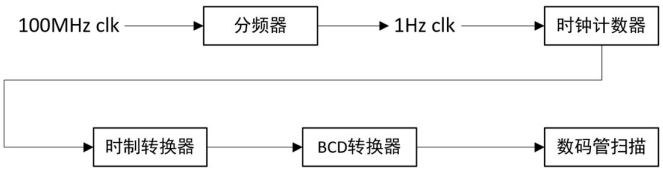
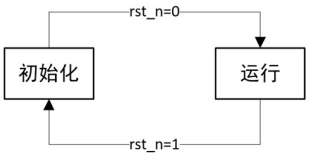
实验三报告 基于 Verilog HDL 的数字电路综合设计

实验一：简易数字钟

系统结构框图



状态图



源程序代码与注释

```
module digital_clock_100M (
    input           clk,        // 100 MHz 系统时钟
    input           rst_n,      // 异步复位，低有效
    output reg [7:0] seg,       // seg[6:0]=A~G 段，seg[7]=DP
    output reg [7:0] an         // 位选 8 位
);

    // -- 1Hz 分频 (100MHz -> 1Hz) --
    reg [26:0] clk_cnt;
    wire       pulse_1hz;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            clk_cnt <= 27'd0;
        else if (clk_cnt == 27'd100_000_000 - 1)
            clk_cnt <= 27'd0;
        else
            clk_cnt <= clk_cnt + 1'b1;
    end
    assign pulse_1hz = (clk_cnt == 27'd100_000_000 - 1);

    // -- 0.5s DP 闪烁 --
    reg blink;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            blink <= 1'b0;
        else if (clk_cnt == 27'd0)
            blink <= 1'b1;
        else if (clk_cnt == 27'd50_000_000)
            blink <= 1'b0;
    end

    // -- 时分秒计数 --
    reg [5:0] sec, min;
    reg [4:0] hr;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            sec <= 6'd0; min <= 6'd0; hr <= 5'd0;
        end else if (pulse_1hz) begin
            if (sec == 6'd59) begin
                sec <= 6'd0;
                if (min == 6'd59) begin
                    min <= 6'd0;
                    hr  <= (hr == 5'd23) ? 5'd0 : hr + 1'b1;
                end else
                    min <= min + 1'b1;
            end else
                sec <= sec + 1'b1;
        end
    end

    // -- BCD 分解 --
    wire [3:0] d0 = sec  % 10;  // 秒低
    wire [3:0] d1 = sec  / 10;  // 秒高
    wire [3:0] d2 = min  % 10;  // 分低
    wire [3:0] d3 = min  / 10;  // 分高
    wire [3:0] d4 = hr   % 10;  // 时低
    wire [3:0] d5 = hr   / 10;  // 时高

    // -- 7 段译码函数 --
    function [6:0] seg_map;
        input [3:0] v;
        case (v)
            4'd0: seg_map = 7'b1000000;
            4'd1: seg_map = 7'b1111001;
            4'd2: seg_map = 7'b0100100;
            4'd3: seg_map = 7'b0110000;
            4'd4: seg_map = 7'b0011001;
            4'd5: seg_map = 7'b0010010;
            4'd6: seg_map = 7'b0000010;
            4'd7: seg_map = 7'b1111000;
            4'd8: seg_map = 7'b0000000;
            4'd9: seg_map = 7'b0010000;
            default: seg_map = 7'b1111111;
        endcase
    endfunction

    // -- 多路扫描 (~125Hz/位) --
    reg [16:0] refresh_cnt;
    reg [2:0]  scan_idx;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            refresh_cnt <= 17'd0;
            scan_idx    <= 3'd0;
        end else if (refresh_cnt == 17'd100_000 - 1) begin
            refresh_cnt <= 17'd0;
            scan_idx    <= scan_idx + 1'b1;
        end else
            refresh_cnt <= refresh_cnt + 1'b1;
    end

    // -- 输出：00.00.00,隐藏末两位 --
    always @(*) begin
        an  = 8'b11111111;
        seg = 8'b11111111;
        case (scan_idx)
            3'd7: begin an[7]=1'b0; seg[6:0]=seg_map(d5); seg[7]=1'b1; end  // 时高（不闪）
            3'd6: begin an[6]=1'b0; seg[6:0]=seg_map(d4); seg[7]=blink; end  // 时低，DP 闪烁(冒号)
            3'd5: begin an[5]=1'b0; seg[6:0]=seg_map(d3); seg[7]=1'b1; end  // 分高
            3'd4: begin an[4]=1'b0; seg[6:0]=seg_map(d2); seg[7]=blink; end  // 分低，DP 闪烁(冒号)
            3'd3: begin an[3]=1'b0; seg[6:0]=seg_map(d1); seg[7]=1'b1; end  // 秒高
            3'd2: begin an[2]=1'b0; seg[6:0]=seg_map(d0); seg[7]=1'b1; end  // 秒低
            default: begin /* scan_idx 1,0 隐藏 */ end
        endcase
    end

endmodule
```

