

# 《数字逻辑与数字系统》实验报告

学院 计算机科学与技术学院 年级 2023 级 班级 4

姓名 张宇晨 学号 3023210076

实验项目名称 算术逻辑单元（ALU）的设计与实现

大模型名称 Claude Sonnet 4.5 问答形式 ☒ 线上 ☐ 本地

是否建立知识库 ☒ 是 ☐ 否

## 一. 实验目的

1. 掌握全加器和行波进位加法器的结构；
2. 熟悉加减法运算及溢出的判断方法；
3. 掌握算术逻辑单元（ALU）的结构；
4. 熟练使用 SystemVerilogHDL 的行为建模和结构化建模方法对 ALU 进行描述实现。

## 二. 实验内容

基于 SystemVerilogHDL 设计并实现一个 32 位 ALU 单元。包括基本逻辑运算、移位运算、加法减法运算、乘法运算和比较运算。

## 三. 实验原理与步骤（请给出所有提示语和回答）

问：按照 task.md 的要求进行实现

注（这不是提问的一部分）：task.md 在项目文件中。具体内容如下图所示。

#### task.md

```
1  目标：使用 SystemVerilog 完成基本 ALU 功能。项目源代码在 ALU_32bits.srcs 目录下。
2
3  端口说明：文件 alu.sv 的 A、B 端口为操作数，aluop 为操作选择符，alures 为计算结果，ZF 是零标志位，所有运算都要
   设置 ZF 标志位，OF 是溢出标志位，有符号数加法和有符号数减法需要设置 OF 标志位，其余运算默认设置 OF 为 0。
4
5  具体步骤：
6
7  1. 在文件 fulladder.sv 中实现全加器
8  2. 在文件 rca.sv 中，调用全加器，实现 32 位行波进位加法器
9  3. 在文件 alu.sv 中，通过调用行波进位加法器和添加必要的逻辑电路，实现以下运算：
10
11     1. aluop 为 0000 时，实现按位与
12     2. aluop 为 0001 时，实现按位或
13     3. aluop 为 0010 时，实现按位异或
14     4. aluop 为 0011 时，实现按位与非
15     5. aluop 为 0100 时，实现对 A 的逻辑非
16     6. aluop 为 0101 时，实现对 A 的逻辑左移，其中 B 取低 5 位
17     7. aluop 为 0110 时，实现对 A 的逻辑右移，其中 B 取低 5 位
18     8. aluop 为 0111 时，实现对 A 的算术右移，其中 B 取低 5 位
19     9. aluop 为 1000 时，实现无符号数乘法
20     10. aluop 为 1001 时，实现有符号数乘法
21     11. aluop 为 1010 时，实现有符号数加法
22     12. aluop 为 1011 时，实现无符号数加法
23     13. aluop 为 1100 时，实现有符号数减法
24     14. aluop 为 1101 时，实现无符号数减法
25     15. aluop 为 1110 时，实现有符号数比较，alures=(A < B ? 1 : 0)
26     16. aluop 为 1111 时，实现无符号数比较
27
28  注意：ALU 单元的输入 A 和 B 均是补码形式。实现加法和减法时，不能使用 + 和 - 两种运算符，只能通过一个行波进位加法
   器和其它必要的逻辑电路实现。
29
```

---

```
module fulladder(
```

```
    input wire a,
    input wire b,
    input wire cin,
    output wire sum,
    output wire cout
);

// combinational full adder
assign sum = a ^ b ^ cin;
assign cout = (a & b) | (a & cin) | (b & cin);
```

```
endmodule
```

---

```
module rca(
```

```
    input wire [31:0] a,
    input wire [31:0] b,
    input wire        cin,
    output wire [31:0] sum,
    output wire        cout
```

```

);

// carry chain: c[0] is cin, c[32] is final carry-out
wire [32:0] c;
assign c[0] = cin;

genvar i;
generate
    for (i = 0; i < 32; i = i + 1) begin : FA_LOOP
        fulladder fa(
            .a  (a[i]),
            .b  (b[i]),
            .cin (c[i]),
            .sum (sum[i]),
            .cout(c[i+1])
        );
    end
endgenerate

assign cout = c[32];

endmodule
-----
module alu(
    input      [31 : 0] A,
    input      [31 : 0] B,
    input      [3  : 0] aluop,
    output logic [31 : 0] alures,
    output logic          ZF,
    output logic          OF
);

    localparam AND    = 4'b0000;
    localparam OR     = 4'b0001;
    localparam XOR    = 4'b0010;
    localparam NAND   = 4'b0011;
    localparam NOT    = 4'b0100;
    localparam SLL    = 4'b0101;
    localparam SRL    = 4'b0110;
    localparam SRA    = 4'b0111;
    localparam MULU   = 4'b1000;
    localparam MUL    = 4'b1001;
    localparam ADD    = 4'b1010;
    localparam ADDU   = 4'b1011;

