实验六 Verilog：ALU

一、 实验目的

用 Verilog 实现一个简单 ALU。（组合逻辑）

二、 实验内容

输入：两个 4 位二进制数，代表两个操作数 A，B；一个 3 位控制信号 operation，代表 ALU 要进行的运算。本实验中，ALU 可以实现 8 种运算：

输出：4 位结果，1 位进位

operation | F

000 | A + B

001 | A - B

010 | B + 1

011 | B - 1

100 | NOT A

101 | A XOR B

110 | A AND B

111 | A OR B

三．实验源码及分析（见注释）

```
module ALU(
    input [3:0] A,       // 4 位操作数 A
    input [3:0] B,       // 4 位操作数 B
    input [2:0] operation,// 3 位控制信号
    output reg [3:0] result, // 4 位结果，使用 reg 是因为在 always 块中赋值
    output reg cout       // 1 位进位，使用 reg 是因为在 always 块中赋值
);

  // 使用 always @(*) 块描述组合逻辑
  always @(*) begin
    // 初始化变量（如果需要）
    cout = 0; // 默认没有进位
    case(operation)
      // A + B
      3'b000: begin
        {cout, result} = A + B; // 使用花括号来接收加法操作的进位和结果
      end
      // A - B
      3'b001: begin
        {cout, result} = A - B + 4'b0001;
        if (A < B) begin // 如果 A 小于 B，则需要调整进位和结果（根据是否有借位）
          cout = 0;
          result = 4'b1111 - (B - A); // 假设我们处理无符号数，这里只是简单示例
```

```
        end
      end

      3'b010: begin
        result = B + 1'b1;
        cout = (B == 4'b1111); // 只在 B 为 4 位全 1 时进位
      end
      // B - 1
      3'b011: begin
        result = B - 1'b1;
        cout = 0; // 对于无符号数，B-1 不会产生进位
      end
      // NOT A
      3'b100: begin
        result = ~A;
        cout = 0; // NOT 操作不产生进位
      end
      // A XOR B
      3'b101: begin
        result = A ^ B;
        cout = 0; // XOR 操作不产生进位
      end
      // A AND B
      3'b110: begin
        result = A & B;
        cout = (result == 4'b1111); // 只有在结果全为 1 时才产生进位（这里的进位定义可能不符合常规逻辑，仅作示例）
      end
      // A OR B
      3'b111: begin
        result = A | B;
        cout = 0; // OR 操作不产生进位
      end
      default: begin
        result = 4'bXXXX; // 使用 X 表示未定义的操作结果
        cout = 0;
      end
    endcase
  end
Endmodule
```

Test Bench

```
`timescale  1ns / 1ps

`include "ALU.v"

module tb_ALU;
```

```
// ALU Parameters
parameter PERIOD  = 10;
```

```
// ALU Inputs
reg   [3:0]  A                            = 0 ;
reg   [3:0]  B                            = 0 ;
reg   [2:0]  operation                     = 0 ;
```

```
// ALU Outputs
wire  [3:0]  result                        ;
wire  cout                                 ;
```

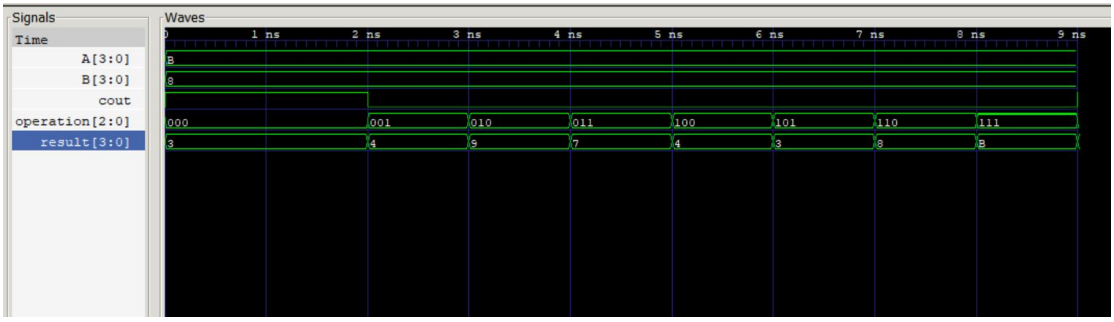
```
ALU  u_ALU (
    .A                      ( A          [3:0] ),
    .B                      ( B          [3:0] ),
    .operation              ( operation  [2:0] ),
```

```
    .result                 ( result     [3:0] ),
    .cout                   ( cout             )
);
```

```
initial
begin
    $dumpfile("ALUtb.vcd");
    $dumpvars;
    A=4'b1011;B=4'b1000;
    #1 operation=3'b000;
    #1 operation=3'b001;
    #1 operation=3'b010;
    #1 operation=3'b011;
    #1 operation=3'b100;
    #1 operation=3'b101;
    #1 operation=3'b110;
    #1 operation=3'b111;
    #1 operation=3'b000;
    $finish;
end
```

```
endmodule
```

