《计算机组成原理》实验报告

年级、专业、班级	2020 级计算机科学与技术 01 班 2020 级计算机科学与技术 02 班	**姓名**	陈鹏宇 徐小龙
实验题目	实验一简单流水线与运算器实验		
实验时间	2022 年 04 月 16 日	**实验地点**	Ds1410
实验成绩	优秀/良好/中等	**实验性质**	□验证性 ☑设计性 □综合性

教师评价：

□算法/实验过程正确； □源程序/实验内容提交； □程序结构/实验步骤合理;

□实验结果正确； □语法、语义正确； □报告规范;

其他:

评价教师: 钟将

实验目的

（1）理解流水线 (Pipeline) 设计原理；

（2）了解算术逻辑单元 ALU 的原理；

（3）熟悉并运用 Verilog 语言设计 ALU；

（4）熟悉并运用 Verilog 语言设计流水线全加器；

报告完成时间: 2022 年 5 月 28 日

1 实验内容

1.1 ALU 设计实验

实验要求实现以下算术运算功能，其对应的控制码及功能如下：

$F_{2:0}$	功能	$F_{2:0}$	功能
000	A + B(Unsigned)	100	$\overline{A}$
001	A - B	***101***	***SLT***
010	A AND B	110	未使用
011	A OR B	111	未使用

表 1: 算数运算控制码及功能

实验要求：

1. 根据 ALU 原理图，使用 Verilog 语言定义 ALU 模块，其中输入输出端口参考实验原理，运算指令码长度为 [2:0]。
2. 内置一个 32 位 num2（值为 32h'01）作为输入到运算器端口 A；
3. 将 sw0-sw7 输入到 num1, 经过无符号扩展至 32 位后，输入到运算器的端口 B；
4. 运算器支持“加、减、与、或、非”5 种运算，需要 3 位（8 个操作）。将 sw15-sw14 输入到 op 作为运算器的控制信号；
5. 实现 SLT 功能。
6. 将计算 32 位结果 s 显示到七段数码管 (16 进制)。
7. 验证表 1 中所有功能。
8. 给出 RTL 源程序（.v 文件）

1.2 流水线实验

本次实验为仿真实验，设计完成后仅需进行**行为仿真**。

实验要求：

1. 实现 4 级流水线 8bit 全加器，需带有流水线暂停和刷新；
2. 模拟流水线暂停，仿真时控制 10 周期后暂停流水线 2 周期（第 2 级），流水线恢复流动；
3. 模拟流水线刷新，仿真时控制 15 周期时流水线刷新（第 3 级）。

2 实验设计

2.1 ALU

2.1.1 功能描述

在 optional code 的控制下，输入的 8 位操作数 A 能在该算数逻辑单元中与内置操作数 B 进行加、减、与、或、非和 SLT 的运算。

2.1.2 接口定义

表 2: 接口定义模版

信号名	方向	位宽	功能描述
clk	Input	1-bit	时钟信号
reset	Input	1-bit	分时复用重置
num1	Input	8-bit	操作数 A
op	Input	3-bit	operation code 000 表示加法，001 表示减法，010 表示与，011 表示或，100 表示非，101 表示 ALT
seg	Output	7-bit	七段数码管选通
ans	Output	8-bit	八个数码管显示运算结果

2.1.3 逻辑控制

1. ALU 的设计采用 always 实现组合逻辑。通过 op 来控制 ALU 实现不同运算功能。其中 op 通过 sw[15:13] 控制，num1 通过 sw[7:0] 控制。
2. 输入八位 num1 后，会将该八位操作数无符号拓展为 32 位，再输入 ALU 进行运算。
3. 七段数码管使用时序逻辑方法，采取分时复用使运算结果显示到八个七段数码管上。

2.2 有阻塞 4 级 8bit 全加器

2.2.1 功能描述

实现 4 级流水线 32bit 全加器，每一级进行 8bit 加法运算，带有各级流水线暂停和刷新。

2.2.2 接口定义

表 3: 接口定义

信号名	方向	位宽	功能描述
clk	Input	1-bit	时钟信号
a	Input	32-bit	操作数 A
b	Input	32-bit	操作数 B
cin	Input	1-bit	该加法器初始进位信号
rst	Input	1-bit	全局重置信号
validin	Input	1-bit	一级流水线接受新操作数信号
out_allowin	Input	1-bit	四级流水线输出最终运算结果信号
pipeX_ready_go	Input	1-bit	描述第 X 级的状态，1 表示第 X 级的处理任务已经完成，可以传给 X+1 级
fresh_X	Input	1-bit	各级流水线刷新信号
sum1	Output	8-bit	一级流水线运算结果
sum2	Output	16-bit	二级流水线运算结果
sum3	Output	24-bit	三级流水线运算结果
sum4	Output	32-bit	四级流水线运算结果 (最终结果)
validout	Output	1-bit	1 代表最终输出结果有效
cout	Output	1-bit	该加法器是否产生进位