```

```

localparam SUB    = 4'b1100;
localparam SUBU   = 4'b1101;
localparam SLT    = 4'b1110;
localparam SLTU   = 4'b1111;

// TODO: Finish the ALU_32bits design
// Internal wires for arithmetic using rca

wire [31:0] sum_add; // A + B
wire        cout_add;
wire [31:0] sum_add_bneg; // A + (~B) + 1 (used for subtraction)
wire        cout_add_bneg;

// instantiate rca for addition (A + B)
rca u_rca_add(
    .a    (A),
    .b    (B),
    .cin  (1'b0),
    .sum  (sum_add),
    .cout (cout_add)
);

// instantiate rca for A + (~B) + 1 (used for subtraction)
rca u_rca_sub(
    .a    (A),
    .b    (~B),
    .cin  (1'b1),
    .sum  (sum_add_bneg),
    .cout (cout_add_bneg)
);

always_comb begin
    ZF = 1'b0;
    OF = 1'b0;
    alures = 32'b0;
    case (aluop)
        AND: begin
            alures = A & B;
            OF = 1'b0;
        end
        OR: begin
            alures = A | B;
            OF = 1'b0;
        end
    end
end

```

```

XOR: begin
    alures = A ^ B;
    OF = 1'b0;
end
NAND: begin
    alures = ~(A & B);
    OF = 1'b0;
end
NOT: begin
    alures = ~A;
    OF = 1'b0;
end
SLL: begin
    alures = A << B[4:0];
    OF = 1'b0;
end
SRL: begin
    alures = A >> B[4:0];
    OF = 1'b0;
end
SRA: begin
    alures = $signed(A) >>> B[4:0];
    OF = 1'b0;
end
MULU: begin
    alures = ($unsigned(A) * $unsigned(B))[31:0];
    OF = 1'b0;
end
MUL: begin
    alures = ($signed(A) * $signed(B))[31:0];
    OF = 1'b0;
end
ADD: begin // signed addition using rca
    alures = sum_add;
    // signed overflow: when A and B have same sign and result
has different sign
    OF = (A[31] & B[31] & ~sum_add[31]) | (~A[31] & ~B[31] &
sum_add[31]);
end
ADDU: begin // unsigned add
    alures = sum_add;
    OF = 1'b0;
end
SUB: begin // signed subtraction A - B using rca (A + ~B + 1)

```

```

        alures = sum_add_bneg;
        // signed overflow on subtraction: when A and B have
different signs and result sign differs from A
        OF = (A[31] & ~B[31] & ~sum_add_bneg[31]) | (~A[31] & B[31]
& sum_add_bneg[31]);
    end
    SUBU: begin // unsigned subtraction
        alures = sum_add_bneg;
        OF = 1'b0;
    end
    SLT: begin // signed compare
        alures = ($signed(A) < $signed(B)) ? 32'd1 : 32'd0;
        OF = 1'b0;
    end
    SLTU: begin // unsigned compare
        alures = (A < B) ? 32'd1 : 32'd0;
        OF = 1'b0;
    end
    default: begin
        alures = 32'b0;
        OF = 1'b0;
    end
endcase

// Zero flag for all operations: set when alures == 0
ZF = (alures == 32'b0);
end
endmodule

```

---

问: alu.sv 的第 93 行和第 97 行在 vivado 中会报错: Syntax error near "["

---

```

...
    // 64-bit products for multiplication (avoid slicing expressions
directly)
    wire [63:0] prod_u;
    wire signed [63:0] prod_s;

    assign prod_u = $unsigned(A) * $unsigned(B);
    assign prod_s = $signed(A) * $signed(B);
...
    MULU: begin
        alures = prod_u[31:0];
    end

```

```

        OF = 1'b0;
    end
    MUL: begin
        alures = prod_s[31:0];
        OF = 1'b0;
    end
endmodule

