实验 6 报告

学号　2019K8009929019　2019K8009929026
姓名　桂庭辉　高梓源
箱子号　44

1 实验任务

本次实验在流水线中添加部分普通用户态指令，主要包括 9 条算术逻辑运算类指令和 7 条乘除运算类指令。由于实现的指令主要针对运算，故而主要的实现基于已有的流水线框架与已实现指令的数据通路即可，较为复杂的处理在乘除法指令上。

2 实验设计

2.1 总体设计思路

9 条算术逻辑运算类指令的实现主要复用此前流水线中已实现的部分指令功能。`slti, sltui, andi, ori, xori` 指令分别复用 `slt, sltu, and, or, xor` 指令数据通路，处理立即数内容与相关控制信号即可。`sll, srl, sra` 指令分别复用 `slli, srli, srai` 指令数据通路，调整操作数来源与相关控制信号即可。`pcaddu12i` 指令主要通路复用 `add` 指令，操作数处理上来自 PC 寄存器与 `lu12i` 指令结果。由于此部分几乎全部为对已有数据通路的复用，仅添加了部分控制信号，故而本篇报告中不再详述该部分的实现细节。

7 条乘除法运算类指令中，乘法的处理我们使用自己编写的流水化乘法器，在两拍内计算出乘法结果；除法与取模的处理我们选择调用 Xilinx IP 定制除法器运算部件进行实现。具体运算的进行在 ALU 部件中完成，由于引入了新的运算与相关控制信号，需要维护流水级间的寄存器宽度、`alu_op` 宽度以及流水线控制的部分信号。

2.2 重要模块设计：mul_top 与流水化后乘法实现

本小节所有模块设计均在 tools.v 文件中。

2.2.1 功能描述

乘法器顶层模块，例化 17 个数相加华莱士树模块 `wallace` 与 33 位 Booth 两位乘部分积生成器 `boothgen` 完成乘法器整体搭建与流水化。

2.2.2 工作原理

考虑硬件资源的节省，本次设计一个乘法器以完成有符号与无符号 32 位补码乘法。实现上如实验讲义所说将输入扩展成 33 位，通过 33 位补码乘法器实现对有无符号运算的同时支持。整体结构同样来自实验讲义。

Booth 两位乘部分积生成器的设计来自理论课教材，17 个数相加的华莱士树设计来自实验课讲义。此处均不再赘述。在流水级划分上，考虑 Booth 部分积生成逻辑与华莱士树中 6 层 1 位全加器的总延迟约为 20 余级门（具体延迟大小取决于 Vivado 对 1 位加法器的实现），最终合并华莱士树结果的 66 位加法器将产生相当级别的延迟，故而选择将流水级切分在 66 位加法器入口处实现。采用一个 133 位寄存器将加法器的两个 66 位输入与 1 位进位输入锁一拍以实现流水化，在两拍内计算得到乘法结果。

2.2.3 接口定义

表 1: 乘法器顶层模块接口

名称	方向	位宽	描述
clk	In	1	时钟信号
rst	In	1	复位信号，高电平有效
mul_signed	In	1	是否做有符号乘法
src1	In	32	源操作数 1
src2	In	32	源操作数 2
res	Out	64	乘法运算结果

表 2: 华莱士树模块接口

名称	方向	位宽	描述
in	In	17	待相加的 17 个 1 位数
cin	In	14	来自先前华莱士树或转置部件的进位输入
cout	Out	14	传递给后续华莱士树的进位输入
carry	Out	1	压缩结果：进位输出
sum	Out	1	压缩结果：和

表 3: 部分积生成模块接口

名称	方向	位宽	描述
y	In	3	来自乘数的部分积生成标志信号
x	In	66	移位处理后的被乘数
c	Out	1	部分积的末位进位输入（取反“加一”）
p	Out	66	部分积

2.2.4 乘法实现的其他逻辑

由于乘法器与其他运算逻辑一样集成到 ALU 部件中，而流水化后的乘法指令为了避免影响流水线性能，应如访存指令一样在指令进入流水级时取得乘法结果，为了实现该功能，需要将乘法运算结果导出至执行级外传递给访存级。乘法实现里引入的时序逻辑除此前介绍的乘法计算中的流水级切分外，乘法所需结果是高 32 位还是低 32 位所需的控制信号也需要时序逻辑保存到下一拍，即乘法指令位于访存级时。该信号的传递可以通过执行级与访存级间的流水槽实现，也可以单独缓存该信号利用传递乘法结果的通路进入访存级参与结果选择。

