# 模块实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 叶福伦 | 学号 | 21051331 |
| 专业 | 计算机科学与技术 | 班级 | 21052313 |
| 指导教师 | 髙志刚 | 课程名称 | 计算机组成原理课程设计（甲） |
| 实验序号 | 03 | 实验名称 | 多功能ALU设计实验 |
| 实验时间 | 05.05 | 实验地点 | 1教223 |

# 成绩表

|  |  |  |
| --- | --- | --- |
| **考核项目** | **考查分项** | **得分** |
| **一、实验方案设计（30分）** | **实验目的与要求（5分）** |  |
| **模块构成与连接（15分）** |  |
| **板级验证方案（10分）** |  |
| **二、FPGA程序设计（20分）** | **程序源代码（15分）** |  |
| **管脚约束程序源代码（5分）** |  |
| **三、仿真实验（15分）** | **仿真代码（5分）** |  |
| **仿真波形及结果分析（10分）** |  |
| **四、板级实验（15分）** | **实验操作（5分）** |  |
| **结果记录（5分）** |  |
| **结果分析（5分）** |  |
| **五、思考与探索（10分）** | **拓展提升（10分）** |  |
| **格式（10分）** | **书写规范、图表清晰、表达清楚** |  |
| **合计** | |  |

# 一、实验方案设计

1. 实验目的与要求（5分）

学习多功能ALU的工作原理，掌握多功能ALU的设计方法；

掌握暂存器的设计方法，及其与ALU的连接方法；

掌握运用Verilog HDL语言进行组合逻辑电路与时序逻辑电路混合设计的方法；

学会输入输出设备不足时的处理方法。

使用行为描述方式建模多功能ALU模块，并添加标志位产生电路FlagU。仿真验证其功能

设计一个暂存器模块，包含A、B、F和标志寄存器FR的寄存电路及控制逻辑（这里也可设计为各自独立的4个模块）

编写一个带暂存器的多功能ALU模块，将多功能ALU模块与暂存器模块连接在一起。

针对使用的实验板卡，设计带暂存器的多功能ALU模块的板级验证实验方案，编写顶层测试模块；

选择典型的输入数据，验证你的ALU的各种功能，将实验结果记录到表格，要求所有运算功能和标志位都被有效测试

撰写实验报告，格式见附录，重点内容包括：

对仿真结果进行分析；

描述你设计的板级验证实验方案、模块结构与连接；

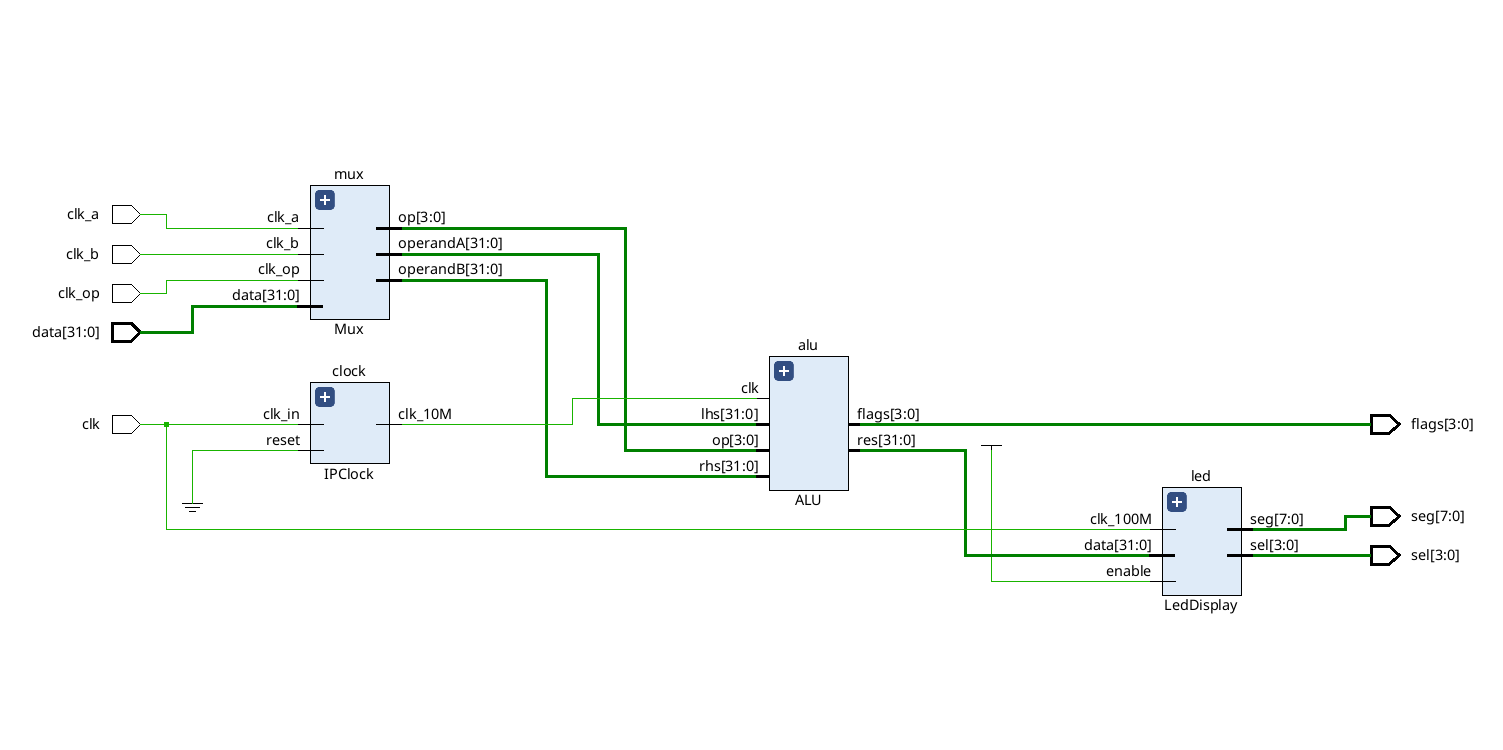
说明你的板级操作过程；

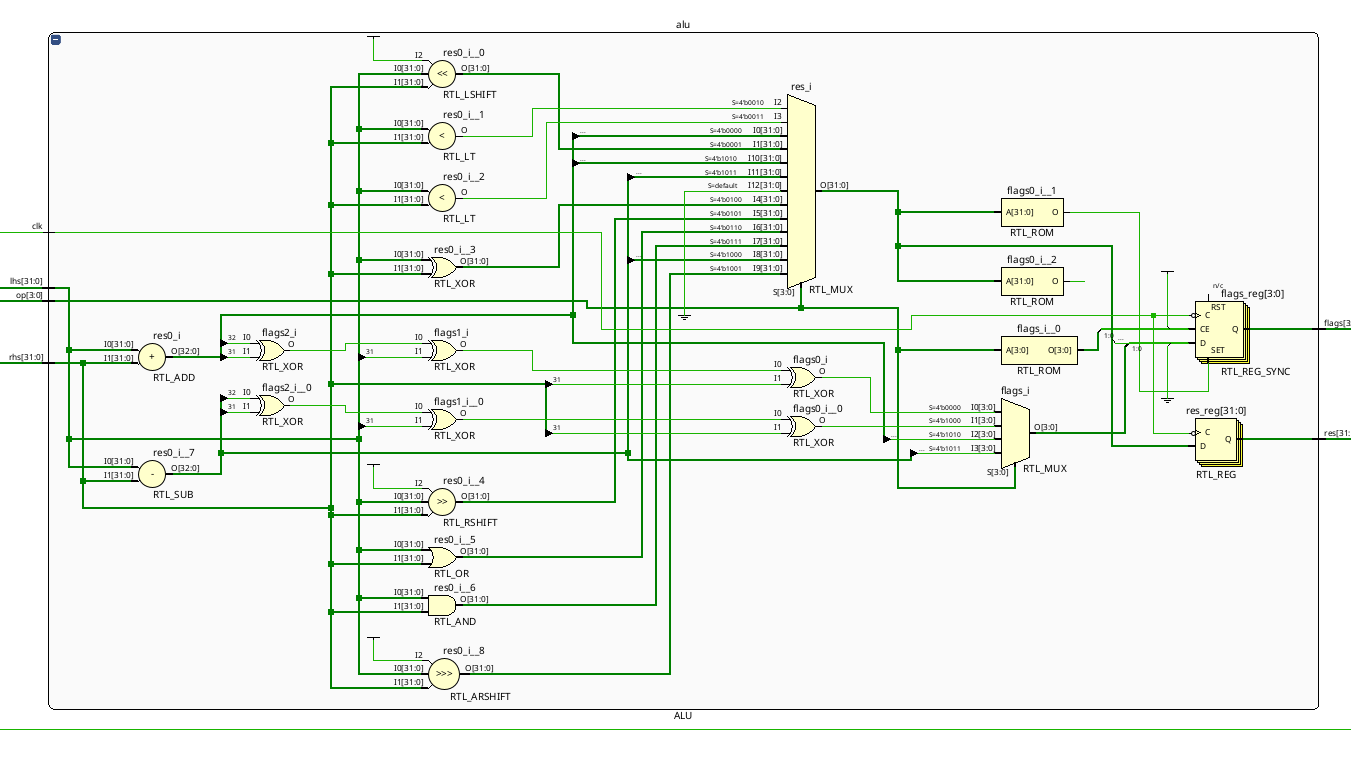
分析记录下来的板级实验结果、得到有效结论。

请力所能及回答或实践本实验的“思考与探索”部分。

1. 模块构成与连接（15分）

（请附图，并说明各模块的功能，以及模块间的连接关系。也可附开发工具中显示的电路模块连接图，加以说明。）





