实验七：EDA多功能数字钟

专业班级：**通信2101班**

姓名：　　**罗畅**

学号：　　**U202113940**

实验名称

EDA多功能数字钟

实验目的：

- 了解数字钟的功能要求及设计方法；
- 了解CPLD/FPGA的一般结构及开发步骤；
- 掌握ISE 13.4软件的使用；
- 熟悉用FPGA器件取代传统的中规模集成器件实现数字电路与系统的方法。

实验仪器

Vivado2018.03*，*Nexys4 DDR* 开发板，数据线，*Visual Studio Code-Insiders

实验任务

基础

- 设计多功能数字钟，使用组合逻辑，能显示小时，分钟，秒
- 小时为24进制，分和秒用60进制
- 可以调整小时，分的时间

提高

- 可以切换12/24小时制
- 可以设定任意时刻闹钟
- 整点报数（几点LED灯就闪烁几次）

验收

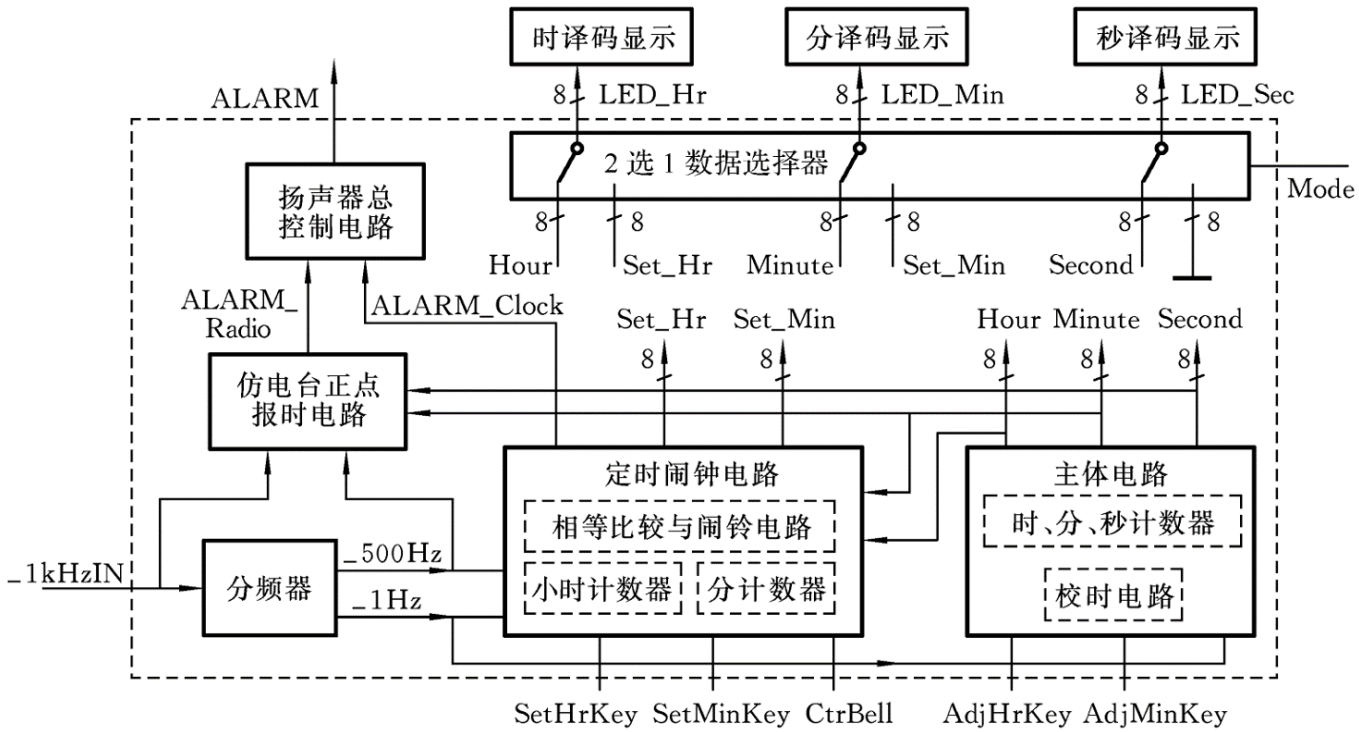
- 教师现场要求增加功能

实验原理

自顶向下的设计模式

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。

数字钟整体框图



实验代码

注：代码缩进和注释均进行过调整，和实际项目地址中的代码略有差别

GitHub项目地址：GitHub - HUSTerCH/digital_clock

Clock.v

```
`timescale 1ns / 1ps

module Clock(
    input clock_en,
    input CR,
    input CP,
    input adjust_hour_en,
    input adjust_minute_en,
    input second_continue,
    input show_mode,
    input punctually_report_en,
    input alarm_switch,
    input set_alarm_en,
    input set_alarm_time_hour,
    input set_alarm_time_minute,
    output [7:0] tubePosSignal,
    output [6:0] tubeShowSignal,
    output punctuallyReportSignal,
    output alarmReportSignal
    );
    wire CLK_1k,CLK_1Hz,CLK_2Hz;
    wire secondInputSignal,minuteInputSignal,hourInputSignal;
    wire isPunctuallyReporting;
// lower 3 bits is for ones bit,and upper 3bits is for tens bit
    wire [7:0] second;
    wire [7:0] minute;
    wire [7:0] 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;
    assign hour_real_num = hour[3:0] + 10 * hour[7:4];

    wire [7:0] alarm_hour_set;
    wire [7:0] alarm_minute_set;

    reg [7:0] alarm_hour,alarm_minute;
    always @(*) alarm_hour = alarm_hour_set;
    always @(*) alarm_minute = alarm_minute_set;


    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]);

    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,alarmReportSign
al);
    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
```

AlarmAdjuster.v

```
`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
```

AlarmReporter.v

```
  `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
```

FrequencyDivider_1k.v

```
`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
```

FrequencyDivider_1hz.v

```
`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 sim, we 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
```

FrequencyDivider_2Hz.v

```
`timescale 1ns / 1ps

module FrequencyDivider_2hz(
    input CLK_1k,
    input CR,
    input EN,
    output reg CLK_2Hz = 0
);
    reg [7:0] state;
    always @(posedge CLK_1k)
        begin
            if (~CR)
                begin
                    CLK_2Hz <= 0;
                    state <= 0;
