《数字逻辑》实验报告

姓名	简沅晞	年级	2024 级
学号	20241212	专业、班级	计算机科学与技术 计卓 2 班

实验名称	电梯控制器设计		
实验时间	2025 年 11 月 20 日	实验地点	DS1401
实验成绩		实验性质	□验证性 □设计性 □综合性

教师评价：

□算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；

□实验结果正确； □语法、语义正确； □报告规范；

评语：

评价教师签名（电子签名）：

一、实验目的

（1） 巩固有限状态机设计方法，并设计实现一个电梯控制器。

（2） 掌握 7 段数码管显示的工作原理及应用。

（3） 掌握按键消抖原理及实现方法。

二、实验项目内容

设计一个多楼层的电梯控制器系统，并能在开发板上模拟电梯运行状态。可以利用按键作为呼叫按键，数码管显示电梯运行时电梯所在楼层，led 灯显示楼层叫梯状态。具体要求如下：

1) 利用开发板的 5 个按键作为电梯控制器的呼叫按钮；

2) 利用 led 灯分别显示楼层 1~5 的呼梯状态；

3) 利用数码管显示电梯运行时电梯所在楼层；

4) 利用时钟分频设计电梯控制器控制电梯每秒运行一层。

三、实验设计（实验原理、真值表、原理图等）

实验原理：

1. 有限状态机 (FSM) 设计 电梯的运行逻辑是通过有限状态机实现的 。系统定义了电梯的几种基本状态，如：

IDLE (空闲/停止)：电梯在某一层静止，等待呼叫 。

UP (上升)：电梯正在向高楼层运行 。

DOWN (下降)：电梯正在向低楼层运行 。 状态机根据当前楼层与目标请求楼层的比较结果进行状态跳转，并在状态转换过程中控制楼层计数器的加减。

2. 按键消抖原理：机械按键在按下和松开的瞬间会产生电气抖动，可能导致系统误判为多次按键。为了保证系统的稳定性，必须进行消抖处理 。

原理：利用时钟分频产生一个较低频率的采样时钟（如 100Hz 或 200ms 周期） 。

实现：使用移位寄存器连续采样按键状态，只有当连续多个采样周期（如 key_rrr, key_rr, key_r）电平保持一致时，才认定按键状态有效 。

3. 叫梯请求与存储 (异或逻辑) 实验要求按键具有"叫梯"和"取消"功能。

实现：利用异或（XOR）逻辑 。当检测到有效的按键上升沿时，将当前的叫梯状态寄存器与 1 进行异或操作，从而实现状态翻转（亮灯变灭灯，灭灯变亮灯） 。

自动清除：当电梯到达目标楼层后，状态机会自动清除该层的叫梯请求位。

4. 时钟分频与定时

运行速度控制：FPGA 系统时钟为 100MHz，为了满足"电梯每秒运行一层"的要求 ，需要设计一个计数器将 100MHz 分频为 1Hz 的脉冲信号，作为电梯移动的使能信号。

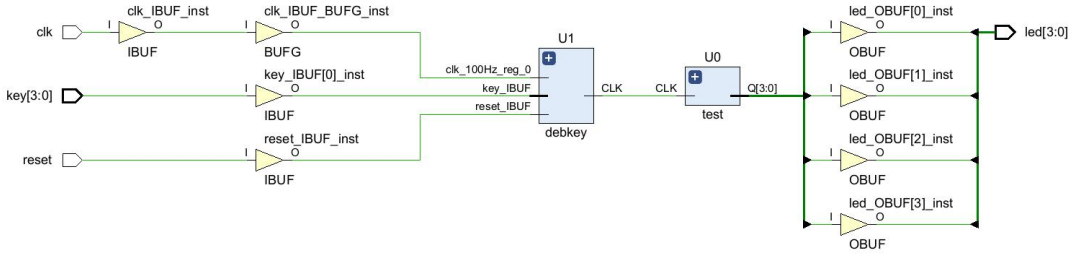
数码管扫描：数码管的动态扫描也需要通过分频时钟来控制刷新频率，但在单个数码管显示时可直接驱动。

5. 输入输出显示

数码管：用于显示电梯当前所在的楼层数字（1-5）。

LED 灯：用于指示各个楼层的叫梯请求状态（亮起表示有请求）。

原理图：



四、实验过程或算法(关键步骤、核心代码注解等)