4. 仿真图及分析



A 为 B（11），B 为 8。Count 为进位。000：11+8=19=16+3=count：1，result：3

001：11-8=4，result：4

010：8+1=9；result：9

011：B-1=8-1=7 result：7

100：NOTA = 1011（非）=0100=4　result：4

101：A XOR B =1000XOR1011 =3 result：3

110：A And B = B AND 8 =8 result：8

111：A or B =B OR 8 = B result：B

5. 总结与分析：

1：ALU 在现代计算机中的作用：

ALU：Arithmetic Logic Unit，即算术逻辑单元，其是在计算机中是专门执行算术和逻辑运算的数字电路。ALU 是计算机中央处理器的最重要组成部分，甚至连最小的微处理器也包含 ALU 作计数功能。在现代 CPU 和 GPU 处理器中已含有功能强大和复杂的 ALU；一个单一的元件也可能含有 ALU。

2：列举 ALU 的其他功能：

移位运算（算术移位，逻辑移位，循环移位，循环移位（带进位））

其他特殊运算：FZ FS FC

实验七 Verilog：时序电路

实验目的：

学生通过用 Verilog 实现 4 位计数器，进一步熟悉 Verilog 的语法和时序逻辑电路

实验描述：

输入：

Clock：如果计数器 enable 信号为 1，那么在时钟上升沿，count 加 1

Enable：如果 enable 为 1，那么在时钟上升沿，count 加 1；如果 enable 为 0，count 保持不变

Reset：重置信号，如果 reset 为 0，count 重置为 0

输出：

Count[3:0]：4 位计数信号，范围：4‘b0000 – 4’b1111

源码及分析：

```
module counter(
    input clk,       // 时钟信号
    input reset,     // 复位信号（假设低电平有效）
    input enable,    // 使能信号
    output reg [3:0] count  // 计数器输出，4 位二进制数
);

    // 如果 reset 为 0（低电平有效），count 设置为 0；
    // 如果 enable 为 1，在时钟上升沿，count 加 1；
    // 如果 enable 为 0 或 reset 为高，count 不变

    always @(posedge clk or negedge reset) // 触发器在 clk 上升沿或 reset 下降沿触发
    begin
        if(!reset)   // 如果 reset 为 0（低电平有效），则复位
            count <= 0;
        else if(enable) // 如果 enable 为 1 且 reset 不为 0
            if(clk_edge) // 需要一个标志位来检测 clk 的上升沿，但在 Verilog 中通常直接在 always 块中检测
                count <= count + 1'b1; // 在时钟上升沿加 1
    end

    // 注意：这里没有直接检测 clk 的上升沿，因为 Verilog 的 always 块已经通过 posedge clk 指定了这一点。
    // 如果需要在一个更复杂的逻辑中检测 clk 的上升沿，可以添加一个额外的变量来跟踪它。

endmodule
```

Test bench

```
`timescale  1ns / 1ps
`include "Time.v"
module tb_counter;

// counter Parameters
parameter PERIOD  = 10;
```

```
// counter Inputs
```

```
reg   reset                                = 0 ;
reg   enable                               = 1 ;
```

```
// counter Outputs
wire  [3:0]  count                         ;
```

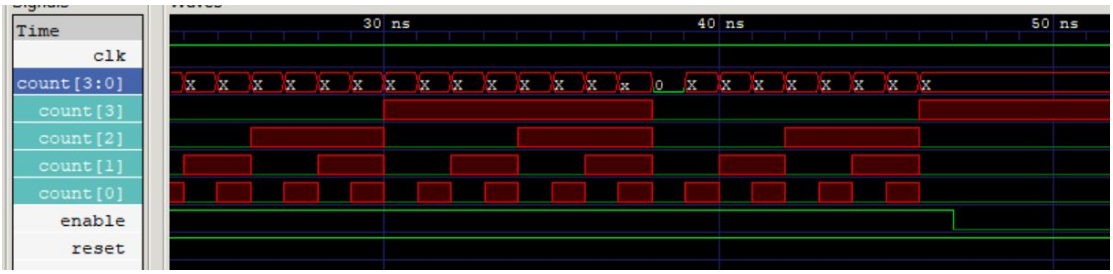
```
counter  u_counter (

    .reset                  ( reset         ),
    .enable                 ( enable        ),
```

```
    .count                  ( count   [3:0] )
);
```

```
initial
begin
    $dumpfile("twave.vcd");
   $dumpvars;
    #2 reset =0;
    #5 reset=1;
    #40 enable=0;
    #41 enable=1;
    #50 reset=0;
    #51 reset=1;
    #6000 $stop;
    $finish;
end
    reg clk;
    initial
        clk=0;
        always #(clk)
        clk=~clk;
        counter ctr(clk,reset,enable,count);
endmodule
```

仿真图及分析



当 reset 为一，enable 为一，计数器工作，随着 cp 脉冲下降计数，count+1/当 enable 为 0，不再计数

当 reset 置零，全体置零。