```

---

问：接着在 ALU\_32bits.srcs/sim\_1 目录下的 ALU\_32bits\_tb.sv 文件中，使用 SystemVerilog 完成对 alu.sv 的完整的自动测试

---

```

module ALU_32bits_tb;

    reg [31:0] A;
    reg [31:0] B;
    reg [3:0] aluop;
    wire [31:0] alures;
    wire ZF;
    wire OF;

    // instantiate DUT
    alu uut(
        .A      (A),
        .B      (B),
        .aluop   (aluop),
        .alures  (alures),
        .ZF      (ZF),
        .OF      (OF)
    );

    integer total_tests;
    integer passed_tests;

    // helper task: run single test and compare expected values
    task run_test(input [31:0] ta, input [31:0] tb, input [3:0] top, input
string name);
        reg [31:0] exp;
        reg expZF;
        reg expOF;
        reg signed [31:0] sA;
        reg signed [31:0] sB;
        reg signed [63:0] sprod;
        reg [63:0] uprod;
        begin

```

```

    A = ta; B = tb; aluop = top;
    #1; // wait for combinational logic to settle

    // default
    exp = 32'h00000000;
    expZF = 1'b0;
    expOF = 1'b0;

    case (top)
        4'b0000: begin exp = ta & tb; expOF = 1'b0; end // AND
        4'b0001: begin exp = ta | tb; expOF = 1'b0; end // OR
        4'b0010: begin exp = ta ^ tb; expOF = 1'b0; end // XOR
        4'b0011: begin exp = ~(ta & tb); expOF = 1'b0; end // NAND
        4'b0100: begin exp = ~ta; expOF = 1'b0; end // NOT A
        4'b0101: begin exp = ta << tb[4:0]; expOF = 1'b0; end //
SLL
        4'b0110: begin exp = ta >> tb[4:0]; expOF = 1'b0; end //
SRL
        4'b0111: begin exp = $signed(ta) >>> tb[4:0]; expOF = 1'b0;
end // SRA
        4'b1000: begin uprod = $unsigned(ta) * $unsigned(tb); exp
= uprod[31:0]; expOF = 1'b0; end // MULU
        4'b1001: begin sprod = $signed(ta) * $signed(tb); exp =
sprod[31:0]; expOF = 1'b0; end // MUL (signed)
        4'b1010: begin // signed add
            exp = ta + tb;
            // signed overflow when signs of a and b same and result
sign differs
            expOF = (ta[31] & tb[31] & ~exp[31]) | (~ta[31] &
~tb[31] & exp[31]);
            end
        4'b1011: begin exp = ta + tb; expOF = 1'b0; end // unsigned
add
        4'b1100: begin // signed sub (A - B)
            exp = ta - tb;
            // signed overflow when signs of a and b different and
result sign differs from a
            expOF = (ta[31] & ~tb[31] & ~exp[31]) | (~ta[31] &
tb[31] & exp[31]);
            end
        4'b1101: begin exp = ta - tb; expOF = 1'b0; end // unsigned
sub
        4'b1110: begin exp = ($signed(ta) < $signed(tb)) ? 32'd1 :
32'd0; expOF = 1'b0; end // SLT

```



```

        4'b1111: begin exp = (ta < tb) ? 32'd1 : 32'd0; expOF =
1'b0; end // SLTU
        default: begin exp = 32'h0; expOF = 1'b0; end
    endcase

    expZF = (exp == 32'h0);

    total_tests = total_tests + 1;

    if (alures === exp && ZF === expZF && OF === expOF) begin
        passed_tests = passed_tests + 1;
        $display("[PASS] %s A=%h B=%h op=%b -> res=%h ZF=%b OF=%b",
name, ta, tb, top, alures, ZF, OF);
    end else begin
        $display("[FAIL] %s A=%h B=%h op=%b -> got res=%h ZF=%b
OF=%b, expected res=%h ZF=%b OF=%b", name, ta, tb, top, alures, ZF, OF,
exp, expZF, expOF);
    end
end
endtask

initial begin
    $display("Starting ALU automated test...");
    total_tests = 0;
    passed_tests = 0;

    // Basic logical operations
    run_test(32'hA5A5A5A5, 32'h5A5A5A5A, 4'b0000, "AND");
    run_test(32'hA5A5A5A5, 32'h5A5A5A5A, 4'b0001, "OR");
    run_test(32'hA5A5A5A5, 32'h5A5A5A5A, 4'b0010, "XOR");
    run_test(32'hA5A5A5A5, 32'h5A5A5A5A, 4'b0011, "NAND");
    run_test(32'hFF00FF00, 32'h00000000, 4'b0100, "NOT A");

    // Shifts
    run_test(32'h000000FF, 32'd4, 4'b0101, "SLL by 4");
    run_test(32'h000000FF, 32'd4, 4'b0110, "SRL by 4");
    run_test(32'h80000000, 32'd1, 4'b0111, "SRA negative by 1");

    // Multiplication
    run_test(32'd2, 32'd3, 4'b1000, "MULU small");
    run_test(32'hFFFFFFFF, 32'hFFFFFFFF, 4'b1001, "MUL signed (-1 * -
1)");

    // Addition: normal and overflow

