# 《数字逻辑与EDA》实验

# 课程设计

课程名称： 《数字逻辑与EDA》实验

设计题目： 数字钟

指导老师： 赵逸之

专 业： 计算机科学与技术

班 级： 1803

姓 名： 邢广杰

2019年12月19日

目录

[1. 功能介绍 3](#_Toc27691971)

[1.1时钟功能 3](#_Toc27691972)

[1.2 闹钟设置功能 3](#_Toc27691973)

[1.3 整点报时功能 4](#_Toc27691974)

[1.4 秒表功能 4](#_Toc27691975)

[2. 设计思路 4](#_Toc27691976)

[2.1 整体设计思路 4](#_Toc27691977)

[2.2 功能切换设计思路 5](#_Toc27691978)

[2,3 设置时间切换模块 6](#_Toc27691979)

[2.4 时间自增模块 6](#_Toc27691980)

[2.5 整点报时和闹铃模块 7](#_Toc27691981)

[2.6 数码管显示模块 7](#_Toc27691982)

[3. 模块结构 8](#_Toc27691983)

[4．代码 8](#_Toc27691984)

[4.1clock.v 8](#_Toc27691985)

[4.2 original\_to\_1HZ.v 15](#_Toc27691986)

[4.3 original\_to\_2HZ.v 15](#_Toc27691987)

[4.4 original\_to\_100HZ.v 16](#_Toc27691988)

[4.5 original\_to\_1kHZ.v 16](#_Toc27691989)

[4.6 alarm\_signal.v 17](#_Toc27691990)

[4.7 clock\_signal.v 18](#_Toc27691991)

[4.8 show.v 19](#_Toc27691992)

[4.9 display.v 20](#_Toc27691993)

[4.10 selfadder.v 20](#_Toc27691994)

[5. 效果展示 21](#_Toc27691995)

[6．问题分析和总结 28](#_Toc27691996)

[附录：视频文件 29](#_Toc27691997)

# 功能介绍

## 1.1时钟功能

数字钟在时钟模式下，可以在开发板的6个七段数码管上以24小时制显示设置好的时间，格式为HH:MM:SS。时钟可以按照时间的进位规则进行进位，且每24个小时完成一次循环。

在时钟模式下可以通过开关切换到设置模式，设置模式下可以通过按钮切换更改时、分或秒，且更改哪个时间部分那个部分显示的七段数码管就会进行闪烁进行提示。通过另一个按钮可以对时间进行更改，每按一下对应部分的数值加一，以达到设置时间的效果。

## 闹钟设置功能

数字钟可以通过开关切换到闹钟设置模式，在闹钟模式下，可以设置一个时分闹钟，和设置时间一样，闹钟设置也是通过按钮实现，同时也会有对应的提示，由于闹钟只用设置时分，此时显示秒的七数码管将不显示任何数码。在设置闹钟的过程中时钟仍在不断走，当时钟达到闹钟的设定值时，位于开发板右下侧的8个LED绿灯将会以一定的花型闪烁，提示已达到设定的闹钟时间。

## 整点报时功能

数字种在时钟走到整点，即XX:00:00时，位于开发板下方的8个红色LED灯将会以一定的花型闪烁，提示已经到达整点。

## 秒表功能

通过开关可以将数字钟切换到秒表模式，秒表模式将会用时钟模式显示秒的七段数码管显示毫秒，同时不设置正常时间进位，处于秒表模式下，可以通过开关暂停秒表以记录时间。

# 设计思路

## 2.1 整体设计思路

数字钟利用开发板上的50MHZ时钟信号通过设计分屏器得到不同频率的时钟信号。

数字钟以一个可以设置不同进制的加法器来实现时间的前进，同时为了满足闹钟的基本功能，数字钟分别以12个4位reg型变量寄存当前跑到的时间和设置好的闹钟以及当前跑到的秒表时钟，同时不断捕捉信号，当当前时间和闹钟时间相等时或当前时间达到整点时即传给LED灯闪烁模块一个使能信号使闹钟产生作用。

