32位ALU设计

17307130178 宁晨然

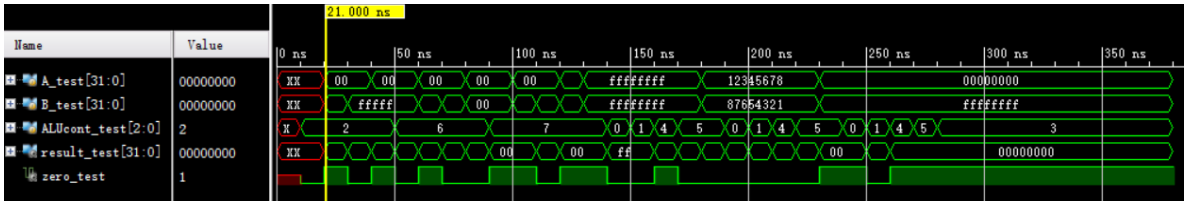
1. ALU结果展示

本次作业实现了**32位ALU设计**，并且使用**仿真**模拟完整测试ALU的所有功能，与**要求一致**。

本人认为本次作业达到**A水平**，具备以下实验亮点。

实验亮点

- ALU功能完整，符合要求
- ALU测试完整可靠，使用仿真完成**所有情况**测试
- 代码附有**注释**，清晰易懂
- 实验报告使用markdown撰写，详细细致
- 涵盖实验原理/实验过程/实验结果/实验改进等方面

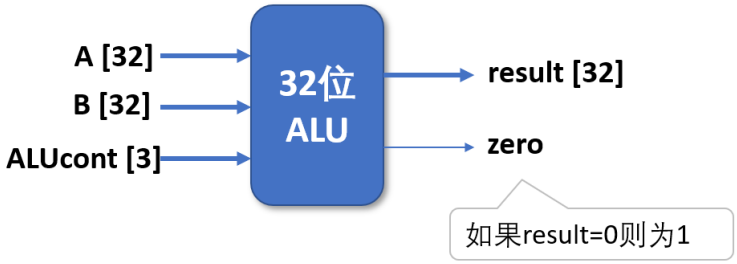


2. ALU实现过程

2.1 实验原理

原理如图所示，该ALU部件拥有7个功能。

$$result = A \pm B$$



ALUcont	功 能
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\bar{B}$
101	A OR $\bar{B}$
110	A − B
111	SLT

Set if Less Than. 小于则置1

该ALU部件简单易懂，代码实现需要分情况判断即可。

2.2 实现代码

ALU功能简单，输入三个值，输出两个值，触发条件为输入值改变时。实现过程如下：

- 初始化：先定义输入值A/B/ALUcont和输出值result/zero，注意always语句中赋值需要reg。
- 触发条件：任意输出值改变时。
- 情况分类：根据ALUcont值不同，分别为实验原理中的7个功能与1个默认状态。其中详细讲解：
  - and/or/and not/or not：此类实现直接使用& | ~ 的按位与或非操作即可。
  - add/sub：实现直接使用+ - 操作即可。注意进位直接舍去。
  - SLT：小于时置result为0。
- zero位更新：等result更新完毕后，判断是否为0即可。

```
`timescale 1ns / 1ps

module ALU(A, B, ALUcont, result, zero);
    input [31:0]A;
    input [31:0]B;
    input [2:0]ALUcont;

    output reg [31:0]result;
    output reg zero;
    reg [31:0]res;

    // then implement functions
    always@(A or B or ALUcont)
    begin
        case(ALUcont)
            'b000: result = A & B;
            'b001: result = A | B;
            'b010: result = A + B;
            'b100: result = A & (~B);
```

```
            'b101: result = A | (~B);
            'b110: result = A - B;
            'b111:
                begin
                // signed comparison
                    res = A-B;
                    result = res[31];
                end
             default: result = 0;
        endcase
        if(result == 0) zero = 1;
        else zero = 0;
    end
endmodule
```

3. ALU测试过程

本实验采用`vivado`内置的仿真模拟功能进行测试。测试内容覆盖ALU所有功能。

3.1 测试代码

测试过程:

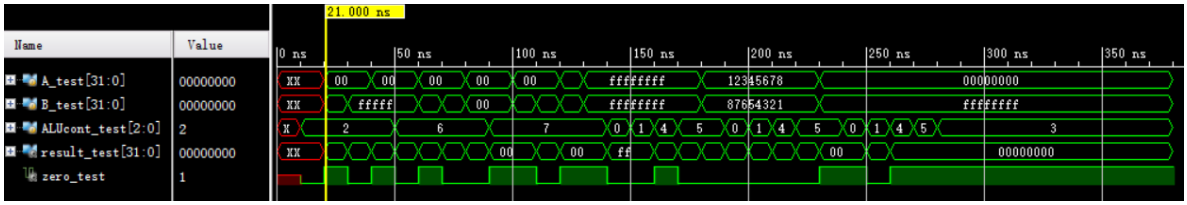
- 测试程序的各个变量与设计程序相同
- 使用`initial`中赋值，并且每次测试用`#10`的方式延时赋值
- 分别测试每个功能，并多次改变输入的A/B值

```
`timescale 1ns / 1ps

module test_ALU();
    reg [31:0] A_test;
    reg [31:0] B_test;
    reg [2:0] ALUcont_test;
    wire [31:0] result_test;
    wire zero_test;


    initial
    begin
        // test add
        #10 ALUcont_test = 2;
        #10 A_test = 0;B_test = 0;
        #10 A_test = 0;B_test = -1;
        #10 A_test = 1;B_test = -1;
        #10 A_test = 'h12345678; B_test = 'h87654321;

        // test sub
        #10 ALUcont_test = 6;
        #10 A_test = 0;B_test = 0;
        #10 A_test = 0;B_test = -1;
        #10 A_test = 100;B_test = 1;

        // test SLT
        #10 ALUcont_test = 7;
        #10 A_test = 0;B_test = 0;
        #10 A_test = 0;B_test = -1;
```

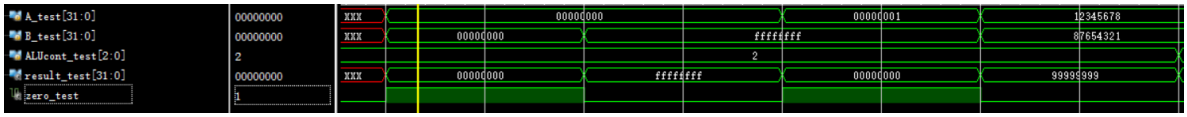
```
        #10 A_test = 1;B_test = 0;
        #10 A_test = -1;B_test = 0;

        // test and/or/and not/or not 0145
        #10 A_test = -1;B_test = -1;
        #10 ALUcont_test = 0;
        #10 ALUcont_test = 1;
        #10 ALUcont_test = 4;
        #10 ALUcont_test = 5;

        #10 A_test = 'h12345678; B_test = 'h87654321;
        #10 ALUcont_test = 0;
        #10 ALUcont_test = 1;
        #10 ALUcont_test = 4;
        #10 ALUcont_test = 5;

        #10 A_test = 0; B_test = -1;
        #10 ALUcont_test = 0;
        #10 ALUcont_test = 1;
        #10 ALUcont_test = 4;
        #10 ALUcont_test = 5;
        #10 ALUcont_test = 3;

        #100 $stop;
    end

    ALU
t(.A(A_test),.B(B_test),.ALUcont(ALUcont_test),.result(result_test),.zero(zero_t
est));
endmodule
```

3.2 仿真模拟结果

总体结果如图所示，下面从各个功能出发解释。

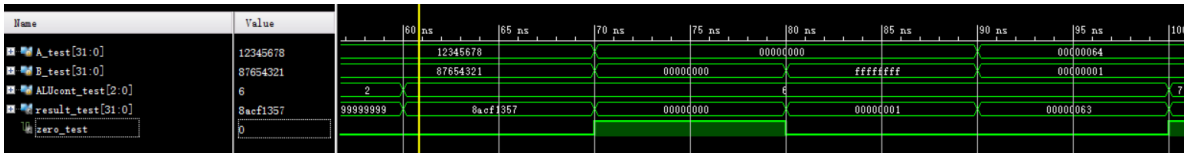


3.2.1 ADD



ALUcont	A	B	result	zero
2	0	0	0	1
	0	-1	-1	0
	1	-1	0	1
	12345678	87654321	99999999	0

