《数字逻辑与数字系统》实验报告

学院 计算机科学与技术 年级 2022 级 班级 4 班

姓名 陆子毅 学号 3022206045

实验项目名称 算术逻辑单元 ALU 的设计与实现

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 SystemVerilog HDL 的行为建模和结构化建模方法对 ALU 进行描述实现；

5. 为“单周期 MIPS 处理器的设计与实现”奠定基础。

二. 实验内容

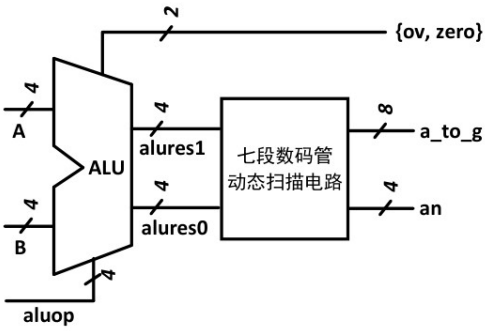


图 2-4 实验的顶层模块

基于 SystemVerilog HDL 设计并实现一个 4 位 ALU 单元。整个工程的顶层模块如图 2-4 所示，输入/输出端口如表 2-2 所示。

满足以下要求：

1. ALU 单元的输入 A 和 B 均是补码形式。

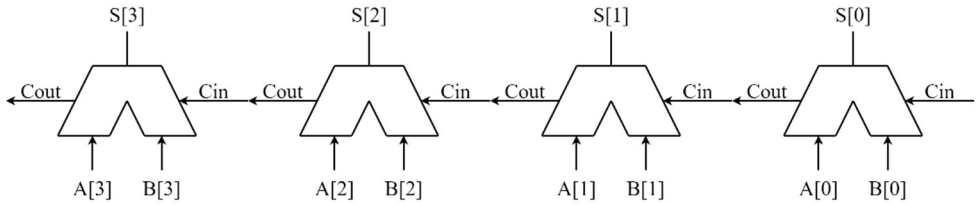
2. 实现加法和减法时，不能使用“+”和“-”两种运算符，且只能通过一个行波进位加法器和其它必要的逻辑电路实现。

3. 可以使用“*”运算符实现乘法，但该运算符在只适用无符号数的乘法，有符号数的乘法需要同学们考虑如何处理。

4. 实现算术右移时，可以使用运算符“>>>”。

三. 实验原理与步骤（注：不用写工具的操作步骤，而是设计步骤）

1. 画出实现加/减法运算的逻辑电路原理图，并说明为什么加/减法可以只使用一个加法器进行实现？



使用多个全加法器构建四位行波进位加法器。

加法器通过逐位相加，处理进位和借位来实现两个数的加法。

减法能够使用加法器运算的原理是将第二个操作数编程相反数然后与第一个数相加，这样减法就可以使用加法器来运算了，减法的本质是加上一个负数。

实现步骤，先将减数的每一位取反，然后+1，得到减数的相反数与被减数相加得到结果。

2. 给出有符号数加/减法溢出的判断规则？

有符号数加法判断溢出：（1）两个很大的数相加溢出，会出现得到的结果是负数。（2）两个很小的负数相加，得到正数，或者两个很小的负数相加得到一个正数。

减法同理。

用 SystemVerilog 语言表示：

加法：A[3] == B[3] && A[3] != S[3]

减法：A[3] == ~B[3] && A[3] != S[3]