如图该实验的顶层模块大致可分为四个模块。mux时分复用输入模块主要解决了开发板上的输入设备不足的问题。IPclock是ip时钟核为led和alu模块提供工作时钟。alu是运算主模块。led的功能是把运算结果转换成数码管显示。

1. 板级验证方案（10分）

（说明顶层测试模块的输入/输出信号，及对应使用的I/O设备；可附图说明）

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 信号名 | 使用的IO设备 | 说明 |
| input | clk\_a | 按键 | 把数据打入operandA寄存器 |
| clk\_b | 按键 | 把数据打入operandB寄存器 |
| clk\_op | 按键 | 把数据打入op寄存器 |
| Data[31:0] | 一组逻辑开关 | 待写入operandA/B或者op的数据 |
| output | Flags[3:0] | 一组LED灯 | 标志寄存器 |
| sel[3:0] | 数码管 | 显示运算结果 |
| Seg[7:0] |

# 二、FPGA程序设计

1. 实验程序源代码（25分）

（实验各个模块的代码，包含功能注释）

//alu\_op.v

//alu\_op编码

`define ALU\_ADD\_OP 4'b0000

`define ALU\_SLL\_OP 4'b0001

`define ALU\_SLT\_OP 4'b0010

`define ALU\_SLTU\_OP 4'b0011

`define ALU\_XOR\_OP 4'b0100

`define ALU\_SRL\_OP 4'b0101

`define ALU\_OR\_OP 4'b0110

`define ALU\_AND\_OP 4'b0111

`define ALU\_SUB\_OP 4'b1000

`define ALU\_SRA\_OP 4'b1001

`define ALU\_ADDU\_OP 4'b1010 //扩展不是标准实现,与add区别在标志寄存器的溢出和进位判断上

`define ALU\_SUBU\_OP 4'b1011 //扩展不是标准实现,与sub区别在标志寄存器的溢出和进位判断上

`define ALU\_UNVALID\_OP 4'b1111

//ALU.v

`include "../define/alu\_op.v"

module ALU (

input [31:0] lhs,

input [31:0] rhs,

input clk,

input [3:0] op,

output reg [31:0] res,

output reg [3:0] flags // [is\_zero, is\_positive, carry, overflow]

);

reg \_c;

always @(negedge clk) begin

case(op)

`ALU\_ADD\_OP :begin { \_c, res } = lhs + rhs; flags[0] = \_c ^ res[31] ^ lhs[31] ^ rhs[31]; end

