实验 3-4 报告

第 27 小组
姚子薇、陈乐滢

一、实验任务

本次实验我们主要分为了三部分：

第一部分是用自己的除法器代替除法器 IP，重新跑过 func3，第二部分是添加 10 条非对齐访问指令，跑过 func4 并上板测试，最后一部分是优化代码，使得电路的时序更好，主频更高。

验证方法即通过仿真 68 个测试的 PASS，以及上板验证现象符合预期（LED 灯及数码管如任务书中所述），timing report 中的 WNS 等数据未出现红色，即时序违约。

以下为性能测试相关数据：

MyCPU 上板最高可运行频率	Dhrystone total_ns	Coremark total_ns	DMIPS/MHz	Coremark/MHz
100MHz	48660	3827030	1.18	2.61

二、实验设计

（一）除法器设计

上一周的实验中我们用的是 IP 核除法器进行调试，但是将 div 模块的接口封装为参考文档里的接口，以方便这一周替换为我们自己写的除法器。自己写的除法器每一次需要 34 拍，IP 核的 latency 是 37，这样一来还减少了运行时长。

除法的过程分为三部分：

第一步是确定符号，计算绝对值。有符号除法中，如果是正数，就不变，如果是负数，取反加 1；无符号除法则直接使用除数和被除数原值，不做调整。

用({32{x_sig}} ^ x) + x_sig 这种写法修正有符号除法的操作数比较好，正负数情况统一不需要选择器。

第二步是进行迭代除法，这一块看起来复杂，实际上就是一个时序逻辑，每一拍重复做一样的事情。

我们认为一个比较好的设计在于，参考文档上写的是每一次取 64 位扩展的被除数 A 的不同区间，和 33 位的除数 B 相减，这样的设计比较复杂。我们是将 B 也扩展到 64 位，A-B 之后 A 不用改变，直接将 B 右移一位，就可以进行下一次迭代了，得出的结果是完全一致的，但是操作要更简洁。

第三步是恢复商和余数的符号，如果是无符号数除法，直接使用第二步算出来的值即可。

比较重要的一点就是商、余数以及是否是有符号除法的信号都是在第一步确定出来的，需要随时钟信号一直保存到第 34 拍，如下图代码段：

```
always @(posedge div_clk) begin
    if (~resetn) begin
        s_sig_reg <= 1'b0;
        r_sig_reg <= 1'b0;
        div_signed_reg <= 1'b0;
    end
    else if(div && step==0) begin
        s_sig_reg <= s_sig;
        r_sig_reg <= r_sig;
        div_signed_reg <= div_signed;
    end
end
```

图 1. 除法器符号信号延续代码

其他调试中的问题见错误记录。

（二）非对齐访存指令设计

1、非对齐 load 指令

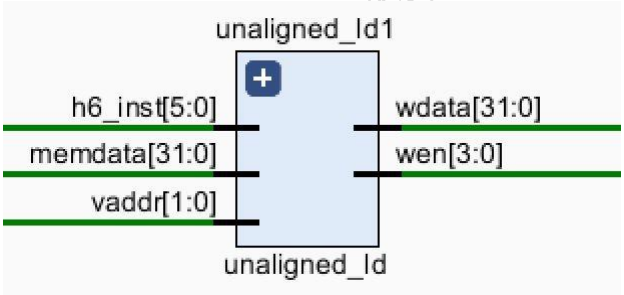

图 2. 非对齐 load 译码设计

由于 load 指令中有非对齐访问，而 load 指令写回数据是在 mem 级产生的，所以我们在 mem 级添加模块 unaligned_ld，输入当前 inst 的高 6 位 h6_inst[5:0]，从 mem 中读取的数据 memdata[31:0]，以及地址的低两位 vaddr[1:0]，输出给寄存器的写回数据 wdata[31:0]以及给寄存器的字节写使能[3:0]，其中写使能流入 wb 级再处理。