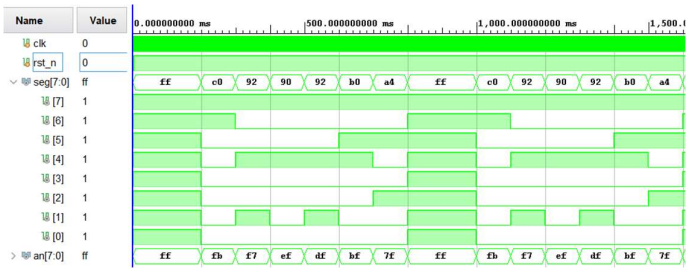
实验结果与分析

- 在时钟的驱动下，观察到 sec 计数器每经过 1 秒递增一次。
- sec 计数器从 0 递增至 59，并在下一个 pulse_1hz 信号到来时归零，同时 min 计数器递增 1 。

- 类似地，min 计数器从 0 递增至 59，并在下一个周期归零，同时 hr 计数器递增 1 。
- hr 计数器从 0 递增至 23，并在下一个周期归零，实现 24 小时制循环 。

仿真波形与分析

```
`timescale 1ns/1ps

module tb_digital_clock_100M;

    reg  clk;
    reg  rst_n;
    wire [7:0] seg;
    wire [7:0] an;

    digital_clock_100M uut (
        .clk(clk),
        .rst_n(rst_n),
        .seg(seg),
        .an(an)
    );

    // 时钟生成 (100 MHz)
    always begin
        #500 clk = ~clk;
    end

    integer i; // 循环变量声明为 integer

    initial begin
        clk = 1'b0;
        rst_n = 1'b0;
        #100;
        rst_n = 1'b1;

        // 强制更改内部变量的值 (例如，跳到 23:59:50)
        #500;
        $display("----------------------------------------");
        $display("强制更改时间...");
        force uut.hr  = 5'd23;
        force uut.min = 6'd59;
        force uut.sec = 6'd50;
        $display("强制时间为: %0d:%0d:%0d", uut.hr, uut.min,
uut.sec);
        $display("----------------------------------------");

        #1000; // 观察强制后的状态

        // 释放强制，让时钟开始自然运行
        release uut.hr;
        release uut.min;
        release uut.sec;

        // 让时钟自然运行一段时间，看看它是否能从 23:59:50 开始递
增
        // 运行 20 秒 (20 * 100,000,000 = 2,000,000,000 ns)
        // 这样可以观察到时钟从 50 秒 -> 59 秒 -> 00 秒 -> 00 分
-> 00 时 (午夜跳变)
        for(i=0;i<10;i=i+1)#2_000_000_000;

        $finish;
    end

endmodule
```



由仿真图可知 an 从低位开始扫描，seg 显示的内容是 235950，与设置的相符。

实验现象与测试照片

如图所示。

遇到的问题与解决方法

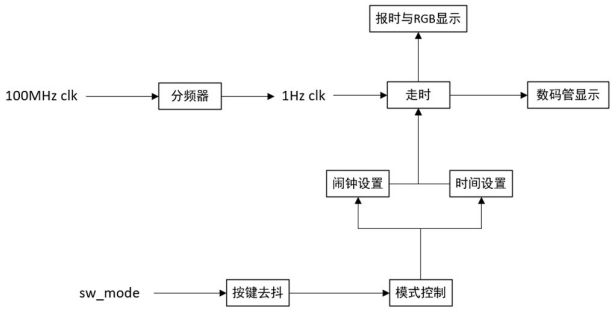
问题	解决方法
秒跳变不连贯	检查分频器精度，确保 1Hz 脉冲稳定
显示闪烁或重影	调整扫描频率，加入死区帧
时分误差	确认进位判断条件正确、无竞争冒险