`ALU\_SLL\_OP :begin res = lhs << rhs; end

`ALU\_SLT\_OP :begin res = $signed(lhs) < $signed(rhs); end

`ALU\_SLTU\_OP:begin res = lhs < rhs; end

`ALU\_XOR\_OP :begin res = lhs ^ rhs; end

`ALU\_SRL\_OP :begin res = lhs >> rhs; end

`ALU\_OR\_OP :begin res = lhs | rhs; end

`ALU\_AND\_OP :begin res = lhs & rhs; end

`ALU\_SUB\_OP :begin { \_c, res } = lhs - rhs; flags[0] = \_c ^ res[31] ^ lhs[31] ^ rhs[31]; end

`ALU\_SRA\_OP :begin res = $signed(lhs) >>> rhs; end

`ALU\_ADDU\_OP:begin { flags[1], res } = lhs + rhs; end

`ALU\_SUBU\_OP:begin { flags[1], res } = lhs - rhs; end

default: begin res = 0; end

endcase

flags[3] <= res == 32'h0000\_0000 ? 1 : 0;

flags[2] <= res[31];

end

endmodule

//ALU.top.v

`include "../include/LedDisplay.v"

module Mux(

input [31:0] data,

input clk\_a,

input clk\_b,

input clk\_op,

output reg [31:0] operandA,

output reg [31:0] operandB,

output reg [3:0] op

);

always @(posedge clk\_a) operandA <= data;

always @(posedge clk\_b) operandB <= data;

always @(posedge clk\_op) op <= data[31:28];

endmodule

//顶层模块

module ALUTopper(

input [31:0] data,

input clk\_a,

input clk\_b,

input clk\_op,

input clk,

output [3:0] flags,

output [3:0] sel,

output [7:0] seg

);

wire [31:0] operandA;

wire [31:0] operandB;

wire [3:0] op;

wire [31:0] res;

wire alu\_clk;

wire led\_clk;

