一、简单alu

源文件

```
`timescale 1ns / 1ps


module simple_alu(
    input [31:0] alu_control,
    input [31:0] alu_src1,
    input [31:0] alu_src2,
    output [31:0] alu_result
    );

    wire op_add;
    wire op_sub;
    wire op_slt;
    wire op_sltu;//
    wire op_and;
    wire op_or;
    wire op_xor;
    wire op_nor;
    wire op_sll;//shift left logic
    wire op_srl;// shift right logic
    wire op_sra;//shift right arithmetic
    wire op_lui;

    assign op_add = alu_control[0];//加法操作
    assign op_sub = alu_control[1];//减法操作
    assign op_slt = alu_control[2];//有符号比较，小于置位
    assign op_sltu = alu_control[3];//无符号比较，小于置位
    assign op_and = alu_control[4];//按位与
    assign op_or  = alu_control[5];//按位或非
    assign op_xor = alu_control[6];//按位或
    assign op_nor = alu_control[7];//按位异或
    assign op_sll = alu_control[8];//逻辑左移
    assign op_srl = alu_control[9];//逻辑右移
    assign op_sra = alu_control[10];//算术右移
    assign op_lui = alu_control[11];//高位加载

    //result
    wire [31:0] add_sub_result;//add=sub because two's complement
    wire [31:0] slt_result;//set less than
    wire [31:0] sltu_result;//set less than unsigned
    wire [31:0] and_result;

    wire [31:0] or_result;
    wire [31:0] xor_result;
    wire [31:0] nor_result;
    wire [31:0] sll_result;

    wire [31:0] srl_result;
    wire [31:0] sra_result;
    wire [31:0] lui_result;
```

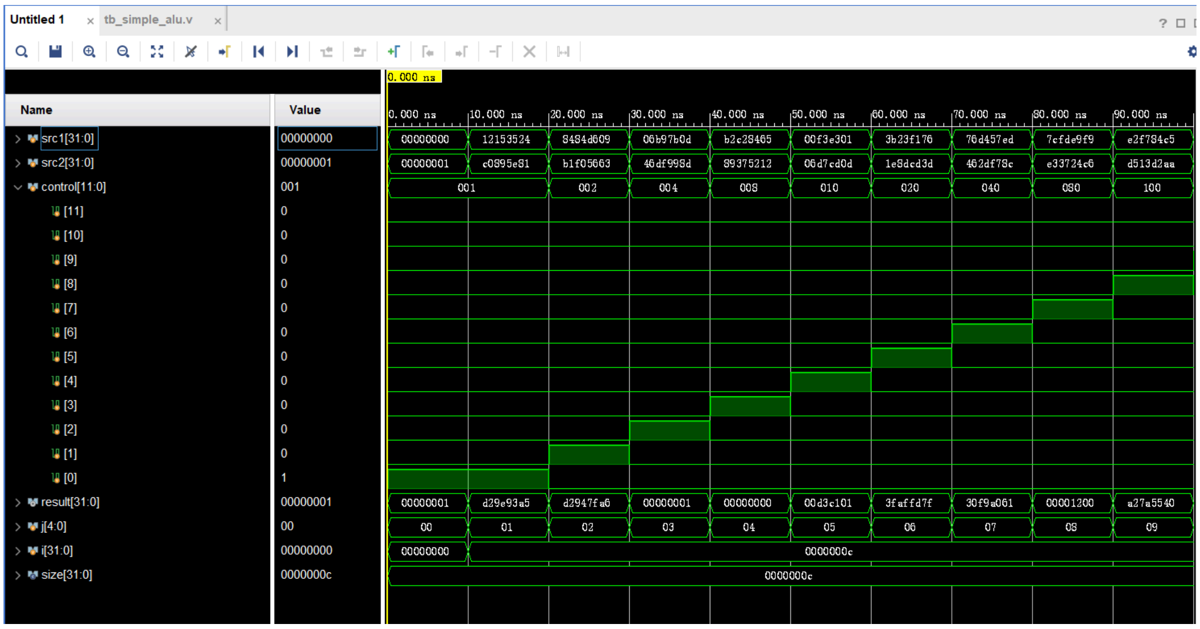
```verilog
//how to get result
    assign and_result = alu_src1 & alu_src2;
    assign or_result = alu_src1 | alu_src2;
    assign xor_result = alu_src1 ^ alu_src2;
    assign nor_result = ~ or_result;
    assign lui_result = {alu_src2[15:0],16'b0};//??can't learn

    wire [31:0] adder_a;//
    wire [31:0] adder_b;
    wire        adder_cin;
    wire [31:0] adder_result;
    wire        adder_cout;

    assign adder_a = alu_src1;
    assign adder_b =  (op_sub|op_slt|op_sltu)?~alu_src2:alu_src2;
    //judgement op_sub if true then ~alu_src2 else alu_src2
    //(a+b)'s implement =(a)'s implement + (b)'s implement
    //(a-b)'s implement =(a)'s implement + (-b)'s implement=(a)'s implement +
(b)'s implement
    // relation? b's implement and (-b)'s implement
    // if b>0, (-b)'s implement=~(b'simplement)+1
    //if b<0, (-b)'s implement=~(b'simplement)+1
    assign adder_cin = (op_sub|op_slt|op_sltu)? 1'b1:1'b0;
    assign {adder_cout,adder_result}=adder_a+adder_b+adder_cin;

    assign add_sub_result =  adder_result;
    assign slt_result[31:1]=31'b0;
    assign slt_result[0]=(alu_src1[31]&~alu_src2[31])
    |(~(alu_src1[31]^alu_src2[31])&adder_result[31]);//***
    //one
    // - and -
    //+ and +
    //judgement with (alu_src1[31]&~alu_src2[31])
    //two
    //+ and - or - and +
    //judgement with (~(alu_src1[31]^alu_src2[31])&adder_result[31])

    assign sltu_result[31:0]=31'b0;
    assign sltu_result[0]=~adder_cout;

    assign sll_result = alu_src2 << alu_src1[4:0];
    assign srl_result =alu_src2 >> alu_src1[4:0];
    assign sra_result = ($signed(alu_src2)) >>> alu_src1[4:0]; //>>> sra

    assign alu_result = ({32{op_add|op_sub}}&add_sub_result)
    |    ({32{op_slt}}&slt_result)
    |    ({32{op_sltu}}&sltu_result)
    |    ({32{op_and}}&and_result)
    |    ({32{op_or}}&or_result)
    |    ({32{op_xor}}&xor_result)
    |    ({32{op_nor}}&nor_result)
    |    ({32{op_sll}}&sll_result)
    |    ({32{op_srl}}&srl_result)
    |    ({32{op_sra}}&sra_result)
    |    ({32{op_lui}}&lui_result);
```

```
endmodule
```