在模块中我们分别对 inst 及 vaddr 做译码，分离出每一种指令以及 addr 低两位代表的值，根据 addr 取 mem 不同的 Byte/Half 等，再形成最终的 wdata 及 wen 信号。其中 lb 指令所取 Byte 的偏移量由地址确定，具体代码如下：

```
assign lbdata = addr_0 ? memdata[7:0]:
                addr_1 ? memdata[15:8]:
                addr_2 ? memdata[23:16]:
                addr_3 ? memdata[31:24]:
                         8'b0;
```

图 3. lb 指令代码设计

2、非对齐 store 指令

Store 的译码部分和 load 十分类似，输入数据改为从寄存器读出的数据 regdata[31:0]，且由于 store 指令的目标地址是在 exe 产生的，且对内存的写入数据需要在 mem 级前产生，将 unaligned_st 模块置于 exe 级，其他输入输出信号与 load 相同，不再赘述。我们仍旧根据 inst 和 vaddr 低两位进行处理，其中 sb 指令的数据形成代码如下所示：

```
assign sbdata = addr_0 ? {24'b0, regdata[7:0]}:
                addr_1 ? {16'b0, regdata[7:0],  8'b0}:
                addr_2 ? { 8'b0, regdata[7:0], 16'b0}:
                addr_3 ? {regdata[7:0], 24'b0}:
                         32'b0;
```

图 4. sb 指令代码设计

三、实验过程

（一）实验流水账

1. 姚子薇

2017.10.28，20:00 - 23:00，写除法器。

2017.10.29， 9:30 - 12:30，除法器 debug，将除法器加入流水仿真调试。

2017.10.30，20:00 - 23:30，参与 func4 的调试。

2017.10.31，10:00 - 13:20，跑性能测试，写实验报告。

15:20 - 17:30，调试 Coremark，测上板最高频率，完善实验报告。

2. 陈乐滢

2017.10.30，16:00 - 24:00，添加非对齐访问指令，仿真通过。

2017.10.31，10:00 - 12:00，进行综合、布局布线，func4 上板通过。

2017.10.31，10:00 - 13:20，跑性能测试，写实验报告。

15:20 - 17:30，优化代码，上板，完善实验报告。

（二）错误记录

0、错误 0

（1）错误现象

（来自 lab3-3 的遗留错误）仿真 pass，综合及布局布线均通过，但上板时低两位在 2b 处卡了四个测试之后与高两位同时累加，最后停留在 3a000036

（2）分析定位过程

在助教及老师均尝试 debug 无果后，听到同学说 input output 写反可能会导致这样的问题

（3）错误原因

把 input 写成了 output qwq。

（4）修正效果

将该前递信号的 output 改成 input 后上板正确。

1、错误 1

（1）错误现象

[time@455000]Error: x= 2097015289, y= 3812041926, signed=0, s=x, r=x, s_ref=0, r_ref= 2097015289, s_OK=x, r_OK=x

（2）分析定位过程

s_reg 和 s_reg_next 都是 xxxx,查这两个信号的代码段。

（3）错误原因

首先 s_reg 没初始化；

其次 s_reg_next 最后一位 dif_sig 没初始化；

最后发现 dif 位数还写错了,写的[32:0]，应为[63:0]。

（4）修正效果

添加初始化，修改位数后正确。

2、错误 2

（1）错误现象

[time@455000]Error: x= 2097015289, y= 3812041926, signed=0, s= 2419535833, r= 2096456739, s_ref= 0, r_ref= 2097015289, s_OK=0, r_OK=0

（2）分析定位过程

r 的计算值不对，往前定位，dif_sig 信号无误，但是修改剩余被除数有问题。

（3）错误原因

```
assign s_reg_next = {s_reg[30:0], dif_sig};

A_reg <= (dif_sig) ?  dif : A_reg;
```

以上两句关于 dif_sig 的判断和使用都弄反了。

dif_sig 为 1 时，差为负数，上商应为 0，剩余被除数不修改。

（4）修正效果

修改为

```
assign s_reg_next = {s_reg[30:0], ~dif_sig};

A_reg <= (dif_sig) ? A_reg:dif;
```