数字钟以几个开关的打开闭合状态的组合信号定义数字钟当前的模式。

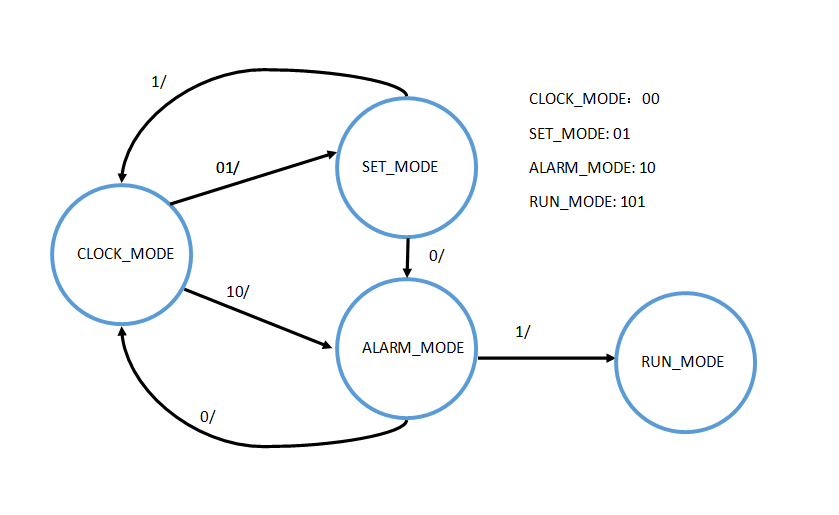
数字钟将六个4位数值传给七段数码管显示模块，显示模块根据当前设置的部分给不同的数码管不同的使能信号使得设置的部分能够进行闪烁。

数字钟的4-7译码器可以使得数码管显示对应的数码。

数字钟通过一个开关给秒表模式下的加法器一个使能信号，若该使能信号处于高电平则秒表时钟暂停不前进，以达到秒表停表的效果。

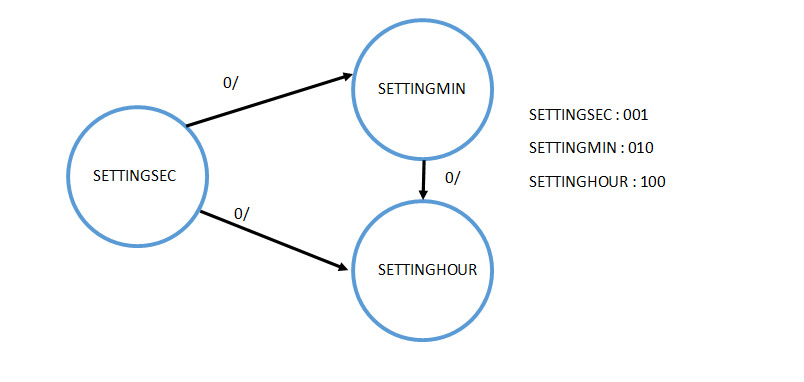
## 2.2 功能切换设计思路

数字钟的功能切换功能以有限状态机的设计实现方法设计，将数字钟的几种模式定义为几种状态，通过捕捉开关信号的变化进行状态转移，同时捕捉状态转移的信号以在七段数码管上显示不同模式下应显示的数值。该模块的状态转移图如图：



## 2,3 设置时间切换模块

通过捕捉一个按钮的下降沿信号来切换设置时间时的设置部分，其由有限状态机的设计思路进行设计，其状态转移图如下：



## 2.4 时间自增模块

时间自增通过一个可以设置进制（max值）的加法器实现，当处于该时间位的4位数值接收到上一位传来的进位信号，将会自增，若达到max值，将会对该位的数值清零，并产生一个进位信号，该时间位的下一位将会捕捉这个时间进位信号完成自增，并且通过24小时制时间的限制，对不同位的时间数值赋予不同的max值，以达到模拟正常时间的效果。最低位时间位（时钟下是秒，秒表下是毫秒）将会捕捉分频得到的时钟信号作为自己的自增信号。

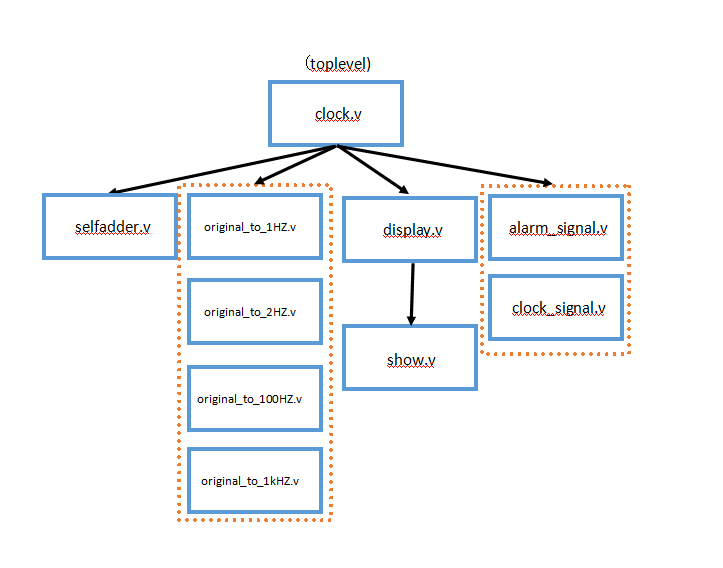
## 2.5 整点报时和闹铃模块

该模块利用有限状态自动机的设计方法设计了不同花型的LED闪烁效果分别作为整点报时和触发闹钟的信号。该模块会接收两个使能信号作为两个灯光闪烁是否启动的标志，在主模块中会不断捕捉当前时间判断是否触发闹铃或达到整点以改变此两个使能信号的值以达到触发闹钟或报时。