IPClock clock(

// Clock out ports

.clk\_10M(alu\_clk), // output clk\_10M

.clk\_100M(led\_clk), // output clk\_100M

// Status and control signals

.reset(1'b0), // input reset 0时不重置

.locked(), // output locked

// Clock in ports

.clk\_in(clk) // input clk\_in

);

Mux mux(

data,

clk\_a,

clk\_b,

clk\_op,

operandA,

operandB,

op

);

ALU alu(

operandA,

operandB,

alu\_clk,

op,

res,

flags

);

LedDisplay led(

clk,

res,

1'b1,

sel,

seg

);

endmodule

1. FPGA管脚约束代码（5分）

（引脚约束文件的内容，备注使用的具体设备）

clk\_a, clk\_b, clk\_op使用按键, data使用一组32个逻辑开关

clk使用全局时钟，flags使用led灯，sel和seg共同驱动数码管

set\_property PACKAGE\_PIN T5 [get\_ports clk\_a]

set\_property PACKAGE\_PIN V8 [get\_ports clk\_b]

set\_property PACKAGE\_PIN AA8 [get\_ports clk\_op]

set\_property PULLDOWN true [get\_ports {data[31]}]

set\_property PULLDOWN true [get\_ports {data[30]}]

set\_property PULLDOWN true [get\_ports {data[29]}]

set\_property PULLDOWN true [get\_ports {data[28]}]

set\_property PULLDOWN true [get\_ports {data[27]}]

set\_property PULLDOWN true [get\_ports {data[26]}]

set\_property PULLDOWN true [get\_ports {data[25]}]

set\_property PULLDOWN true [get\_ports {data[24]}]

set\_property PULLDOWN true [get\_ports {data[23]}]

set\_property PULLDOWN true [get\_ports {data[22]}]

set\_property PULLDOWN true [get\_ports {data[21]}]

set\_property PULLDOWN true [get\_ports {data[20]}]

set\_property PULLDOWN true [get\_ports {data[19]}]

set\_property PULLDOWN true [get\_ports {data[18]}]

set\_property PULLDOWN true [get\_ports {data[17]}]

set\_property PULLDOWN true [get\_ports {data[16]}]

set\_property PULLDOWN true [get\_ports {data[15]}]

set\_property PULLDOWN true [get\_ports {data[14]}]

set\_property PULLDOWN true [get\_ports {data[13]}]

set\_property PULLDOWN true [get\_ports {data[12]}]

set\_property PULLDOWN true [get\_ports {data[11]}]

set\_property PULLDOWN true [get\_ports {data[10]}]

set\_property PULLDOWN true [get\_ports {data[9]}]

set\_property PULLDOWN true [get\_ports {data[8]}]

set\_property PULLDOWN true [get\_ports {data[7]}]

set\_property PULLDOWN true [get\_ports {data[6]}]

set\_property PULLDOWN true [get\_ports {data[5]}]

set\_property PULLDOWN true [get\_ports {data[4]}]

set\_property PULLDOWN true [get\_ports {data[3]}]

set\_property PULLDOWN true [get\_ports {data[2]}]

set\_property PULLDOWN true [get\_ports {data[1]}]

set\_property PULLDOWN true [get\_ports {data[0]}]

set\_property PACKAGE\_PIN T3 [get\_ports {data[31]}]

set\_property PACKAGE\_PIN U3 [get\_ports {data[30]}]

set\_property PACKAGE\_PIN T4 [get\_ports {data[29]}]

set\_property PACKAGE\_PIN V3 [get\_ports {data[28]}]

set\_property PACKAGE\_PIN V4 [get\_ports {data[27]}]

set\_property PACKAGE\_PIN W4 [get\_ports {data[26]}]

set\_property PACKAGE\_PIN Y4 [get\_ports {data[25]}]

set\_property PACKAGE\_PIN Y6 [get\_ports {data[24]}]

set\_property PACKAGE\_PIN W7 [get\_ports {data[23]}]

set\_property PACKAGE\_PIN Y8 [get\_ports {data[22]}]

set\_property PACKAGE\_PIN Y7 [get\_ports {data[21]}]

set\_property PACKAGE\_PIN T1 [get\_ports {data[20]}]

set\_property PACKAGE\_PIN U1 [get\_ports {data[19]}]