跑除法器的 tb 正确。

3、错误 3

（1）错误现象

将除法器放入 CPU 流水级中，计算结果错误。

（2）分析定位过程

查看波形，是两个操作数不对，用的是 decode 级信号。

（3）错误原因

用除法器 IP 核时 div 信号在 decode 级，操作数、div_signed 是在 exe 级输出给除法器的，但是自己写的除法器要求这几个量同时输出。

（4）修正效果

将被除数、除数、有无符号计算改为在 decode 级输出，修改后正确。

4、错误 4

（1）错误现象

计算结果正确，但是第二个 MF 取数错误。

（2）分析定位过程

计算结果出来的下一拍的确将商和余数写回相应寄存器了，但是下一拍又一次进行写入，更改了寄存器内的值，导致错误。

（3）错误原因

由于流水线阻塞，在写入的下一拍仍是除法的 wb 级，此时写使能 wen 仍保持 f，所以再次写入。

（4）修正效果

complete 输出到写回级，限制写使能 wen，只写入除法结束时得出的商和余数。

5、错误 5

（1）错误现象

有符号数除法，商正确是 0，余数错误。

（2）分析定位过程

检查进行除法的整个过程，第一步取绝对值无误，商的计算过程无误，但是计算完毕没有恢复余数的符号，再定位到商和余数的符号位只在最开始的一拍出现了，所以最后默认为不恢复。

修改之后发现还是没有恢复符号位，原因是进行判断是无符号还是有符号除法的 div_signed 也没有延续到恢复的那一拍。

（3）错误原因

s 和 r 的符号位信号没有传递下去，导致没有绝对值的恢复调整；

div_signed 也只出现了一拍，没有延续下去。

（4）修正效果

给 s_sig，r_sig，div_signed 都加一个寄存器变量，将它们的值从开始一拍一拍延续到最后恢复符号，修改后加入字写除法器的 fun3 仿真通过。

6、错误 6

（1）错误现象

[1249195 ns] Error!!!

reference: PC = 0xbfc57e20, wb_rf_wnum = 0x02, wb_rf_wdata = 0x0000000b

mycpu　: PC = 0xbfc57e20, wb_rf_wnum = 0x02, wb_rf_wdata = 0xffffffe0

（2）分析定位过程

PC = 0xbfc57e20 的 exe 级的两个用来算地址的操作数是正确的，从 data_sram 中读出的数据为 0xc83b0be0，wdata 用来扩展的数是最后两位 0xe0，导致 wdata 错误。观察 test.s 发现存入地址是(t0)+14240，而 lb 的目标地址是(t0)+14241，对应位为 0x0b。

（3）错误原因

没有考虑到 data_sram 是按照对齐访问的，应该按照低两位地址来取从 data_sram 中读出来的对应的 Byte。

（4）修正效果

在执行 lb(u)指令时按照低两位地址取对应 Byte，即更改为：

```
assign lbdata = addr_0 ? memdata[7:0]:
                addr_1 ? memdata[15:8]:
                addr_2 ? memdata[23:16]:
                addr_3 ? memdata[31:24]:
                    8'b0;
```

后 lb 及 lbu 的 test 通过。。

（5）归纳总结

我们坚信其他非对齐访问指令也会出现相似的问题，但是因为害怕 lh 的偏移量是 1 或者 3 没有更改。

7、错误 7

（1）错误现象

[1301535 ns] Error!!!

reference: PC = 0xbfc51968, wb_rf_wnum = 0x02, wb_rf_wdata = 0x00004932

mycpu　: PC = 0xbfc51968, wb_rf_wnum = 0x02, wb_rf_wdata = 0x0000330c

（2）分析定位过程

在该指令的 mem 级从 data_sram 中读出的数据为 0x4932330c，我们的 CPU 取了低 16 位而非高 16 位，推测还是偏移量的问题。

（3）错误原因

没有按照低两位地址来取从 data_sram 中读出来的对应的 Byte，lh(u)指令中偏移量只有 0 或者 2 两种情况。

（4）修正效果

做与错误 1 中类似更改后 lh 及 lhu 的 test 通过。

8、错误 8

（1）错误现象

[1343015 ns] Error!!!