3. 给出 ALU 单元的 SystemVerilog HDL 代码。

```
module alu(
    input [3:0] A,
    input [3:0] B,
    input [3:0] aluop,
    output [3:0] alures0,
    output [3:0] alures1,
    output logic ov,
    output logic zero
);
    reg [3:0] r_alures0;
    reg [3:0] r_alures1;

    assign alures0 = r_alures0;
    assign alures1 = r_alures1;
    wire [3:0] S;
    wire [3:0] Cout;

    reg [3:0]a,b;
    assign a=A;
    rca rca0(.A(a), .B(b), .S(S), .Cout(Cout));

    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;

always @(*) begin

      if(aluop==SUBU ||aluop==SUB)
            b=~B+1;
      else
            b=B;
end



logic [3:0] A_mux,B_mux;
logic [7:0] Prod_tmp;
logic [7:0] Prod;
assign A_mux = (A[3])?(~A+1):A;
assign B_mux = (B[3])?(~B+1):B;
assign Prod_tmp = A_mux * B_mux;
assign Prod = (A[3]^B[3])?(~Prod_tmp+1):(Prod_tmp);



// Logic for setting zero and overflow flags
```

```
// Select operation based on aluop
always_comb begin
    ov= 1'b0;
    case(aluop)
        AND: begin
            r_alures0 = A & B;
            r_alures1 = 4'b0000;
        end
        OR: begin
            r_alures0 = A | B;
            r_alures1 = 4'b0000;
        end
        XOR: begin
            r_alures0 = A ^ B;
            r_alures1 = 4'b0000;
        end
        NAND: begin
            r_alures0 = ~(A & B);
            r_alures1 = 4'b0000;
        end
        NOT: begin
            r_alures0 = ~A;
            r_alures1 = 4'b0000;
        end
        SLL: begin
            r_alures0 = A << B[1:0];
            r_alures1 = 4'b0000;
        end
        SRL: begin
```

```
                    r_alures0 = A >> B[1:0];
                    r_alures1 = 4'b0000;
                end
                SRA: begin
                    r_alures0 = $signed(A) >>> B[1:0];
                    r_alures1 = 4'b0000;
                end
                MULU: {r_alures1,r_alures0} = A * B;
                MUL:    {r_alures1,r_alures0} = Prod;
                ADD: begin
                    r_alures0 = S;
                    r_alures1 = 4'b0000;
                    ov =    A[3] == B[3] && A[3] != S[3];
                end
                ADDU: begin r_alures0 = S; r_alures1 = 4'b0000; end
                SUB: begin
                    r_alures0 = S;
                    ov = A[3] == ~B[3] &&    A[3] != S[3];
                end
                SUBU: r_alures0 = S;
                SLT: r_alures0 = A[3]?(B[3]?(A<B?1:0):1):(B[3]?0:(A<B?1:0));
                SLTU: r_alures0 = (A < B) ? 1 : 0;
                // Add cases for other operations based on the table provided
                default: begin r_alures1 = 4'b0000; r_alures0 = A; end // Default case,
can be modified
            endcase
        end
        assign zero = (r_alures1 == 4'b0000);
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）

注：远程 FPGA 硬件云平台截图只需要一个测试激励即可

给出具有自动化测试功能的仿真程序和对应的波形图截图，并说明为什么选取这些测试向量？

按位与，按位或，按位异或，按位与非，逻辑非等逻辑运算使用 4’b1100 和 4’b1010 作为测试向量，涵盖了四种可能的情况。可以比较全面地测试运算是否正确。

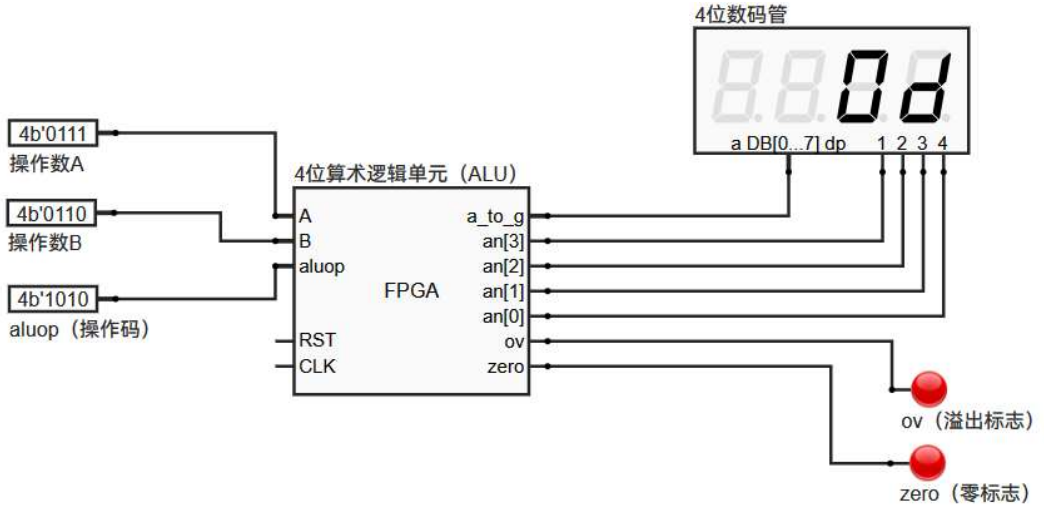
无符号数乘法使用 4’b1111 和 4’b1111，查看结果的高位是否也有结果。

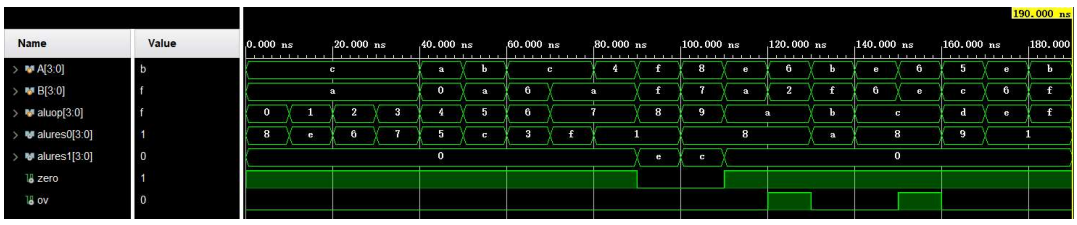
有符号数乘法使用 4’b1000 和 4’b0111，检验结果是否产生预期的负数。

有符号数加法使用 4’b1110 和 4’b1010，检查负数溢出是否正确判断。

使用 4’b0110 和 4’b0010 判断正数是否溢出。

有符号数减法，被减数使用有符号数加法的取反+1。





五．实验中遇到的问题和解决办法

ov 位的设置，判断，重新翻阅了计算机系统那本书以后，不断修改得到。

有符号数和无符号数减法的时候，如何将被减数变为~B+1 然后实例化一个行波进位加法器实现。加入 assign 语句，检测到操作是 SUB 或者 SUBU 时候，将实际交给实例化的数据 b 赋值为~B+1。同时，解决了只使用一个 rca 实例表示加减法的问题。

模块代码编写完成以后，生成 bin 文件时遇到 vivado 报错，最后根据错误信息，在 xcd 文件中加入了

set_property SEVERITY {Warning} [get_drc_checks NSTD-1]

set_property SEVERITY {Warning} [get_drc_checks UCIO-1]

忽略错误解决。

生成的 bin 文件烧写到远程平台后，运行实验发现没有输出。在老师指导下，检查了模块的接口，发现定义的 alu 与调用 alu 的接口参数不一致。

由于仿真时将 Top 模块设置为 alu，所以在生成比特流的时候，文件名也是 alu.bin，并不是最后提交到远程平台的文件。将 Top 模块更改为 ALU_4bits 后解决。

六．附加题

无