数码显示模块：

```
module seg_display(
    input wire [3:0] floor_num,
    output reg [6:0] seg,
    output wire [3:0] an
);

    assign an = 4'b1110;

    always @(*) begin
        case(floor_num)
            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;
            default: seg = 7'b1111111;
        endcase
    end
endmodule
```

消抖模块：

```
module debkey (
    input wire clk,
    input wire reset,
    input wire [4:0] key_in,
    output wire [4:0] key_out
);


    parameter T100Hz = 499999;
    integer cnt_100Hz;
    reg clk_100Hz;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            cnt_100Hz <= 32'd0;
            clk_100Hz <= 1'b0;
        end else begin
            if (cnt_100Hz == T100Hz) begin
                cnt_100Hz <= 32'd0;
                clk_100Hz <= ~clk_100Hz;
            end else begin
                cnt_100Hz <= cnt_100Hz + 1'b1;
            end
        end
    end


    reg [4:0] key_rrr, key_rr, key_r;
    always @(posedge clk_100Hz or posedge reset) begin
        if (reset) begin
            key_rrr <= 5'b00000;
            key_rr  <= 5'b00000;
            key_r   <= 5'b00000;
        end else begin
            key_rrr <= key_rr;
            key_rr  <= key_r;
            key_r   <= key_in;
        end
    end

    assign key_out = key_rrr & key_rr & key_r;

endmodule
```

电梯主控模块：

```
module elevator_ctrl(
    input wire clk,
    input wire reset,
    input wire [4:0] key_clean,
    output reg [4:0] floor_led,
    output reg [3:0] current_floor_out
);

    parameter IDLE = 2'b00;
    parameter UP   = 2'b01;
    parameter DOWN = 2'b10;

    reg [1:0] state;
    reg [3:0] current_floor;
    reg arrive_flag;

    reg [4:0] key_prev;
    wire [4:0] key_posedge;
    always @(posedge clk) key_prev <= key_clean;
    assign key_posedge = key_clean & (~key_prev);
```

```
always @(posedge clk or posedge reset) begin
    if (reset) begin
        floor_led <= 5'b0;
    end else begin
        if (key_posedge[0]) floor_led[0] <= ~floor_led[0];
        if (key_posedge[1]) floor_led[1] <= ~floor_led[1];
        if (key_posedge[2]) floor_led[2] <= ~floor_led[2];
        if (key_posedge[3]) floor_led[3] <= ~floor_led[3];
        if (key_posedge[4]) floor_led[4] <= ~floor_led[4];
        if (arrive_flag && current_floor >= 1 && current_floor <= 5) begin
            floor_led[current_floor - 1] <= 1'b0;
        end
    end
end

parameter T1S = 99_999_999;
integer cnt_1s;
reg move_pulse;
```

```
    always @(posedge clk or posedge reset) begin
        if(reset) begin
            cnt_1s <= 0;
            move_pulse <= 0;
        end else if (state != IDLE) begin
            if (cnt_1s == T1S) begin
                cnt_1s <= 0;
                move_pulse <= 1;
            end else begin
                cnt_1s <= cnt_1s + 1;
                move_pulse <= 0;
            end
        end else begin
            cnt_1s <= 0;
            move_pulse <= 0;
        end
    end
```

```
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            state <= IDLE;
            current_floor <= 4'd1;
            arrive_flag <= 1'b0;
        end else begin
            arrive_flag <= 1'b0;

            case (state)
                IDLE: begin
                    if (floor_led > 0) begin
                        if (floor_led[4] && 5 > current_floor) state <= UP;
                        else if (floor_led[3] && 4 > current_floor) state <= UP;
                        else if (floor_led[2] && 3 > current_floor) state <= UP;
                        else if (floor_led[1] && 2 > current_floor) state <= UP;
                        else if (floor_led[0] && 1 > current_floor) state <= UP;

                        else if (floor_led[0] && 1 < current_floor) state <= DOWN;
                        else if (floor_led[1] && 2 < current_floor) state <= DOWN;
                        else if (floor_led[2] && 3 < current_floor) state <= DOWN;
                        else if (floor_led[3] && 4 < current_floor) state <= DOWN;
                        else if (floor_led[4] && 5 < current_floor) state <= DOWN;
                        else begin
                            arrive_flag <= 1'b1;
                        end
                    end
                end

                UP: begin
                    if (move_pulse) begin
                        current_floor <= current_floor + 1;
                        arrive_flag <= 1'b1;
                    end
                end

                DOWN: begin
                    if (move_pulse) begin
                        current_floor <= current_floor - 1;
                        arrive_flag <= 1'b1;
                    end
                endcase
            end
        end

        always @(*) begin
            current_floor_out = current_floor;
        end

    endmodule
```

