同济大学计算机系

数字逻辑课程实验报告

学　　号 **2253156**

姓　　名 闫浩扬

专　　业 计算机科学与技术（精英班）

授课老师 郭玉臣

一、 实验内容

实验介绍:

在本次实验中，我们将使用 Verilog HDL 语言实现行为级 ALU 的设计和仿真。

实验目标:

深入了解 ALU 的原理

学习使用 Verilog HDL 语言进行行为级 ALU 的设计与仿真。

二、 硬件逻辑图

实验原理: ALU 是负责运算的电路。ALU 必须实现以下几个运算：加（ADD）、减（SUB）、与（AND）、或（OR）、异或（XOR）、置高位立即数（LUI）、逻辑左移与算数左移（SLL）、逻辑右移（SRL）以及算数右移（SRA）、 SLT、SLTU 等操作。输出 32 位计算结果、carry(借位进位标志位)、zero(零标志位)、negative(负数标志位)和 overflow(溢出标志位)。本实验实现 ALU 的基本思想是：在操作数输入之后将所有可能的结果都计算出来，通过操作符 aluc 的输入来判别需要执行的操作来选择需要的结果进行输出。图 6.9.1 所示为本实验的 ALU 参考原理图。表 6.9.1 所示为 aluc 的值所对应的运算。表 6.9.2 所示为 addsub32 标志位规则（仅供参考）。

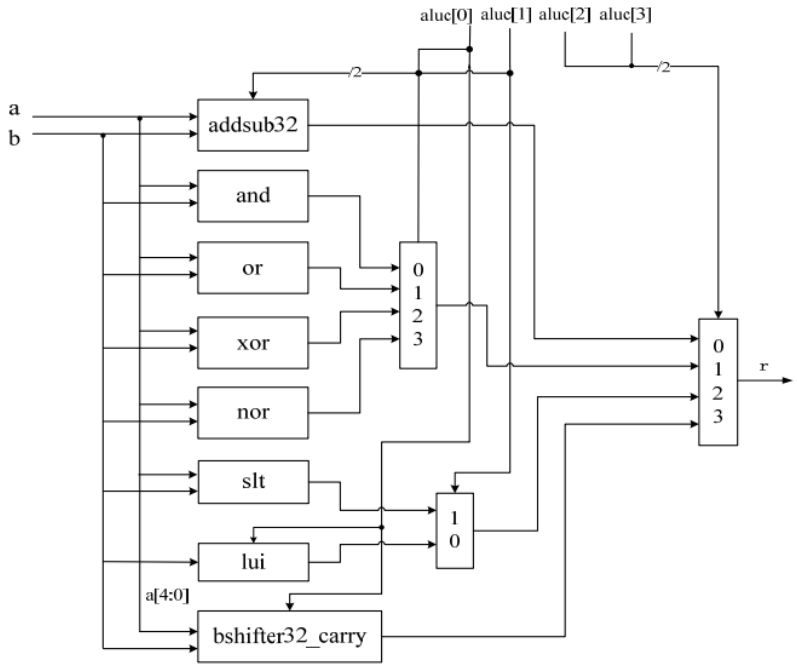


图 6.9.1 ALU 的原理图供参考

表 6.9.1 aluc 的值所对应的运算

	aluc[3]	aluc[2]	aluc[1]	aluc[0]
Addu r=a+b 无符号	0	0	0	0
Add r=a+b 有符号	0	0	1	0
Subu r=a-b 无符号	0	0	0	1
Sub r=a-b 有符号	0	0	1	1
And r=a & b	0	1	0	0
Or r=a | b	0	1	0	1
Xor r=a ^ b	0	1	1	0
Nor r=~ (a | b)	0	1	1	1
Lui r={b[15:0],16'b0}	1	0	0	X
Slt r=(a<b)?1:0 有符号	1	0	1	1
Sltu r=(a<b)?1:0 无符号	1	0	1	0
Sra r=b>>>a	1	1	0	0
Sll/Slr r=b<<a	1	1	1	X
Srl r=b>>a	1	1	0	1