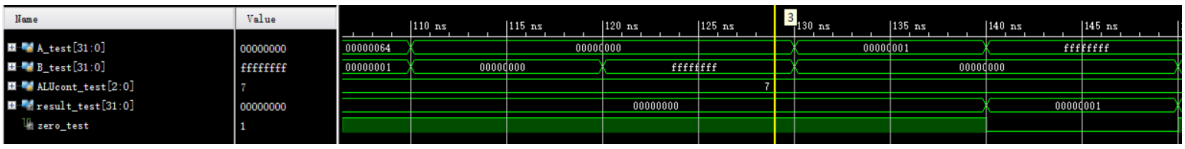
3.2.2 SUB



ALUcont	A	B	result	zero
6	h12345678	h87654321	8acf1357	0
	0	0	-1	1
	0	-1	1	0
	h64	1	h63	0

3.2.3 SLT

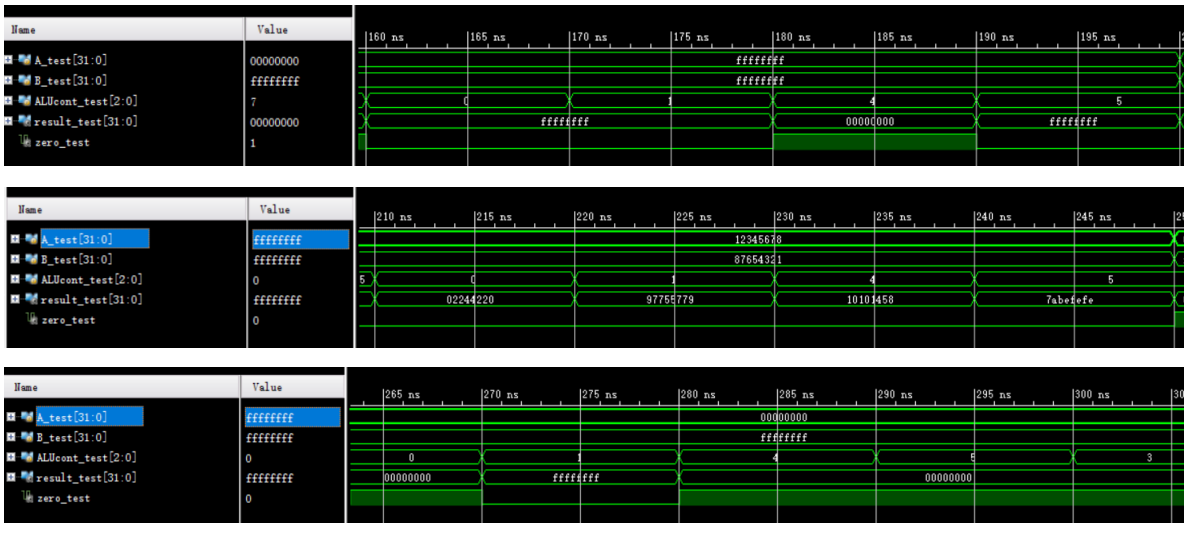
此处实现的是有符号比较。



ALUcont	A	B	result	zero
7	h64	1	0	1
	0	0	0	1
	0	-1	0	1
	1	0	0	1
	-1	0	1	0

3.2.4 AND OR

3为default状态，设置为0。



ALUcont	A	B	result	zero
0	-1	-1	-1	0
1			-1	0
4			0	1
5			-1	0
0	12345678	87654321	02244220	0
1			97755779	0
4			10101458	0
5			7abefefe	0
0	0	-1	0	1
1			-1	0
4			0	1
5			0	1
3			0	1

4. 实验改进

本次实现的ALU较为简单，改进方法可以从几个方面入手：

- 添加clk控制ALU触发变为时钟上升沿
- SLT可以加入控制条件为 有符号比较/无符号比较
- ALU设计可添加更多控制条件，输出可以添加cout进位等