set\_property PACKAGE\_PIN U2 [get\_ports {data[18]}]

set\_property PACKAGE\_PIN W1 [get\_ports {data[17]}]

set\_property PACKAGE\_PIN W2 [get\_ports {data[16]}]

set\_property PACKAGE\_PIN Y1 [get\_ports {data[15]}]

set\_property PACKAGE\_PIN AA1 [get\_ports {data[14]}]

set\_property PACKAGE\_PIN V2 [get\_ports {data[13]}]

set\_property PACKAGE\_PIN Y2 [get\_ports {data[12]}]

set\_property PACKAGE\_PIN AB1 [get\_ports {data[11]}]

set\_property PACKAGE\_PIN AB2 [get\_ports {data[10]}]

set\_property PACKAGE\_PIN AB3 [get\_ports {data[9]}]

set\_property PACKAGE\_PIN AB5 [get\_ports {data[8]}]

set\_property PACKAGE\_PIN AA6 [get\_ports {data[7]}]

set\_property PACKAGE\_PIN R2 [get\_ports {data[6]}]

set\_property PACKAGE\_PIN R3 [get\_ports {data[5]}]

set\_property PACKAGE\_PIN T6 [get\_ports {data[4]}]

set\_property PACKAGE\_PIN R6 [get\_ports {data[3]}]

set\_property PACKAGE\_PIN U7 [get\_ports {data[2]}]

set\_property PACKAGE\_PIN AB7 [get\_ports {data[1]}]

set\_property PACKAGE\_PIN AB8 [get\_ports {data[0]}]

set\_property PACKAGE\_PIN R1 [get\_ports {flags[3]}]

set\_property PACKAGE\_PIN P2 [get\_ports {flags[2]}]

set\_property PACKAGE\_PIN P1 [get\_ports {flags[1]}]

set\_property PACKAGE\_PIN N2 [get\_ports {flags[0]}]

set\_property PACKAGE\_PIN H19 [get\_ports {seg[7]}]

set\_property PACKAGE\_PIN G20 [get\_ports {seg[6]}]

set\_property PACKAGE\_PIN J22 [get\_ports {seg[5]}]

set\_property PACKAGE\_PIN K22 [get\_ports {seg[4]}]

set\_property PACKAGE\_PIN K21 [get\_ports {seg[3]}]

set\_property PACKAGE\_PIN H20 [get\_ports {seg[2]}]

set\_property PACKAGE\_PIN H22 [get\_ports {seg[1]}]

set\_property PACKAGE\_PIN J21 [get\_ports {seg[0]}]

set\_property PACKAGE\_PIN L21 [get\_ports {sel[3]}]

set\_property PACKAGE\_PIN M22 [get\_ports {sel[2]}]

set\_property PACKAGE\_PIN M21 [get\_ports {sel[1]}]

set\_property PACKAGE\_PIN N22 [get\_ports {sel[0]}]

set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets clk\_b\_IBUF]

set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets clk\_op\_IBUF]

set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets clk\_a\_IBUF]

set\_property IOSTANDARD LVCMOS18 [get\_ports clk\_a]

set\_property IOSTANDARD LVCMOS18 [get\_ports clk\_b]

set\_property IOSTANDARD LVCMOS18 [get\_ports clk\_op]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[31]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[30]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[29]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[28]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[27]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[26]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[25]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[24]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[23]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[22]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[21]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[20]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[19]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[18]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[17]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[16]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[15]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[14]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[13]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[5]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[6]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[7]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[12]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[11]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[10]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[9]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[8]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[4]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {data[0]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {flags[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {flags[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {flags[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {flags[0]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[7]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[6]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[5]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[4]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[0]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {sel[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {sel[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {sel[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {sel[0]}]

set\_property BITSTREAM.GENERAL.COMPRESS TRUE [current\_design]

set\_property -dict {IOSTANDARD LVCMOS18 PACKAGE\_PIN H4} [get\_ports clk]

set\_property SLEW SLOW [get\_ports enable]

# 三、仿真实验

（只对CPU模块仿真，无需对板级验证的顶层测试模块仿真）

1. 仿真代码（5分）

（含仿真源代码、仿真验证方案）

`timescale 1ns/1ps

module ALUTester();

reg [3:0] op;

reg [31:0] lhs, rhs;

wire [31:0] res;

wire ZF,SF,CF,OF;

ALU alu(

lhs,

rhs,

clk,

op,

res,

{ ZF, SF, CF, OF }

);

reg clk;

parameter PERIOD = 10;

initial begin

clk = 0;

forever begin

#(PERIOD/2) clk = ~clk;

end

end

initial begin

op = 4'b0000; //add

lhs = 32'hffff\_ffff;

rhs = 32'h0000\_0001;