此外，由于乘法指令与访存指令一样，在访存级才能得到写回结果，故而如访存指令一样，也需在前递机制中引入乘法指令于执行级与译码级指令冲突时的阻塞信号。

2.3 除法器 IP 调用与时序处理

笔者将乘除法所有实现放在 alu.v 中，这就意味着 alu_op 需要拓宽至 19 位，囊括所有乘除法指令信息，对于除法需要有无符号的信息、取商或余数的信息，并且这些信息彼此互斥，很难提取共同特征而缩减，因此可直接对乘除法每类指令设置一位 alu_op，求得的结果也可以根据 alu_op 直接选取。

2.3.1 接口定义

根据讲义描述，调用并设置好 Vivado 提供的有无符号除法 IP 核，注意 IP 核暴露的所有接口都有作用，需要全部连接，接口如下：

接口	位宽	I/O	说明
aclk	1	input	时钟驱动，迭代除法
s_axis_divisor_tdata	32	input	除数输入
s_axis_dividend_tdata	32	input	被除数输入
s_axis_divisor_tready	1	output	是否准备好读取除数
s_axis_dividend_tready	1	output	是否准备好读取被除数
s_axis_divisor_tvalid	1	input	除数输入是否有效，开始运算
s_axis_dividend_tvalid	1	input	被除数输入是否有效，开始运算
m_axis_dout_tdata	64	output	商和余数结果输出
m_axis_dout_tvalid	1	output	输出是否有效，当前除法是否算完

2.3.2 使用说明

正如讲义强调的，除法 IP 核采用 AXI 总线接口，Ready 和 Valid 信号同时有效时开始输入，经过观察，我们设置的 clocks per Division 选项是 Ready 信号每次拉高间隔的时钟周期数，我们需要保证 Valid 信号第一次和 Ready 信号同时拉高后必须在该除法计算整个完成后 (m_axis_dout_tvalid 信号拉高) 之后才能有条件再度拉高，进行下一次计算，在此过程中流水级也需要进行阻塞。

笔者在实验中使用了另一个信号 div_valid，它为高电平当且仅当整个除法运算开始且未结束，于是一旦它为高电平，除法器输入的两个 Valid 信号就不能拉高。因为除数被除数的 Ready 和 Vallid 信号需要同时拉高，用同一个 div_data_valid 信号进行控制即可，于是下面是控制信号的时序赋值过程：

```
always @(posedge clk) begin
    if (div_valid) begin
        div_data_valid <= 1'b0;
```

```
        end else if (es_valid & use_div & (~divisor_data_ready | ~dividend_data_ready))
            begin
            div_data_valid <= 1'b1;
        end else begin
            div_data_valid <= 1'b0;
        end


        if (div_valid) begin
            divu_data_valid <= 1'b0;
        end else if (es_valid & use_divu & (~u_divisor_data_ready | ~
            u_dividend_data_ready)) begin
            divu_data_valid <= 1'b1;
        end else begin
            divu_data_valid <= 1'b0;
        end

        if (rst) begin
            div_valid <= 1'b0;
        end else if (div_res_valid | divu_res_valid) begin
            div_valid <= 1'b0;
        end else if ((div_data_valid & divisor_data_ready) | (divu_data_valid &
            u_divisor_data_ready)) begin
            div_valid <= 1'b1;
        end
    end
```

