这种方法。

这次实验虽然不要在老师写下的茫茫代码中去寻找几个 bug，但自己设计的代码中隐藏的问题往往更加难以发现，需要我在熟悉整个电路时序和组合逻辑，以及各种数据之间的关联的情况下，才能找出这些隐蔽的问题。借助 trace 对比和代码不断地调试，最终上板效果如下，表明调试成功。

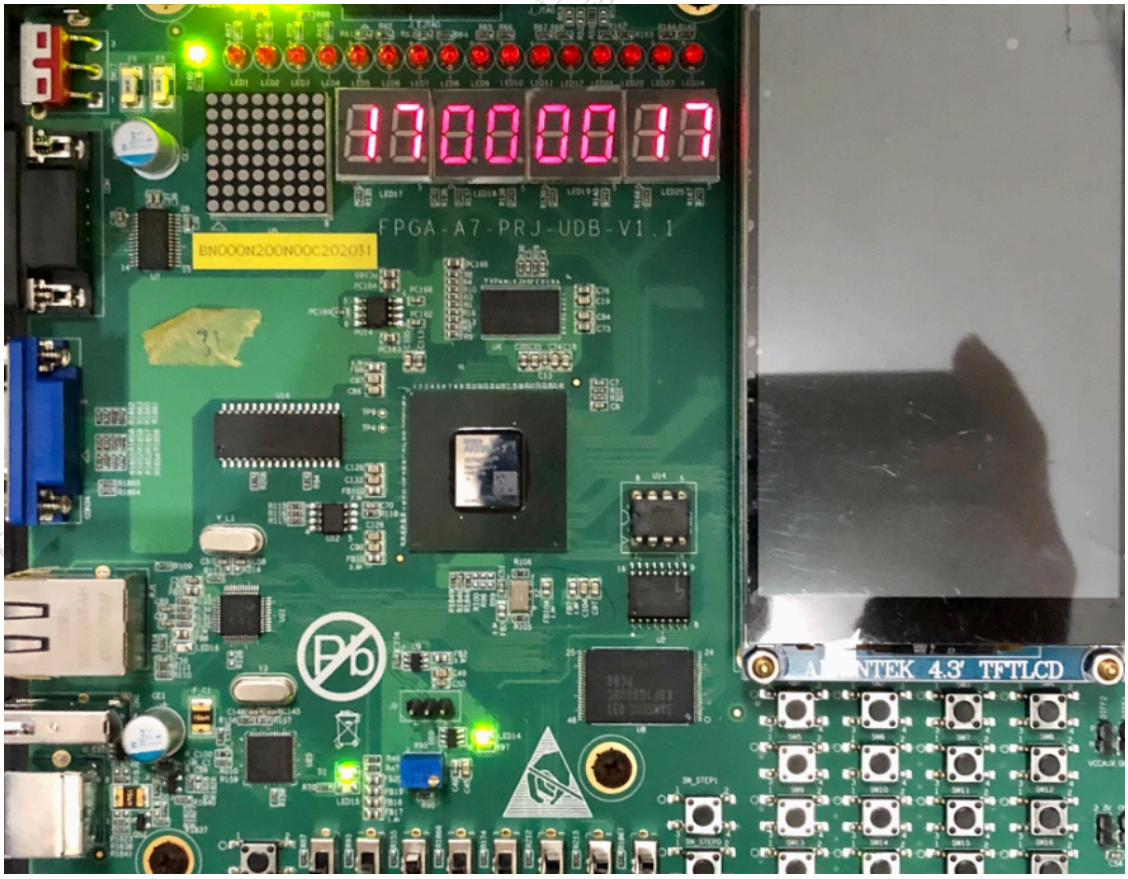

图 4.1 上板成功运行