

《计算机组成原理》实验报告

年级、专业、班级	2023 级计算机科学与技术 06 班 2023 级 2023 级	姓名	饶格奇
实验题目	实验一简单流水线与运算器实验		
实验时间	2025 年 4 月 13 日	实验地点	DS1410
实验成绩	优秀/良好/中等	实验性质	<input type="checkbox"/> 验证性 <input checked="" type="checkbox"/> 设计性 <input type="checkbox"/> 综合性
教师评价： <input checked="" type="checkbox"/> 算法/实验过程正确； <input checked="" type="checkbox"/> 源程序/实验内容提交； <input checked="" type="checkbox"/> 程序结构/实验步骤合理； <input checked="" type="checkbox"/> 实验结果正确； <input checked="" type="checkbox"/> 语法、语义正确； <input checked="" type="checkbox"/> 报告规范； 其他： <div>评价教师：任骛</div>			
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；			

表 1: 实验报告封面

报告完成时间: 2025 年 4 月 20 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 2: 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. 仿真时 B 端口输入为 32h'01, A 端口输入参照 4.1 中表格
3. 实现 SLT 功能。
4. 验证表2中所有功能。
5. 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

该 ALU 实现加法、减法、按位与、按位或、取反及无符号小于判断等基础运算功能，支持 8 位与 32 位输入，输出 32 位结果，并在数码管上进行显示。

2.1.2 接口定义

表 3: 接口定义

信号名	方向	位宽	功能描述
num1	Input	8-bit	操作数 1
num2	Input	32-bits	操作数 2
op	Output	3-bits	操作码
result	Output	32-bits	计算结果

2.1.3 逻辑控制

本 ALU 模块采用组合逻辑控制方式，不使用时序触发的有限状态机(FSM)。由于 ALU 运算为即时反应型逻辑(无时序依赖)，其控制流程可简化为单周期组合逻辑响应，无需状态跳转。模块通过将 8 位输入零扩展为 32 位，并根据 3 位操作码执行加、减、与、或、取反和小于判断等运算，非法操作码输出全 x。ALU 的控制逻辑使用组合逻辑电路实现，具体如下：

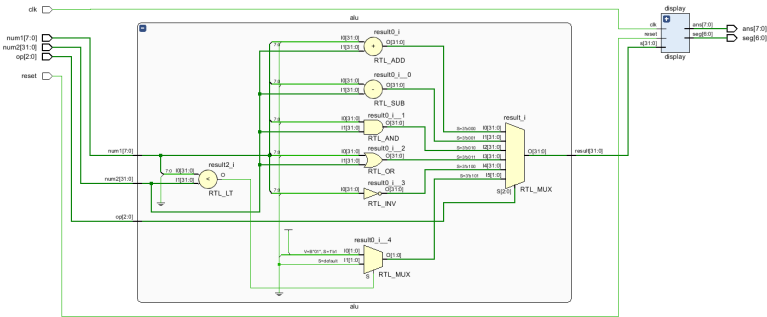


图 1: ALU 控制逻辑电路

2.2 有阻塞 4 级 8bit 全加器

2.2.1 功能描述

本模块 `adder` 是一个采用四级流水线结构实现的 32 位加法器。模块接收两个 32 位输入操作数 `num1` 和 `num2`, 通过每阶段处理 8 位数据逐步完成加法, 并支持输入初始进位 `cin`。模块支持异步复位 `reset`, 每级流水线的刷新控制 `refresh` 以及暂停控制 `stop`, 并提供各阶段调试输出与流水线状态指示。最终输出结果为 32 位加法结果 `ans` 和最终进位 `cout`。

2.2.2 接口定义

见下表

2.2.3 逻辑控制

本模块采用四级流水线结构, 逐步完成 32 位加法操作, 每级处理 8 位数据, 并传递进位信息, 详细逻辑如下:

- **流水线允许控制:** 每一级通过 `allowin` 信号判断是否能接收数据, 条件为本级未被 `stop[i]` 阻塞, 且下一级允许传输数据, 或本级当前无效。
- **数据传递与寄存:** 每一阶段使用寄存器保存当前级别的部分和与操作数剩余高位, 下一阶段继续处理更高位加法并拼接形成完整结果。
- **进位传递逻辑:** 初始进位 `cin` 输入到第一阶段, 后续每级进位通过前一级的进位累加传递。
- **复位与刷新机制:** 模块支持复位信号 `reset`, 用于清除所有阶段的 `valid` 状态。单独阶段可通过对应的 `refresh[i]` 控制信号进行刷新清除。
- **输出控制:** 最终输出由第四级流水线完成, 其输出受 `out_allow` 控制。若该阶段有效且输出允许, 则输出 `ans` 和 `cout`, 同时置 `validout` 为高。
- **调试输出:** 模块提供每一级流水的调试输出 `debug_sum*`, 用于观察中间和的变化情况, 辅助调试与验证。

表 4: adder 模块接口定义

信号名	方向	位宽	功能描述
clk	Input	1	时钟信号, 驱动流水线各阶段时序逻辑
reset	Input	1	异步复位信号, 高电平有效, 清除所有状态
refresh	Input	4	各流水线级刷新信号, 高电平有效清除对应阶段状态
stop	Input	4	各流水线级暂停信号, 高电平阻止流水前传
cin	Input	1	初始加法进位
validin	Input	1	输入数据有效标志
out_allow	Input	1	输出允许信号, 决定是否将结果输出
num1	Input	32	加法器第一个操作数
num2	Input	32	加法器第二个操作数
validout	Output	1	输出数据有效标志, 表示结果已准备好
ans	Output	32	最终加法结果
cout	Output	1	最终进位输出
debug_sum1	Output	8	第一阶段的部分加法结果 (用于 Debug)
debug_sum2	Output	16	第二阶段的部分加法结果 (用于 Debug)
debug_sum3	Output	24	第三阶段的部分加法结果 (用于 Debug)
debug_sum4	Output	32	第四阶段 (最终) 的加法结果 (用于 Debug)
pipe1_valid_out	Output	1	第一阶段有效状态指示
pipe2_valid_out	Output	1	第二阶段有效状态指示
pipe3_valid_out	Output	1	第三阶段有效状态指示
pipe4_valid_out	Output	1	第四阶段有效状态指示

3 实验过程记录

3.1 问题 1:ALU 计算错误

问题描述:误以为运算会自动零扩展,未对 num1 作位数扩展,与 num2 相加时,结果错误。

解决方案:对 num1 进行位数扩展,使用 Verilog 的零扩展语句,将 8 位数扩展为 32 位:
`assign num1_ext = {24'b0, num1};`

3.2 问题 2:ALU 实现各指令码所对应的功能

问题描述:实现过程中,需要根据操作码(op)执行不同的运算逻辑,确保 ALU 能正确完成加法、减法、位运算和比较等功能。

解决方案:使用 Verilog 中的 case 语句对操作码进行译码,根据不同的值执行相应的算术或逻辑运算,例如加法、减法、按位与、按位或、取反和无符号小于比较。通过组合逻辑块 always @(*) 实现功能控制。

3.3 问题 3: 验证 ALU 各功能正确性

问题描述:需要验证 ALU 模块是否按照设计正确执行加法、减法、与、或、取反、小于判断等功能。

解决方案:编写 Testbench 仿真文件,采用 initial 块在每个时钟上升沿提供输入数据,包括 8 位 num1、32 位 num2 和 3 位操作码 op,通过 \$display 打印输出结果,在控制台查看各功能的执行情况并与预期对比验证正确性。

3.4 问题 4: 流水线加法器仿真图难以 Debug

问题描述:

```
module adder(  
    input clk,reset,  
    input [3:0]refresh,  
    input [3:0]stop,  
    input cin,           // 初始进位  
    input validin,       // 初始值有效  
    input out_allow,     // 允许输出  
    input [31:0] num1,  
    input [31:0] num2,  
  
    output validout,  
    output [31:0] ans,  
    output cout,
```

```
);
```

