**实验报告: Lab3 - 算术逻辑部件实验**

**一、实验目的**

* 本实验的主要目的是帮助学生掌握先行进位加法器、桶形移位器和 ALU（算术逻辑单元）的设计和应用。具体目标包括：
* 理解先行进位加法器的工作原理，并能够设计和实现 32 位加法器。
* 掌握桶形移位器的结构和功能，并将其扩展为 32 位桶形移位器。
* 理解 ALU 的操作控制信号编码方案，并能够设计 32 位 ALU。

**二、实验环境**

* 软件: Vivado 2021.1
* 硬件: Xilinx Artix-7 FPGA 开发板

**三、实验原理**

1. **串行进位加法器（SCLA）原理**

串行进位加法器（SCLA） 是一种经典的加法器，它以串行的方式计算加法操作。其主要原理如下：

SCLA 处理位的顺序是从最低位（最右侧）到最高位（最左侧）。

对于每一位，SCLA 计算局部和 P = A ⊕ B ⊕ C\_in，并传递到下一位的 C\_in。

进位 C\_in 从最低位开始是零，但会随着位的处理逐渐传递。

C\_in 是下一位的局部和 P 的进位，从而完成对下一位的计算。

1. **先行进位加法器（CLA）原理**

先行进位加法器（CLA）是一种高效的加法器，能够在一个时钟周期内执行加法操作。其原理如下：

每一位的和 S 取决于输入 A、B 和上一位的进位 C\_in。

先计算局部和 P = A ⊕ B ⊕ C\_in 和进位 G = (A ∧ B) ∨ ((A ⊕ B) ∧ C\_in)。

利用 P 和 G，计算本位的和 S = P ⊕ C\_in 和下一位的进位 C\_out = G ∨ (P ∧ C\_in)。

1. **32 位 ALU 原理**

\*\*ALU（算术逻辑单元）\*\*执行不同的算术和逻辑运算，具体操作通过 4 位控制信号 ALUctr 来确定。控制信号的编码如下：

SUBctr：减法操作控制。当 SUBctr=1 时，执行减法运算；当 SUBctr=0 时，执行加法运算。

SIGctr：带符号/无符号比较控制。当 SIGctr=1 时，执行“带符号整数比较小于置 1”；当 SIGctr=0 时，执行“无符号数比较小于置 1”。

ALctr：算术/逻辑移位控制。当 ALctr=0 时，执行逻辑移位；当 ALctr=1 时，执行算术移位。

OPctr[2:0]：运算结果的选择，有加法器和、按位或、按位与、按位异或、移位器输出、操作数 B 输出、小于置 1 等 7 种运算结果。

**四、实验内容**

**1. 带标志位的加减运算部件**

* 在这一部分，我们将实现一个带标志位的 32 位加减法器。具体内容包括：
* 实现带符号整数的加法和减法。
* 生成溢出标志 OF、符号位标志 SF、进位/借位标志 CF 和结果为 0 标志 ZF。
* 设计 Adder32 模块，模块端口包括输入、输出和控制信号。

代码实现：

module Adder32(

output [31:0] f,

output OF, SF, ZF, CF,

output cout,

input [31:0] x, y,

input sub

);

wire carry;

wire [31:0] out\_y;

assign out\_y = sub ? ~y : y ;

CLA\_16 cla16\_0(f[15:0], carry, x[15:0], out\_y[15:0], sub);

CLA\_16 cla16\_1(f[31:16], cout, x[31:16], out\_y[31:16], carry);

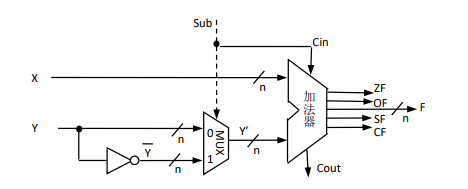
assign ZF = (f == 32'd0);

assign SF = f[31];

assign CF = sub ^ cout;

assign OF = (~x[31] & ~y[31] & f[31]) | (x[31] & y[31] & ~f[31]);

endmodule



仿真测试：

图形用户界面

描述已自动生成

**2. 桶形移位器**

* 在这一部分，我们将实现一个 32 位桶形移位器，可以进行左移和右移操作。具体内容包括：
* 了解 8 位桶形移位器的结构和原理。
* 扩展 8 位桶形移位器为 32 位版本。
* 实现 barrelsft32 模块，模块端口包括输入、输出和控制信号。

