实验二报告 基于 Verilog HDL 的数字电路设计（二）

实验目的

1. 使用 Verilog HDL 语言设计一个 4 位迭代乘法器，并在数码管上显示计算结果。
2. 使用 Verilog HDL 语言设计一个 8 位有符号乘法器，并在数码管上显示计算结果。

实验一：4 位迭代乘法器

实验原理

该模块实现一个 4 位无符号乘法器，通过迭代的移位相加操作完成乘法。输入 a 和 b 是 4 位无符号数，乘积最大为 15×15=225。模块使用状态机控制乘法过程，并通过 BCD 码转换将 8 位乘积转换为十进制，最终在数码管上动态显示。

状态机包括以下状态：

- IDLE：等待输入变化。
- TRIGGER：检测到输入变化，准备开始计算。
- CALC：执行移位相加的乘法运算。
- DONE：计算完成，结果准备显示。

BCD 转换采用"Double Dabble"算法，将二进制结果转换为 BCD 码，以便于数码管显示。数码管显示部分采用动态扫描方式，依次刷新百位、十位、个位，并加入空白帧和死区以消除重影。

程序代码

```
module multiplier_display (
    input clk,           // 系统时钟
    input [7:0] sw,      // 拨码开关输入：SW[3:0] = A, SW[7:4] = B
    output reg [6:0] seg, // 数码管段选输出
    output reg [3:0] an  // 数码管位选输出
);

    // === 输入同步 ===
    reg [7:0] sw_sync_0, sw_sync_1;
    always @(posedge clk) begin
        sw_sync_0 <= sw;
        sw_sync_1 <= sw_sync_0;
    end
    wire [7:0] sw_stable = sw_sync_1;
    wire [3:0] a = sw_stable[3:0]; // 乘数 A
    wire [3:0] b = sw_stable[7:4]; // 被乘数 B

    // === 检测输入变化 ===
    reg [7:0] prev_sw = 0;
    wire input_changed = (sw_stable != prev_sw);

    // === 状态机定义 ===
    reg [3:0] state = 0;
    parameter IDLE = 0, TRIGGER = 1, CALC = 2, DONE = 3;

    reg [7:0] multiplicand;  // 乘法运算中的被加数
    reg [3:0] multiplier;    // 乘法运算中的乘数
    reg [7:0] accumulator;   // 累加器，存储部分积
    reg [3:0] count;         // 计数器，控制迭代次数
    reg [7:0] result = 0;    // 最终的乘法结果

    always @(posedge clk) begin
        case (state)
            IDLE: begin
                if (input_changed) begin
                    prev_sw <= sw_stable;
                    state <= TRIGGER;
                end
            end
            TRIGGER: begin
                multiplicand <= {4'b0000, a}; // 扩展到 8 位
                multiplier <= b;
                accumulator <= 0;
                count <= 4; // 4 位乘法迭代 4 次
                state <= CALC;
            end
            CALC: begin
                if (count > 0) begin
                    if (multiplier[0]) // 如果乘数最低位为 1，则累加
                        accumulator <= accumulator + multiplicand;
                    multiplicand <= multiplicand << 1; // 被加数左移
                    multiplier <= multiplier >> 1;     // 乘数右移
                    count <= count - 1;
                end else begin
                    result <= accumulator;
                    state <= DONE;
                end
            end
            DONE: begin
                state <= IDLE;
            end
        endcase
    end

    // === BCD 转换 ===
    reg [19:0] shift; // 用于 BCD 转换的移位寄存器
    reg [3:0] hundreds, tens, ones; // 百位、十位、个位的 BCD 码
    integer i;

    always @(*) begin
        shift = 0;
        shift[7:0] = result; // 加载 8 位结果

        for (i = 0; i < 8; i = i + 1) begin // 循环 8 次
            // Double Dabble 算法：如果 BCD 位大于等于 5，则加 3
            if (shift[11:8] >= 5) shift[11:8] = shift[11:8] + 3;
```

```
            if (shift[15:12] >= 5) shift[15:12] = shift[15:12] + 3;
            if (shift[19:16] >= 5) shift[19:16] = shift[19:16] + 3;
            shift = shift << 1; // 整体左移一位
        end

        hundreds = shift[19:16]; // 百位 BCD
        tens     = shift[15:12]; // 十位 BCD
        ones     = shift[11:8];  // 个位 BCD
    end

    // === 数码管动态刷新（死区 + 空白帧）===
    reg [15:0] clkdiv = 0; // 时钟分频器
    reg [2:0] scan_idx = 0; // 扫描索引
    reg [3:0] digit;        // 当前要显示的数字（BCD 码）

    always @(posedge clk) begin
        clkdiv <= clkdiv + 1;
        scan_idx <= clkdiv[15:13]; // 控制扫描速度
    end

    always @(*) begin
        an = 4'b1111;     // 默认不亮
        seg = 7'b1111111; // 默认不亮

        case (scan_idx)
            3'd0: begin an = 4'b1110; digit = ones; end      // 扫描个位
            3'd1: begin an = 4'b1101; digit = tens; end      // 扫描十位
            3'd2: begin an = 4'b1011; digit = hundreds; end  // 扫描百位
            3'd3: begin an = 4'b1111; digit = 4'd0; end      // 空白帧
            3'd7: begin an = 4'b1111; digit = 4'd0; end      // 死区
            default: begin an = 4'b1111; digit = 4'd0; end
        endcase

        // BCD 码到七段码的转换
        case (digit)
            4'd0: seg = 7'b1000000;
            4'd1: seg = 7'b1111001;
            4'd2: seg = 7'b0100100;
            4'd3: seg = 7'b0110000;
            4'd4: seg = 7'b0011001;
            4'd5: seg = 7'b0010010;
            4'd6: seg = 7'b0000010;
            4'd7: seg = 7'b1111000;
            4'd8: seg = 7'b0000000;
            4'd9: seg = 7'b0010000;
            default: seg = 7'b1111111; // 空白
        endcase
    end
endmodule
```