除法器实例化连接后，对于结果直接取 $[63:32]$ 位作为商，$[31:0]$ 作为余数。

3 实验过程

3.1 实验流水账

2021.09.30 08:00 ~ 2021.09.30 09:00 ： 完成第一部分，对 9 条算术逻辑类指令的实现

2021.10.01 14:00 ~ 2021.10.01 16:00 ： 利用 Xilinx IP 完成乘除法指令实现

2021.10.02 15:00 ~ 2021.10.02 16:45 ： 完成流水化乘法器支持乘法指令

2021.10.02 18:00 ~ 2021.10.03 00:00 ： 撰写实验报告

3.2 错误记录

3.2.1 错误 1：流水化后乘法结果获取出错

3.2.1.1 错误现象

对于乘法指令，在访存级获取到的计算结果为 0。

3.2.1.2 分析定位过程

无论是访存级还是执行级，其中获取到的乘法结果均为 0，但乘法器结果正确，说明问题来自 ALU 中对乘法结果的处理。

3.2.1.3 错误原因

在最初的设计中，我们在 ALU 中根据 `alu_op` 译码结果对乘法结果的高低位进行选择，然后传递 32 位数据给访存级，这一过程的“选择”使用与或逻辑实现，同时该选择过程为纯组合逻辑。那么在乘法指令进入访存级时，下一条指令进入执行级，`op_mul` 等乘法译码结果均被拉低，导致选择器结果为 0 无意义。

3.2.1.4 修正效果

注意到选择信号的时序问题，我们选择引入寄存器保存高低选择信号至下一拍，同时传递的乘法结果为 64 位的完整结果以及保存下来的高低选择信号，将选择器的实现放在访存级中完成。

3.2.1.5 归纳总结

该问题来自对乘法器流水化导致的时序问题认识不足，意味着在设计时未能准确完善地把握控制信号的维护与传递。

3.2.2 错误 2: 除法器 aclk 信号未接入

3.2.2.1 错误现象

除法一直阻塞在 exe 流水级，div 模块中 aclk 显示黄色 U，s_axis_divisor_tready 信号始终未能拉高。

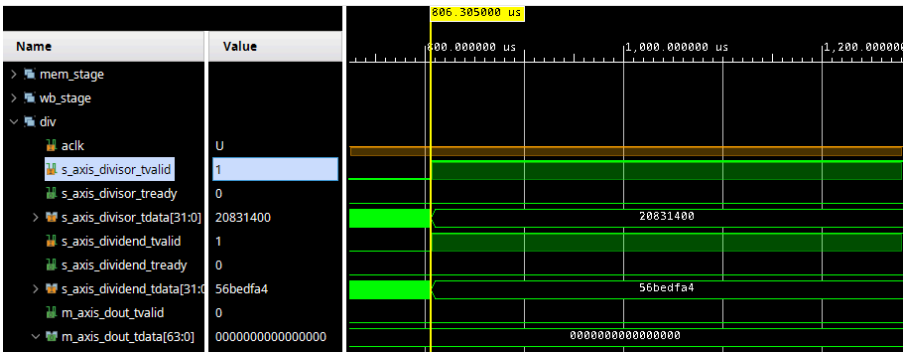

图 1: 除法器 aclk 信号悬空

3.2.2.2 分析定位过程 & 错误原因

因为错误为除法指令，去查看 div 模块如何出错，发现一个未赋值信号

3.2.2.3 修正效果

只需要将 div 和 divu 模块接口全部定义就能避免这种问题

```
div_gen div(
    .aclk                   (clk),
    .s_axis_divisor_tdata   (alu_src2),
    .s_axis_dividend_tdata  (alu_src1),
    .s_axis_divisor_tready  (divisor_data_ready),
    .s_axis_dividend_tready (dividend_data_ready),
    .s_axis_divisor_tvalid  (div_data_valid),
```

```
        .s_axis_dividend_tvalid (div_data_valid),
        .m_axis_dout_tdata      (div_result),
        .m_axis_dout_tvalid     (div_res_valid)
    );
```