代码实现：

module barrelsft32(

    output [31:0] dout,

    input [31:0] din,

    input [4:0] shamt,     //移动位数

    input LR,           // LR=1时左移，LR=0时右移

    input AL      // AL=1时算术右移，AR=0时逻辑右移

    );

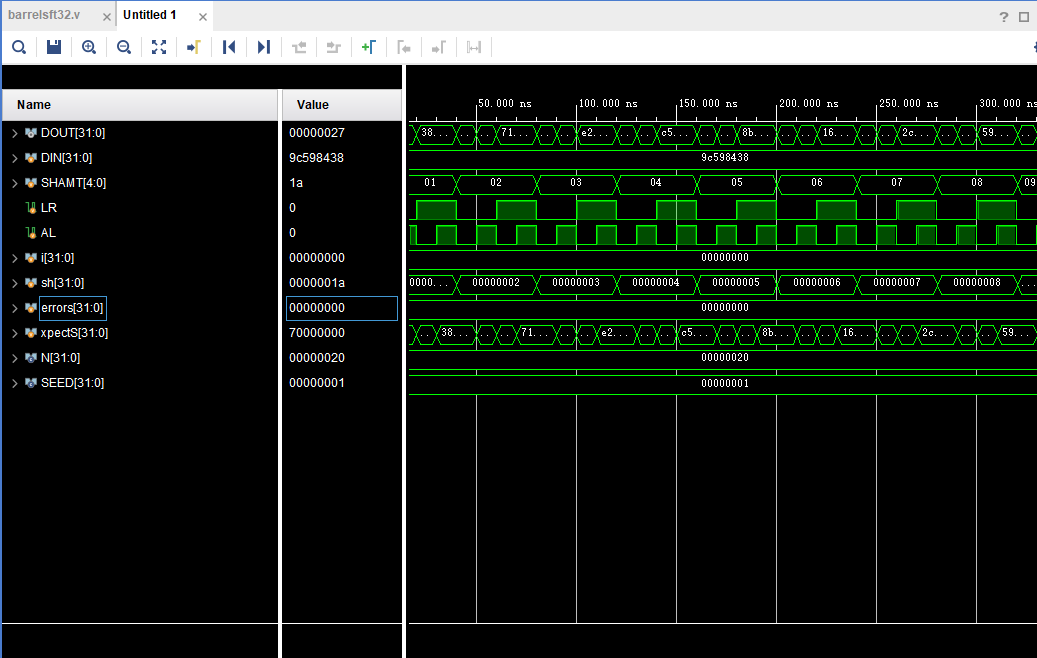
    wire [31:0] ALR;

    assign ALR = din[31] ? (din >> shamt) | (32'hFFFFFFFF << (32 - shamt)) : din >> shamt;

    assign dout = LR ? din << shamt : (AL ? ALR : din >> shamt);

endmodule

仿真测试：



**3. 32 位 ALU**

* 在这一部分，我们将设计一个 32 位 ALU，可以执行各种算术和逻辑运算。具体内容包括：
* 理解 ALU 的操作控制信号编码。
* 实现 ALU32 模块，模块端口包括输入、输出和控制信号。
* 创建 ALU32\_top 父级模块，将 ALU32 模块实例化并连接输入开关和输出设备。
* 在实验开发板上验证 ALU32 的功能和正确性。

代码实现：

32位加法器：

module ALU32(

output [31:0] result, //32位运算结果

output zero, //结果为0标志位

input [31:0] dataa, //32位数据输入，送到ALU端口A

input [31:0] datab, //32位数据输入，送到ALU端口B

input [3:0] aluctr //4位ALU操作控制信号

);

reg [31:0] \_result;

assign result = \_result;

reg sub;

wire OF, SF, ZF, CF;

wire [31:0] real\_y;

assign real\_y = sub ? ~datab : datab;

wire real\_OF;

assign real\_OF = (~dataa[31] & ~real\_y[31] & adder\_result[31]) | (dataa[31] & real\_y[31] & ~adder\_result[31]);

reg LR, AL, SIG;

wire [31:0] adder\_result;

wire [31:0] sft\_result;

wire [31:0] cmp\_result;

assign cmp\_result = {31'b0, (SIG ? (real\_OF ^ SF) : CF)};

wire cout;

Adder32 my\_adder(adder\_result, OF, SF, ZF, CF, cout, dataa, datab, sub);

barrelsft32 my\_barrel(sft\_result, dataa, datab[4:0], LR, AL);

always @(\*) begin

case(aluctr)

4'b0000: begin //and

sub = 0;

\_result = adder\_result;

end

4'b0001: begin //left shift

LR = 1; AL = 0;