实验总结与建议

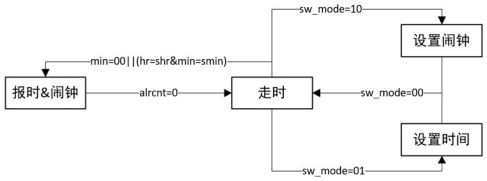
- 使用模块化方法划分功能，提高可读性与调试效率；
- 分频精度是整个系统稳定运行的关键，应重点验证；
- 可扩展暂停、清零、校时等功能，增强可用性。

实验二：高级时钟

系统结构框图



状态图



源程序代码与注释

```
module clock_pro (
    input           clk,
    input           rst_n,
    input   [1:0]   sw_mode,    // 00=正常, 01=设置时间, 10=设置闹钟
    input           sw_field,   // 0=小时, 1=分钟
    input           btn_inc,
    input           btn_dec,
    output          rgb_r,
    output          rgb_g,
    output          rgb_b,
    output reg [7:0] seg,       // [7]=DP active low
    output reg [7:0] an
);

    // ---------------------- 1Hz 分频 ----------------------
    reg [26:0] clk_cnt;
    wire pulse_1hz;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)               clk_cnt <= 0;
        else if (clk_cnt==100_000_000-1) clk_cnt <= 0;
        else                      clk_cnt <= clk_cnt + 1;
    end
    assign pulse_1hz = (clk_cnt == 100_000_000-1);

    // ---------------------- 按键去抖+长按 ----------------------
    reg inc_s0,inc_s1,dec_s0,dec_s1;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) {inc_s0,inc_s1,dec_s0,dec_s1} <= 0;
        else        {inc_s0,inc_s1,dec_s0,dec_s1} <=
{btn_inc,inc_s0,btn_dec,dec_s0};
    end
    wire inc_db = inc_s1, dec_db = dec_s1;
    parameter LONG_THR = 27'd50_000_000, REP_INT = 27'd10_000_000;
    reg inc_pressed, dec_pressed;
    reg [26:0] inc_cnt, dec_cnt;
```

```
    wire inc_pulse = inc_db && (!inc_pressed || (inc_cnt>=LONG_THR &&
inc_cnt%REP_INT==0));
    wire dec_pulse = dec_db && (!dec_pressed || (dec_cnt>=LONG_THR &&
dec_cnt%REP_INT==0));
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) {inc_pressed,inc_cnt,dec_pressed,dec_cnt} <= 0;
        else begin
            if (inc_db) begin inc_pressed<=1; inc_cnt<=inc_cnt+1; end
            else         begin inc_pressed<=0; inc_cnt<=0;       end
            if (dec_db) begin dec_pressed<=1; dec_cnt<=dec_cnt+1; end
            else         begin dec_pressed<=0; dec_cnt<=0;       end
        end
    end

    // ---------------------- 时分秒计数 ----------------------
    reg [5:0] sec, min_cnt;
    reg [4:0] hr_cnt;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)               {sec,min_cnt,hr_cnt} <= 0;
        else begin
            if (sw_mode==2'b00 && pulse_lhz) begin
                if (sec==59) begin sec<=0;
                    if (min_cnt==59) begin min_cnt<=0;
hr_cnt<=(hr_cnt==23)?0:hr_cnt+1; end
                    else min_cnt<=min_cnt+1;
                end else sec<=sec+1;
            end else if (sw_mode==2'b01) begin
                if (inc_pulse)
                    if (!sw_field) hr_cnt<=(hr_cnt==23)?0:hr_cnt+1;
                    else           min_cnt<=(min_cnt==59)?0:min_cnt+1;
                if (dec_pulse)
                    if (!sw_field) hr_cnt<=(hr_cnt==0)?23:hr_cnt-1;
                    else           min_cnt<=(min_cnt==0)?59:min_cnt-1;
                sec<=0;
            end else if (sw_mode==2'b10 && pulse_lhz) begin
                // 在闹钟设置模式后台走秒
                if (sec==59) begin sec<=0;
                    if (min_cnt==59) begin min_cnt<=0;
hr_cnt<=(hr_cnt==23)?0:hr_cnt+1; end
                    else min_cnt<=min_cnt+1;
                end else sec<=sec+1;
            end
        end
    end

    // ---------------------- 闹钟设置 ----------------------
    reg [4:0] alarm_hr;
    reg [5:0] alarm_min;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)              alarm_hr<=0;
        else if (sw_mode==2'b10 && !sw_field) begin
            if (inc_pulse) alarm_hr<=(alarm_hr==23)?0:alarm_hr+1;
            if (dec_pulse) alarm_hr<=(alarm_hr==0)?23:alarm_hr-1;
        end
    end
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)              alarm_min<=0;
        else if (sw_mode==2'b10 && sw_field) begin
            if (inc_pulse) alarm_min<=(alarm_min==59)?0:alarm_min+1;
            if (dec_pulse) alarm_min<=(alarm_min==0)?59:alarm_min-1;
        end
    end

    // ---------------------- 报时 & 闹钟触发 ----------------------
    reg [3:0] alarm_counter;
    wire alarm_active = (alarm_counter > 0);
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            alarm_counter <= 0;
        else if (pulse_lhz) begin
            // 原闹钟触发条件 + 新增整点报时条件（正常模式每小时整点报时，排
除 00:00）
            if (
                (sec==0 && min_cnt==alarm_min && hr_cnt==alarm_hr &&
(hr_cnt!=0 || min_cnt!=0))
                || (sw_mode==2'b00 && sec==0 && min_cnt==0 && hr_cnt!=0)
