电子线路设计、测试与实验
实验报告（七）

实验名称：EDA 多功能数字钟

学院：	电子信息与通信学院
专业班级：	提高 2201 班
姓名：	王翎羽
学号：	U202213806

目录

第七次实验：有限状态机及数字钟

一、 实验名称

有限状态机及数字钟

二、 实验目的

1. 掌握用 verilog HDL 描述数字逻辑电路与系统的方法;
2. 有限状态机概念;
3. 掌握用 verilog HDL 描述有限状态机的方法;
4. 掌握分层次电路设计方法;
5. 熟练掌握数字钟的设计与调试方法。

三、 实验任务

步进电机脉冲分配器设计与多功能数字钟设计

1. 功能要求

1.使用组合逻辑，能显示小时、分钟、秒钟
2.能调整小时、分钟的时间。

2. 验收内容

代码及实验板实操结果

四、 实验原理

1. 状态图的描述方法

1.利用 parameter 描述状态机中各个状态的名称，并指定状态编码。例如，对序列检测器的状态分配使用最简单的自然二进制码，其描述如下:parameter S0=2'b00,S1=2'b01,S2=2'b10,S3 = 2'b11;

2.用 always 块描述状态触发器实现状态存储。

3.使用敏感表和 case 语句(也可以采用 if-else 等价语句)描述的状态转换逻辑。

4.描述状态机的输出逻辑。

2. 脉冲分配器设计步骤

1.创建子目录 E:\EDA_Lab\Lab5，并新建一个工程项目。

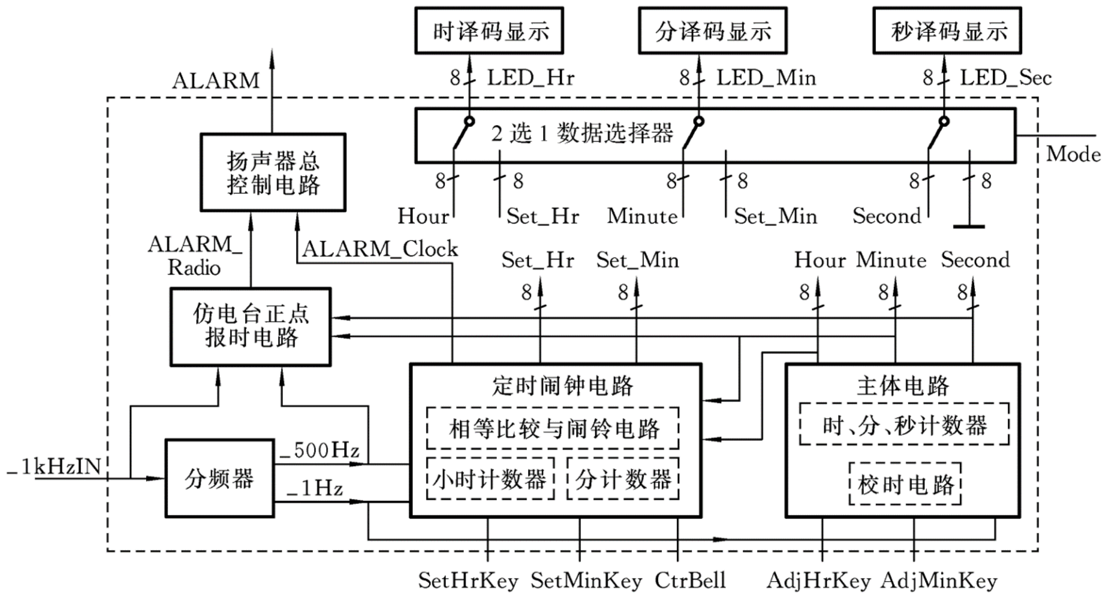
2.使用 Verilog HDL 设计电路，并进行仿真分析。

3.用 FPGA 开发板实现步进电机脉冲分配器，并实际测试逻辑功能。(A、B、C 用发光二极管代替)。

4.根据实验流程和实验结果，写出实验总结报告，并对波形图和实验现象进行说明。

3. 自顶而下的设计方法

先设计顶层总框图,该框图由若干个具有特定功能的源模块组成。下一步针对这些具有不同功能的模块进行设计,对于有些功能复杂的模块,还可以将该模块继续化分为若干个功能子模块，这样就形成模块套模块的层次化设计方法。



