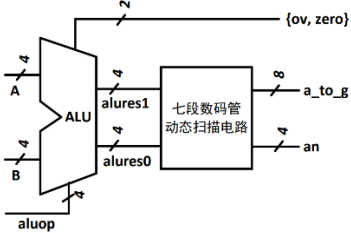


学部/院	智能与计算学部	年级	2021	班级	软工 3 班
姓名	陈昊昆	学号	3021001196	实验日期	4.25
<div>实验项目名称 算术逻辑单元（ALU）的设计与实现</div> <div><div>一. 实验目的</div><div>1. 掌握全加器和行波进位加法器的结构；</div><div>2. 熟悉加减法运算及溢出的判断方法；</div><div>3. 掌握算术逻辑单元（ALU）的结构；</div><div>4. 熟练使用 SystemVerilog HDL 的行为建模和结构化建模方法对 ALU 进行 描述实现；</div><div>5. 为“单周期 MIPS 处理器的设计与实现”奠定基础。</div></div> <div><div>二. 实验内容</div><div></div><div>图 2-4 实验的顶层模块</div><div>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。注意，顶层模块由两个子模块组成，其中，一个是 ALU 单元，另一个是 7 段数码管动态显示扫描单元。</div></div>					
<div>三 . 实验原理与步骤（注：步骤不用写工具的操作步骤，而是设计步骤）</div> <div>1. 写出全加器 fulladder.sv 代码。</div> <div><pre>module fulladder(     input  A,     input  B,     input  Cin,     output S,     output Cout );      assign S = A ^ B ^ Cin;     assign Cout = A &amp; B   (A ^ B) &amp; Cin;  endmodule</pre></div> <div>2. 给出行波进位加法器 rsa.sv 的代码。</div> <div><pre>module rca(     input  [3 : 0] A,     input  [3 : 0] B,     input          Cin,     output [3 : 0] S,     output          Cout );      logic [2 : 0] out;     fulladder f1(A[0], B[0], Cin, S[0], out[0]);     fulladder f2(A[1], B[1], out[0], S[1], out[1]);     fulladder f3(A[2], B[2], out[1], S[2], out[2]);     fulladder f4(A[3], B[3], out[2], S[3], Cout);  endmodule</pre></div>					

3. 写出 ALU 模块 alu.sv 代码。

```
module alu(  
    input [3 : 0] A,  
    input [3 : 0] B,  
    input [3 : 0] aluop,  
    output logic [7 : 0] alures,  
    output logic ZF,  
    output logic OF  
);  
  
logic [3 : 0] alures0, alures1, alures4;  
logic [7 : 0] alures2, alures3;  
logic [3 : 0] b;  
logic temp, cin;  
always_comb begin  
    cin = 0;  
  
    if(aluop == 4'b1010) begin  
        b = B;  
    end  
  
    if(aluop == 4'b1011) begin  
        b = B;  
    end  
  
    if(aluop == 4'b1100) begin  
        b = ~B;  
        cin = 1;  
    end  
  
    if(aluop == 4'b1101) begin  