顶层模块：

```
module top_elevator(
    input wire clk,
    input wire btnU,
    input wire btnL,
    input wire btnC,
    input wire btnR,
    input wire btnD,
    input wire sw0,

    output wire [4:0] led,
    output wire [6:0] seg,
    output wire [3:0] an
);

    wire [4:0] raw_keys;
    wire [4:0] clean_keys;
    wire [3:0] current_floor;

    assign raw_keys = {btnD, btnR, btnC, btnL, btnU};
```

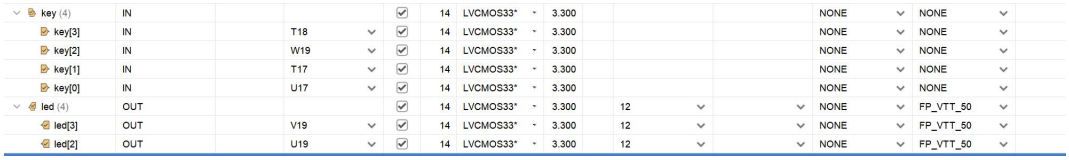
```
    debkey U_Debounce (
        .clk(clk),
        .reset(sw0),
        .key_in(raw_keys),
        .key_out(clean_keys)
    );

    elevator_ctrl U_Ctrl (
        .clk(clk),
        .reset(sw0),
        .key_clean(clean_keys),
        .floor_led(led),
        .current_floor_out(current_floor)
    );

    seg_display U_Disp (
        .floor_num(current_floor),
        .seg(seg),
        .an(an)
    );

endmodule
```

引脚分配：



key (4)	IN			14	LVCMOS33*	3.300			NONE	NONE	
key[3]	IN	T18		14	LVCMOS33*	3.300			NONE	NONE	
key[2]	IN	W19		14	LVCMOS33*	3.300			NONE	NONE	
key[1]	IN	T17		14	LVCMOS33*	3.300			NONE	NONE	
key[0]	IN	U17		14	LVCMOS33*	3.300			NONE	NONE	
led (4)	OUT			14	LVCMOS33*	3.300	12		NONE	FP_VTT_50	
led[3]	OUT	V19		14	LVCMOS33*	3.300	12		NONE	FP_VTT_50	
led[2]	OUT	U19		14	LVCMOS33*	3.300	12		NONE	FP_VTT_50	

五、实验过程中遇到的问题及解决情况(主要问题及解决情况)

1. 消抖模块的位宽适配问题

问题描述：参考资料中的消抖模块 debkey 仅设计了 3 位按键输入 (input [2:0] key)，而本次实验需要控制 5 个楼层，原有模块无法满足需求。

解决情况：

修改了 debkey.v 模块，将输入输出端口位宽扩展为 5 位 ([4:0])。

相应地增加了内部移位寄存器 (key_rrr, key_rr, key_r) 的位宽，使其能够同时对 5 个按键信号进行并行消抖处理。

2. 数码管显示乱码（显示 "E" 而非 "3"）

问题描述：在下载程序后，按下 3 楼按键，逻辑功能正常（电梯到达 3 楼），但数码管显示出字母 "E" 而不是数字 "3"。

原因分析：这是由于约束文件（XDC）中数码管段选信号 seg[6:0] 与物理引脚的映射顺序颠倒所致。FPGA 输出的段码 0110000 (数字 3) 被错

误的引脚顺序解释为了 0110000 的镜像，从而点亮了错误的管脚。

解决情况：重新编写了 .xdc 约束文件，调整了 seg[0] 到 seg[6] 的引脚绑定顺序，使其正确对应开发板上数码管的 a 到 g 段（例如 seg[0] 对应引脚 W7），修正后显示正常。

3. 生成比特流失败 (DRC NSTD-1 错误)