五、 实验过程

1. 有限状态机设计代码

1. 分频器模块

```
module clk_divider (
    input clk_125MHz,
    input rst_n,
    output reg clk_1Hz
);

    reg [26:0] count; // 27 位计数器, 用于计数 125M 时钟的周
期数


    always @(posedge clk_125MHz or negedge rst_n) begin
        if (!rst_n) begin
            count <= 0;
            clk_1Hz <= 0;
        end else begin
            if (count == 100_000_000) begin
                count <= 0;
                clk_1Hz <= ~clk_1Hz; // 1 秒翻转一次
            end else begin
                count <= count + 1;
            end
        end
    end

endmodule
```

2. 步进模块

```
module fsm (
    input clk, // 时钟信号
    input rst_n, // 复位信号（低电平有效）
    input A, // 输入信号 A
    output reg [2:0] state // 输出当前状态
);

    parameter S0 = 3'b000,
              S1 = 3'b001,
              S2 = 3'b010,
              S3 = 3'b011,
              S4 = 3'b100,
              S5 = 3'b101,
              S6 = 3'b110,
              S7 = 3'b111;

    reg [2:0] next_state; // 下一个状态

    // 状态机逻辑
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            state <= S0; // 复位时状态置为 S0
        end else begin
            state <= next_state; // 正常情况下根据
next_state 更新状态
        end
    end

    // 组合逻辑，根据当前状态和输入信号 A 更新下一个状态
    always @(*) begin
        case(state)
            S0: next_state = A ? S6 : S6;
            S1: next_state = A ? S3 : S5;
            S2: next_state = A ? S6 : S3;
            S3: next_state = A ? S2 : S1;
            S4: next_state = A ? S5 : S6;
            S5: next_state = A ? S1 : S4;
            S6: next_state = A ? S4 : S2;
            S7: next_state = A ? S0 : S0;
            default: next_state = S0;
        endcase
    end

endmodule
```

3. 顶层模块

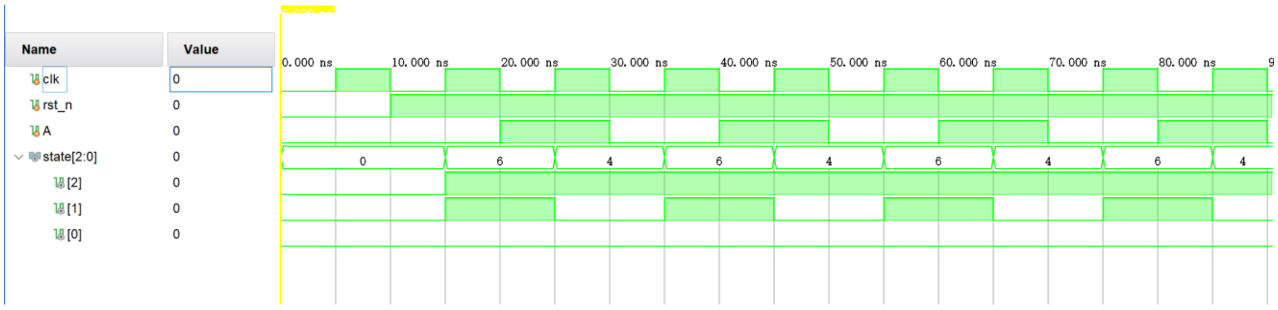
```
module top (
    input clk_125MHz,
    input rst_n,
    input A,
    output wire [2:0] state
);

    wire clk_1Hz;

    clk_divider divider (
        .clk_125MHz(clk_125MHz),
        .rst_n(rst_n),
        .clk_1Hz(clk_1Hz)
    );

    fsm fsm_inst (
        .clk(clk_1Hz),
        .rst_n(rst_n),
        .A(A),
        .state(state)
    );

endmodule
```