## 2.6 数码管显示模块

数码管显示模块将接受6个4位数值并分别传给4-7译码器进行译码，同时译码器留有使能信号的接口，显示模块会捕捉一个分频器得到的时钟信号，同时解码处于设置位置的信号，周期性地改变传入译码器的使能信号，若译码器得到使能信号和设置处于该位的信号则不显示译码结果，否则显示，已达到可以使得正在设置该时间位的数码管闪烁，以提示正在处于设置的时间位。

# 模块结构



# 4．代码

## 4.1clock.v

module clock(original\_clk, clr, modeswitch, func, posswitch, changetime, ledhour1, ledhour0, ledmin1, ledmin0, ledsec1, ledsec0, signal\_led, signal\_led2);  
(\*chip\_pin = "Y2"\*)input original\_clk;  
(\*chip\_pin = "Y23"\*)input clr;//秒表模式的暂停开关  
(\*chip\_pin = "AA23"\*)input func;  
(\*chip\_pin = "Y24, AA22"\*)input [1:0] modeswitch;//模式切换开关  
(\*chip\_pin = "N21"\*)input posswitch;//位置切换按钮  
(\*chip\_pin = "M21"\*)input changetime;//时间调节按钮  
(\*chip\_pin = "AA14, AG18, AF17, AH17, AG17, AE17, AD17"\*)output [6:0] ledhour1;  
(\*chip\_pin = "AC17, AA15, AB15, AB17, AA16, AB16, AA17"\*)output [6:0] ledhour0;  
(\*chip\_pin = "AH18, AF18, AG19, AH19, AB18, AC18, AD18"\*)output [6:0] ledmin1;  
(\*chip\_pin = "AE18, AF19, AE19, AH21, AG21, AA19, AB19"\*)output [6:0] ledmin0;  
(\*chip\_pin = "W28, W27, Y26, W26, Y25, AA26, AA25"\*)output [6:0] ledsec1;  
(\*chip\_pin = "U24, U23, W25, W22, W21, Y22, M24"\*)output [6:0] ledsec0;   
(\*chip\_pin = "G21, G22, G20, H21, E24, E25, E22, E21"\*)output [7:0] signal\_led;//闹钟  
(\*chip\_pin = "H17, J16, H16, J15, G17, J17, H19, J19"\*)output [7:0] signal\_led2;//整点报时  
wire clk\_1HZ;  
wire clk\_1kHZ;  
wire clk\_2HZ;  
wire clk\_100HZ;  
wire [3:0] clockhour1, clockhour0, clockmin1, clockmin0, clocksec1, clocksec0;//存储当前时间  
wire [3:0] runmin1, runmin0, runsec1, runsec0, runms1, runms0;//存储秒表时间  
wire[6:0] carryout;//当前时间进位信号  
wire[6:0] carryrun;//秒表时间进位信号  
wire[3:0] hour0\_system;  
  
parameter  
clock\_mode = 2'b00,  
set\_mode = 2'b01,  
alarm\_mode = 2'b10,  
  
noneset = 3'b000,  
secset = 3'b001,  
minset = 3'b010,  
hourset = 3'b100;  
  
  
reg[1:0] now\_mode;//当前模式  
reg[3:0] showhour1, showhour0, showmin1, showmin0, showsec1, showsec0;//4位输出信号  
reg [3:0] alarmhour1, alarmhour0, alarmmin1, alarmmin0;//存储设置闹钟时间  
reg[2:0] setmode;//存储设置状态  
reg[2:0] alarmmode;//存储闹钟设置状态  
reg [2:0] whichtoset;//存储设置位置  
reg [6:0] carryclk;  
reg alarmon;//闹钟触发使能信号  
reg clockon;//整点报时使能信号

//获取当前模式  
always@(modeswitch)  
begin  
now\_mode = modeswitch;  
end

//获取小时十位的最大值  
assign hour0\_system = (clockhour1 == 4'b0010) ? (4'b0011) : (4'b1001);  
//分屏器得到不同频率的信号  
original\_to\_1HZ get1HZ(original\_clk, clk\_1HZ);//get 1HZ clock signal  
original\_to\_2HZ get2HZ(original\_clk, clk\_2HZ);//get 1HZ clock signal  
original\_to\_100HZ get100HZ(original\_clk, clk\_100HZ);//get 100HZ clock signal  
original\_to\_1kHZ get1kHZ(original\_clk, clk\_1kHZ);//get 1kHZ clock signal  
  