表 6.9.2 ALU 标志位规则

zero 标志位	1.Z=1 表示运算结果是零，Z=0 表示运算结果不是零。 2.对于 Slt 和 Sltu 运算，如 a-b=0，则 Z=1，表示进行比较的两个数大小相等。 3.所有运算均影响此标志位。
carry 标志位	1.无符号数加法运算（Addu）发生上溢出，则该标志位为 1。 2.无符号数减法运算（Subu）发生下溢出，则该标志位为 1。 3.无符号数比较运算（Sltu），如 a-b<0，则该标志位为 1。 4.移位运算，该标志位为最后一次被移出的位的数值(在移位模块实现)。 5.其他运算不影响此标志位。
negative 标志位	1.有符号数运算 Add 和 Sub，操作数和运算结果均采用二进制补码的形式表示，N=1 表示运算的结果为负数，N=0 表示结果为正数或零。 2.有符号数比较运算（Slt），如果 a-b<0，则 N=1。 3.其他运算，运算最终结果的最高位 r[31]为 1，则 N=1。
overflow 标志位	1.对于有符号加减法运算（Add 和 Sub），操作数和运算结果均采用二进制补码的形式表示，有溢出时该标志位 o=1。 2.只有有符号加减法运算影响此标志位。

三、 模块建模

● 接口定义

```
module alu(
input [31:0] a, //32 位输入，操作数 1
input [31:0] b, //32 位输入，操作数 2
input [3:0] aluc, //4 位输入，控制 alu 的操作
output [31:0] r, //32 位输出，由 a、b 经过 aluc 指定的操作生成
output zero, //0 标志位
output carry, // 进位标志位
output negative, // 负数标志位
output overflow // 溢出标志位
);
```

● Verilog 代码描述

提示：本次实验允许使用行为级建模方式实现 ALU，可以使用“+”“-”“<>”等运 算符号实现 ALU 中的计算模块

```
module alu(
    input [31:0] a,
    input [31:0] b,
    input [3:0] aluc,
    output reg [31:0] r,
    output reg zero,
    output reg carry,
    output reg negative,
    output reg overflow
    );
    parameter Addu = 4'b0000; //r=a+b unsigned
    parameter Add  = 4'b0010; //r=a+b signed
    parameter Subu = 4'b0001; //r=a-b unsigned
    parameter Sub  = 4'b0011; //r=a-b signed
    parameter And  = 4'b0100; //r=a&b
    parameter Or   = 4'b0101; //r=a|b
    parameter Xor  = 4'b0110; //r=a^b
    parameter Nor  = 4'b0111; //r=~(a|b)
    parameter Lui1 = 4'b1000; //r={b[15:0],16'b0}
    parameter Lui2 = 4'b1001; //r={b[15:0],16'b0}
    parameter Slt  = 4'b1011; //r=(a-b<0)?1:0 signed
    parameter Sltu = 4'b1010; //r=(a-b<0)?1:0 unsigned
    parameter Sra  = 4'b1100; //r=b>>>a
    parameter Sll  = 4'b1110; //r=b<<a
    parameter Slr  = 4'b1111; //r=b<<a
    parameter Srl  = 4'b1101; //r=b>>a
    reg [32 : 0] temp;
    reg signed [31:0] temp_b;
    always @(*)
        casex (aluc)
            Addu: begin
                r = a + b;
                temp = {1'b0, a} + {1'b0, b};
                zero = (r == 0) ? 1 : 0;
                carry = temp[32];
                negative = r[31];
                overflow = 0;
            end

            Add: begin
                r = a + b;
                zero = (r == 0) ? 1 : 0;
                carry = 0;
                negative = r[31];
                overflow = ((a[31] == b[31]) && (r[31] !=
a[31])) ? 1 : 0;
            end

            Subu: begin
                r = a - b;
                zero = (r == 0) ? 1 : 0;
                carry = (a < b) ? 1 : 0;
                negative = r[31];