```

```

run_test(32'd100, 32'd23, 4'b1010, "ADD normal signed");
run_test(32'h7FFFFFFF, 32'd1, 4'b1010, "ADD signed overflow");
run_test(32'hFFFFFFF, 32'd1, 4'b1011, "ADDU unsigned wrap
(OF=0)");

// Subtraction: normal and overflow
run_test(32'd50, 32'd20, 4'b1100, "SUB normal signed");
run_test(32'h80000000, 32'd1, 4'b1100, "SUB signed overflow (-2^31
- 1)");
run_test(32'd0, 32'd1, 4'b1101, "SUBU unsigned (wrap)");

// Comparisons
run_test(32'hFFFFFFF, 32'd1, 4'b1110, "SLT (-1 < 1)");
run_test(32'd0, 32'd1, 4'b1111, "SLTU (0 < 1)");

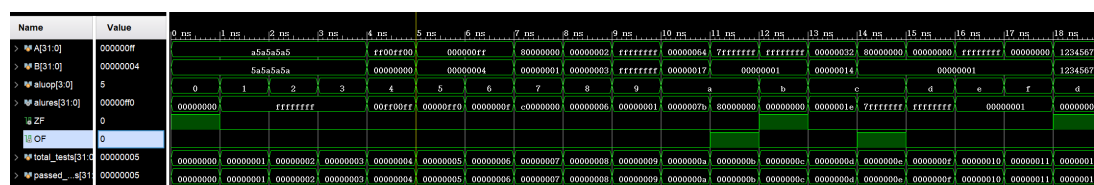
// Zero result check
run_test(32'h12345678, 32'h12345678, 4'b1101, "SUBU result zero
check");

$display("ALU test finished: %0d/%0d passed", passed_tests,
total_tests);
if (passed_tests != total_tests) begin
    $display("Some tests failed. See above for details.");
end
$finish;
end

endmodule

```

四. 仿真与实验结果（注：仿真需要给出波形图截图，截图要清晰，如果波形过长，可以分段截取；实验结果为远程 FPGA 硬件云平台的截图）



从左到右分别测试了按位与、按位或、按位异或……其中还测试了 aluop 等于 0xa 和 0xc 时，即有符号加减法能否正确设置 OF 标志位。最后还测试了无

符号数减法的结果为 0 时能否正确将 ZF 设置为 1。

实际上还缺少以下测试：

1. 0 与 0 的 and 操作，1 与 1 的 and 操作；
2. 0 与 0 的 or 操作，1 与 1 的 or 操作；
3. 0 与 0 的 xor 操作，1 与 1 的 xor 操作；
4. 0 与 0 的 nand 操作，1 与 1 的 nand 操作；
5. 对于逻辑左移，还需测试 B 的高位不为 0 时能否正确取低五位进行左移；
6. 对于逻辑右移，还需测试 B 的高位不为 0 时能否正确取低五位进行右移；
7. 对于算术右移，还需测试 B 的高位不为 0 时能否正确取低五位进行右移；
8. 对于所有移位操作，还需测试移位结果为 0 时能否正确设置 ZF 标志位
9. 对于比较操作，还需测试 A 大于或等于 B 的情况。