#20

lhs = 32'h7fff\_ffff;

rhs = 32'h1000\_0003;

#20

op = 4'b0001; //sll

lhs = 32'h000f\_0000;

rhs = 32'h0000\_0002;

#20

op = 4'b0010; //slt

lhs = 32'hffff\_0000;

rhs = 32'h0003\_0001;

#20

lhs = 32'h0fff\_ffff;

rhs = 32'h0009\_0003;

#20

op = 4'b0011; //sltu

lhs = 32'hf000\_0000;

rhs = 32'h0000\_0001;

#20

op = 4'b0100; //xor

lhs = 32'h0000\_0000;

rhs = 32'h0000\_0001;

#20

op = 4'b0100; //srl

lhs = 32'hf000\_0000;

rhs = 32'h0000\_0002;

#20

op = 4'b0110; //or

lhs = 32'h0000\_0000;

rhs = 32'h0000\_0001;

#20

op = 4'b0111; //and

lhs = 32'h0d00\_0001;

rhs = 32'h0f00\_0001;

#20

op = 4'b1000; //sub

lhs = 32'h8000\_0000;

rhs = 32'h0fff\_ffff;

#20

lhs = 32'h0000\_0001;

rhs = 32'h0000\_0002;

#20

op = 4'b1001; //sra

lhs = 32'h0000\_00f0;

rhs = 32'h0000\_0001;

#20

lhs = 32'hffff\_ffff;

rhs = 32'h0000\_0003;

#20

op = 4'b1010; //addu

lhs = 32'h0000\_0000;

rhs = 32'h0000\_0001;

#20

lhs = 32'hffff\_ffff;

rhs = 32'h0000\_0001;

#20

op = 4'b1011; //subu

lhs = 32'h0000\_0000;

rhs = 32'h0000\_0001;

#20

lhs = 32'h7fff\_ffff;

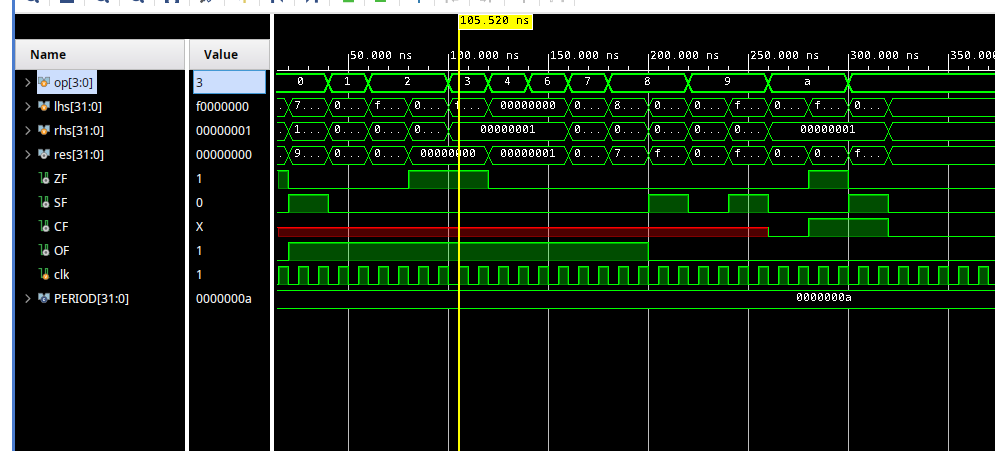
rhs = 32'h7000\_0001;