            )
                alarm_counter <= 10;
            else if (alarm_counter > 0)
                alarm_counter <= alarm_counter - 1;
        end
    end

    // ---------------------- 彩色超平滑过渡 & PWM ----------------------
    // 使用高精度色轮实现平滑过渡
    reg [14:0] color_pos; // 更高精度的色轮位置
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) color_pos <= 0;
        else color_pos <= color_pos + 1; // 不管闹钟是否激活都持续变化
    end

    wire [9:0] wheel_pos = color_pos[14:5]; // 取高 10 位作为 0-1023 范围的色轮
位置
    wire [7:0] pos_in_segment = wheel_pos[7:0]; // 0-255 位置
    wire [1:0] segment = wheel_pos[9:8]; // 4 象限

    // 根据象限和位置计算 RGB 值
    reg [7:0] r_value, g_value, b_value;
    always @(*) begin
        case(segment)
            2'b00: begin // 红→绿
                r_value = 255 - pos_in_segment;
                g_value = pos_in_segment;
                b_value = 0;
            end
            2'b01: begin // 绿→蓝
                r_value = 0;
                g_value = 255 - pos_in_segment;
                b_value = pos_in_segment;
            end
            2'b10: begin // 蓝→红
                r_value = pos_in_segment;
                g_value = 0;
                b_value = 255 - pos_in_segment;
            end
            default: begin // 兼容性处理
                r_value = 255;
                g_value = 0;
                b_value = 0;
            end
        endcase
    end

    // PWM 生成
    reg [7:0] pwm_cnt;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) pwm_cnt <= 0;
        else pwm_cnt <= pwm_cnt + 1;
    end

    // RGB 输出 - 仅在闹钟激活时显示
    assign rgb_r = alarm_active ? (pwm_cnt < r_value) : 1'b0;
    assign rgb_g = alarm_active ? (pwm_cnt < g_value) : 1'b0;
    assign rgb_b = alarm_active ? (pwm_cnt < b_value) : 1'b0;

    // ---------------------- 七段显示 ----------------------
    wire [3:0] d0 = sec % 10, d1 = sec / 10;
    wire [3:0] d2 = min_cnt % 10, d3 = min_cnt / 10;
    wire [3:0] d4 = hr_cnt % 10, d5 = hr_cnt / 10;
    wire [3:0] a0 = alarm_min % 10, a1 = alarm_min / 10;
    wire [3:0] a2 = alarm_hr % 10, a3 = alarm_hr / 10;
    function [6:0] seg_map; input [3:0] v; case(v)
        4'd0:seg_map=7'b1000000; 4'd1:seg_map=7'b1111001;
        4'd2:seg_map=7'b0100100; 4'd3:seg_map=7'b0110000;
        4'd4:seg_map=7'b0011001; 4'd5:seg_map=7'b0010010;
        4'd6:seg_map=7'b0000010; 4'd7:seg_map=7'b1111000;
        4'd8:seg_map=7'b0000000; 4'd9:seg_map=7'b0010000;
        default:seg_map=7'b1111111;
    endcase endfunction

    reg [16:0] refresh_cnt;
    reg [2:0]  scan_idx;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)                {refresh_cnt,scan_idx}<=0;
        else if (refresh_cnt==100_000-1)
{refresh_cnt,scan_idx}<={17'd0,scan_idx+1};
        else                       refresh_cnt<=refresh_cnt+1;
    end

    always @(*) begin
        an  = 8'hFF;
        seg = 8'hFF;
        case(scan_idx)
            // 英文提示: TM / AL
            3'd7: begin
                an[7]=0;
                if (sw_mode==2'b01) seg={1'b1,7'b1000111}; // 'T'
                else if (sw_mode==2'b10) seg={1'b1,7'b0001000}; // 'A'
            end
            3'd6: begin
                an[6]=0;
                if (sw_mode==2'b01) seg={1'b1,7'b1000010}; // 'M'
                else if (sw_mode==2'b10) seg={1'b1,7'b1110001}; // 'L'
            end
            // 时分秒 hh.mm.ss
            3'd5: begin // 小时十位
```

```
                an[5]=0;
                if (sw_mode==2'b10) seg={1'b1, seg_map(a3)}; // 闹钟小时十位
                else                seg={1'b1, seg_map(d5)}; // 时间小时十位
            end
            3'd4: begin // 小时个位
                an[4]=0;
                if (sw_mode==2'b10) seg={1'b0, seg_map(a2)}; // 闹钟小时个
位，带小数点
                else                seg={1'b0, seg_map(d4)}; // 时间小时个
位，带小数点
            end
            3'd3: begin // 分钟十位
                an[3]=0;
                if (sw_mode==2'b10) seg={1'b1, seg_map(a1)}; // 闹钟分钟十位
                else                seg={1'b1, seg_map(d3)}; // 时间分钟十位
            end
            3'd2: begin // 分钟个位
                an[2]=0;
                if (sw_mode==2'b10) seg={1'b0, seg_map(a0)}; // 闹钟分钟个
位，带小数点
                else                seg={1'b0, seg_map(d2)}; // 时间分钟个
位，带小数点
            end
            3'd1: begin an[1]=0; seg={1'b1, seg_map(d1)}; end // 秒十位
            3'd0: begin an[0]=0; seg={1'b1, seg_map(d0)}; end // 秒个位
        endcase
    end
endmodule
```

