**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**

[一、 实验名称 1](#_Toc100238706)

[二、 实验目的 1](#_Toc100238707)

[三、 实验任务 1](#_Toc100238708)

[1. 功能要求 1](#_Toc100238709)

[2. 验收内容 1](#_Toc100238710)

[四、 实验原理 1](#_Toc100238711)

[1. 状态图的描述方法 1](#_Toc100238712)

[2. 脉冲分配器设计步骤 1](#_Toc100238713)

[3. 自顶而下的设计方法 2](#_Toc100238714)

[五、 实验过程 2](#_Toc100238715)

[1. 步进电机代码 2](#_Toc100238716)

[2. 步进电机仿真输出结果 4](#_Toc100238717)

[3. 数字钟代码 6](#_Toc100238718)

[4. 数字钟的插板结果 **错误!未定义书签。**](#_Toc100238731)

[六、 实验小结 19](#_Toc100238732)

# 第七次实验：有限状态机及数字钟

# 实验名称

有限状态机及数字钟

# 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。

# 实验任务

步进电机脉冲分配器设计与多功能数字钟设计

## 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。

## 验收内容

代码及实验板实操结果

# 实验原理

## 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 SO=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。

## 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。

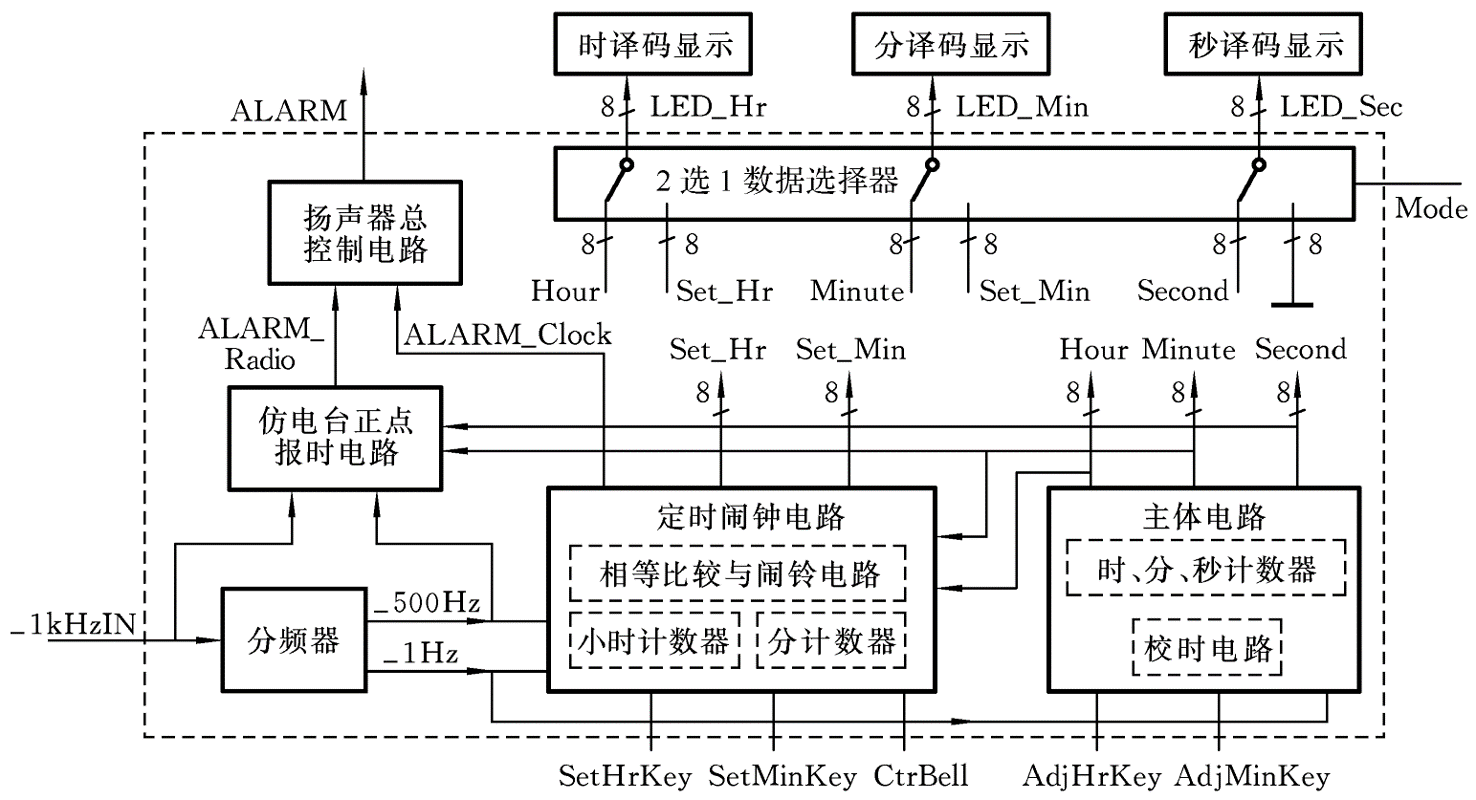
2.使用Verilog HDL设计电路，并进行仿真分析。

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。

## 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。



# 实验过程

## 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

* + - 1. 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

* + - 1. 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

## 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

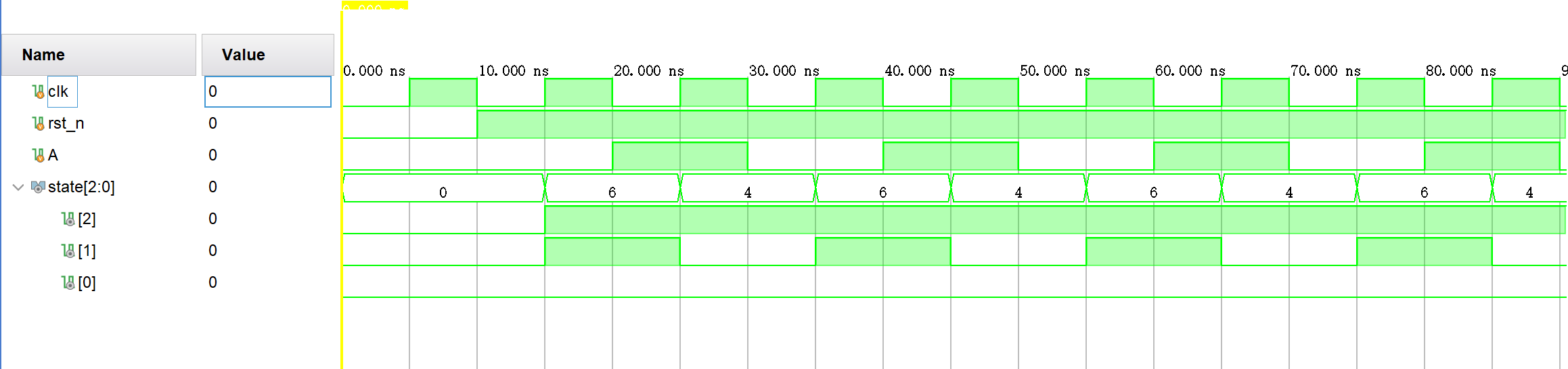
always @(state) begin

$display("State: %b", state);

end

endmodule

* + - 1. 仿真结果



## 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

### 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

### 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

# 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。