实验一报告 基于 Verilog HDL 的数字电路设计（一）

实验目的

使用 Verilog HDL 语言设计以下功能模块，并在 FPGA 或仿真环境中验证其逻辑正确性：

1. 8 位带进位输入/输出的全加器
2. 3-8 线译码器
3. 8 选 1 多路选择器
4. 4 位可配置移位寄存器（含去抖模块）

实验一：8 位带进位输入输出的全加器

实验原理

全加器实现两个 8 位输入的加法，并处理进位输入（cin）与输出（sum[8]）。通过控制信号 sel 选择是否累加先前存储的值，实现简单的带状态计算功能。

程序代码

```
module full_adder_8bit(
    input        clk,    // 系统时钟
    input        rst,    // 异步复位，高电平有效
    input  [7:0] a,      // 8 位输入
    input        sel,    // 控制信号：
                         // sel = 0：更新存储值并输出当前输入
                         // sel = 1：使用存储值与当前输入相加
    input        cin,    // 进位输入
    output reg [8:0] sum // 9 位输出（含进位）
);

reg [7:0] stored_a;

always @(posedge clk or posedge rst) begin
    if (rst) begin
        stored_a <= 8'b0;
        sum      <= 9'b0;
    end else if (sel == 1'b0) begin
        stored_a <= a;
        sum <= {1'b0, a} + cin;
    end else begin
        sum <= a + stored_a + cin;
    end
end
endmodule
```

实验结果及分析

如图，加数已被设置为 4，被加数为 8，进位 1，结果为 8+4+1=13。

问题与解决

- **问题**：初始未考虑复位条件。
- **解决**：添加异步复位逻辑。

实验二：3-8 线译码器

实验原理

译码器将 3 位二进制输入转换为 8 位输出，每次仅一个输出位为 0，其余为 1。设定特定使能条件。

程序代码

```
module Decoder(
    input [2:0] data_i,    // 3 位输入数据
    input [2:0] en_i,      // 3 位使能信号
    output reg [7:0] data_o // 8 位输出
);

always @(*) begin
    if(en_i[0] || en_i[1] || !en_i[2])
        data_o = 8'b1111_1111; // 使能无效，全部高电平
    else
        case (data_i)
            3'b000: data_o = 8'b1111_1110;
            3'b001: data_o = 8'b1111_1101;
            3'b010: data_o = 8'b1111_1011;
            3'b011: data_o = 8'b1111_0111;
            3'b100: data_o = 8'b1110_1111;
            3'b101: data_o = 8'b1101_1111;
            3'b110: data_o = 8'b1011_1111;
            3'b111: data_o = 8'b0111_1111;
        endcase
end
endmodule
```

实验结果及分析

如图，选择位输入 101，5 号位灯熄灭。

实验三：8 选 1 多路选择器

实验原理

根据 3 位选择信号 sel，从 8 位输入 in 中选出一个输出。

程序代码

```
module MUX(
    input [2:0] sel,    // 选择信号
    input [7:0] in,     // 8 路输入
    output reg out      // 单比特输出
);

always @(sel) begin
    case(sel)
        3'b000: out <= in[0];
        3'b001: out <= in[1];
        3'b010: out <= in[2];
        3'b011: out <= in[3];
        3'b100: out <= in[4];
        3'b101: out <= in[5];
        3'b110: out <= in[6];
        3'b111: out <= in[7];
    endcase
end

endmodule
```

实验结果及分析

如图，状态位为 101（图误），输出了 5 位上的数据。

实验四：4 位移位寄存器（含可配置功能与按键去抖）

实验原理

该模块支持：

- 可配置的移位方向（左/右）
- 移位模式（循环/逻辑补 0）
- 支持按钮控制加载/移位动作
- 使用 debounce 去抖模块稳定按键信号

程序代码（分模块）

1. debounce（去抖模块）

```
module debounce(
    input clk,
    input rst,
    input noisy_in,
    output reg clean_out
);
    reg [127:0] shift_reg;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            shift_reg <=
128'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF;
            clean_out <= 0;
        end else begin
            shift_reg <= {shift_reg[126:0], noisy_in};
            if (shift_reg ==
128'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)
                clean_out <= 1;
            else if (shift_reg ==
128'h00000000000000000000000000000000)
                clean_out <= 0;
        end
    end
endmodule
```