1. 有限状态机机测试代码和仿真结果

1. 仿真代码

```
`timescale 1ns / 1ps

module fsm_tb;

    reg clk;
    reg rst_n;
    reg A;
    wire [2:0] state;

    fsm dut (
        .clk(clk),
        .rst_n(rst_n),
        .A(A),
        .state(state)
    );

    // 时钟信号生成
    always #5 clk = ~clk;

    // 初始化
 initial begin
    clk = 0;
    rst_n = 0;
    A = 0;
    #10;
    rst_n = 1;
    #10;

    // 循环激励状态机
    repeat (5) begin
        A = 1;
        #10;
        A = 0;
        #10;
    end

    $finish;
end


    // 打印状态
    always @(state) begin
        $display("State: %b", state);
    end

endmodule
```

2. 仿真结果



3. 数字钟代码

1.顶层文件

```
    `timescale 1ns / 1ps

    module Clock(
        input wire clock_en,
        input wire CR,
        input wire CP,
        input wire adjust_hour_en,
        input wire adjust_minute_en,
        input wire second_continue,
        input wire show_mode,
        input wire punctually_report_en,
        input wire alarm_switch,
        input wire set_alarm_en,
        input wire set_alarm_time_hour,
        input wire set_alarm_time_minute,
        output reg [7:0] tubePosSignal,
        output reg [6:0] tubeShowSignal,
        output reg punctuallyReportSignal,
        output reg alarmReportSignal
        );

        wire CLK_1k, CLK_1Hz, CLK_2Hz;
        wire secondInputSignal, minuteInputSignal, hourInputSignal;
        wire isPunctuallyReporting;
        wire [7:0] second, minute, hour;
        wire [6:0] secondOnesShowCode, secondTensShowCode,
minuteOnesShowCode, minuteTensShowCode, hourOnesShowCode,
hourTensShowCode;
        wire [6:0] alarmHourOnesShowCode, alarmHourTensShowCode,
alarmMinuteOnesShowCode, alarmMinuteTensShowCode;
        wire secondToMinuteCarryBit, minuteToHourCarryBit;
        wire [4:0] hour_real_num;
        reg [7:0] alarm_hour, alarm_minute;

        FrequencyDivider_1k divider_1k(CP, CLK_1k);
        FrequencyDivider_1hz divider_1Hz(CLK_1k, CR, clock_en,
CLK_1Hz);
        FrequencyDivider_2hz divider_2Hz(CLK_1k, CR, clock_en,
CLK_2Hz);

        TwoToOneSelector minuteInput(secondToMinuteCarryBit, CLK_1Hz,
~adjust_minute_en, minuteInputSignal);
        TwoToOneSelector hourInput(minuteToHourCarryBit, CLK_1Hz,
~adjust_hour_en, hourInputSignal);
        TwoToOneSelector secondInput(CLK_1Hz, 1'b0, ~adjust_hour_en
&& ~adjust_minute_en, secondInputSignal);

        Counter_60 secondCounter(secondInputSignal, second_continue,
CR, second, secondToMinuteCarryBit);
        Counter_60 minuteCounter(minuteInputSignal, clock_en, CR,
minute, minuteToHourCarryBit);
        Counter_24 hourCounter(hourInputSignal, CR, clock_en,
hour[3:0], hour[7:4]);

        assign hour_real_num = hour[3:0] + 10 * hour[7:4];
        always @(*) begin
            alarm_hour = alarm_hour_set;
            alarm_minute = alarm_minute_set;
        end

        PunctuallyReporter PunctuallyReporter (minute, hour_real_num,
punctually_report_en, CLK_1Hz, isPunctuallyReporting,
punctuallyReportSignal);

        TubeDecoder secondOnesDecoder(second[3:0],
secondOnesShowCode);
        TubeDecoder secondTensDecoder(second[7:4],
secondTensShowCode);
        TubeDecoder minuteOnesDecoder(minute[3:0],
minuteOnesShowCode);
        TubeDecoder minuteTensDecoder(minute[7:4],
minuteTensShowCode);
        TubeDecoder HourOnesDecoder(hour[3:0], hourOnesShowCode);
        TubeDecoder hourTensDecoder(hour[7:4], hourTensShowCode);
        TubeDecoder alarmMinuteOnesDecoder(alarm_minute[3:0],
alarmMinuteOnesShowCode);
```

```
        TubeDecoder alarmMinuteTensDecoder(alarm_minute[7:4],
alarmMinuteTensShowCode);
        TubeDecoder alarmHourOnesDecoder(alarm_hour[3:0],
alarmHourOnesShowCode);
        TubeDecoder alarmHourTensDecoder(alarm_hour[7:4],
alarmHourTensShowCode);

        TubeShower shower(CLK_1k, CLK_2Hz, show_mode, ~set_alarm_en,
isPunctuallyReporting, hour_real_num, hour,
                           secondOnesShowCode, secondTensShowCode,
minuteOnesShowCode, minuteTensShowCode,
                           hourOnesShowCode, hourTensShowCode,
alarmHourTensShowCode, alarmHourOnesShowCode,
                           alarmMinuteTensShowCode,
alarmMinuteOnesShowCode, tubePosSignal, tubeShowSignal);

        AlarmReporter alarmReporter(alarm_switch, alarm_hour,
alarm_minute, hour, minute, CLK_1Hz, alarmReportSignal);
        AlarmSetter alarmSetter(set_alarm_time_hour && ~set_alarm_en,
set_alarm_time_minute && ~set_alarm_en, CLK_1Hz, alarm_hour_set,
alarm_minute_set);

    endmodule
```

