基础-3-数码管实验

在许多项目设计中，我们通常需要一些显示设备来显示我们需要的信息，可以选择的显示设备有很多，而数码管是使用最多，最简单的显示设备之一。

3.1 章节导读

本章将通过数码管驱动实验讲解FPGA数字系统中重要的"选通控制"概念。读者将学习到：

1. 数码管工作原理与动态扫描技术
2. 多路复用（Multiplexing）设计方法
3. 参数化模块设计技巧
4. 外设驱动时序规划
5. ASCII到段码的转换原理

实验将使用Verilog HDL实现一个支持8位数码管显示、包含字符动态滚动和选通控制的完整系统。

3.2 理论学习

3.2.1 数码管结构

- 7段数码管组成：A-G段+DP小数点
- 共阳/共阴类型区分（本实验采用共阳型，低电平有效）

3.2.2 动态扫描原理

```
显示周期 = 刷新周期 × 数码管数量
人眼视觉暂留效应（>60Hz）
扫描频率计算公式：f_scan = f_clk / CLK_CYCLE
```

3.2.3 关键技术

- 时分复用：分时选通数码管
- 段码生成：ASCII字符到七段码转换
- 消隐处理：消除切换时的视觉残留

3.2.4 设计指标

参数	值	说明
位数	8	数码管数量
频率	200Hz	单管刷新频率
分辨率	8bit	段码控制（含小数点）

3.2 实战演练

3.3.1 系统架构

```
系统框图：
[Top模块] → [显示驱动模块] → [选通控制模块]
            ↖ ASCII数据生成 ↙ 时钟分频
```

3.3.2 模块设计

led_display_selector

```
module led_display_selector #(
    parameter NUM = 4,
    parameter VALID_SIGNAL = 1'b0,
    parameter CLK_CYCLE = 1000
)(
    input  wire             clk,
    input  wire             rstn,
    input  wire [NUM*8-1:0] led_in,
    output  reg [7:0]       led_display_seg,//[DP,G,F,E,D,C,B,A]
    output  reg [NUM-1:0]   led_display_sel
);

reg [31:0] clk_cnt;
always @(posedge clk or negedge rstn) begin
    if (!rstn) clk_cnt <= 0;
    else if(clk_cnt == CLK_CYCLE) clk_cnt <= 0;
    else clk_cnt <= clk_cnt + 1;
end

wire seg_change = (clk_cnt == CLK_CYCLE) ? 1'b1 : 1'b0;

always @(posedge clk or negedge rstn) begin
    if(!rstn) led_display_sel <= {{(NUM-1){~VALID_SIGNAL}}, VALID_SIGNAL};
    else if (seg_change) led_display_sel <= {led_display_sel[NUM-2:0],
led_display_sel[NUM-1]};
    else led_display_sel <= led_display_sel;
end

integer i;
always @(*) begin
    for(i=0;i<NUM;i=i+1) begin
        if(led_display_sel[NUM-1-i] == VALID_SIGNAL)
            led_display_seg = led_in[i*8 +: 8] ^ ({8{~VALID_SIGNAL}});
    end
end

endmodule //led_display_ctrl
```