//当前时间自增  
selfadder sec0(carryclk[0], 4'b1001, clocksec0, carryout[0], 0);  
selfadder sec1(carryclk[1], 4'b0101, clocksec1, carryout[1], 0);  
selfadder min0(carryclk[2], 4'b1001, clockmin0, carryout[2], 0);  
selfadder min1(carryclk[3], 4'b0101, clockmin1, carryout[3], 0);  
selfadder hour0(carryclk[4], hour0\_system, clockhour0, carryout[4], 0);  
selfadder hour1(carryclk[5], 4'b0010, clockhour1, carryout[5], 0);

//秒表时间自增  
selfadder run\_ms0(clk\_100HZ, 4'b1001, runms0, carryrun[0], clr);  
selfadder run\_ms1(carryrun[0], 4'b1001, runms1, carryrun[1], clr);  
selfadder run\_sec0(carryrun[1], 4'b1001, runsec0, carryrun[2], clr);  
selfadder run\_sec1(carryrun[2], 4'b0101, runsec1, carryrun[3], clr);  
selfadder run\_min0(carryrun[3], 4'b1001, runmin0, carryrun[4], clr);  
selfadder run\_min1(carryrun[4], 4'b1001, runmin1, carryrun[5], clr);

//调用显示模块  
show(clk\_2HZ,  
 showhour1, showhour0, showmin1, showmin0, showsec1, showsec0,   
 ledhour1, ledhour0, ledmin1, ledmin0, ledsec1, ledsec0,  
 whichtoset, func);  
alarm\_signal alarmring(clk\_1kHZ, alarmon, signal\_led);  
clock\_signal clockring(clk\_1kHZ, clockon, signal\_led2);  
  
  
//对当前设置状态进行解码  
always@(negedge posswitch)  
begin  
 case(setmode)  
 noneset : setmode <= secset;  
 secset : setmode <= minset;  
 minset : setmode <= hourset;  
 hourset : setmode <= secset;  
 default : setmode <= noneset;  
 endcase  
end  
  
always@(negedge posswitch)  
begin  
 case(alarmmode)  
 noneset : alarmmode <= minset;  
 minset : alarmmode <= hourset;  
 hourset : alarmmode <= minset;  
 default : alarmmode <= noneset;  
 endcase  
end  
  
always  
begin  
 if(modeswitch == set\_mode)  
 whichtoset = setmode;  
 else if(modeswitch == alarm\_mode)  
 whichtoset = alarmmode;  
 else  
 whichtoset = noneset;  
end

//触发设置模式  
always  
begin  
 if(modeswitch == set\_mode)  
 begin  
 if(whichtoset == noneset)  
 begin  
 carryclk[0] = clk\_1HZ;  
 carryclk[1] = carryout[0];  
 carryclk[2] = carryout[1];  
 carryclk[3] = carryout[2];  
 carryclk[4] = carryout[3];  
 carryclk[5] = carryout[4];  
 end  
 else  
 begin  
 case(whichtoset)  
 secset:  
 begin  
 carryclk[0] = (~changetime);  
 carryclk[1] = carryout[0];  
 carryclk[2] = carryout[1];  
 carryclk[3] = carryout[2];  
 carryclk[4] = carryout[3];  
 carryclk[5] = carryout[4];  
 end  
 minset:  
 begin  
 carryclk[0] = 0;  
 carryclk[1] = carryout[0];  
 carryclk[2] = (carryout[1] | (~changetime));  
 carryclk[3] = carryout[2];  
 carryclk[4] = carryout[3];  
 carryclk[5] = carryout[4];  
 end  
 hourset:  
 begin  
 carryclk[0] = 0;  
 carryclk[1] = carryout[0];  
 carryclk[2] = carryout[1];  
 carryclk[3] = carryout[2];  
 carryclk[4] = (carryout[3] | (~changetime));  
 carryclk[5] = carryout[4];  
 end  
 endcase  
 end  
 end  
 else  
 begin  
 carryclk[0] = clk\_1HZ;  
 carryclk[1] = carryout[0];  
 carryclk[2] = carryout[1];  
 carryclk[3] = carryout[2];  
 carryclk[4] = carryout[3];  
 carryclk[5] = carryout[4];  
 end  
end  
  
  
//触发闹钟模式对闹钟进行调节  
always@(negedge changetime)  
begin  
 if(modeswitch == alarm\_mode)  
 begin  
 if(whichtoset == minset)  
 begin  
 if({alarmmin1, alarmmin0} == 8'b01011001)  
 begin  
 alarmmin1 <= 4'b0000;  
 alarmmin0 <= 4'b0000;  
 end  
 else if(alarmmin0 == 4'b1001)  
 begin  
 alarmmin0 <= 4'b0000;  
 alarmmin1 <= (alarmmin1 + 1'b1);  
 end  
 else  
 alarmmin0 <= (alarmmin0 + 1'b1);  
 end  
 else if(whichtoset == hourset)  
 begin  
 if({alarmhour1, alarmhour0} == 8'b00100011)  
 begin  
 alarmhour1 <= 4'b0000;  
 alarmhour0 <= 4'b0000;  
 end  
 else if(alarmhour0 == 4'b1001)  
 begin  
 alarmhour1 <= (alarmhour1 + 1'b1);  
 alarmhour0 <= 4'b0000;  
 end  
 else  
 alarmhour0 <= (alarmhour0 + 1'b1);  
 end  
 else  
 alarmmin1 <= (alarmmin1 + 1'b1);  
 end  
end