实验结果及分析

如图所示，通过测试图，可以看到当 sw 输入变化时，状态机会从 IDLE 进入 TRIGGER，然后进入 CALC 状态进行乘法运算。在 CALC 状态，multiplicand 会左移，multiplier 会右移，并且根据 multiplier 的最低位判断是否将 multiplicand 加到 accumulator 中。当 count 减为 0 时，计算完成，结果传递给 result，状态进入 DONE，最终回到 IDLE。BCD 转换模块会将 result 转换为百位、十位、个位的 BCD 码。数码管动态扫描模块会依次点亮相应的数码管位并显示对应的数字。当 sw 输入为 8'b0111_0011 (A=7, B=3) 时，结果显示 21。

问题与解决

- **问题**：初始未考虑输入变化检测，导致数码管显示重影。
- **解决**：添加 input_changed 信号和 prev_sw 寄存器来检测输入变化，确保乘法运算在输入稳定后才开始。

实验二：8 位有符号乘法器

实验原理

该模块实现一个 8 位有符号乘法器。输入的 a 和 b 是 8 位有符号数，乘积最大为 127×127=16129 或 −128×−128=16384，最小为 −128×127=−16256。结果是一个 16 位有符号数。模块直接使用 Verilog 的乘法运算符来计算有符号乘积。

核心部分是将 16 位的有符号乘积转换为 BCD 码以便在数码管上显示。由于结果可能是负数，因此需要先取绝对值。BCD 转换仍然采用"Double Dabble"算法，但需要处理 16 位输入，因此 BCD 寄存器的宽度和循环次数有所增加，以产生 5 位十进制数字。数码管显示部分扩展到 6 位，其中一位用于显示符号（负号为'-'，正数为空白）。