\_result = sft\_result;

end

4'b0010: begin //signed cmp

sub = 1; SIG = 1;

\_result = cmp\_result;

end

4'b0011: begin //unsigned cmp

sub = 1; SIG = 0;

\_result = cmp\_result;

end

4'b0100: begin //xor

\_result = dataa ^ datab;

end

4'b0101: begin //right shift

LR = 0; AL = 0;

\_result = sft\_result;

end

4'b0110: begin

\_result = dataa | datab;

end

4'b0111: begin

\_result = dataa & datab;

end

4'b1000: begin //sub

sub = 1;

\_result = adder\_result;

end

4'b1101: begin //al right shift

LR = 0; AL = 1;

\_result = sft\_result;

end

4'b1111: begin //load imm(datab)

\_result = datab;

end

default: begin

\_result = 32'd0;

end

endcase

end

endmodule

顶层代码：

module ALU32\_top(

output [6:0] segs, //七段数码管字形输出

output [7:0] AN, //七段数码管显示32位运算结果

output [15:0] result\_l, //32位运算结果

output zero, //结果为0标志位

input [3:0] data\_a, //4位数据输入，重复8次后送到ALU端口A

input [3:0] data\_b, //4位数据输入，重复8次后送到ALU端口B

input [3:0] aluctr, //4位ALU操作控制信号

input clk

);

reg [3:0] dis\_cur;

reg [3:0] dis\_pos;

dec7seg led\_driver(segs, AN, dis\_cur, dis\_pos);

wire [31:0] result\_32;

assign result\_l = result\_32[15:0];

wire [31:0] dataa;

wire [31:0] datab;

assign dataa = {8{data\_a}};

assign datab = {8{data\_b}};

ALU32 alu(result\_32, zero, dataa, datab, aluctr);

//display buffer

wire [3:0] display\_buffer [0:7];

assign display\_buffer[0] = result\_32[3:0];

assign display\_buffer[1] = result\_32[7:4];

assign display\_buffer[2] = result\_32[11:8];

assign display\_buffer[3] = result\_32[15:12];

assign display\_buffer[4] = result\_32[19:16];

assign display\_buffer[5] = result\_32[23:20];

assign display\_buffer[6] = result\_32[27:24];

assign display\_buffer[7] = result\_32[31:28];

reg [15:0] trans;

reg [3:0] dis\_cnt;

//Display driving loop

always @(posedge clk) begin

//Transfer clk signal to acceptable fresh rate.

if(trans >= 16'd50000)

trans <= 0;

else

trans <= trans + 1;

if(trans == 0) begin

if(dis\_cnt >= 7)

dis\_cnt <= 0;

else

dis\_cnt <= dis\_cnt + 1;

end

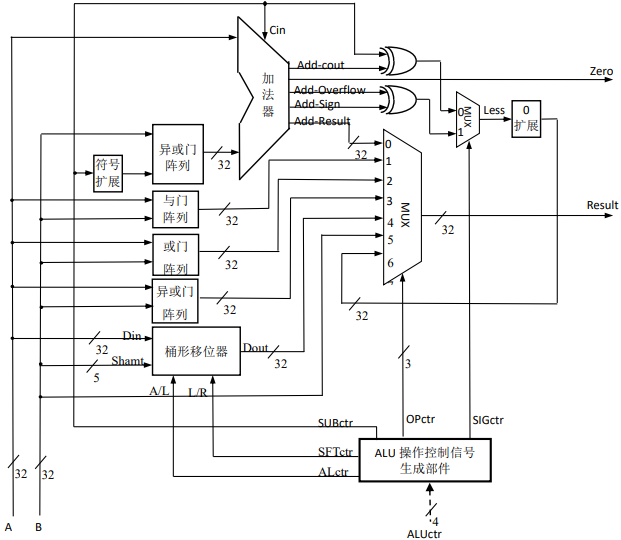
//Display

dis\_pos <= dis\_cnt;

dis\_cur <= display\_buffer[dis\_cnt];

end

endmodule



仿真测试：  
电脑屏幕截图

描述已自动生成

**五、实验步骤**

* 以下是实验的具体步骤：

1. 使用 Vivado 创建一个新工程。
2. 添加设计源码文件，包括 Adder32.v、barrelsft32.v 和 ALU32.v。
3. 添加仿真测试文件，包括 Adder32\_tb.v、barrelsft32\_tb.v 和 ALU32\_tb.v。
4. 添加约束文件，如 ALU32\_top.xdc，以确保正确的时序约束。
5. 根据实验要求，在设计源码文件中完成模块的设计和实例化。
6. 进行仿真测试，分析输入输出时序波形和控制台信息。
7. 一旦仿真通过，进行综合、实现并生成比特流文件。
8. 将比特流文件加载到实验开发板上，进行调试验证。