//根据不同模式使显示模块显示不同的值  
always  
begin  
 if(now\_mode == alarm\_mode)  
 begin  
 if(func)  
 begin  
 showhour1 <= runmin1;  
 showhour0 <= runmin0;  
 showmin1 <= runsec1;  
 showmin0 <= runsec0;  
 showsec1 <= runms1;  
 showsec0 <= runms0;  
 end  
 else  
 begin  
 showhour1 <= alarmhour1;  
 showhour0 <= alarmhour0;  
 showmin1 <= alarmmin1;  
 showmin0 <= alarmmin0;  
 showsec1 <= 4'b1111;  
 showsec0 <= 4'b1111;  
 end  
 end  
 else  
 begin  
 showhour1 <= clockhour1;  
 showhour0 <= clockhour0;  
 showmin1 <= clockmin1;  
 showmin0 <= clockmin0;  
 showsec1 <= clocksec1;  
 showsec0 <= clocksec0;  
 end  
end

//捕捉闹钟或整点报时的触发  
always  
begin  
 if(({clockhour1, clockhour0, clockmin1, clockmin0} == {alarmhour1, alarmhour0, alarmmin1, alarmmin0}) && (clocksec1 < 4'b0011) && ({clockhour1, clockhour0, clockmin1, clockmin0, clocksec1, clocksec0} != 18'b000000000000000000))  
 alarmon = 1;  
 else  
 alarmon = 0;  
end  
  
always  
begin  
 if(({clockmin1, clockmin0} == 6'b000000) && (clocksec1 < 'd1))  
 clockon = 1;  
 else  
 clockon = 0;  
end  
endmodule

## 4.2 original\_to\_1HZ.v

module original\_to\_1HZ(original\_clk, clk1HZ);  
input original\_clk;  
output reg clk1HZ;  
reg[31:0] cnt\_1HZ;  
always@(posedge original\_clk)  
begin  
 if(cnt\_1HZ == 'd25000000-1)  
 begin  
 clk1HZ <= ~clk1HZ;  
 cnt\_1HZ <= 0;  
 end  
 else  
 cnt\_1HZ <= cnt\_1HZ + 1'b1;  
end  
endmodule

## 4.3 original\_to\_2HZ.v

module original\_to\_2HZ(original\_clk, clk2HZ);  
input original\_clk;  
output reg clk2HZ;  
reg[31:0] cnt\_2HZ;  
always@(posedge original\_clk)  
begin  
 if(cnt\_2HZ == 'd12500000-1)  
 begin  
 clk2HZ <= ~clk2HZ;  
 cnt\_2HZ <= 0;  
 end  
 else  
 cnt\_2HZ <= cnt\_2HZ + 1'b1;  
end  
endmodule

## 4.4 original\_to\_100HZ.v

module original\_to\_100HZ(original\_clk, clk100HZ);  
input original\_clk;  
output reg clk100HZ;  
reg[31:0] cnt\_100HZ;  
always@(posedge original\_clk)  
begin  
 if(cnt\_100HZ == 'd250000-1)  
 begin  
 clk100HZ <= ~clk100HZ;  
 cnt\_100HZ <= 0;  
 end  
 else  
 cnt\_100HZ <= cnt\_100HZ + 1'b1;  
end  
endmodule

## 4.5 original\_to\_1kHZ.v

module original\_to\_1kHZ(original\_clk, clk1kHZ);  
input original\_clk;  
output reg clk1kHZ;  
reg[25:0] cnt\_1kHZ;  
always@(posedge original\_clk)  
begin  
 if(cnt\_1kHZ == 'd25000-1)  
 begin  
 cnt\_1kHZ <= 0;  
 clk1kHZ <= ~clk1kHZ;  
 end  
 else  
 cnt\_1kHZ <= cnt\_1kHZ + 1'b1;  
end  
endmodule