2.分频为 1khz

```
`timescale 1ns / 1ps

module FrequencyDivider_1k(
    input CP,
    output reg CLK_1k = 0
    );
    reg [15:0] state;
    always @(posedge CP)
        begin
//        for real use
            if (state < 49999) state <= state + 1'b1;
//          for sim
        //  if (state < 4) state <= state +1'b1;
            else
                begin
                    state <= 0;
                    CLK_1k <= ~CLK_1k;
                end
        end
```

```
endmodule
```

3.分频为 1hz

```
`timescale 1ns / 1ps



module FrequencyDivider_1hz(
    input CLK_1k,
    input CR,
    input EN,
    output reg CLK_1Hz = 0
    );
    reg [8:0] state;
    always @(posedge CLK_1k or negedge CR)
        begin
            if (~CR)
                begin
                    CLK_1Hz <= 0;
                    state <= 0;
                end
            else if (~EN)
                begin
                    CLK_1Hz <= CLK_1Hz;
                    state <= state;
                end
//              for real use, here should be 499,but for test, we can
change it into 49 or 4
            else if (state < 499) state <= state + 1'b1;
//              else if (state < 49) state <= state + 1'b1;
            //  else if (state < 4) state <= state +1'b1;
            else
                begin
                    state <= 0;
                    CLK_1Hz <= ~CLK_1Hz;
                end
        end
endmodule
```

4.二选一选择器

```
    module TwoToOneSelector(
        input inputA,
        input inputB,
        input selectSignal,
        output reg outputSignal
        );
        always @(*)
            begin
                if(selectSignal == 1'b0)
                    outputSignal <= inputA;
                else
                    outputSignal <= inputB;
            end
    endmodule
```

5.60 计数器

```
`timescale 1ns / 1ps



module Counter_60(
    input CP,
    input EN,
    input CR,
    output reg [7:0] Q,
    output reg carryBit
    );
    wire onesToTensCarryBit;
    wire tensToUpperCarryBit;
    wire [3:0] ones,tens;
    Counter_10 OnesPlace(CP,CR,EN,ones,onesToTensCarryBit);
    Counter_6
TensPlace(onesToTensCarryBit,CR,EN,tens,tensToUpperCarryBit);
    always @(*) Q = {tens[3:0],ones[3:0]};
    always @(*) carryBit = tensToUpperCarryBit;
endmodule
```