                end
            else if (~EN)
                begin
                    CLK_2Hz <= CLK_2Hz;
                    state <= 0;
                end
        else if (state < 249) state <= state + 1'b1;
        else
            begin
                state <= 0;
                CLK_2Hz <= ~CLK_2Hz;
            end
    end
endmodule
```

Counter_60.v

```
`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 @(ones,tens) Q = {tens[3:0],ones[3:0]};
    always @(tensToUpperCarryBit) carryBit = tensToUpperCarryBit;
endmodule
```

Counter_10.v

```
`timescale 1ns / 1ps

module Counter_10(
    input CP,
    input CR,
    input EN,
    output reg [3:0] Q = 4'b0000,
    output reg carryBit
    );
    always @(posedge CP or negedge CR)
        begin
            if (~CR) Q <= 4'b0000;
            else if(~EN) Q <= Q;
            else if (Q == 4'b1001)
                begin
                    Q <= 4'b0000;
                    carryBit = 1'b1;
                end
            else
                begin
                    Q <= Q + 1'b1;
                    carryBit = 1'b0;
                end
        end
endmodule
```

Counter_6.v

```
`timescale 1ns / 1ps

module Counter_6(
    input CP,
    input CR,
    input EN,
    output reg [3:0] Q  =4'b0000,
    output reg carryBit
    );
    always @(posedge CP or negedge CR)
        begin
            if (~CR) Q <= 4'b0000;
            else if(~EN) Q <= Q;
            else if (Q == 4'b0101)
                begin
                    Q <= 4'b0000;
                    carryBit = 1'b1;
                end
            else
                begin
                    Q <= Q + 1'b1;
                    carryBit = 1'b0;
                end
        end
endmodule
```

Counter_24.v

```
`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
```

TwoToOneSelector.v

```
`timescale 1ns / 1ps

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

PunctuallyReporter.v

```
`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
```

TubeDecoder.v

```
`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
```

TubeShower.v

```
`timescale 1ns / 1ps

module TubeShower(
    input CLK_1k,
    input CLK_2Hz,
    //for show mode 0, 24h mode clock;mode 1, 12h mode clock
    input showMode,
    input isSettingAlarm,
// when is punctually reporting HH:mm flashs per 0.5 Sec,
// last two bit show flash time
    input isPunctuallyReporting,
//lower 4 bits are for hour ones bit,
//upper 4 bits are for hour tens bit
    input [4:0] hour_real_num,
    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
```

仿真部分：clock_tb.v