只使用上方代码中的接口难以看清流水线加法器内部结构,难以 Debug 并且难以判断功能是否正常。**解决方案:**在上方代码中添加如下代码,添加了流水线加法器内部结构,便于 Debug。

```
module adder(  
    input clk,reset,  
    input [3:0]refresh,  
    input [3:0]stop,  
    input cin,          // 初始进位  
    input validin,      // 初始值有效  
    input out_allow,    // 允许输出  
    input [31:0] num1,  
    input [31:0] num2,  
  
    output validout,  
    output [31:0] ans,  
    output cout,  
  
    output debug_cout1,  
    output debug_cout2,  
    output debug_cout3,  
    output debug_cout4,  
  
    output [7:0] debug_sum1,  
    output [15:0] debug_sum2,  
    output [23:0] debug_sum3,  
    output [31:0] debug_sum4,  
    output pipe1_valid_out,  
    output pipe2_valid_out,  
    output pipe3_valid_out,  
    output pipe4_valid_out  
);
```

3.5 问题 5: 仿真结果不正确

问题描述:由仿真结果观察,仿真结果不正确。无法做到暂停功能并且 ans 没有正确结果。

解决方案:仔细观察代码后发现,将部分变量的变量名写错,导致仿真结果不正确。将变量名改正后,仿真结果正确。

4 实验结果及分析

4.1 ALU 验证实验结果

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

表 5: ALU 结果表

4.2 ALU 仿真

4.3 流水线阻塞(暂停)仿真图



图 2: 流水线加法器暂停

分析: 115ns 时, 时钟上升沿, 流水线加法器 stop 信号为 4'b0011, 表示一二级流水线暂停。由图可知正常的结果延迟 4 周期再出现。而暂停后等待 6 周期才出现, 表示成功暂停。并且此时 0x00000016 只在第一次出现的周期有效, 后两个周期 validout 为 0, 表示此结果无效。

4.4 流水线刷新(清空)仿真图

分析: 165ns 时, 时钟上升沿, 流水线加法器 refresh 信号为 4'b0100, 表示第三级流水线清空。由图可知, pipe3_valid_out 为 0, 表示第三级流水线结果无效。并且本应该是 0x0000001c 的周期, 结果为 0x0000001b, validout 信号为 0, 表示结果无效。由此可知成功刷新。



图 3: 流水线加法器清空

A ALU 代码

```
'timescale 1ns / 1ps

module alu(
    input wire [2:0] op,           // 操作码
    input wire [7:0] num1,        // 8位输入
    input wire [31:0] num2,       // 32位输入
    output reg [31:0] result      // 运算结果
);

    wire [31:0] num1_ext;
    assign num1_ext = {24'b0, num1}; // 8位零扩展到32位

    always @(*) begin
        case (op)
            3'b000: result = num1_ext + num2;
            3'b001: result = num1_ext - num2;
            3'b010: result = num1_ext & num2;
            3'b011: result = num1_ext | num2;
            3'b100: result = ~num1_ext;
            3'b101: result = (num1_ext < num2) ? 32'b1 : 32'b0;
            default: result = 32'hxxxxxxxx;
        endcase
    end
endmodule
```

B 32bit 流水线全加器代码

```
module adder(
    input clk, reset,
    input [3:0] refresh,
    input [3:0] stop,
    input cin,           // 初始进位
    input validin,       // 初始值有效
    input out_allow,     // 允许输出
    input [31:0] num1,
    input [31:0] num2,

    output validout,
    output [31:0] ans,
    output cout,

    output debug_cout1,
    output debug_cout2,
    output debug_cout3,
```

```

output debug_cout4,

output [7:0] debug_sum1,
output [15:0] debug_sum2,
output [23:0] debug_sum3,
output [31:0] debug_sum4,
output pipe1_valid_out,
output pipe2_valid_out,
output pipe3_valid_out,
output pipe4_valid_out

    );

reg [7:0] temp_sum1;
reg pipe1_valid;
reg [15:0] temp_sum2;
reg pipe2_valid;
reg [23:0] temp_sum3;
reg pipe3_valid;
reg [31:0] temp_sum4;
reg pipe4_valid;

reg cout1,cout2,cout3,cout4;

reg [23:0] temp_a_1,temp_b_1;      //1,2 stage
reg [15:0] temp_a_2,temp_b_2;      //2,3 stage
reg [7:0] temp_a_3,temp_b_3;      //3,4 stage

//pipe1_stage

wire pipe1_allowin;
wire pipe1_ready;
wire pipe1_to_pipe2_valid;

assign pipe1_ready=!stop[0];
assign pipe1_allowin=! pipe1_valid || pipe1_ready && pipe2_allowin;
assign pipe1_to_pipe2_valid=pipe1_valid && pipe1_ready;

always @(posedge clk ) begin
    if (reset) begin
        pipe1_valid<=1'b0;
    end
    else if(refresh[0]) begin
        pipe1_valid<=1'b0;
    end
    else if(pipe1_allowin) begin
        pipe1_valid<=validin;
    end
end