6.24 计数器

```
    `timescale 1ns / 1ps
    module Counter_24(CP,CR,EN,Q_ones,Q_tens);
        input CP;
        input CR;
        input EN;
        output reg [3:0] Q_ones = 4'b0000;
        output reg [3:0] Q_tens = 4'b0000;

        always @(posedge CP or negedge CR)
            begin
                if (~CR)
                    begin
                        Q_ones <= 4'b0000;
                        Q_tens <= 4'b0000;
                    end
                else if(~EN)
                    begin
                        Q_ones <= Q_ones;
                        Q_tens <= Q_tens;
                    end
                else if (Q_ones == 4'b1001 && Q_tens < 4'b0010)
                    begin
                        Q_ones <= 4'b0000;
                        Q_tens <= Q_tens +1'b1;
                    end
                else if (Q_ones == 4'b0011 && Q_tens == 4'b0010)
                    begin
                        Q_tens <= 4'b0000;
                        Q_ones <= 4'b0000;
                    end
                else
                    begin
                        Q_ones <= Q_ones + 1'b1;
                    end
            end
    endmodule
```

6. 整点&&闹钟显示

```
    `timescale 1ns / 1ps

    module PunctuallyReporter(
        input [7:0] minute,
        input [4:0] hour,
        input EN,
        input CLK_1Hz,
        output reg isReporting = 0,
        output reg reportSignal = 0
        );
        reg [5:0] flashingTime = 0;
        reg hasReport = 0;
        always @(posedge CLK_1Hz)
            begin
                if (EN == 0) reportSignal = 1'b0;
                else if (minute[7:0] == 0 && flashingTime < 2 *
    (hour[4:0]) && hasReport == 1'b0)
                    begin
                        reportSignal <= ~reportSignal;
                        flashingTime <= flashingTime + 1'b1;
                    end
                else
                    begin
                        reportSignal <= 1'b0;
                        flashingTime <= 1'b0;
                        hasReport <= 1'b1;
                    end

                if (minute[7:0] > 0)
                    hasReport <= 1'b0;
                if (flashingTime > 0)
                    isReporting <= 1;
                else isReporting <= 0;
            end
    endmodule
```

7. 译码显示

```
`timescale 1ns / 1ps

module TubeDecoder(
    input [3:0] number,
    output reg [6:0] code
    );
    always @(number)
        begin
            case(number)
                4'd0: code <= 7'b100_0000;
                4'd1: code <= 7'b111_1001;
                4'd2: code <= 7'b010_0100;
                4'd3: code <= 7'b011_0000;
                4'd4: code <= 7'b001_1001;
                4'd5: code <= 7'b001_0010;
                4'd6: code <= 7'b000_0010;
                4'd7: code <= 7'b111_1000;
                4'd8: code <= 7'b000_0000;
                4'd9: code <= 7'b001_0000;

                4'ha: code <= 7'b000_1000;//show A
                4'hb: code <= 7'b000_1100;//show P
                default: code <= 7'b111_1111;
                endcase
            end
        endmodule
```