3.2.2.4 总结归纳

今后在遇到新的模块进行例化时要将需要操作的接口看全、注意类型和位宽，恰当使用。

3.2.3 错误 3: 除法器运算过程持续读取输入

3.2.3.1 错误现象

在遇到一个除法指令后，Valid 和 Ready 信号仍然同时拉高，读取除法器输入并进行运算，接连输出多个结果。后一条乘法指令的结果被覆盖导致错误。

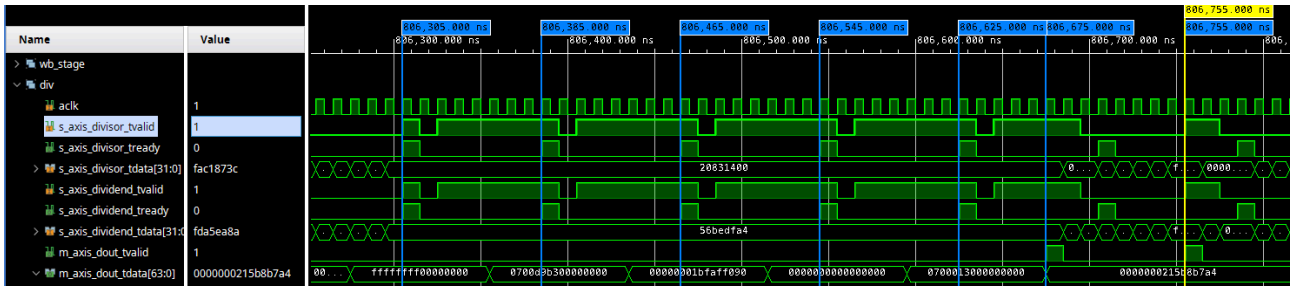

图 2: 除法器 Valid 信号设置错误

3.2.3.2 分析定位过程

最初留意的现象是连续两次除法运算时，后一条指令 Valid 和 Ready 信号已经读取成功，但并不开始运算。与此同时，除法器结果和前一条除法指令相同，并且 m_axis_dout_tvalid 信号为高电平。

多次尝试分析后才理解讲义的意思，Valid 信号并不是每遇到一次 Ready 就拉低，而是在一次除法过程中只能拉高一次，否则会持续读取输入进行运算。

3.2.3.3 错误原因

除法器 Valid 信号设置错误，在一次乘法指令中多次读取操作数输入进行运算，导致结果多次输出，可能会覆盖下一次乘法指令的结果。

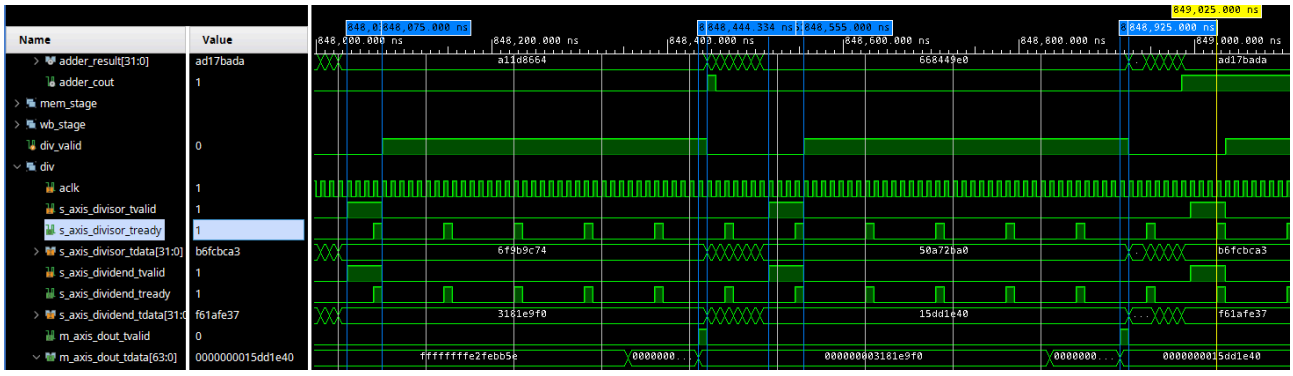

图 3: 正确 Valid 输入信号效果

3.2.3.4 修正效果

采用新的变量 div_valid 来表示除法开始到结束的过程，在此过程中除法器 Valid 输入只能在最开始拉高，紧接着持续拉低，修正代码如 2.3.2中代码所示。修正后效果如上所示。

3.2.3.5 总结归纳

在遇到 AXI 总线多拍实现的部件时，需要对它输入输出接口的时序又深入了解，接合实现功能进行对接和赋值。

4 实验总结

除法器始终使用 Xilinx IP，乘法器使用 Xilinx IP(即 * 号) 时 WNS 为 0.162 ns，使用自行设计的单周期乘法器时 WNS 为 0.073 ns，将乘法器流水化后 WNS（两拍完成乘法）为 1.418 ns