**六、实验验证**

在实验验证部分，我们会描述实验验证的过程，包括在实验开发板上加载比特流文件并记录验证结果。

下载到开发板进行验证：

初始情况

图片包含 游戏机, 电路

描述已自动生成

加法模式

0x22222222 + 0x55555555

图片包含 室内, 桌子, 游戏机, 钟表

描述已自动生成

0x33333333 + 0x88888888

图片包含 物体, 游戏机, 钟表

描述已自动生成

减法模式

0x55555555 – 0x22222222

图片包含 游戏机, 电路, 钟表

描述已自动生成

0x22222222 – 0x55555555

图片包含 游戏机, 桌子, 钟表

描述已自动生成

算术右移

0x11111111 >> 0x11111111

图片包含 钟表, 桌子

描述已自动生成

0x99999999 >> 0x11111111

图片包含 游戏机, 钟表

描述已自动生成

或运算

0x33333333 | 0xaaaaaaaa

图片包含 游戏机, 电路

描述已自动生成

带符号数小于比较

0x33333333 < 0x44444444

图片包含 游戏机, 电路

描述已自动生成

0x88888888 < 0x00000000

黑色的游戏机

低可信度描述已自动生成

**四、思考题**

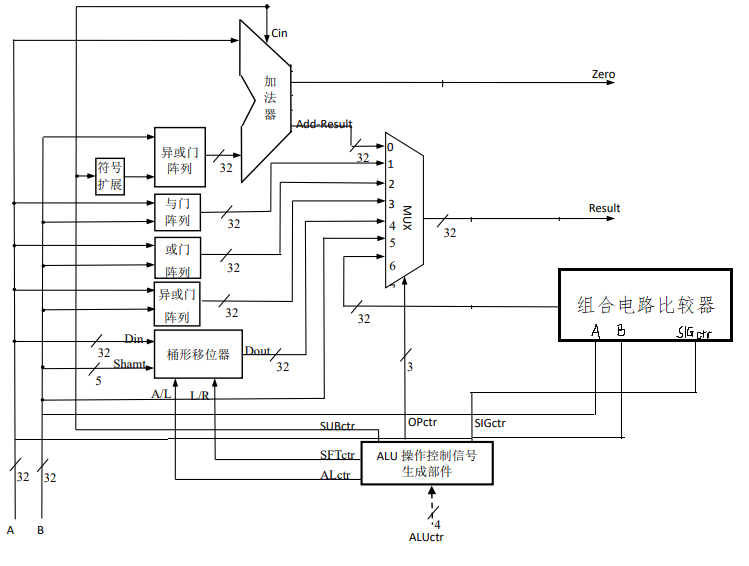
**1、分析 32 位 ALU 的资源占用情况。** 如图：

图形用户界面

低可信度描述已自动生成

图表

描述已自动生成

**2、如果比较运算直接使用组合电路比较器来实现，则 32 位 ALU 电路原理图需要做哪些修改？** 

**3、在 32 位 ALU 的基础上如何实现 64 位的 ALU？**

扩展输入和输出：首先，需要扩展ALU的输入和输出以支持64位操作。这包括将ALU的输入操作数A和B扩展为64位，并将ALU的结果扩展为64位。

扩展操作码：需要修改ALU的操作码以支持新的64位操作。这可能需要增加更多的操作码位，以便支持新的操作，如64位加法、减法、逻辑运算等。

修改运算单元：ALU的运算单元需要进行修改，以支持64位操作。这包括增加更多的运算器件，如64位加法器、64位减法器和其他64位运算电路。

处理溢出：在64位操作中，溢出情况变得更加复杂。需要添加溢出检测和处理电路，以确保正确处理溢出情况。

逻辑和控制单元：修改ALU的控制单元以支持64位操作。这可能需要更复杂的逻辑电路，以确保正确的操作选择和结果生成。

**4、查找资料说明还有哪些并行加法器的设计方法，并详细介绍其中一种方法。**

Kogge-Stone加法器是一种并行加法器，旨在加速大位宽加法操作。它的主要特点是并行性强，能够同时处理多个位的相加，从而减少加法操作的延迟。这种加法器的设计基于一种树形结构，其中位的传递是并行的，而不是逐位进行。