end

endmodule

1. 仿真波形及结果分析（15分）

（仿真波形截图，并对仿真波形进行分析）



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| op | lhs | rhs | 期望的res | res | 期望的flags | flags{ZF,SF,CF,OF} | 说明 |
| 0000  add | 32’hffff\_ffff  -1 | 32’h0000\_0001  1 | 32’h0000\_0000  0 | 32’h0000\_0000 | 10\*\* | 10x1 | 验证加法 |
| 0000  add | 32’h7fff\_ffff | 32’h1000\_0003 | 正溢出成负数 | 32‘h9000\_0002 | OF寄存器为1 | 01X1 | 检验溢出情况是否正常 |
| 0001  sll | 32’h000f\_0000 | 32’h0000\_0002 | 32’h003c\_0000 | 32’h003c\_0000 |  | 00X1 | 验证左移 |
| 0010  slt | ffff\_0000  <0 | 0003\_0001  >0 | 1 | 1 |  | 00X1 | 验证有符号比较 |
| 0010  slt | 0ffff\_ffff | 0009\_0003 | 0 | 0 |  | 10X1 | 验证有符号比较 |
| 0011  sltu | f000\_0000 | 0000\_0001 | 0 | 0 |  | 10X1 | 验证无符号比较 |
| 0100  xor | 0000\_0000 | 0000\_0001 | 0000\_0001 | 0000\_0001 |  | 00X1 | 验证异或运算 |
| 0101  srl | f000\_0000 | 0000\_0002 | 3c00\_0002 | 3c00\_0000 |  | 00X1 | 考察逻辑右移 |
| 0110  or | 0000\_0000 | 0000\_0001 | 0000\_0001 | 0000\_0001 |  | 00X1 | 考察或运算 |
| 0111  and | 0d00\_0001 | 0f00\_0001 | 0d00\_0001 | 0d00\_0001 |  | 00X1 | 考察与运算 |
| 1000  sub | 8000\_0000  负最大值 | 0fff\_ffff | 负溢出成正数 | 7000\_0001 | OF寄存器为1 | 00X1 | 考察有符号减法溢出 |
| 1000  sub | 0000\_0001 | 0000\_0002 | ffff\_ffff | ffff\_ffff |  | 01X0 | 考察有符号减法 |
| 1001  sra | 0000\_00f0 | 0000\_0001 | 0000\_0078 | 0000\_0078 |  | 00X0 | 算术右移 |
| 1001  sra | ffff\_ffff | 0000\_0003 | ffff\_ffff | ffff\_ffff |  | 01X0 | 算术右移 |
| 1010  addu | 0000\_0000 | 0000\_0001 | 0000\_0001 | 0000\_0001 | CF寄存器为0 | 0000 | 无符号加法 |
| 1010  addu | ffff\_ffff | 0000\_0001 | 0000\_0000 | 0000\_0000 | CF寄存器为1 | 1010 | 无符号加法 |
| 1011  subu | 0000\_0000 | 0000\_0001 | ffff\_ffff | ffff\_ffff | CF寄存器为1 | 0110 | 无符号减法 |
| 1011  subu | 7ffff\_ffff | 7000\_0001 | 0fff\_fffe | 0fff\_fffe | CF为0 | 0000 | 无符号减法 |

# 由仿真结果可知，该代码大致符合预期结果

# 四、板级实验

1. 板级实验操作说明（5分）

板级验证：将引脚约束好的FPGA程序下载到芯片中，然后利用FPGA芯片引脚上所连接的输入输出设备（开关、按键、显示灯和数码管等）验证：操作输入设备，读取输出设备上的结果，来判断程序的逻辑是否正确

1. 板级实验结果记录（5分）

（以表格形式展现，具体见教材上实验结果记录表）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| op | 功能 | lhs | rhs | res | flags{ZF,SF,CF,OF} | 说明 |
| 0000 | add | ffff\_ffff | 0000\_0001 | 0000\_0000 | 10x1 | 验证加法 |
| 0000 | add | 7fff\_ffff | 1000\_0003 | 9000\_0002 | 01X1 | 检验溢出情况是否正常 |
| 0001 | sll | 000f\_0000 | 0000\_0002 | 003c\_0000 | 00X1 | 验证左移 |
| 0010 | slt | ffff\_0000 | 0003\_0001 | 0000\_0001 | 00X1 | 验证有符号比较 |
| 0010 | slt | 0ffff\_ffff | 0009\_0003 | 0000\_0000 | 10X1 | 验证有符号比较 |
| 0011 | sltu | f000\_0000 | 0000\_0001 | 0000\_0000 | 10X1 | 验证无符号比较 |
| 0100 | xor | 0000\_0000 | 0000\_0001 | 0000\_0001 | 00X1 | 验证异或运算 |
| 0101 | srl | f000\_0000 | 0000\_0002 | 3c00\_0000 | 00X1 | 考察逻辑右移 |
| 0110 | or | 0000\_0000 | 0000\_0001 | 0000\_0001 | 00X1 | 考察或运算 |
| 0111 | and | 0d00\_0001 | 0f00\_0001 | 0d00\_0001 | 00X1 | 考察与运算 |
| 1000 | sub | 8000\_0000 | 0fff\_ffff | 7000\_0001 | 00X1 | 考察有符号减法溢出 |
| 1000 | sub | 0000\_0001 | 0000\_0002 | ffff\_ffff | 01X0 | 考察有符号减法 |
| 1001 | sra | 0000\_00f0 | 0000\_0001 | 0000\_0078 | 00X0 | 算术右移 |
| 1001 | sra | ffff\_ffff | 0000\_0003 | ffff\_ffff | 01X0 | 算术右移 |
| 1010 | addu | 0000\_0000 | 0000\_0001 | 0000\_0001 | 0000 | 无符号加法 |
| 1010 | addu | ffff\_ffff | 0000\_0001 | 0000\_0000 | 1010 | 无符号加法 |
| 1011 | subu | 0000\_0000 | 0000\_0001 | ffff\_ffff | 0110 | 无符号减法 |
| 1011 | subu | 7ffff\_ffff | 7000\_0001 | 0fff\_fffe | 0000 | 无符号减法 |