```
`timescale 1ns / 1ps

module clock_tb;
    reg clock_en;
    reg CR;
    reg CP;
    reg adjust_hour_en;
    reg adjust_minute_en;
    reg second_stop;
    reg show_mode;
    reg punctually_report_en;
    reg alarm_switch;
    reg set_alarm_en;
    reg set_alarm_time_hour;
    reg set_alarm_time_minute;
    wire [7:0]tubePosSignal;
    wire [6:0]tubeShowSignal;
    wire punctuallyReportSignal;
    wire alarmReportSignal;
Clock clock(
     clock_en,
     CR,
     CP,
     adjust_hour_en,
     adjust_minute_en,
     second_stop,
     show_mode,
     punctually_report_en,
     alarm_switch,
     set_alarm_en,
     set_alarm_time_hour,
     set_alarm_time_minute,
     tubePosSignal,
     tubeShowSignal,
     punctuallyReportSignal,
     alarmReportSignal
    );
    always #1 CP = ~CP;
    initial
        begin
            clock_en = 1'b1;
            CP = 1'b1;
            CR = 1'b1;
            adjust_hour_en = 1'b1;
            adjust_minute_en = 1'b1;
            show_mode = 1'b0;
            punctually_report_en = 1'b1;
            set_alarm_en = 1'b0;
            set_alarm_time_hour = 1'b1;
            set_alarm_time_minute = 1'b1;
```

```
            #4000
            set_alarm_en = 1'b1;
            adjust_hour_en = 1'b0;
            adjust_minute_en = 1'b0;
            #3660
            adjust_hour_en = 1'b1;
            adjust_minute_en = 1'b1;
            alarm_switch = 1;
            $stop;
        end

endmodule
```

约束文件 clock_test.xdc

```
set_property PACKAGE_PIN U13 [get_ports {tubePosSignal[7]}]
set_property PACKAGE_PIN K2 [get_ports {tubePosSignal[6]}]
set_property PACKAGE_PIN T14 [get_ports {tubePosSignal[5]}]
set_property PACKAGE_PIN P14 [get_ports {tubePosSignal[4]}]
set_property PACKAGE_PIN J14 [get_ports {tubePosSignal[3]}]
set_property PACKAGE_PIN T9 [get_ports {tubePosSignal[2]}]
set_property PACKAGE_PIN J18 [get_ports {tubePosSignal[1]}]
set_property PACKAGE_PIN J17 [get_ports {tubePosSignal[0]}]
set_property PACKAGE_PIN L18 [get_ports {tubeShowSignal[6]}]
set_property PACKAGE_PIN T11 [get_ports {tubeShowSignal[5]}]
set_property PACKAGE_PIN P15 [get_ports {tubeShowSignal[4]}]
set_property PACKAGE_PIN K13 [get_ports {tubeShowSignal[3]}]
set_property PACKAGE_PIN K16 [get_ports {tubeShowSignal[2]}]
set_property PACKAGE_PIN R10 [get_ports {tubeShowSignal[1]}]
set_property PACKAGE_PIN T10 [get_ports {tubeShowSignal[0]}]
set_property PACKAGE_PIN V10 [get_ports clock_en]
set_property PACKAGE_PIN E3 [get_ports CP]
set_property PACKAGE_PIN U11 [get_ports CR]
set_property PACKAGE_PIN U12 [get_ports adjust_hour_en]
set_property IOSTANDARD LVCMOS18 [get_ports {tubePosSignal[7]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubePosSignal[6]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubePosSignal[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubePosSignal[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubePosSignal[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubePosSignal[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubePosSignal[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubePosSignal[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubeShowSignal[6]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubeShowSignal[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubeShowSignal[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubeShowSignal[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubeShowSignal[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubeShowSignal[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {tubeShowSignal[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports CP]
set_property IOSTANDARD LVCMOS18 [get_ports CR]
set_property IOSTANDARD LVCMOS18 [get_ports punctually_report_en]
set_property IOSTANDARD LVCMOS18 [get_ports adjust_hour_en]
set_property IOSTANDARD LVCMOS18 [get_ports adjust_minute_en]
set_property IOSTANDARD LVCMOS18 [get_ports clock_en]
set_property IOSTANDARD LVCMOS18 [get_ports show_mode]
set_property IOSTANDARD LVCMOS18 [get_ports punctuallyReportSignal]

set_property PACKAGE_PIN R16 [get_ports show_mode]
set_property PACKAGE_PIN U8 [get_ports punctually_report_en]
set_property PACKAGE_PIN T15 [get_ports punctuallyReportSignal]
set_property PACKAGE_PIN T8 [get_ports alarm_switch]
set_property PACKAGE_PIN N16 [get_ports alarmReportSignal]
set_property IOSTANDARD LVCMOS18 [get_ports alarm_switch]
set_property IOSTANDARD LVCMOS18 [get_ports alarmReportSignal]
```