8.亮灯逻辑

```
    `timescale 1ns / 1ps



    module TubeShower(
        input CLK_1k,
        input CLK_2Hz,
        input showMode,//for show mode 0, 24h mode clock;mode 1, 12h
mode clock
        input isSettingAlarm,
        input isPunctuallyReporting,// when is punctually reporting
HH:mm flashs per 0.5 Sec,last two bit show flash time
        input [4:0] hour_real_num,//lower 4 bits are for hour ones
bit,upper 4 bits are for hour tens bit
        input [7:0] hour,
        input [6:0] second_ones,
        input [6:0] second_tens,
        input [6:0] minute_ones,
        input [6:0] minute_tens,
        input [6:0] hour_ones,
        input [6:0] hour_tens,
        input [6:0] alarm_hour_setting_tens,
        input [6:0] alarm_hour_setting_ones,
        input [6:0] alarm_minute_setting_tens,
        input [6:0] alarm_minute_setting_ones,
        output reg [7:0] tubePos,
        output reg [6:0] showCode
        );
        integer k = 0;
        wire [6:0] convert0;
        wire [7:0] hour_12;
        reg [3:0] convert_ones,convert_tens;
        always@ (*)
            begin
                if (hour_real_num == 5'd20 || hour_real_num == 5'd21)
                    convert_ones <= hour[3:0] + 8;
                else
                    convert_ones <= hour[3:0] - 2;
            end
        TubeDecoder decoder0(convert_ones,convert0);

        always @(posedge CLK_1k)
            begin
                case(k)
                    0:
    //                show A or P or none for 12/24 hour switch or
noting when set alarm
                      begin
                          tubePos <= 8'b1111_1110;
                          // when is setting alarm, show nothing
                          if (isSettingAlarm) showCode <=
7'b111_1111;
                          // when is punctually reporting,show hour
ones bit
                          else if (isPunctuallyReporting) showCode
<= hour_ones;
                          else if (showMode == 0) showCode <=
7'b111_1111;
                          else if (showMode == 1)
                              begin
                                  if (hour_real_num >= 5'd12)
showCode <= 7'b000_1100;
                                  else showCode <= 7'b000_1000;
                              end
                          k <= k + 1;
                      end
                    1:
    //                show nothing
                      begin
                          tubePos <= 8'b1111_1101;
                          // when is punctually reporting,show hour
tens bit
                          if (isPunctuallyReporting) showCode <=
hour_tens;
                          else showCode <= 7'b111_1111;
                          k <= k + 1;
                      end
                    2:
    //                show second ones bit or noting when set alarm
                      begin
                          tubePos <= 8'b1111_1011;
                          if (isSettingAlarm) showCode <=
7'b111_1111;
```

```
                        else
                            begin
                                showCode <= second_ones;
                            end
                        k <= k + 1;
                    end
                3:
    //              show second tens bit or noting when set alarm
                    begin
                        tubePos <= 8'b1111_0111;
                        if (isSettingAlarm) showCode <=
7'b111_1111;
                        else
                            begin
                                showCode <= second_tens;
                            end
                        k <= k + 1;
                    end
                4:
    //              show minute ones bit
                    begin
                        tubePos <= 8'b1110_1111;
                        if (isSettingAlarm) showCode <=
alarm_minute_setting_ones;
                        else if (isPunctuallyReporting)
                            begin
                                if (CLK_2Hz == 1'b1) showCode <=
minute_ones;
                                else showCode <= 7'b111_1111;
                            end
                        else showCode <= minute_ones;
                        k <= k + 1;
                    end
                5:
    //              show mintue tens bit
                    begin
                        tubePos <= 8'b1101_1111;
                        if (isSettingAlarm) showCode <=
alarm_minute_setting_tens;
                        else if (isPunctuallyReporting)
                            begin
                                if (CLK_2Hz == 1'b1) showCode <=
minute_tens;
                                else showCode <= 7'b111_1111;
```

```
                                        end
                                    else showCode <= minute_tens;
                                    k <= k + 1;
                                end
                            6:
    //                        show hour ones bit
                                begin
                                    tubePos <= 8'b1011_1111;
                                    if (isSettingAlarm) showCode <=
alarm_hour_setting_ones;
                                    else if (isPunctuallyReporting)
                                        begin
                                            if (CLK_2Hz == 1'b0) showCode <=
7'b111_1111;
                                            else
                                                begin
                                                    if (showMode == 0 ||
hour_real_num <= 5'd12)
                                                        showCode <=
hour_ones;
                                                    else if (showMode == 1 &&
hour_real_num > 5'd12)
                                                        showCode <= convert0;
                                                end
                                        end
                                    else
                                        begin
                                            if (showMode == 0 ||
hour_real_num <= 5'd12)
                                                showCode <= hour_ones;
                                        else if (showMode == 1 &&
hour_real_num > 5'd12)
                                                showCode <= convert0;
                                        end
                                    k <= k + 1;
                                end
                            7:
    //                        show hour tens bit
                                begin
                                    tubePos <= 8'b0111_1111;
                                    if (isSettingAlarm) showCode <=
alarm_hour_setting_tens;
                                    else if (isPunctuallyReporting)
                                        begin
                                    if (CLK_2Hz == 1'b0) showCode <=
7'b111_1111;
                                    else
                                        if (showMode == 0 ||
hour_real_num <= 5'd12) showCode <= hour_tens;
                                        else if (showMode == 1)
                                            begin
                                                if (hour_real_num <
5'd22 && hour_real_num > 5'd12)
                                                    showCode <=
7'b100_0000;
                                                else
                                                    showCode <=
7'b111_1001;
                                            end
                                end
                            else
                                begin
                                    if (showMode == 0 ||
hour_real_num <= 5'd12) showCode <= hour_tens;
                                    else if (showMode == 1)
                                        begin
                                            if (hour_real_num < 5'd22
&& hour_real_num > 5'd12)
                                                showCode <=
7'b100_0000;
                                            else
                                                showCode <=
7'b111_1001;
                                        end
                                end
                            k <= k + 1;
                        end
                    8: k <= 0;
                endcase
            end
    endmodule
```