led_display_driver

```
module led_display_driver(// 8个数码管显示，阳极管（在selector中已经做了阴阳处理）
    input wire           clk,
    input wire           rstn,
    input wire [8*8-1:0] assic_seg, //ASSIC coding
    input wire [7:0]     seg_point, //显示小数点

    output wire [7:0]    led_display_seg,
    output wire [7:0]    led_display_sel
);

reg [8*8-1:0] led_in;

integer i;
always @(*) begin
    led_in = 0;
    for(i=0;i<8;i=i+1) begin //led_in[i*8 +: 8] <---> assic_seg[i*8 +: 8]
        case (assic_seg[i*8 +: 8])
            "0":     led_in[i*8 +: 8] = (8'h3f) | {seg_point[i],7'b0};
            "1":     led_in[i*8 +: 8] = (8'h06) | {seg_point[i],7'b0};
            "2":     led_in[i*8 +: 8] = (8'h5b) | {seg_point[i],7'b0};
            "3":     led_in[i*8 +: 8] = (8'h4f) | {seg_point[i],7'b0};
            "4":     led_in[i*8 +: 8] = (8'h66) | {seg_point[i],7'b0};
            "5":     led_in[i*8 +: 8] = (8'h6d) | {seg_point[i],7'b0};
            "6":     led_in[i*8 +: 8] = (8'h7d) | {seg_point[i],7'b0};
            "7":     led_in[i*8 +: 8] = (8'h07) | {seg_point[i],7'b0};
            "8":     led_in[i*8 +: 8] = (8'h7f) | {seg_point[i],7'b0};
            "9":     led_in[i*8 +: 8] = (8'h6f) | {seg_point[i],7'b0};
            "A","a": led_in[i*8 +: 8] = (8'h77) | {seg_point[i],7'b0};
            "B","b": led_in[i*8 +: 8] = (8'h7c) | {seg_point[i],7'b0};
            "C","c": led_in[i*8 +: 8] = (8'h39) | {seg_point[i],7'b0};
            "D","d": led_in[i*8 +: 8] = (8'h5e) | {seg_point[i],7'b0};
            "E","e": led_in[i*8 +: 8] = (8'h79) | {seg_point[i],7'b0};
            "F","f": led_in[i*8 +: 8] = (8'h71) | {seg_point[i],7'b0};
            "G","g": led_in[i*8 +: 8] = (8'h3d) | {seg_point[i],7'b0};
            "H","h": led_in[i*8 +: 8] = (8'h76) | {seg_point[i],7'b0};
            "I","i": led_in[i*8 +: 8] = (8'h0f) | {seg_point[i],7'b0};
            "J","j": led_in[i*8 +: 8] = (8'h0e) | {seg_point[i],7'b0};
            "K","k": led_in[i*8 +: 8] = (8'h75) | {seg_point[i],7'b0};
            "L","l": led_in[i*8 +: 8] = (8'h38) | {seg_point[i],7'b0};
            "M","m": led_in[i*8 +: 8] = (8'h37) | {seg_point[i],7'b0};
            "N","n": led_in[i*8 +: 8] = (8'h54) | {seg_point[i],7'b0};
            "O","o": led_in[i*8 +: 8] = (8'h5c) | {seg_point[i],7'b0};
            "P","p": led_in[i*8 +: 8] = (8'h73) | {seg_point[i],7'b0};
            "Q","q": led_in[i*8 +: 8] = (8'h67) | {seg_point[i],7'b0};
            "R","r": led_in[i*8 +: 8] = (8'h31) | {seg_point[i],7'b0};
            "S","s": led_in[i*8 +: 8] = (8'h49) | {seg_point[i],7'b0};
            "T","t": led_in[i*8 +: 8] = (8'h78) | {seg_point[i],7'b0};
            "U","u": led_in[i*8 +: 8] = (8'h3e) | {seg_point[i],7'b0};
            "V","v": led_in[i*8 +: 8] = (8'h1c) | {seg_point[i],7'b0};
            "W","w": led_in[i*8 +: 8] = (8'h7e) | {seg_point[i],7'b0};
            "X","x": led_in[i*8 +: 8] = (8'h64) | {seg_point[i],7'b0};
            "Y","y": led_in[i*8 +: 8] = (8'h6e) | {seg_point[i],7'b0};
            "Z","z": led_in[i*8 +: 8] = (8'h59) | {seg_point[i],7'b0};
```

```
            " ":     led_in[i*8 +: 8] = (8'h00) | {seg_point[i],7'b0};
            "-":     led_in[i*8 +: 8] = (8'h40) | {seg_point[i],7'b0};
            "_":     led_in[i*8 +: 8] = (8'h08) | {seg_point[i],7'b0};
            "=":     led_in[i*8 +: 8] = (8'h48) | {seg_point[i],7'b0};
            "+":     led_in[i*8 +: 8] = (8'h5c) | {seg_point[i],7'b0};
            "(":     led_in[i*8 +: 8] = (8'h39) | {seg_point[i],7'b0};
            ")":     led_in[i*8 +: 8] = (8'h0F) | {seg_point[i],7'b0};
            default: led_in[i*8 +: 8] = (8'h00) | {seg_point[i],7'b0};
        endcase
    end
end

led_display_selector #(
    .NUM            ( 8     ),
    .VALID_SIGNAL   ( 1'b0  ), //阳极管，低电平亮
    .CLK_CYCLE      ( 5000 ))
u_led_display_selector(
    .clk                ( clk             ),
    .rstn               ( rstn            ),
    .led_in             ( led_in          ),
    .led_display_seg    ( led_display_seg  ),
    .led_display_sel    ( led_display_sel  )
);

endmodule //moduleName
```

led_display_top

```
module led_diaplay_top(
    //system io
    input  wire       external_clk ,
    input  wire       external_rstn,
    //led display io
    output wire [7:0] led_display_seg,
    output wire [7:0] led_display_sel
);

reg [43*8-1:0] assic_seg;
reg [7:0]     seg_point;

reg [31:0] clk_cnt;
always @(posedge external_clk or negedge external_rstn) begin
    if(!external_rstn) clk_cnt <= 0;
    else clk_cnt <= clk_cnt + 1;
end

always @(posedge external_clk or negedge external_rstn) begin
    if(!external_rstn) begin
        assic_seg <= "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ -_=+()";
        seg_point <= 8'b00000001;
    end else if({clk_cnt[24]==1'b1} && (clk_cnt[23:0]==25'b0))begin
        assic_seg <= {assic_seg[8*43-8-1:0], assic_seg[8*43-1 -: 8]};
        seg_point <= {seg_point[6:0], seg_point[7]};
    end else begin
```

```
        assic_seg <= assic_seg;
        seg_point <= seg_point;
    end
end

led_display_driver u_led_display_driver(
    .clk                ( external_clk             ),
    .rstn               ( external_rstn            ),
    .assic_seg          ( assic_seg[8*43-1 -: 8*8] ),
    .seg_point          ( seg_point                ),
    .led_display_seg    ( led_display_seg          ),
    .led_display_sel    ( led_display_sel          )
);

endmodule //led_diaplay_top
```

3.3.3 上板验证步骤

1. 设置参数：CLK_CYCLE=5000（对应200Hz扫描频率）
2. 绑定管脚：连接数码管段选/位选信号
3. 观察现象：字符"01234567"应稳定显示
4. 修改assic_seg初始值验证滚动功能

3.4 章末总结

关键收获：

1. 掌握动态扫描消除器件闪烁的原理
2. 理解参数化设计（NUM/VALID_SIGNAL）的优势
3. 学习时序控制中计数器的重要作用
4. 实践ASCII到硬件编码的转换方法

设计亮点：

- 支持阴阳极自动适配（通过VALID_SIGNAL参数）
- 字符环形缓冲区实现无缝滚动
- 参数化设计增强模块复用性

3.5 拓展训练

结合流水灯实验和数码管实验：数码管显示数字，标识出当前流水到了哪一个灯