实验结果与分析

- **设置闹钟时间：** 在闹钟设置模式下，能够通过按键设置 alarm_hr 和 alarm_min 。数码管会相应显示设定的闹钟时间（a3:a2:a1:a0）。
- **闹钟触发：** 当当前时间（hr_cnt:min_cnt:sec）与设定的闹钟时间（alarm_hr:alarm_min:0）精确匹配时，闹钟会被触发. 触发条件包括闹钟时间不为 00:00:00.
- **整点报时：** 在正常模式（sw_mode==2'b00）下，当时间到达每个小时的整点（min_cnt==0 && sec==0）且小时不为 0 时，也会触发报时（类似闹钟触发）.
- **报警指示：** 闹钟触发后，alarm_counter 会被设置为 10，并在接下来的 10 秒内递减 。在此期间，RGB LED 被激活并循环显示彩色光效 ，直至 alarm_counter 归零后停止。

仿真波形与分析

数字钟部分见实验一仿真，闹钟部分见实验结果。

实验现象与测试照片

如图所示。

遇到的问题与解决方法

问题	解决方法
按键无响应	添加去抖和同步电路
闹钟无触发	检查比较逻辑与触发时机
数码管闪烁严重	增加死区帧与分时占空比控制

实验总结与建议

- 用户交互设计中应考虑按钮抖动、视觉反馈等细节；
- 报时输出建议可拓展为蜂鸣器等方式。