# 《数字逻辑与数字系统》实验报告

学院 智算学部 年级 2019 级 班级 计科一班 姓名 李润泽 学号 3019244266

课程名称 数字逻辑与数字系统 实验日期 2021.04.27 成绩_____

同组实验者_____

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

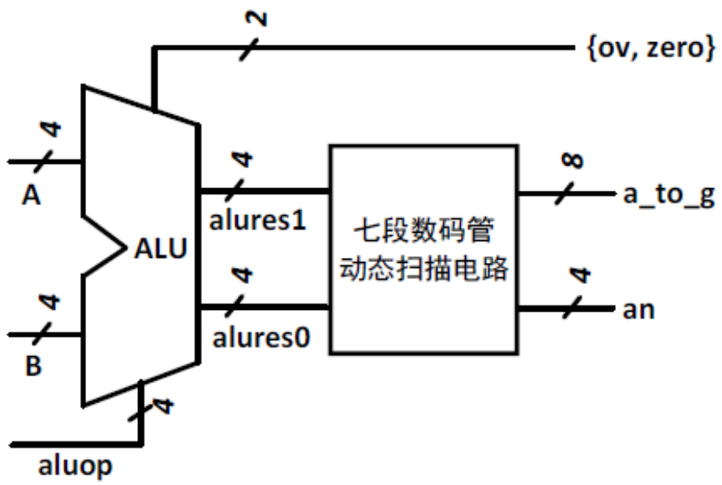



图 2-4 实验的顶层模块

# 天津大学本科生实验报告专用纸

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 端数码管动态显示扫描单元。同学们只需要实现 ALU 单元即可，动态显示扫描单元在工程中直接提供。

表 2-2 输入/输出端口

| 端口名 | 方向 | 宽度（位） | 说明 |
|---|---|---|---|
| A | 输入 | 4 | 操作数 A。 |
| B | 输入 | 4 | 操作数 B。 |
| aluop | 输入 | 4 | 操作类型。 |
| ov | 输出 | 1 | 有符号数加/减法溢出标志。 |
| zero | 输出 | 1 | 零标志（对所有操作均有效）。 |
| a_to_g | 输出 | 8 | 连接七段数码管的数据输入端 CA-CG 和 dp 段，用于显示 ALU 单元的运算结果。 |
| an | 输出 | 4 | 连接 4 个七段数码管的使能端，其中 an[0]对应的数码管显示非乘法结果或乘法结果的低 4 位，an[1]对应的数码管显示乘法结果的高 4 位。剩余两个数码管不点亮。 |

ALU 单元所支持的运算功能如表 2-3 所示。

表 2-3 ALU 单元所支持的运算功能

| aluop | 助记符 | 功能 | 说明[$] |
|---|---|---|---|
| 0000 | AND | 按位与 | alures0 = A & B，alures 1 = 0 |
| 0001 | OR | 按位或 | alures0 = A \| B，alures1 = 0 |
| 0010 | XOR | 按位异或 | alures0 = A ⊕ B，alures1 = 0 |
| 0011 | NAND | 按位与非 | alures0 = −(A & B)，alures1 = 0 |
| 0100 | NOT | 逻辑非 | alures0 = −A，alures1 = 0 |
| 0101 | SLL | 逻辑左移 | alures0 = A << B（B 取低 3 位），alures1 = 0 |
| 0110 | SRL | 逻辑右移[&] | alures0 = A >> B（B 取低 3 位），alures1 = 0 |
| 0111 | SRA | 算术右移 | alures0 = A>>>B（B 取低 3 位），alures1 = 0 |
| 1000 | MULU | 无符号数乘法 | alures0 = $(A * B)_{[3:0]}$，alures1 = $(A * B)_{[7:4]}$ |
| 1001 | MUL | 有符号数乘法 | alures0 = $(A * B)_{[3:0]}$，alures1 = $(A * B)_{[7:4]}$ |
| 1010 | ADD | 有符号数加法 | alures0 = A + B，alures1 = 0，需要设置 ov |
| 1011 | ADDU | 无符号数加法 | alures0 = A + B，alures1 = 0 |
| 1100 | SUB | 有符号数减法 | alures0 = A - B，alures1 = 0，需要设置 ov |
| 1101 | SUBU | 无符号数减法 | alures0 = A - B，alures1 = 0 |
| 1110 | SLT | 有符号数比较 | alures0 = (A < B)? 1 : 0，alures1 = 0 |
| 1111 | SLTU | 无符号数比较 | alures0 = (A < B)? 1 : 0，alures1 = 0 |

$：所有运算均需要设置 zero 位。
&：算数右移高位补符号位的前提是对有符号数进行右移，需要特别注意这一点。

完成上述 ALU 单元的世纪，必需满足如下几点要求：

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、实现算术右移是，可以使用运算符“>>>”.