9.闹钟响铃

```
    `timescale 1ns / 1ps


  module AlarmReporter(
        input EN,
        input [7:0] hour_set_num,
        input [7:0] minute_set_num,
        input [7:0] hour_current_num,
        input [7:0] minute_current,
        input CLK_1Hz,
        output reg reportSignal = 0
        );
        always @(posedge CLK_1Hz)
            begin
                if (EN == 0)
                    reportSignal <= 1'b0;
                else    if    (hour_current_num    ==    hour_set_num    &&
minute_current == minute_set_num)
                    reportSignal <= ~reportSignal;
                else
                    reportSignal <= 1'b0;
            end
  endmodule
```

10. 闹钟设置

```
    `timescale 1ns / 1ps


    module AlarmSetter(
        input set_hour_en,
        input set_minute_en,
        input CLK_1Hz,
        output reg [7:0] hour_set = 8'b0000_0000,
        output reg [7:0] minute_set = 8'b0000_0000
        );
        reg onesToTensCarryBit;
        always @(posedge CLK_1Hz) begin
            if (~set_minute_en) minute_set[3:0] <= minute_set[3:0];
            else if (minute_set[3:0] == 4'b1001)
                begin
                    minute_set[3:0] <= 4'b0000;
                    onesToTensCarryBit = 1'b1;
                end
            else
                begin
                    minute_set[3:0] <= minute_set[3:0] + 1'b1;
                    onesToTensCarryBit = 1'b0;
            end
        end

        always @(posedge onesToTensCarryBit) begin
            if (~set_minute_en) minute_set[7:4] <= minute_set[7:4];
            else if (minute_set[7:4] == 4'b0101)
                minute_set[7:4] <= 4'b0000;
            else
                minute_set[7:4] <= minute_set[7:4] + 1'b1;
        end

        always @(posedge CLK_1Hz) begin
            if (~set_hour_en) hour_set <= hour_set;
            else if (hour_set[3:0] == 4'b1001 && hour_set[7:4] <
4'b0010)
                begin
                    hour_set[3:0] <= 4'b0000;
                    hour_set[7:4] <= hour_set[7:4]+1'b1;
                end
            else if (hour_set[3:0] == 4'b0011 && hour_set[7:4] ==
4'b0010)
                hour_set = 8'b0000_0000;
            else
                hour_set[3:0] <= hour_set[3:0] + 1'b1;
        end
    endmodule
```

六、 实验小结

通过这次实验，我深刻体验到了数字电路设计的难度。在实验过程中，我不断尝试、纠错，逐渐掌握了 Vivado 等软件的使用方法，并领悟了数字电路软件设计的方法论。这次实验让我对常见错误和设计重点有了更深入的理解，为我后续课程的学习打下了坚实的基础。虽然数字钟实验十分考验细致和全面考虑的能力，但我相信通过这样的实践，后面的学习会更加顺利。