reference: PC = 0xbfc12a60, wb_rf_wnum = 0x09, wb_rf_wdata = 0x2b9c0000

mycpu　: PC = 0xbfc13ccc, wb_rf_wnum = 0x09, wb_rf_wdata = 0x3f000000

（2）分析定位过程

我们发现 13ccc 是出现错误后跳到的位置，而跳转之前有一个 lwl 指令，推测是其出现了错误。即如下指令。

bfc12a48:89022a22　　lwl　v0,10786(t0)

根据 test.s 中的指令序列，我们发现 v1 寄存器值为 0xe8d3ee80，我们 lwl 之后 v0 寄存器的值为 0xe8d3ee00，推测是 wen 信号出现问题。之后我们发现在 wb 级没有内存地址的对应变量，wen 没有依据正确的 vaddr 来改变。

（3）错误原因

wen 设置的判断依据选取有误。

（4）修正效果

将产生 wen 信号的功能放在 mem 级，同非对齐 load 的数据一同产生，再输入到 wb 级，错误依然存在。

9、错误 9

（1）错误现象

现象同错误 8。

（2）分析定位过程

我们发现在 mem 级产生的 load_wen 信号为 e，即 4'b1110，是正确的，而寄存器仍然写入了全部 32 位的值，考虑到是 regfile 设计有误。

（3）错误原因

Regfile 更改设计时未考虑字节写使能。

（4）修正效果

更改 regfile 的设计为字节写使能后全部的 load 指令通过。

（5）归纳总结

再久远的 bug 都是要还的。

10、错误 10

（1）错误现象

在 sb 指令后面的 lw 指令时写回值有误。

（2）分析定位过程

观察 test.s 中的指令序列，发现出现了 lb 中类似的非对齐访问，于是做类似更改。

（3）错误原因

没有添加 store 操作中的非对齐访问。

（4）修正效果

分别更改 sbdata，shdata 以及写使能信号，sbdata 的赋值如下：

```
assign sbdata = addr_0 ? {24'b0, regdata[7:0]}:
                addr_1 ? {16'b0, regdata[7:0],  8'b0}:
                addr_2 ? { 8'b0, regdata[7:0], 16'b0}:
                addr_3 ? {regdata[7:0], 24'b0}:
                    32'b0;
```

更改后 func4 的全部仿真通过。

11、错误 11

（1）错误现象

coremark 时报错：

[33165 ns] Error!!!

reference: PC = 0xbfc08c24, wb_rf_wnum = 0x07, wb_rf_wdata = 0x00000002

mycpu : PC = 0xbfc08c24, wb_rf_wnum = 0x07, wb_rf_wdata = 0x800042cf

（2）分析定位过程

相关代码段

bfc08c1c:a5a70002 sh a3,2(t5)

bfc08c20:25290001 addiu t1,t1,1

bfc08c24:24e70001 addiu a3,a3,1

a3 值在寄存器中是对的，但是 decode 级用的是错的，应该是误判断了数据前递。

（3）错误原因

sh 在前时，后面的指令用到相应寄存器，不判断为相关。sw 之前写过，加入了新的 store 指令没有进行更新。

类似的问题还有，BNEQ 在前不判断为相关。

（4）修正效果

修改判断条件后后正确，但在接近 func 结尾处又出现了错误，这是因为只更改了 vsrc1 没有改 vsrc2。

四、实验总结

（一）组员：姚子薇

没有出现奇奇怪怪的问题已经心满意足了。Coremark 出现 error 的时候冷汗都要下来了。。。

看着时序里长长的还绕一圈路径十分糟心，估计是前递造成的，但是真的不知道还能怎么优化，感觉写的已经比较精简了来着。

（二）组员：陈乐滢

非对齐访问是真的乱，不过总体来说这边的 bug 比之前要好 de 多了。前递造成数据通路乱七八糟的而且很长。