## 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）

1. 写出全加器的 SystemVerilog HDL 代码。

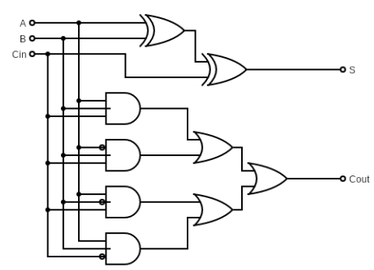
```
module fulladder(
      input logic A,
      input logic B,
      input logic Cin,
      output logic S,
      output logic Cout
      );
      always_comb begin
            if((A==1&&B==1&&Cin==1)||(A==1&&B==0&&Cin==0)||
(A==0&&B==1&&Cin==0)||(A==0&&B==0&&Cin==1)) begin
                  S = 1;
            end
            else begin S = 0; end
            if((A==1&&B==1&&Cin==1)||(A==1&&B==1&&Cin==0)||
(A==0&&B==1&&Cin==1)||(A==1&&B==0&&Cin==1)) begin
                  Cout = 1;
            end
            else begin Cout = 0; end
      end
endmodule
```

2. 写出行波进位加法器的结构化建模的 SystemVerilog HDL 代码。

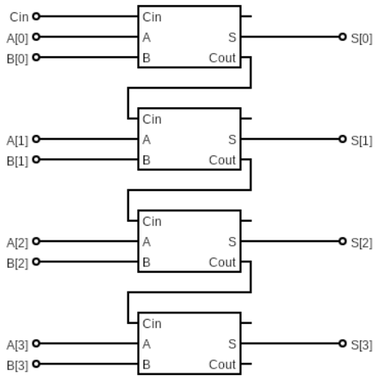
```
module rca(
      input [3:0]A,
      input [3:0]B,
      input Cin,
      output [3:0]S,
      output Cout
      );
      wire [4:0] temp;
      assign temp[0] = Cin;
      genvar i;
      for(i=0; i<4; i=i+1) begin
            fulladder add(
                  .A          (A[i]),
                  .B          (B[i]),
                  .Cin        (temp[i]),
                  .Cout       (temp[i+1]),
                  .S          (S[i])
            );
      end
      assign cout = temp[4];
endmodule
```

3. 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。

全加器：



行波进位加法器：




为什么可以只用一个加法器就可以同时进行加减法运算：

因为输入值 B 将自己与 aluop[2]相结合（可以看到，需要减法时，aluop[2]=1，需要加法时，aluop[2]=0）。我们就可以设置以下语句：

assign Cin=aluop[2]==0 ? 4'b0000 : 4'b1111;