                overflow = 0;
            end
```

```
Sub:  begin
            r = a - b;
            zero = (r == 0) ? 1 : 0;
            carry = 0;
            negative = r[31];
        overflow = ((a[31] != b[31]) && (r[31] != a[31])) ? 1 : 0;
         end
         And: begin
            r = a & b;
            zero = (r == 0) ? 1 : 0;
            carry = 0;
            negative = r[31];
            overflow = 0;
         end
         Or: begin
            r = a | b;
            zero = (r == 0) ? 1 : 0;
            carry = 0;
            negative = r[31];
            overflow = 0;
         end
         Xor: begin
            r = a ^ b;
            zero = (r == 0) ? 1 : 0;
            carry = 0;
            negative = r[31];
            overflow = 0;
         end
         Nor: begin
            r = ~(a | b);
            zero = (r == 0) ? 1 : 0;
            carry = 0;
            negative = r[31];
            overflow = 0;
         end
         Lui1,Lui2: begin
            r = {b[15:0], 16'b0};
            zero = (r == 0) ? 1 : 0;
            carry = 0;
            negative = r[31];
            overflow = 0;
         end
         Slt: begin
            r = a - b;
            zero = (r == 0) ? 1 : 0;
            carry = 0;
            negative = r[31];
        overflow = ((a[31] != b[31]) && (r[31] != a[31])) ? 1 : 0;
            r = (overflow == 1 || (r[31] == 1)) ? 1 : 0;
         end
```

```
 Sltu: begin
                r = (a < b) ? 1 : 0;
                negative = r[31];
                r = a - b;
                zero = (r == 0) ? 1 : 0;
                carry = (a < b) ? 1 : 0;
                overflow = 0;
                r = carry;
            end

            Sra: begin
                temp_b = b;
                r = temp_b >>> a;
                zero = (r == 0) ? 1 : 0;
                carry = (a >= 32) ? b[31] : b[a];
                negative = r[31];
                overflow = 0;
            end

             Sll,Slr: begin
                r = b << a;
                zero = (r == 0) ? 1 : 0;
                carry = (a >= 32) ? 0 : b[31-a];
                negative = r[31];
                overflow = 0;
            end

            Srl:  begin
                r = b >> a;
                zero = (r == 0) ? 1 : 0;
                carry = (a >= 32) ? 0 : b[a];
                negative = r[31];
                overflow = 0;
            end
            default: begin
                r = 1'bz; zero = 1'bz; carry = 1'bz; negative = 1'bz;
overflow = 1'bz;
            end
        endcase
endmodule
```