2.2.3 逻辑控制

1. 采用时序逻辑控制整个加法器。各级流水线有 rst 和 fresh 两个信号（高电平有效）用于各级流水线的重置和刷新。
2. 我们将 32 位相加分为四次进行，即每次相加 8 位，然后用寄存器存储每次相加后的结果、进位情况。其中 cout 来存储是否进位，sum1、sum2、sum3、sum4 存储每次累加后的结果，a/b_X 储存上一级参与运算的操作数。
3. pipeX_allowin = !pipeX_valid || pipeX_ready_go pipeX+1_allowin,pipe1 只有当 pipe1 的值赋给 pipe2 或者 pipe1 数值为无效的时候才允许被赋值。
4. 在一级流水线中，首先判断 rst 信号是否有效，有效则置零；其次判断是否允许被赋值，即 pipeX_allowin 是否有效，若 pipeX_allowin 有效且刷新信号无效，再去判断下一级流水线是否能够接受数据，若可以，则这一级流水线的工作才结束。

表 4: 逻辑控制

信号名	位宽	功能描述
carry_X	1-bit	各级流水线进位情况
pipeX_valid	1-bit	当前级是否存在有效的数据，高有效
pipeX_allowin	1-bit	第 X 级传给第 X-1 级的状态，是否可以接收上一级的数据
pipeX_to_pipeX+1_valid	1-bit	从第 X 级传递给第 X+1 级，1 表示下一时刻第 X 级有数据传递给第 X+1 级

3 实验过程记录

3.1 ALU

1. 使用组合逻辑实现 ALU，其中包括加、减、与、或、非、SLT 六种运算。在仿真和上板过程中均验证 ALU 设计功能正常。

3.2 阻塞流水线加法器

1. 实现了 4 级流水线 32bit 全加器，每一级进行 8bit 加法运算且带有流水线暂停和刷新。
2. 使用四个 always 语句块形成四级流水线
3. 第一级：做 [7:0] 位与进位位的加法操作，并将运算结果和操作数 AB 传给下一级。
4. 第二级：做 [15:8] 位与进位位的加法操作，并将本级运算结果和操作数 AB 传给下一级。
5. 第三级：做 [23:16] 位与进位位的加法操作，并将运算结果和操作数 AB 传给下一级。
6. 第四级：做 [31:24] 位与进位位的加法操作，并将结果组合输出。
7. 经仿真验证，如后图，所设计流水线全加器功能正常。

3.3 问题 1:Verilog 语法问题和 vivado 操作问题

问题描述：由于较长时间未使用 Verilog 语言和 vivado，导致语法和软件操作生疏，具体有无符号拓展时语法报错，仿真时出错。

解决方案：自行复习拼接操作符的使用规范，发现是忘了在最后加上大括号而报错；仿真时未将文件设为顶层而无法完成仿真操作。

3.4 问题 2: 流水线加法器仿真图不正确

问题描述：具体有两种情况，一是运算结果并不是在第四个周期后得出，而是该周期立即得出；二是各级流水线并未存储上一级流水线未参与计算的数据，导致每一级流水线操作数混乱，运算结果出错。

解决方案：第一种情况应该是没有流水线的效果，重新编写为带流水线的加法器；第二种情

况因为没有存储各级流水线未参与计算的数据，导致每一级流水线都是加上最新的操作数的各数位，重现添加寄存器存储上一级流水线参与运算的操作数即可。

4 实验结果及分析

4.1 ALU 验证实验结果

8 位 num1 输入后无符号拓展为 32 位

表 5: ALU 结果表

操作	Num1	Result
A + B(Unsigned)	8’b00000010	32’h00000003
A - B	8’b11111111	32’h000000FE
A AND B	8’b11111110	32’h00000000
A OR B	8’b10101010	32’h000000AB
$\overline{A}$	8’b11110000	32’hFFFFFF0F
SLT	8’b10000001	32’h00000000

