浙江大学实验报告

课程名称：数字系统　　任课老师：李宇波

实验名称：Design a ALU　　实验日期：2023/3/27

1 实验目的和要求

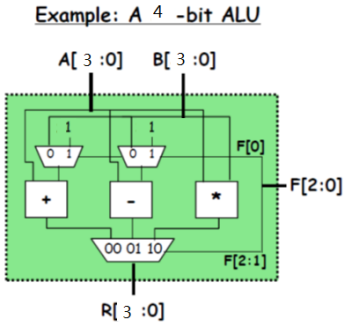
1.1 实验目的

To learn how to use the behavior model Verilog HDL description to design and implement an ALU.

了解如何使用 behavior 模型 Verilog HDL 描述语言来设计和实现 ALU。

1.2 实验要求

Design an ALU with the following functions. 用下列函数设计一个 ALU。



Function Table

F2	F1	F0	Function
0	0	0	A + B
0	0	1	A + 1
0	1	0	A - B
0	1	1	A - 1
1	0	X	A * B

1) Design the modules of 4-bit 2-1 Mux, 4-bit 3-1 Mux, 4-bit Adder, 4-bit Subtracter, and 4-bit Multiplier. You do not need to consider the situation of overflow.

   设计 4 比特 2-1Mux 和 3-1Mux，4 比特加法器，减法器和乘法器的模块。不用考虑溢出的情况。

2) Design the top module for the ALU with hierarchy structure of the modules defined in step 1).

   设计 ALU 的顶层模块，要求使用 step1 定义的子模块。

3) Implement the design in FPGA, with the data input A[3:0], B[3:0], and the control input F[2:0] connected to the switches on Basys3 board, and the output R[3:0] to the LEDs on board.

   在 FPGA 板实现设计，数据输入为 A[3:0], B[3:0]，控制输入为 F[2:0]，通过 Basys3 板的开关连接；输出 R[3:0]通过 LED 实现。

4) (optional) Try to solve the overflow problem in this design with your own method.

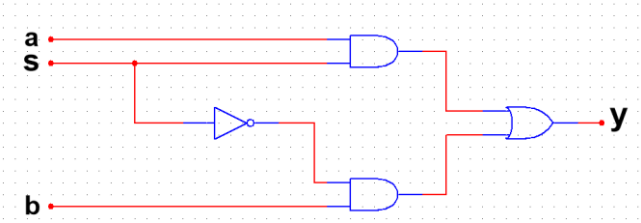
   （可选）尝试解决溢出的问题。

5) (optional) Try to extend the Function table with new functions.

（可选）尝试用新的函数扩展函数表。

2 实现代码

2.1 4bit 2-1 Mux



图一 2-1 Mux 电路图

```
module mux21 (
    input [3:0]d0,
    input [3:0]d1,
    input s,
    output reg [3:0]y
);
    always @(*) begin
        case(s)
        1'b0: y=d0;
        1'b1: y=d1;
        endcase
    end

endmodule
```

2.2 4bit 3-1 Mux

原理和 2-1 Mux 一致。

```
module mux31(
    input [7:0]d0,
    input [7:0]d1,
    input [7:0]d2,
    input [1:0]s,
    output reg [7:0]y
);
    always @(*) begin
        case(s)
        2'b00: y=d0;
        2'b01: y=d1;
        2'b10: y=d2;
        endcase
    end
endmodule
```

2.3 4bit adder

```
module adder (
    input [3:0]a,
    input [3:0]b,
    input c_in,    //set for building multiplier at first, but I found
another way to implement. So, it's useless.
    output [7:0]result,
    output c_out
);

    wire [3:0] t;
    assign {cout,t}=c_in + a + b;
    //keep the same output bits as multiplier.
    assign result = {3'b000, cout, t};

endmodule
```

事实上，不需要定义 c_out 变量，因为输出是 8bit，完全可以把 c_out 的任务交给 t 完成。但是 c_out 可以留作进位（溢出）提示，为后续功能升级提供便利，因此笔者并未删除。

2.4 4bit subtractor

```
module subtractor (
    input [3:0]a,
    input [3:0]b,
    output [3:0]result,
    output cout
);

    wire [3:0] t;
    assign {cout,t}=a - b;
    //keep the same output bits as multiplier.
    assign result = {3'b000, cout, t};

endmodule
```

减法器的实现和加法器如出一辙。

2.5 4bit multiplier

```
module multiplier (
    input [3:0]a,
    input [3:0]b,
    output reg [7:0] result
);
    integer i;
    reg [7:0] t;
    reg [7:0] bt;
```

```
    always @(*)
    begin
        t = 8'b00000000;
        bt = {4'b0000, b};
        for(i = 0; i < 4; i = i + 1)
            begin
                if(a[i] == 1)
                    t = t + bt;
                bt = {bt[6:0], 1'b0};
            end
        result = t;
    end
endmodule
```

2.6 implementation of ALU

```
module ALU(
    input [3:0] A,
    input [3:0] B,
    input [2:0] F,
    output [7:0] R
);
    wire [3:0] y_mux;
    wire [7:0] r_add, r_sub, r_mul;
    //select B or 1 depends on F[0]
    mux21 mux211(.d0(B), .d1(4'b0001), .s(F[0]), .y(y_mux));

    //compute answers.
    adder a1(.a(A), .b(y_mux), .c_in(0), .result(r_add));
    subtractor s1(.a(A), .b(y_mux), .result(r_sub));
    multiplier m1(.a(A), .b(B), .result(r_mul));

    //select an answer above depends on F[2]F[1].
    mux31 mux311(.d0(r_add), .d1(r_sub), .d2(r_mul), .s({F[2],
F[1]}), .y(R));

endmodule
```