1. 结果分析与结论（5分）

（分析实验结果，给出实验结论）

多功能alu的各种运算都是有效的，产生的标志寄存器信号正确

# 五、思考与探索

（10分）

1. 问题与解决方案

（整个实验过程中发生了什么问题？你是如何解决的。）

对于add和sub的溢出信号的产生不熟悉：温习计算机组成原理的判溢方法。

对如何驱动数码管不熟悉：借鉴老师代码，学习段选、片选和时钟周期信号如何驱动数码管显示数据

1. 思考题

（力所能及，尝试实践或回答教材上的思考与探索题目，至少完成1道）

1. 代码如下

`ALU\_SLT\_OP :begin res = $signed(lhs) < $signed(rhs); end

`ALU\_SLTU\_OP:begin res = lhs < rhs; end

用的测试数据如下

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0010 | slt | ffff\_0000 | 0003\_0001 | 0000\_0001 | 00X1 | 验证有符号比较 |
| 0010 | slt | 0ffff\_ffff | 0009\_0003 | 0000\_0000 | 10X1 | 验证有符号比较 |
| 0011 | sltu | f000\_0000 | 0000\_0001 | 0000\_0000 | 10X1 | 验证无符号比较 |

# 2. 代码如下

`ALU\_SRL\_OP :begin res = lhs >> rhs; end

`ALU\_SRA\_OP :begin res = $signed(lhs) >>> rhs; end

用的测试数据如下

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0101 | srl | f000\_0000 | 0000\_0002 | 3c00\_0000 | 00X1 | 考察逻辑右移 |
| 1001 | sra | 0000\_00f0 | 0000\_0001 | 0000\_0078 | 00X0 | 算术右移 |
| 1001 | sra | ffff\_ffff | 0000\_0003 | ffff\_ffff | 01X0 | 算术右移 |

# 5. blt指令根据运算结果的标志位来判断是否小于。在RV32I指令集中，blt指令的具体实现方法如下：

1. 首先，从寄存器rs1中读取一个有符号整数A，从寄存器rs2中读取一个有符号整数B。
2. 然后，将A和B进行有符号减法，即计算A-B的结果。
3. 最后，根据运算结果中的标志位（即负数标志位）来判断A是否小于B。如果负数标志位为1，表示结果是一个负数，也就是A小于B，此时执行跳转指令，否则继续顺序执行下一条指令。

在RV32I指令集中，blt指令使用的是有符号数比较，因此需要考虑符号位的影响。具体而言，blt指令使用有符号减法计算A-B的结果，结果的符号位表示A-B的符号，从而确定A和B的大小关系。如果结果为负数，表示A小于B；如果结果为非负数，表示A大于等于B。因此，blt指令根据运算结果的负数标志位来判断A是否小于B。

# 六、实验心得体会、意见建议

（如果是小组合作，请给出具体分工、分别给出心得体会）

在进行多功能ALU设计的实验过程中，我深刻认识到了数字电路设计的重要性。以下是我在实验中的心得体会：

1. 熟悉ALU的基本功能和设计思路：在进行实验前，需要深入了解ALU的基本功能和设计思路，包括算术运算、逻辑运算、移位运算等概念，这有助于更好地理解电路的功能和设计。
2. 设计合理的电路结构：根据实验要求，我设计了一个包括多路选择器、加法器、逻辑门、移位器等模块的多功能ALU电路。需要注意的是，电路结构的设计需要综合考虑电路的性能和复杂度，同时需要考虑电路的稳定性和可靠性。
3. 编写Verilog HDL代码：在实验中，我使用Verilog HDL编写了电路的描述代码，实现了电路的逻辑功能。需要注意的是，代码的编写需要符合Verilog HDL的语法规范，并且需要考虑模块之间的连接和数据传输等问题。
4. 验证电路的功能和性能：我使用不同的测试数据对电路进行了验证，包括测试算术运算、逻辑运算、移位运算等功能。同时，我还对电路的运行速度和功耗等性能进行了评估，以确保电路的稳定性和可靠性。

通过这次实验，我不仅深入了解了多功能ALU的原理和设计方法，同时也掌握了Verilog HDL的基本使用方法，并对数字电路的设计和实现有了更深刻的认识