4.2 流水线阻塞(暂停)仿真图

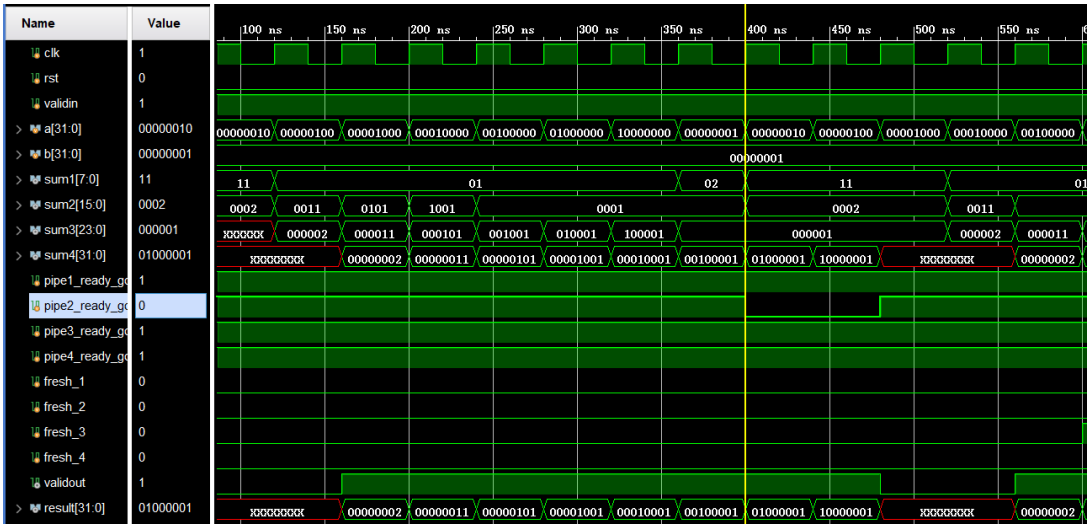

图 1: 暂停

4.3 结果分析

通过该仿真图，我们发现每一个周期，流水线将计算各部分的值，在第四个周期得到最终的运算结果。而第二级暂停时，我们发现第一级流水线也将暂停，保留在一级流水线中的数据不变。第三级，第四级流水线继续运转直到无数据推进，此时，validout 信号变为低电平。

4.4 流水线刷新(清空)仿真图

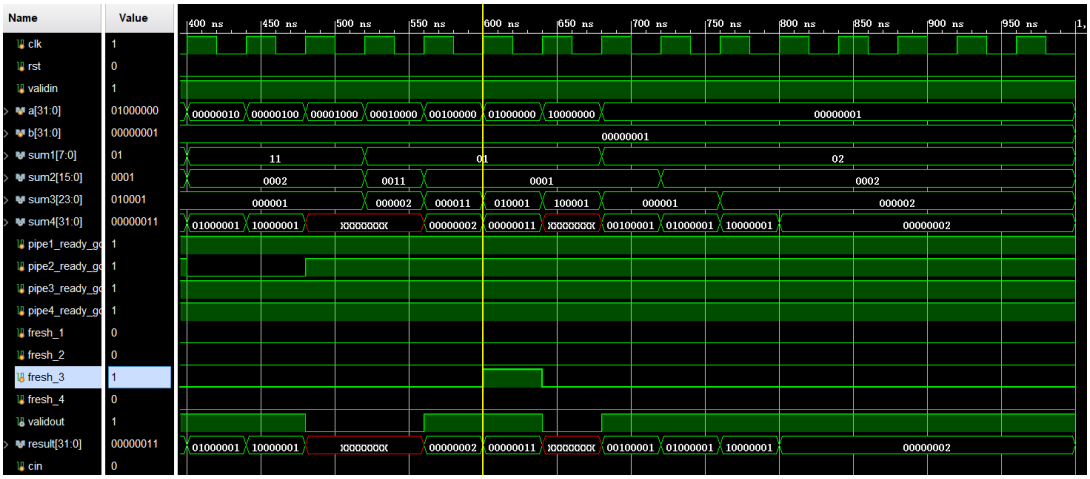

图 2: 刷新

4.5 结果分析

第三级刷新时，我们发现第一、二级流水线不受影响，但由于第三级流水线的刷新，导致下一周期第四级流水线无法得到数据，此时，validout 信号为低电平。

A ALU 代码

```
module ALU(num1,op,ans);
    input [7:0] num1; //operand A
    input [2:0] op;   //Operation code
    output reg [31:0] ans; //Operation result
    wire signed [31:0] num2 = 32'h01;  //num2 value is built-in 1
    wire signed [31:0] num1_32 = {{24{1'b0}},num1};  //unsigned extension to 32-bit
        operand A
    always@(*)
        begin
            case(op)
                3'b000  : ans = num1_32 + num2 ;
                3'b001  : ans = num1_32 - num2 ;
                3'b010  : ans = num1_32 & num2 ;
                3'b011  : ans = num1_32 | num2 ;
                3'b100  : ans = ~ num1_32 ;
                3'b101  : ans = num1_32 < num2;
                default : ans = 32 'h00000000 ;  //Default is 0
            endcase
         end
endmodule
```