如果是加法，没有任何变化，Cin=0；

如果是减法，B 取反加一（加的“一”为 Cin[0]）

这样就可以实现了用一个加法器同时实现加减法运算

4. 给出有符号数加/减法溢出的判断规则。

有符号数加法：(A[3]==B[3])&&(A[3]!=S[3])，

即当且仅当 A、B 符号相同且 A、S 符号相反。

有符号数减法：(A[3]!=B[3])&&(S[3]!=A[3])，

即当且仅当 A、B 符号相反且 A、S 符号相反。

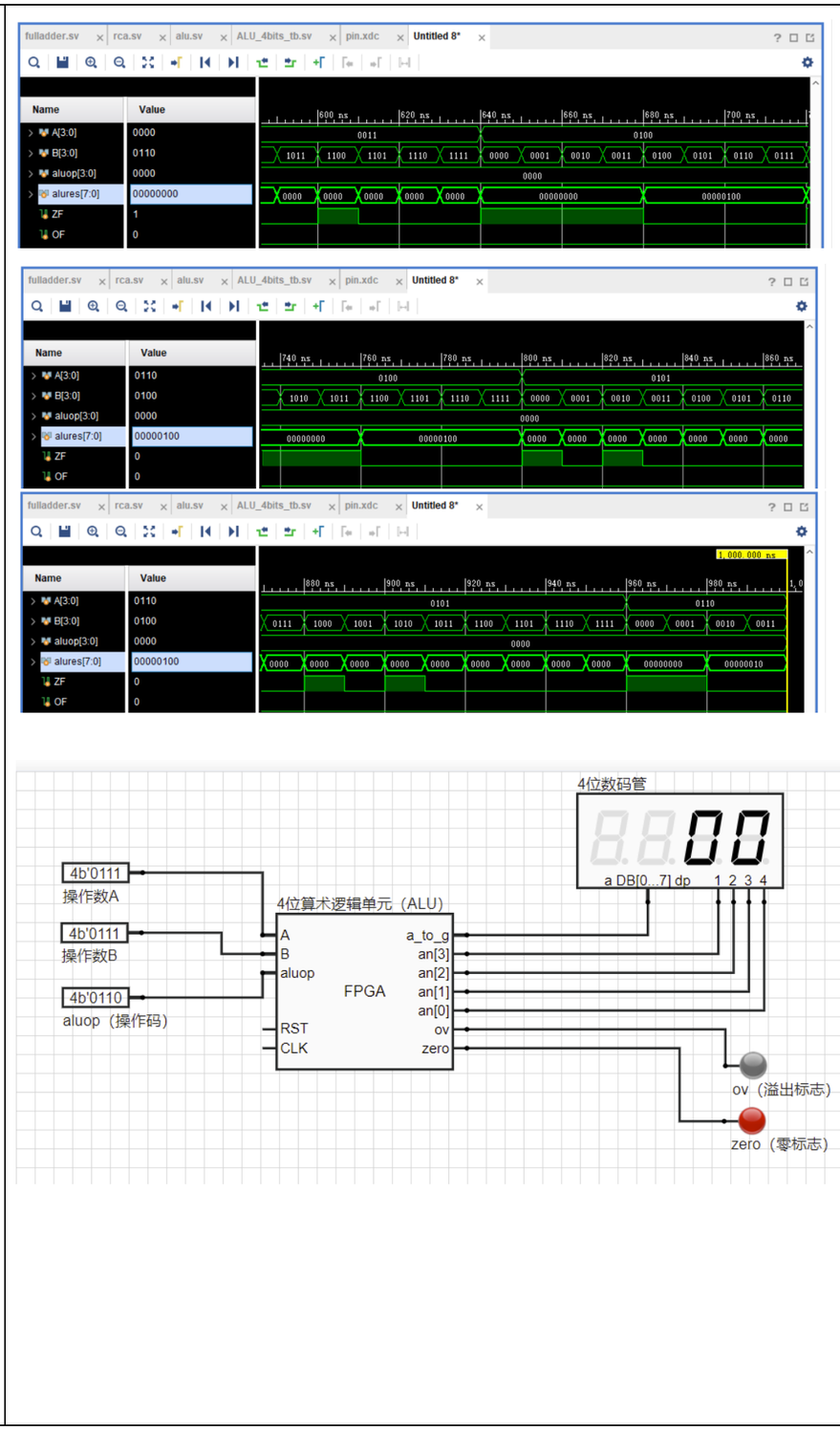
5. 给出 ALU 单元的 SystemVerilog HDL 代码。

```
module alu(
    input [3:0] A,
    input [3:0] B,
    input [3:0] aluop,
    output logic [7:0] alures,
    output logic ZF,
    output logic OF
    );
    logic Cout;
    logic [3:0] S;
    logic [3:0] Cin;

    assign Cin=aluop[2]==0 ? 4'b0000 : 4'b1111;

    rca ADD(
        .A      (A),
        .B      ((~B&Cin)|(B&(~Cin))),
        .Cin    (Cin[0]),
        .S      (S),
        .Cout   (Cout)
    );

    always_comb  begin
        case(aluop)
            4'b0000:begin alures[3:0]=A&B; alures[7:4]=4'b0000; OF=0; ZF=!alures; end
            4'b0001:begin alures[3:0]=A|B; alures[7:4]=4'b0000; OF=0; ZF=!alures; end
            4'b0010:begin alures[3:0]=A^B; alures[7:4]=4'b0000; OF=0; ZF=!alures; end
            4'b0011:begin alures[3:0]=~(A&B); alures[7:4]=4'b0000; OF=0; ZF=!alures; end
            4'b0100:begin alures[3:0]=~A; alures[7:4]=4'b0000; OF=0; ZF=!alures; end
            4'b0101:begin alures[3:0]=A<<B[2:0]; alures[7:4]=4'b0000; OF=0; ZF=!alures; end
            4'b0110:begin alures[3:0]=A>>B[2:0]; alures[7:4]=4'b0000; OF=0; ZF=!alures; end
            4'b0111:begin alures[3:0]=$signed(A)>>>B[2:0]; alures[7:4]=4'b0000; OF=0;
ZF=!alures; end
            4'b1000:begin alures[7:0]=A*B; OF=0; ZF=!alures; end
            4'b1001:begin alures[7:0]={{4{A[3]}},A}*{{4{B[3]}},B}; OF=0; ZF=!alures; end
            4'b1010:begin
                if((A[3]==B[3])&&(A[3]!=S[3])) begin