问题描述：在综合实现阶段，Vivado 报错 [DRC NSTD-1] Unspecified I/O Standard，提示端口未指定 I/O 标准。

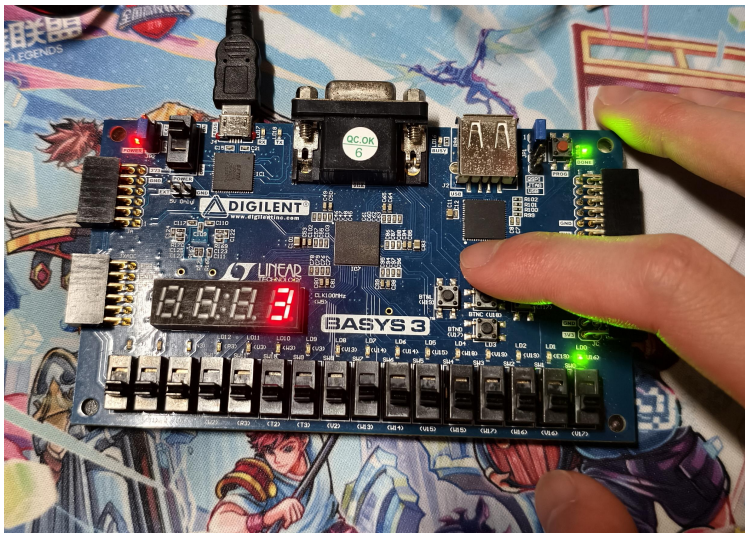
原因分析：初步编写约束文件时，仅指定了引脚位置 (PACKAGE_PIN)，遗漏了电压标准定义，或者在修改 XDC 文件时覆盖了原有的定义。

解决情况：在 .xdc 文件中为每一个端口（clk, reset, btn, led, seg, an）补充了电气标准约束 set_property IOSTANDARD LVCMOS33 [get_ports port_name]，确保所有引脚均工作在 3.3V 逻辑电平下。

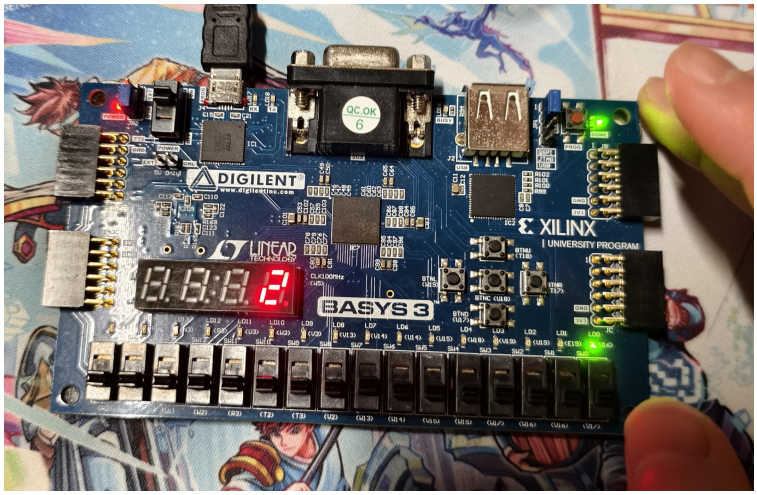
六、实验结果及分析和（或）源程序调试过程

上板结果：

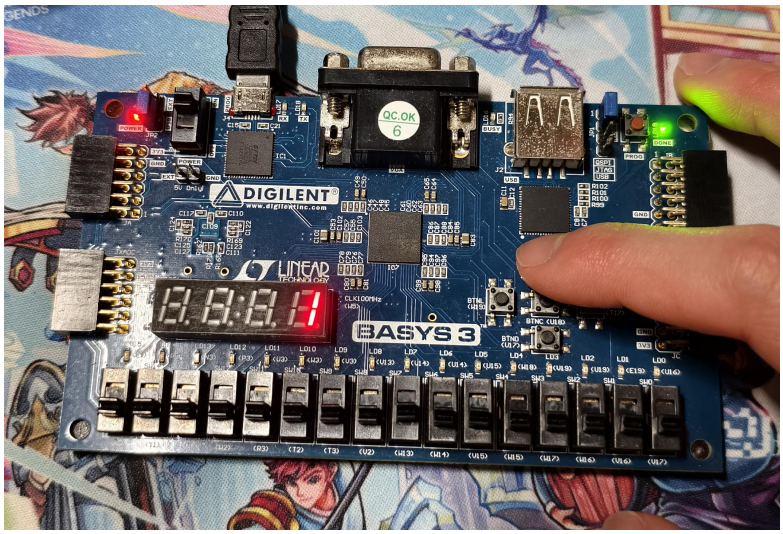
1、电梯停在 3 楼，人在 1 楼召唤，灯亮



2、电梯运行至 2 楼



3、电梯到达 1 楼，灯灭



七、小组分工情况说明

简沅晞：代码编写，仿真测试，综合，引脚分配，上板测试，撰写报告。