        b = ~B;
```

```
        cin = 1;  
    end  
  
    if(aluop == 4'b1110) begin  
        if(A[3] == B[3]) begin  
            b = ~B;  
            cin = 1;  
        end  
    end  
  
    end  
  
    rca adder(A, b, cin, alures4, temp);  
  
    always_comb begin  
  
        if(aluop == 4'b0000) begin  
            alures0 = A & B;  
            alures1 = 0;  
        end  
  
        if(aluop == 4'b0001) begin  
            alures0 = A | B;  
            alures1 = 0;  
        end  
  
        if(aluop == 4'b0010) begin  
            alures0 = A ^ B;  
            alures1 = 0;  
        end  
  
        if(aluop == 4'b0011) begin  
            alures0 = ~(A & B);
```

```
    alures1 = 0;
end

if(aluop == 4'b0100) begin
    alures0 = ~A;
    alures1 = 0;
end

if(aluop == 4'b0101) begin
    alures0 = A << B[2 : 0];
    alures1 = 0;
end

if(aluop == 4'b0110) begin
    alures0 = A >> B[2 : 0];
    alures1 = 0;
end

if(aluop == 4'b0111) begin
    alures0 = $signed(A) >>> B[2 : 0];
    alures1 = 0;
end

if(aluop == 4'b1000) begin
    alures2 = A * B;
    alures1 = alures2[7 : 4];
    alures0 = alures2[3 : 0];
end

if(aluop == 4'b1001) begin
    logic [3 : 0] A_orig, B_orig;
    A_orig = (A[3] == 1) ? ~A + 1 : A;
```

```
    B_orig = (B[3] == 1) ? ~B + 1 : B;

    alures2 = A_orig * B_orig;
    alures3 = (A[3] == 1) ^ (B[3] == 1) ? ~alures2 + 1 : alures2;

    alures1 = alures3[7 : 4];
    alures0 = alures3[3 : 0];
end

OF = 0;
if(aluop == 4'b1010) begin
    OF = ((A[3] == b[3]) && (A[3] != alures4[3])) ? 1 : 0;
    alures0 = alures4;
    alures1 = 0;
end

if(aluop == 4'b1011) begin
    alures0 = alures4;
    alures1 = 0;
end

if(aluop == 4'b1100) begin
    OF = ((A[3] != B[3]) && (A[3] != alures4[3])) ? 1 : 0;
    alures0 = alures4;
    alures1 = 0;
end

if(aluop == 4'b1101) begin
    alures0 = alures4;