                    alures[3:0]=S; alures[7:4]=4'b0000; OF=1;ZF=!alures;
                end
                else begin
                    alures[3:0]=S; alures[7:4]=4'b0000; OF=0; ZF=!alures;
                end
            end
            4'b1011:begin alures[3:0]=S; alures[7:4]=4'b0000; OF=0; ZF=!alures; end
            4'b1100:begin
                if((A[3]!=B[3])&&(S[3]!=A[3]))begin
                    alures[3:0]=S[3:0]; alures[7:4]=4'b0000; OF=1; ZF=!alures;
                end
                else begin
                    alures[3:0]=S[3:0]; alures[7:4]=4'b0000; OF=0; ZF=!alures;
                end
            end
            4'b1101:begin alures[3:0]=S; alures[7:4]=4'b0000; OF=0; ZF=!alures; end
```

```
                4'b1110:begin alures[0]=$signed(A)<$signed(B) ? 1:0;
alures[7:1]=7'b0000000; OF=0; ZF=!alures; end
                4'b1111:begin alures[0]=A<B ? 1:0; alures[7:1]=7'b0000000; OF=0;
ZF=!alures; end
            endcase
        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）**

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

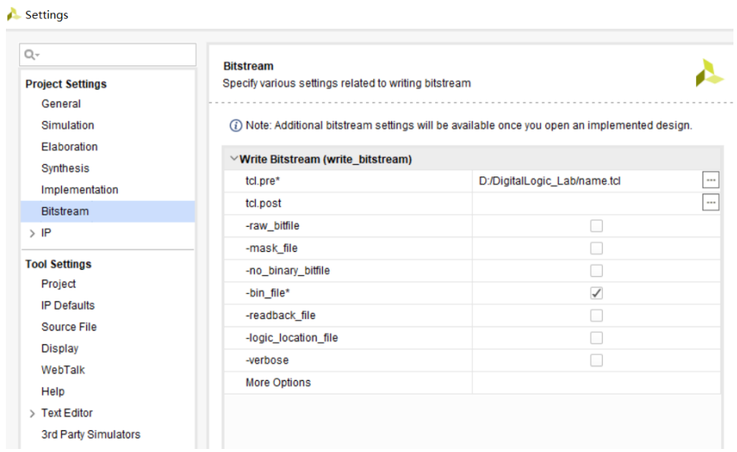
1、Vivado 无法生成比特流：
在进行 Vivado 实验中，其中一个步骤：Generate Bitstream 时发生报错，其中，重点句如下：

NOTE: When using the Vivado Runs infrastructure (e.g. launch_runs Tcl command), add this command to a .tcl file and add that file as a pre-hook for write_bitstream step for the implementation run.

在进行网络查询之后，我发现缺少一个文件，因此，我们可以将如下三行代码：

```
set_property SEVERITY {Warning} [get_drc_checks NSTD-1]
set_property SEVERITY {Warning} [get_drc_checks UCIO-1]
set_property SEVERITY {Warning} [get_drc_checks RTSTAT-1]
```

写到一个.tcl 文件中并保存，再将保存文件导入，便可正确生成比特流，操作如下：




2、实验无法进行模拟测试：
我们将 ALU_4bits_tb.sv 文件置顶就行。

## 六．附加题（若实验指导书无要求，则无需回答）

**教师签字：**

年　月　日