这个模块使用行为型描述，实现了一个算数逻辑单元 ALU，根据指定的操作码 aluc 执行不同的操作。具有两个输入（a 和 b）、一个操作码（aluc）和五个输出（r、zero、carry、negative、overflow）。ALU 根据操作码执行不同的操作，包括加法（有符号和无符号）、减法（有符号和无符号）、位运算（与、或、异或、非）、左右位移、以及比较操作（有符号和无符号），并根据结果更新输出信号以反映运算结果的状态（零、进位、负数、溢出）。

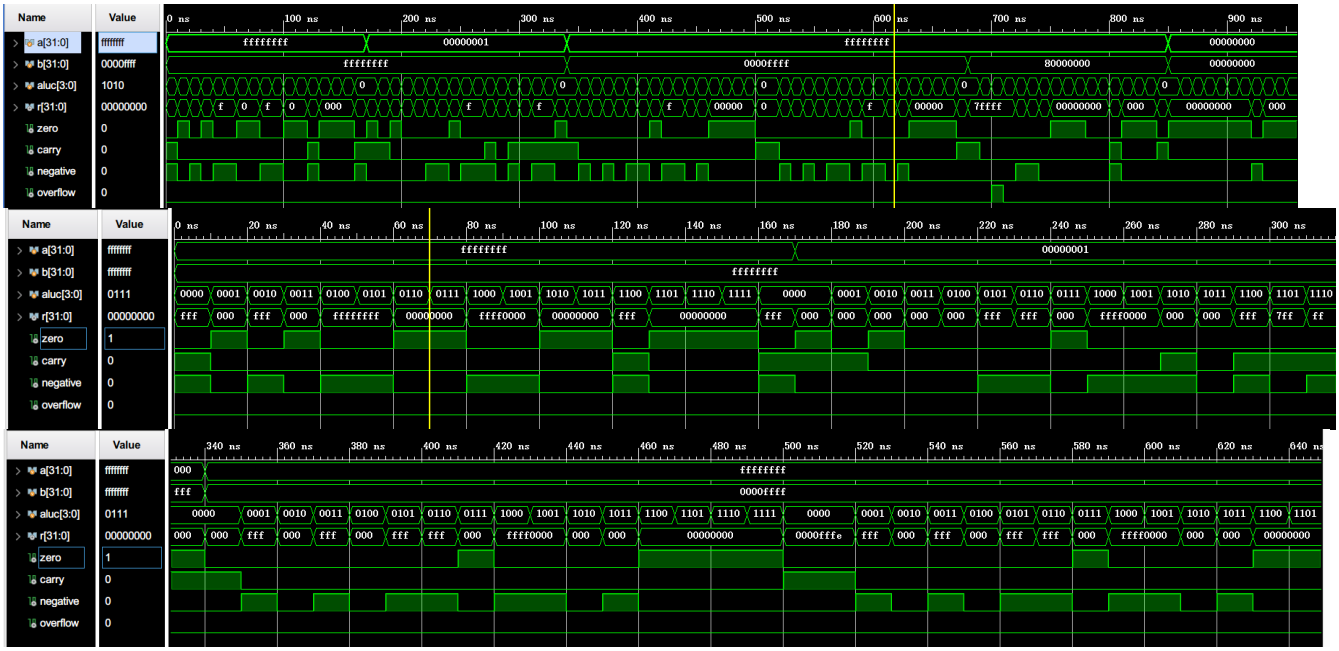
使用了“+”“-”“<>”等运 算符号实现 ALU 中的计算模块

四、 测试模块建模

```
 `timescale 1ns / 1ps
module alu_tb();
reg [31:0] a;
reg [31:0] b;
reg [3:0] aluc;
wire [31:0] r;
wire zero;
wire carry;
wire negative;
wire overflow;

alu uut(
.a(a),
.b(b),
.aluc(aluc),
.r(r),
.zero(zero),
.carry(carry),
.negative(negative),
.overflow(overflow)
);

initial  begin
   a <= 32'hffffffff;
   b <= 32'hffffffff;
   aluc=4'b0000;
    repeat(16) begin
          #10 aluc=aluc+1;
    end

    #10 a=32'h0000_0001;
    b=32'hffff_ffff;
    aluc=4'b0000;
    repeat(16) begin
           #10 aluc=aluc+1;
    end
    #10 a=32'hffff_ffff;
    b=32'h0000_ffff;
    aluc=4'b0000;
    repeat(16) begin
           #10 aluc=aluc+1;
    end
     #10 a=32'hffff_ffff;
    b=32'h8000_0000;
    aluc=4'b0000;
    repeat(16) begin
           #10 aluc=aluc+1;
    end
     #10 a=32'h0000_0000;
    b=32'h0000_0000;
    aluc=4'b0000;
    repeat(16) begin
           #10 aluc=aluc+1;
    end
    #10 a=32'h0000_0008;
    b=32'hffff_ffff;
    aluc=4'b0000;
    repeat(16) begin
           #10 aluc=aluc+1;
    end
    #10 a=32'h0000_ffff;
    b=32'h0000_0000;
    aluc=4'b0000;
    repeat(16) begin
               #10 aluc=aluc+1;
    end
end
endmodule
```

五、 实验结果

- 波形图仿真


Name
Value
a[31:0]
b[31:0]
aluc[3:0]
r[31:0]
zero
carry
negative
overflow
ffffffff
0000ffff
1010
00000000
0
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
00000001
0000ffff
80000000
7fffff
000
f

Name
Value
a[31:0]
b[31:0]
aluc[3:0]
r[31:0]
zero
carry
negative
overflow
ffffffff
0111
00000000
1
0
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
240 ns
260 ns
280 ns
300 ns
00000001
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
fff
000
ffff0000
00000000
7ff
ff

Name
Value
a[31:0]
b[31:0]
aluc[3:0]
r[31:0]
zero
carry
negative
overflow
ffffffff
0111
00000000
1
0
340 ns
360 ns
380 ns
400 ns
420 ns
440 ns
460 ns
480 ns
500 ns
520 ns
540 ns
560 ns
580 ns
600 ns
620 ns
640 ns
0000ffff
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
000
fff
ffff0000
00000000
0000ffffe