测试文件

```
`timescale 1ns / 1ps

module tb_simple_alu(

    );
    reg[31:0] src1,src2;
    reg [11:0] control;
    wire[31:0] result;

    simple_alu alu(.alu_control(control),
                   .alu_src1(src1),
                   .alu_src2(src2),
                   .alu_result(result)
                 );

    initial begin
         src1 = 0;
         src2 = 1;
         control = 12'b0000_0000_0001;//assign语句只能用于驱动wire信号类型，不能用于驱
动reg类型，这里不能用assign src = 0 ;
         // 打印初始值
        $display("Initial values: src1 = %d, src2 = %d, control = %b", src1,
src2, control);

        // 让模拟进行一段时间
        #200;

        // 执行多个测试
        $finish;
    end

    parameter size = 12;

    reg [4:0] j=0;
    integer i=0;
    always #10 begin


                 src1 = $random;
                 src2 = $random;//always语句也不能在always语句中使用，这在 Verilog 中
是非法的。assign 应该在顶层模块之外使用，
    //控制信号变化

               for (i=0;i<size;i=i+1) begin
                    if(i==j)
                       begin
                           control[i] = 1'b1;
                       end
                    else
                        begin
                            control[i] = 1'b0;
                        end
                end
                j = (j+1) % 12;// 使 j 循环在 [0, 11] 范围内
        end
endmodule
```

波形图



Name	Value
src1[31:0]	00000000
src2[31:0]	00000001
control[11:0]	001
[11]	0
[10]	0
[9]	0
[8]	0
[7]	0
[6]	0
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	1
result[31:0]	00000001
j[4:0]	00
i[31:0]	00000000
size[31:0]	0000000c

```
0.000 ns  10.000 ns  20.000 ns  30.000 ns  40.000 ns  50.000 ns  60.000 ns  70.000 ns  80.000 ns  90.000 ns
00000000  12153524   8484d609   06b97b0d   b2c28465   00f3e301   3b23f176   76d457ed   7cfde9f9   e2f784c5
00000001  c0895e81   b1f05663   46df998d   89375212   06d7cd0d   1e8dcd3d   462df78c   e33724c6   d513d2aa
001                  002        004        008        010        020        040        080        100
00000001  d29e93a5   d2947fa6   00000001   00000000   00d3c101   3faffd7f   30f9a061   00001200   a27a5540
00         01         02         03         04         05         06         07         08         09
00000000  0000000c
0000000c
```

1.