B 8bit 全加器代码

```
    module stallable_pipeline_adder(
        input cin, clk, rst,
        input [31:0] a, b,
        input validin, out_allowin,
        input pipe1_ready_go, pipe2_ready_go, pipe3_ready_go, pipe4_ready_go,
        input fresh_1, fresh_2, fresh_3, fresh_4,
        output validout, cout,
        output [31:0] result,
        output reg [7:0] sum1,
        output reg [15:0] sum2,
        output reg [23:0] sum3,
        output reg [31:0] sum4
        );
    reg carry_1, carry_2, carry_3, carry_4;
    reg pipe1_valid, pipe2_valid, pipe3_valid, pipe4_valid;
    wire pipe1_allowin, pipe2_allowin, pipe3_allowin, pipe4_allowin;
    wire pipe1_to_pipe2_valid, pipe2_to_pipe3_valid, pipe3_to_pipe4_valid;
    reg [31:0] a2,b2,a3,b3,a4,b4;
    ////////////////////pipe 1/////////////////////
    assign cin = 0;
    assign pipe1_allowin = !pipe1_valid || pipe1_ready_go && pipe2_allowin;
    assign pipe1_to_pipe2_valid = pipe1_valid && pipe1_ready_go;
```

```
always@(posedge clk)
    begin
        if(rst) pipe1_valid <= 1'b0; //reset pipeX_valid >> invalid
        else if(pipe1_allowin) pipe1_valid <= (fresh_1 == 1)? 0 : validin;
        if(validin && pipe1_allowin) begin
            {carry_1, sum1} <= {1'b0, a[7:0]} + {1'b0, b[7:0]} + cin;
            a2 <= a;
            b2 <= b;
        end
    end
//////////////////pipe 2//////////////////
assign pipe2_allowin = !pipe2_valid || pipe2_ready_go && pipe3_allowin;
assign pipe2_to_pipe3_valid = pipe2_valid && pipe2_ready_go;
always@(posedge clk)
    begin
        if(rst) pipe2_valid <= 1'b0;
        else if(pipe2_allowin) pipe2_valid <= (fresh_2 == 1)? 0 :
            pipe1_to_pipe2_valid;
        if(pipe1_to_pipe2_valid && pipe2_allowin) begin
            {carry_2, sum2} <= {{1'b0, a2[15:8]} + {1'b0, b2[15:8]} + carry_1,
                sum1};
            b3 <= b2;
            a3 <= a2;
        end
    end
//////////////////pipe 3//////////////////
assign pipe3_allowin = !pipe3_valid || pipe3_ready_go && pipe4_allowin;
assign pipe3_to_pipe4_valid = pipe3_ready_go && pipe3_valid;
always@(posedge clk)
    begin
        if(rst) pipe3_valid <= 1'b0;
        else if(pipe3_allowin) pipe3_valid <= (fresh_3 == 1)? 0 :
            pipe2_to_pipe3_valid;
        if(pipe2_to_pipe3_valid && pipe3_allowin) begin
            {carry_3, sum3} <= {{1'b0, a3[23:16]} + {1'b0, b3[23:16]} + carry_2
                , sum2};
            a4 <= a3;
            b4 <= b3;
        end
    end
//////////////////pipe 4//////////////////
assign pipe4_allowin = !pipe4_valid || pipe4_ready_go && out_allowin;
always@(posedge clk)
    begin
        if(rst) pipe4_valid <= 1'b0;
        else if(pipe4_allowin)  pipe4_valid <= (fresh_4 == 1)? 0 :
            pipe3_to_pipe4_valid;
        if(pipe3_to_pipe4_valid && pipe4_allowin && pipe4_ready_go)
```

```
            {carry_4, sum4} <= {{1'b0, a4[31:24]} + {1'b0, b4[31:24]} + carry_3
                , sum3};
        else
            {carry_4, sum4} = 32'hxxxxxxxx;
    end
assign validout = (pipe4_valid && pipe4_ready_go);
assign cout = carry_4;
assign result = sum4;
```