    alures1 = 0;
end
```

```
if(aluop == 4'b1110) begin
    alures1 = 0;
    if(A[3] == 1 && B[3] == 0) alures0 = 1;
    if(A[3] == 0 && B[3] == 1) alures0 = 0;
    if(A[3] == B[3]) begin
        if(alures4[3] == 0) alures0 = 0;
        else alures0 = 1;
    end
end

if(aluop == 4'b1111) begin
    alures0 = (A < B)? 1 : 0;
    alures1 = 0;
end

if(alures0 == 0 && alures1 == 0) ZF = 1;
else ZF = 0;

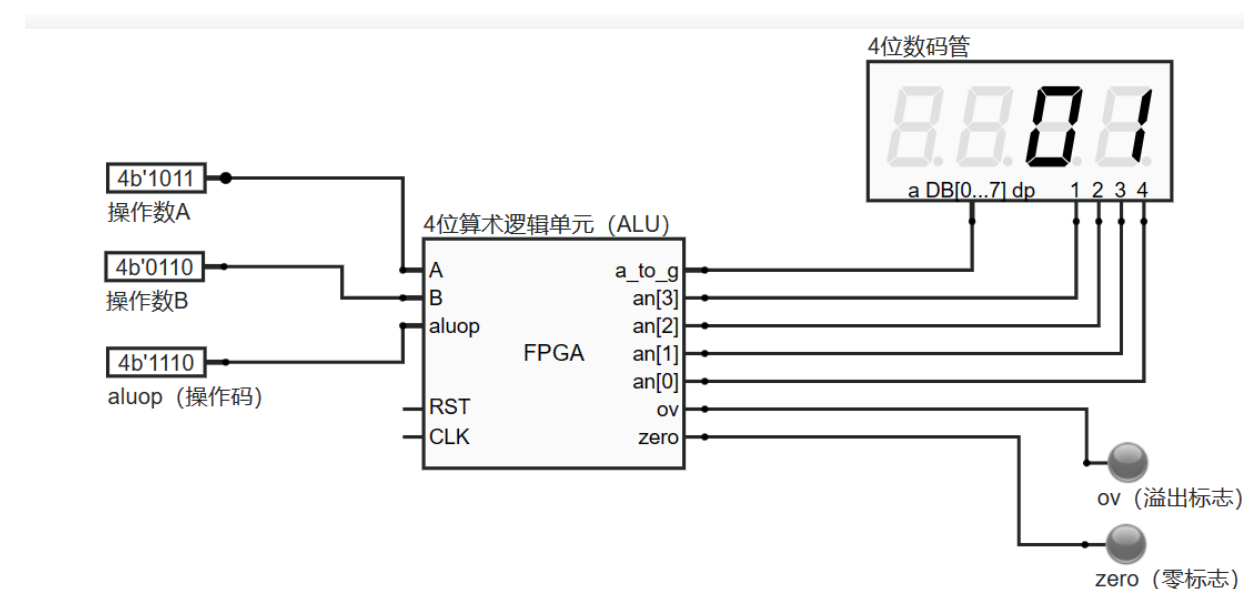
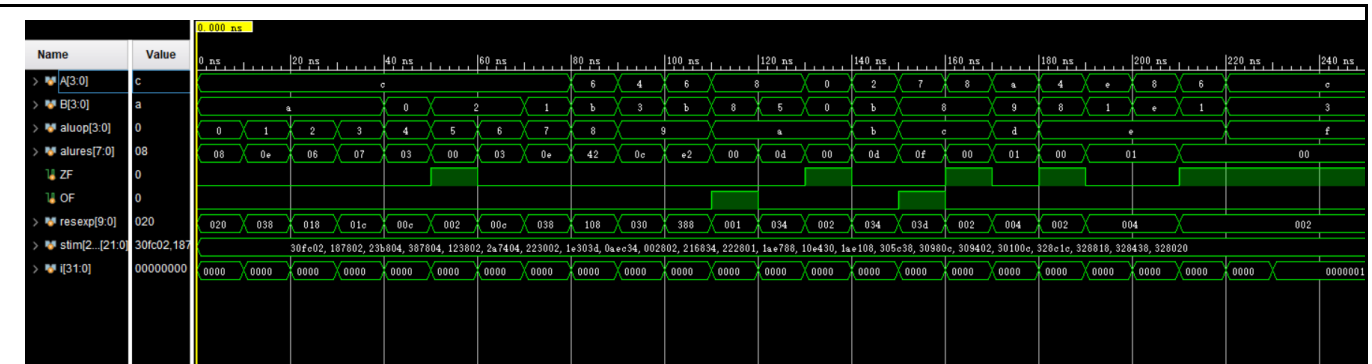
if(OF == 1) ZF = 0;
end

assign alures = {alures1, alures0};

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



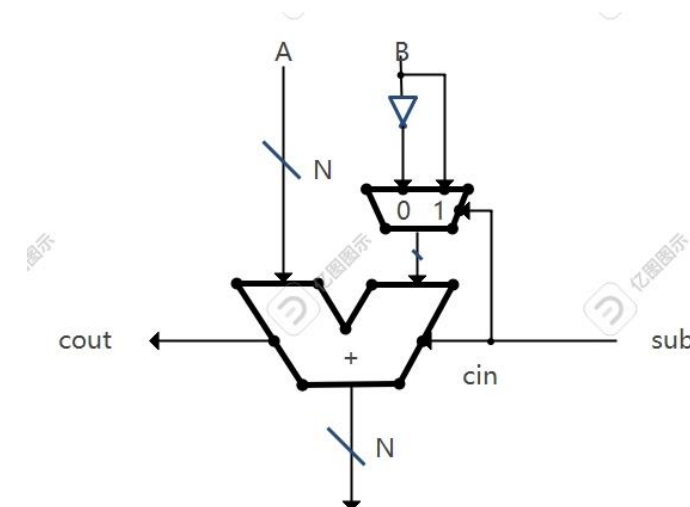
## 五 . 实验中遇到的问题和解决办法

问题：实现时报错，alures0 和 alures1 都显示多元驱动

解决：不能在两个行为建模的模块中都出现对 alures0 和 alures1 的赋值，将其都合并到一个行为建模中

六．附加题

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？



减法的补码为  $[A - B]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}} = [\overline{A}]_{\text{补}} + [B]_{\text{补}} + 1$

所以减法可以由加法器实现

2. 给出有符号数加/减法溢出的判断规则？

有符号加法溢出：同号相加，结果异号

有符号减法溢出：异号相减，结果与被减数异号

3. 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？

module ALU\_4bits\_tb(

);

logic [3:0] A;

logic [3:0] B;

logic [3:0] aluop;

logic [7:0] alures;

logic ZF;

logic OF;

logic [9 : 0] resexp;

logic [21 : 0] stim [22 : 0];

int i;

alu dut (

.A(A),

.B(B),

.aluop(aluop),

.alures(alures),

.ZF(ZF),

.OF(OF)

);

initial begin

\$readmemb

("C:/Users/hkhk3/Desktop/ALU\_4bits\_stu/ALU\_4bits/ALU\_4bits.sim/sim\_1/behav/xsim/test.txt"

stim);

for(i = 0; i < 23; i = i + 1) begin

{A, B, aluop, resexp} = stim[i]; #10;

if ({alures, ZF, OF} == resexp)

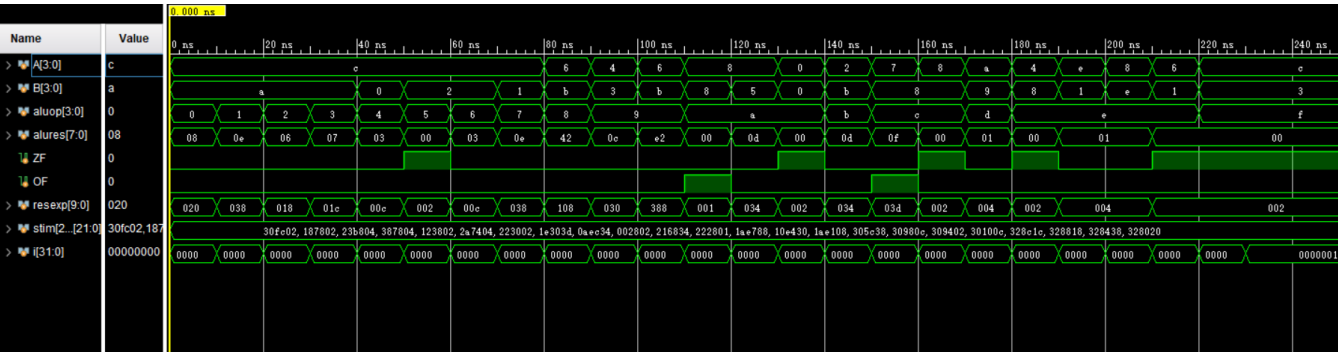
\$display (\$time, "test pass!");

else

\$display (\$time, "Error : inputs = %b[4] %b[4] %b[4]", A, B, aluop);

end

```
end
endmodule
```



选取原因：选取了比较典型的例子，其中带有有符号溢出的情况，有符号不溢出的情况，结果未来的情况，结果不为零的情况等等。

教师签字：

年 月 日