```

```

        end
        if(pipe1_allowin&&validin) begin
            {cout1,temp_sum1} <={1'b0,num1[7:0]}+{1'b0,num2[7:0]}+cin;
            temp_a_1<=num1[31:8];
            temp_b_1<=num2[31:8];
        end
    end

//pipe2_stage

wire pipe2_allowin;
wire pipe2_ready;
wire pipe2_to_pipe3_valid;

assign pipe2_ready=!stop[1];
assign pipe2_allowin=! pipe2_valid || pipe2_ready && pipe3_allowin;
assign pipe2_to_pipe3_valid=pipe2_valid && pipe2_ready;

always @(posedge clk) begin
    if (reset) begin
        pipe2_valid<=1'b0;
    end
    else if(refresh[1]) begin
        pipe2_valid<=1'b0;
    end
    else if(pipe2_allowin) begin
        pipe2_valid<=pipe1_to_pipe2_valid;
    end
    if(pipe2_allowin&&pipe1_to_pipe2_valid) begin
        {cout2,temp_sum2}<={cout1+{1'b0,temp_a_1[7:0]}+{1'b0,temp_b_1
            [7:0]},temp_sum1};
        temp_a_2<=temp_a_1[23:8];
        temp_b_2<=temp_b_1[23:8];
    end
end

//pipe3_stage

wire pipe3_allowin;
wire pipe3_ready;
wire pipe3_to_pipe4_valid;

assign pipe3_ready=!stop[2];
assign pipe3_allowin=! pipe3_valid || pipe3_ready && pipe4_allowin;
assign pipe3_to_pipe4_valid=pipe3_valid && pipe3_ready;

always @(posedge clk) begin
    if (reset) begin

```

```

        pipe3_valid<=1'b0;
    end
    else if(refresh[2]) begin
        pipe3_valid<=1'b0;
    end
    else if(pipe3_allowin) begin
        pipe3_valid<=pipe2_to_pipe3_valid;
    end
    if(pipe3_allowin&&pipe2_to_pipe3_valid) begin
        {cout3,temp_sum3}<={cout2+{1'b0,temp_a_2[7:0]}+{1'b0,temp_b_2
            [7:0]},temp_sum2};
        temp_a_3<=temp_a_2[15:8];
        temp_b_3<=temp_b_2[15:8];
    end
end

//pipe4_stage

wire pipe4_allowin;
wire pipe4_ready;

assign pipe4_ready=!stop[3];
assign pipe4_allowin=! pipe4_valid || pipe4_ready && out_allow;

always @(posedge clk) begin
    if (reset) begin
        pipe4_valid<=1'b0;
    end
    else if(refresh[3]) begin
        pipe4_valid<=1'b0;
    end
    else if(pipe4_allowin) begin
        pipe4_valid<=pipe3_to_pipe4_valid;
    end
    if(pipe4_allowin&&pipe3_to_pipe4_valid) begin
        {cout4,temp_sum4}<={cout3+{1'b0,temp_a_3}+{1'b0,temp_b_3},temp_sum3
            };
    end
end

assign debug_sum1 = temp_sum1;
assign debug_sum2 = temp_sum2;
assign debug_sum3 = temp_sum3;
assign debug_sum4 = temp_sum4;

assign pipe1_valid_out = pipe1_valid;
assign pipe2_valid_out = pipe2_valid;
assign pipe3_valid_out = pipe3_valid;
assign pipe4_valid_out = pipe4_valid;

```

```
assign validout=pipe4_ready&&pipe4_valid;  
assign ans=temp_sum4;  
assign cout=cout4;  
endmodule
```