2. 移位寄存器模块

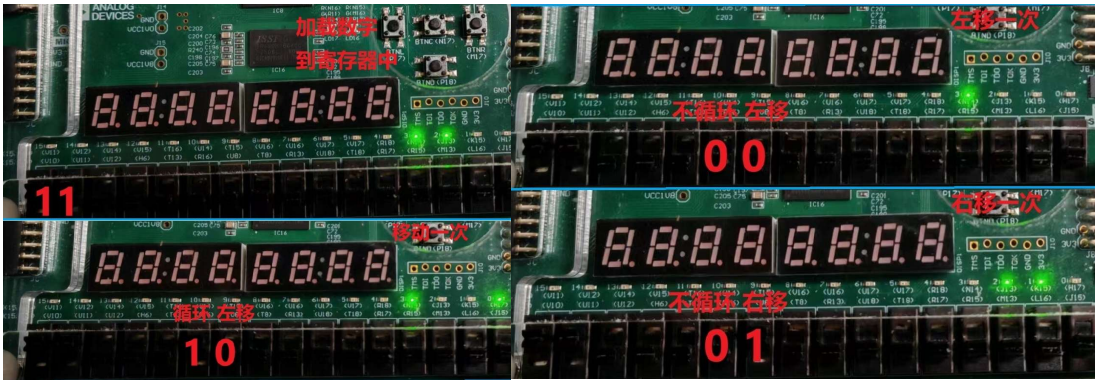
```
module configurable_shift_reg_4bit(
    input clk,
    input rst,
    input start,
    input load,
    input dir,
    input mode,
    input [3:0] d,
    output reg [3:0] q
);

reg start_d, load_d;
wire start_pulse = ~start_d & start;
wire load_pulse  = ~load_d & load;

always @(posedge clk or posedge rst) begin
    if (rst) begin
        start_d <= 0;
        load_d <= 0;
    end else begin
        start_d <= start;
        load_d <= load;
    end
end

always @(posedge clk or posedge rst) begin
    if (rst)
        q <= 4'b0000;
    else if (load_pulse)
        q <= d;
    else if (start_pulse) begin
        if (dir) // 左移
            q <= mode ? {q[2:0], q[3]} : {q[2:0],
1'b0};
        else     // 右移
            q <= mode ? {q[0], q[3:1]} : {1'b0,
q[3:1]};
    end
end

endmodule
```

3. 顶层模块（整合输入输出）

```
module top(
    input clk_100mhz,
    input rst,
    input btn_start_raw,
    input btn_load_raw,
    input dir,
    input mode,
    input [3:0] d,
    output [3:0] q
);

wire btn_start_clean;
wire btn_load_clean;

debounce u_debounce_start(
    .clk(clk_100mhz),
    .rst(rst),
    .noisy_in(btn_start_raw),
    .clean_out(btn_start_clean)
);

debounce u_debounce_load(
    .clk(clk_100mhz),
    .rst(rst),
    .noisy_in(btn_load_raw),
    .clean_out(btn_load_clean)
);

configurable_shift_reg_4bit
u_shift_reg(
    .clk(clk_100mhz),
    .rst(rst),
    .start(btn_start_clean),
    .load(btn_load_clean),
    .dir(~dir),  // 与硬件方向开关
对应
```

```
    .mode(mode),                    .q(q)
    .d(d),                      );                                  endmodule
```

实验结果与测试图



如图所示。

问题与解决办法

问题描述	解决方法
按钮误触反复触发	添加 debounce 去抖模块
多模块连接逻辑不清	分层设计 + 顶层整合
忘记复位条件	每个 always 块加 rst 分支