```
set_property PACKAGE_PIN H6 [get_ports adjust_minute_en]
set_property PACKAGE_PIN T13 [get_ports second_continue]
set_property IOSTANDARD LVCMOS18 [get_ports second_continue]

set_property PACKAGE_PIN R13 [get_ports set_alarm_en]
set_property PACKAGE_PIN U18 [get_ports set_alarm_time_hour]
set_property PACKAGE_PIN T18 [get_ports set_alarm_time_minute]
set_property IOSTANDARD LVCMOS18 [get_ports set_alarm_time_minute]
set_property IOSTANDARD LVCMOS18 [get_ports set_alarm_time_hour]
set_property IOSTANDARD LVCMOS18 [get_ports set_alarm_en]
```

代码分析

整个数字钟的核心分为三部分：分频器、计数器、译码显示器。除此之外，为了实现整点报时功能、12/24h制切换和闹钟功能，还外设了整点报时模块，在译码显示部分加入额外的译码显示分支。

1. **分频器**

Nexys 4 DDR开发板E3管脚内置100MHz时钟信号，需要将100MHz信号分频为1kHz和1Hz信号，1kHz信号用于数码管译码显示，1Hz信号用于输入模60和模24计数器等

2. **计数器**

构造模10和模6计数器，并由其组成模60计数器，为时计数单独构造模24计数器

3. **译码显示器**

分为译码模块和显示模块，译码模块将数字译成7段共阳数码管可识别的7位二进制信号，显示模块在1kHz时钟信号的驱动下，轮流显示从译码模块传输过来的7位2进制信号，利用人眼的视觉暂留效应，让8位数码管看起来是在一起显示数字

4. **整点报时模块**

在1Hz时钟信号驱动下，当分信号为00时，驱动二极管以2秒一次的频率闪烁，闪烁次数即为当前时信号大小，24进制。

5. **12/24h进制转换**

不额外增加模12计数模块，而是识别时信号的数字大小：当时信号小于12时，显示正常时信号，当时信号大于等于12、且进制模式为12h时，在显示模块分别对时信号的两位进行处理。

6. **闹钟模块**

增设寄存器，用于储存设置的闹钟时间，当此时的时间（hh:mm）和设定的闹钟时间一样时，LED灯将以2s一次的频率闪烁1分钟；设置闹钟时间时，在译码显示模块添加更多分支，以达到设置闹钟时间的效果

7. **仿真代码**

为辅助查找bug，设计数字钟仿真模块，模块较为简单，但是在进行仿真时，有一个重要的问题需要注意：**基于计算机算力和显示受限，在进行仿真时，需修改分频频率：**

FrequencyDivider_1hz.v

```
//    for real use, here should be 499,but for test, we 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;
```

FrequencyDIvider_1k.v

```
//    for real use
    if (state < 49999) state <= state + 1'b1;
//    for sim
//    if (state < 4) state <= state +1'b1;
```

否则将很难进行仿真

8. **教师要求现场修改代码**

验收时要求现场改代码，实现新的功能：**当整点报时时，hh:mm以2Hz的频率闪烁，秒照常，最后两位显示报时次数**

思路：增加2Hz分频器，在译码显示模块再添分支，当在进行整点报时时，以2Hz信号为时和分显示的使能信号，实现闪烁，最后两位直接改接显示和时一样的数字即可

实验小结

经过一个多月艰苦卓绝的奋斗，我最终在最后一周完成了数字钟的验收，切实体会到了自己编写一个略有规模的系统的难度，由于本人是Java开发者，所以verilog代码中不可避免的带上了一些Java的代码风格（如变量命名，缩进等），有点惭愧哈哈哈。

除此之外，无论是MIPS_CPU的实验，还是本次数字钟实验，我还真正切实感受到了“**工欲善其事，必先利其器**”这句话的真正意义：前期写verilog代码时，用的是vivado自带的Text Editor，没有代码自动缩进和自动补全，语法高亮单一，这对于我一个用习惯了JetBrains编译器的开发者无

疑是一场灾难。于是我下载了一个**Visual Studio Code**，配置好verilog环境后，代码编写速度直接翻倍，现场改功能也应付的得心应手，感谢VS Code！

一次难忘，但收获巨大的实验！

2023年4月1日 罗畅 于 华中工学院