## 4.6 alarm\_signal.v

module alarm\_signal(clk1khz, on, cout);  
input clk1khz, on;  
output reg [7:0] cout;  
  
parameter starlength = 200;//停留200ms  
parameter spacelength = 100;//两次停留之间空100ms  
parameter longlength = 400;//空400ms  
  
reg[31:0] cnt\_length;  
  
always@(posedge clk1khz)  
begin  
 if(cnt\_length == (starlength\*5+spacelength\*4+longlength-1))  
 cnt\_length <= 0;  
 else  
 cnt\_length <= cnt\_length + 1'b1;  
end  
  
always@(cnt\_length)  
begin  
 if(on)  
 begin  
 if((cnt\_length >= 0) && (cnt\_length < starlength))  
 cout <= 8'b11111111;  
 else if((cnt\_length >= starlength) && (cnt\_length < (starlength + spacelength)))  
 cout <= 8'b00000000;  
 else if((cnt\_length >= (starlength+spacelength)) && (cnt\_length < (starlength\*2+spacelength)))  
 cout <= 8'b11111111;  
 else if((cnt\_length >= (starlength\*2+spacelength)) && (cnt\_length < (starlength\*2+spacelength\*2)))  
 cout <= 8'b00000000;  
 else if((cnt\_length >= (starlength\*2+spacelength\*2)) && (cnt\_length < (starlength\*3+spacelength\*2)))  
 cout <= 8'b11111111;  
 else if((cnt\_length >= (starlength\*3+spacelength\*2)) && (cnt\_length < (starlength\*3+spacelength\*3)))  
 cout <= 8'b00000000;  
 else if((cnt\_length >= (starlength\*3+spacelength\*3)) && (cnt\_length < (starlength\*4+spacelength\*3)))  
 cout <= 8'b11111111;  
 else if((cnt\_length >= (starlength\*4+spacelength\*3)) && (cnt\_length < (starlength\*4+spacelength\*4)))  
 cout <= 8'b00000000;  
 else if((cnt\_length >= (starlength\*4+spacelength\*4)) && (cnt\_length < (starlength\*5+spacelength\*4)))  
 cout <= 8'b11111111;  
 else  
 cout <= 8'b00000000;  
 end  
 else  
 cout <= 8'b00000000;  
end  
endmodule

## 4.7 clock\_signal.v

module clock\_signal(clk1khz, on, cout);  
input clk1khz, on;  
output reg [7:0] cout;  
  
parameter starlength = 1000;//停留1000ms  
parameter spacelength = 800;//空白800ms  
  
reg[31:0] cnt\_length;  
  
always@(posedge clk1khz)  
begin  
 if(cnt\_length == (starlength+spacelength-1))  
 cnt\_length <= 0;  
 else  
 cnt\_length <= cnt\_length + 1'b1;  
end  
  
always@(cnt\_length)  
begin  
 if(on)  
 begin  
 if((cnt\_length >= 0) && (cnt\_length < starlength))  
 cout <= 8'b11111111;  
 else if((cnt\_length >= starlength) && (cnt\_length < (starlength + spacelength)))  
 cout <= 8'b00000000;  
 else if((cnt\_length >= (starlength+spacelength)) && (cnt\_length < (starlength\*2+spacelength)))  
 cout <= 8'b11111111;  
 end  
 else  
 cout <= 8'b00000000;  
end  
endmodule

## 4.8 show.v

module show(clk,  
 showhour1, showhour0, showmin1, showmin0, showsec1, showsec0,   
 ledhour1, ledhour0, ledmin1, ledmin0, ledsec1, ledsec0,  
 whichtoset, flashenable);  
input clk, flashenable;//flashenable为若处于秒表状态则不闪烁  
input [3:0] showhour1, showhour0, showmin1, showmin0, showsec1, showsec0;  
input [2:0] whichtoset;  
output wire [6:0] ledhour1, ledhour0, ledmin1, ledmin0, ledsec1, ledsec0;  
  
reg flag;  
  
wire settinghour, settingmin, settingsec;

//获取各个位是否处于设置状态  
assign {settinghour, settingmin, settingsec} = whichtoset;  
  
display(showhour1, flag, settinghour, ledhour1);  
display(showhour0, flag, settinghour, ledhour0);  
display(showmin1, flag, settingmin, ledmin1);  
display(showmin0, flag, settingmin, ledmin0);  
display(showsec1, flag, settingsec, ledsec1);  
display(showsec0, flag, settingsec, ledsec0);  
  
always@(posedge clk)  
begin  
 if(flashenable)  
 flag = 0;  
 else  
 flag = ~flag;  
end  
endmodule

## 4.9 display.v

module display(in, on, flash, out);  
//on 和flag 都为1 时不显示，实现闪烁的效果  
input on, flash;  
input [3:0] in;  
output reg [6:0] out;  
  
always  
begin  
 if({on, flash} == 2'b11)  
 out<= 7'b1111111;  
 else  
 begin  
 case(in)  
 0: out <= 7'b1000000;  
 1: out <= 7'b1111001;  
 2: out <= 7'b0100100;  
 3: out <= 7'b0110000;  
 4: out <= 7'b0011001;  
 5: out <= 7'b0010010;  
 6: out <= 7'b0000010;  
 7: out <= 7'b1111000;  
 8: out <= 7'b0000000;  
 9: out <= 7'b0010000;  
 15: out <= 7'b1111111;  
 endcase  
 end  
end  
endmodule