程序代码

```
    module multiplier_display_signed8 (
        input             clk,  // 系统时钟
        input      [15:0] sw,   // 拨码开关输入：SW[7:0]=A (signed),
SW[15:8]=B (signed)
        output reg  [6:0] seg,  // 七段显示段选
        output reg  [5:0] an    // 七段显示位选（6 位）
    );

        // === 输入同步 ===
        reg [15:0] sw_sync_0, sw_sync_1;
        always @(posedge clk) begin
            sw_sync_0 <= sw;
            sw_sync_1 <= sw_sync_0;
        end
        wire [15:0] sw_stable = sw_sync_1;

        // === 有符号输入 ===
        wire signed [7:0] a = sw_stable[7:0];   // 8 位有符号乘数 A
        wire signed [7:0] b = sw_stable[15:8];  // 8 位有符号被乘数 B

        // === 直接乘法运算 ===
        wire signed [15:0] prod = a * b; // 8 位有符号乘法结果为 16 位

        // === BCD 转换：16 位绝对值 -> 5 位十进制 ===
        reg [35:0] shift; // 用于 BCD 转换的移位寄存器
        // 5 位十进制数字的 BCD 码
        reg  [3:0] digit0, digit1, digit2, digit3, digit4;
        integer    i;

        always @(*) begin
            shift = 36'd0;
            // 加载乘积的绝对值
            if (prod < 0)
                shift[15:0] = -prod;
            else
                shift[15:0] = prod;

            // Double Dabble 算法，循环 16 次
            for (i = 0; i < 16; i = i + 1) begin
                // 如果 BCD 位大于等于 5，则加 3
                if (shift[35:32] >= 5) shift[35:32] = shift[35:32] + 3;
                if (shift[31:28] >= 5) shift[31:28] = shift[31:28] + 3;
                if (shift[27:24] >= 5) shift[27:24] = shift[27:24] + 3;
                if (shift[23:20] >= 5) shift[23:20] = shift[23:20] + 3;
                if (shift[19:16] >= 5) shift[19:16] = shift[19:16] + 3;
                shift = shift << 1; // 整体左移一位
            end
            digit4 = shift[35:32]; // 万位 BCD
            digit3 = shift[31:28]; // 千位 BCD
            digit2 = shift[27:24]; // 百位 BCD
            digit1 = shift[23:20]; // 十位 BCD
            digit0 = shift[19:16]; // 个位 BCD
        end

        // === 数码管刷新：5 位数字 + 1 位符号 ===
        reg [20:0] clkdiv;   // 时钟分频器
        reg  [2:0] scan_idx; // 扫描索引
        reg  [3:0] cur_digit; // 当前要显示的数字（BCD 码）

        always @(posedge clk) begin
            clkdiv   <= clkdiv + 1;
            scan_idx <= clkdiv[20:18]; // 控制扫描速度
        end

        always @(*) begin
            an = 6'b111111; // 默认不亮
            case (scan_idx)
                3'd0: begin an[0] = 1'b0; cur_digit = digit0; end // 扫描个位
                3'd1: begin an[1] = 1'b0; cur_digit = digit1; end // 扫描十位
                3'd2: begin an[2] = 1'b0; cur_digit = digit2; end // 扫描百位
                3'd3: begin an[3] = 1'b0; cur_digit = digit3; end // 扫描千位
                3'd4: begin an[4] = 1'b0; cur_digit = digit4; end // 扫描万位
                3'd5: begin // 扫描符号位
                    an[5] = 1'b0;
                    if (prod < 0)
                        cur_digit = 4'hE; // 负号 '-'
                    else
                        cur_digit = 4'hF; // 空白
                end
                default: cur_digit = 4'hF; // 默认空白
            endcase

            // 段码映射
            case (cur_digit)
                 4'h0: seg = 7'b1000000;
                 4'h1: seg = 7'b1111001;
                 4'h2: seg = 7'b0100100;
                 4'h3: seg = 7'b0110000;
                 4'h4: seg = 7'b0011001;
                 4'h5: seg = 7'b0010010;
                 4'h6: seg = 7'b0000010;
                 4'h7: seg = 7'b1111000;
                 4'h8: seg = 7'b0000000;
                 4'h9: seg = 7'b0010000;
                 4'hE: seg = 7'b0111111; // '-'
                default: seg = 7'b1111111; // 空白
            endcase
        end
    endmodule
```

实验结果与测试图

如图，通过实验，可以验证 8 位有符号乘法器的功能。当 sw 输入为 16'b100011100000011 (A=-121, B=3) 时，显示 -363。

BCD 转换逻辑能够正确处理 16 位有符号数的绝对值，并生成 5 位十进制数字。数码管动态刷新模块能够正确显示 5 位数字以及符号位。 问题与解决办法

问题描述	解决方法
8 位有符号乘法器 BCD 转换需要处理负数。	在 BCD 转换前先取乘积的绝对值。
8 位有符号乘法器结果位宽增加，BCD 转换逻辑需调整。	增加 BCD 转换的 shift 寄存器位宽和循环次数，以适应 16 位输入和 5 位输出。

问题描述	解决方法
8 位有符号乘法器需显示负号。	在数码管扫描中添加一个符号位，并根据乘积的正负显示'-'或空白。

意见建议

- 在设计复杂的组合逻辑时，推荐先进行手算验证，确保算法逻辑正确。
- 对于带有状态机的设计，明确每个状态的进入和退出条件，以及在该状态下执行的操作，有助于逻辑的清晰性。
- 数码管驱动频率和扫描刷新频率需要合理设计，以避免闪烁或重影。
- 在进行验证时，应设计全面的测试用例，包括边界条件和正负数的组合，以确保设计的鲁棒性。
- 对于有符号数运算，需要注意 Verilog 中 signed 关键字的使用以及位宽的扩展。