提交评测

实验总得分：100.00

Accept | 得分：100 | 2025-11-18 19:13:16

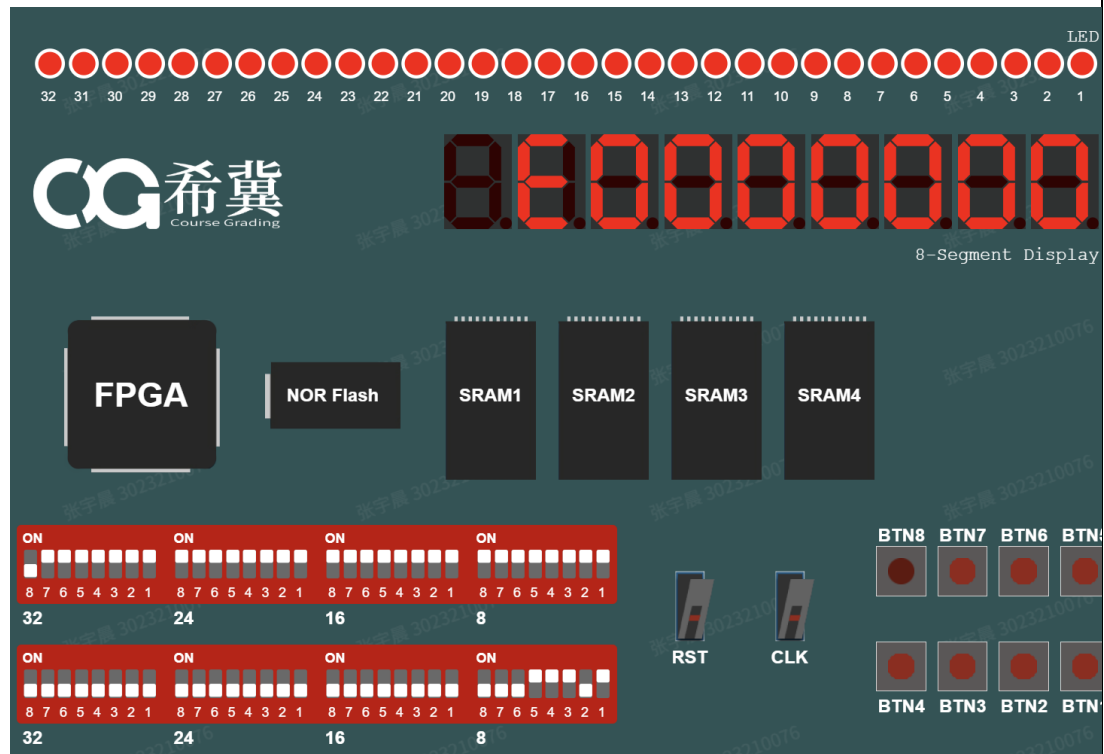
自动评测得分（百分制）：100

— 当前分数是百分制，最后转换为实际得分。

```
-----Test #1-----
srcA: ffffffff, srcB: 12345678, op: and
Expected ---- ans: 12345678, zero: 0, overflow: 0
Yours ----- ans: 12345678, zero: 0, overflow: 0
-----Test #2-----
srcA: 00000000, srcB: 12345678, op: or
Expected ---- ans: 12345678, zero: 0, overflow: 0
Yours ----- ans: 12345678, zero: 0, overflow: 0
-----Test #3-----
srcA: ffffffff, srcB: ffffffff, op: xor
Expected ---- ans: 00000000, zero: 1, overflow: 0
Yours ----- ans: 00000000, zero: 1, overflow: 0
-----Test #4-----
srcA: aaaaaaaa, srcB: 55555555, op: nand
Expected ---- ans: ffffffff, zero: 0, overflow: 0
Yours ----- ans: ffffffff, zero: 0, overflow: 0
-----Test #5-----
srcA: ffffffff, srcB: 00000000, op: not
Expected ---- ans: 00000000, zero: 1, overflow: 0
Yours ----- ans: 00000000, zero: 1, overflow: 0
```

自动评测的结果如上图所示。

在远程平台中，将 `aluop` 设置为 `0111`（即算术右移），`A` 设置为 `0x80000000`，`B` 设置为 `0xFFFFFE2`，也就是将 `A` 算术右移 2 位，结果显示为 `0xE0000000`，且 `ZF` 和 `OF` 均为 0，结果正确。



五. 请评价大模型在本实验中的表现（例如：优势、劣势、使用建议等。）

大模型对工具环境的了解可能与实际略有偏差，但是对于这类简单的问题而言，它完成得非常快而且准确。就目前而言，在提示词简单的情况下，它在写测试的时候可能很难考虑到所有情况，尤其是当一个项目极其复杂时。

六. 附加题

无