## 4.10 selfadder.v

module selfadder(clk, system, clocktime, cout, reset);  
//reset为秒表模式下的暂停信号

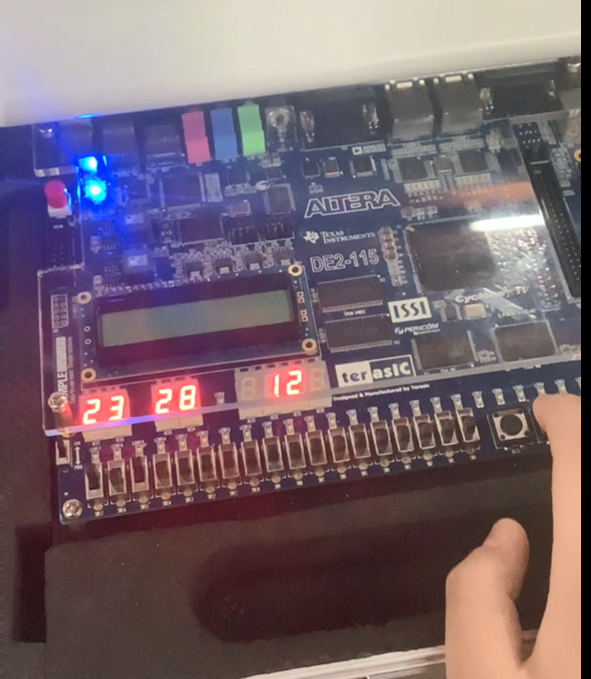
//system为进制值

input clk, reset;  
input [3:0] system;  
output reg cout;  
output reg [3:0] clocktime;  
  
always@(posedge clk)  
begin  
 if(reset)  
 begin  
 clocktime <= 0;  
 cout <= 0;  
 end  
 else  
 begin  
 if((clocktime == system) || (clocktime > system))  
 begin  
 clocktime <= 0;  
 cout <= 1;  
 end  
 else  
 begin  
 clocktime <= clocktime + 1;  
 cout <= 0;  
 end  
 end  
end  
endmodule

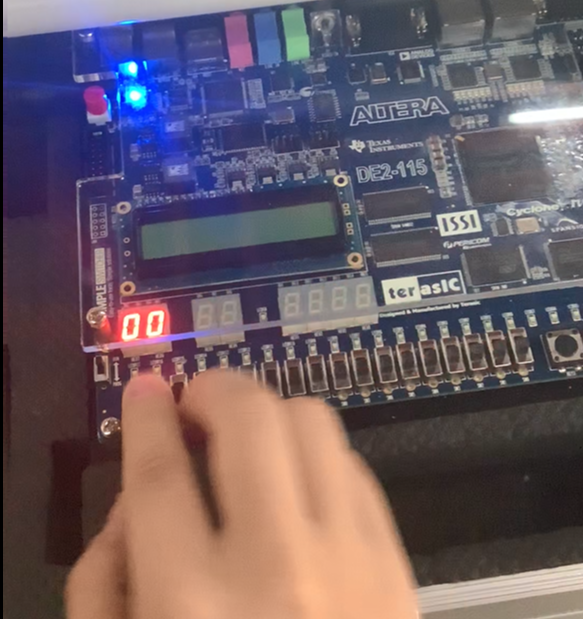
# 效果展示

（具体演示在视频附件中，该视频文件部分时间段经过加速处理）

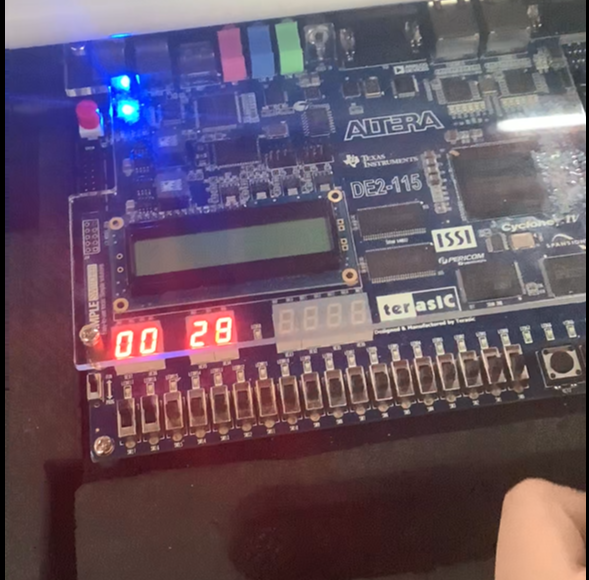
时钟调节：



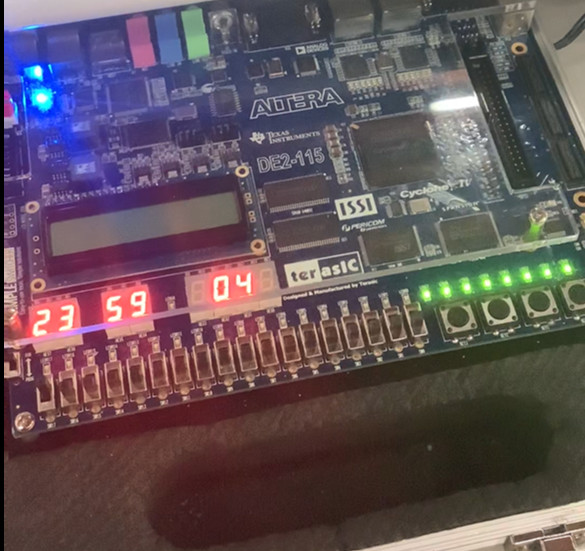
切换模式（利用开关切换数字钟所处的模式）：



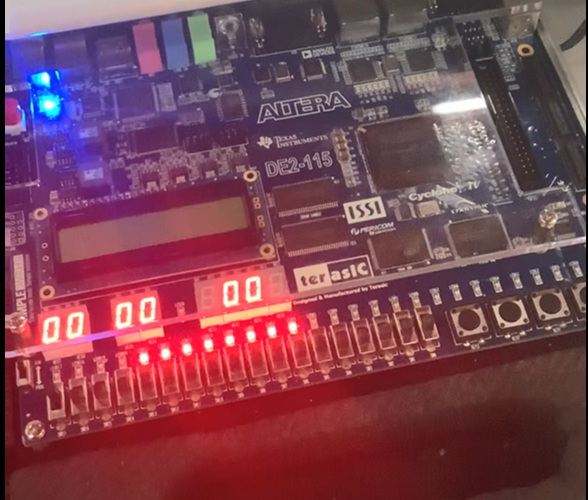
闹钟调节（调节所处的位置会闪烁提示）：



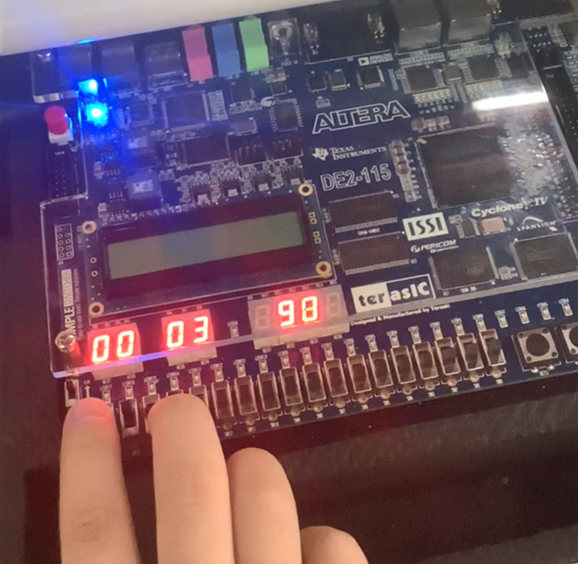
闹钟触发：8路绿色LED灯以特定花型闪烁：



整点报时：8路红色LED灯以特定花型闪烁



秒表模式：开关控制启动与暂停：



# 6．问题分析和总结

本次课程设计旨在提高我们设计和实现电路的能力，需要我们综合运用各种电路设计的方法，理解并运用组合电路、时序电路和有限状态机的设计方法。理解top-down的设计模式，熟悉FPGA开发板开发的流程和设计思想，帮助我们更好地掌握数字电路和EDA技术。

在设计和实现的过程中，遇到了许多问题，比如说基础语法掌握不牢，电路设计的变量意义没有完全理解等等，每每遇到这些问题，设计就很难进行下去，通过查阅各种书籍和网络资源，每一个问题都被有效地解决掉了。但是在整个设计的流程中，还留存了许多疑惑，也是因为并没有很好地理解电路的设计方法和思想而造成的。

做出来的数字钟虽然实现了比较基础的功能，但仍然存在许多可以改进的地方，包括一些潜在的BUG没有被发现，希望今后掌握更加精湛的设计技巧和更加牢固的数字电路知识之后可以做到遇到问题迎刃而解。

总的来说，通过此次课程设计，使我们更加扎实地掌握了有关数字电路和EDA方面的知识，实践出真知，通过亲自动手操作，是我掌握的知识不再是纸上谈兵。在课程设计过程中，我不断发现错误，不断改正，不断领悟，不断获取。最终的检测调试环节，本身就是在践行“过而能改，善莫大焉”的知行观。在今后社会的发展和学习实践过程中，一定要不懈努力，不能遇到问题就想到要退缩，一定要不厌其烦的发现问题所在，然后一一进行解决，只有这样，才能成功的做成想做的事，才能在今后的道路上劈荆斩棘，

附录：视频文件

