目录

[一、方案与设计 3](#_Toc154822283)

[二、实现的功能 3](#_Toc154822284)

[1、显示时间 3](#_Toc154822285)

[2、整点报时 6](#_Toc154822286)

[3、时制切换 6](#_Toc154822287)

[4、设置时间 8](#_Toc154822288)

[5、设置闹钟 9](#_Toc154822289)

[6、整点及闹钟响铃 10](#_Toc154822290)

[7、串口通讯 11](#_Toc154822291)

[8、完整的代码 13](#_Toc154822292)

[top.v 13](#_Toc154822293)

[count.v 15](#_Toc154822294)

[uart\_loopback.v 21](#_Toc154822295)

[Debounce.v 23](#_Toc154822296)

[recoder.v 24](#_Toc154822297)

[三、实验结果 29](#_Toc154822298)

[四、遇到的问题 29](#_Toc154822299)

五、实验总结..............................................................................................................................30

# 一、方案与设计

实现具有报时、闹钟、时制切换以及串口停启闹钟功能的数字钟的方案如下：  
（1）构造一个数字钟，时间的小时位用5个LED灯进行二进制表示，分位和秒位用4个7段数码管表示，并将BTN0作为清零信号。（注：为了方便观察整点时数字种现象，将数字钟的初始化时间设为16：59：55）

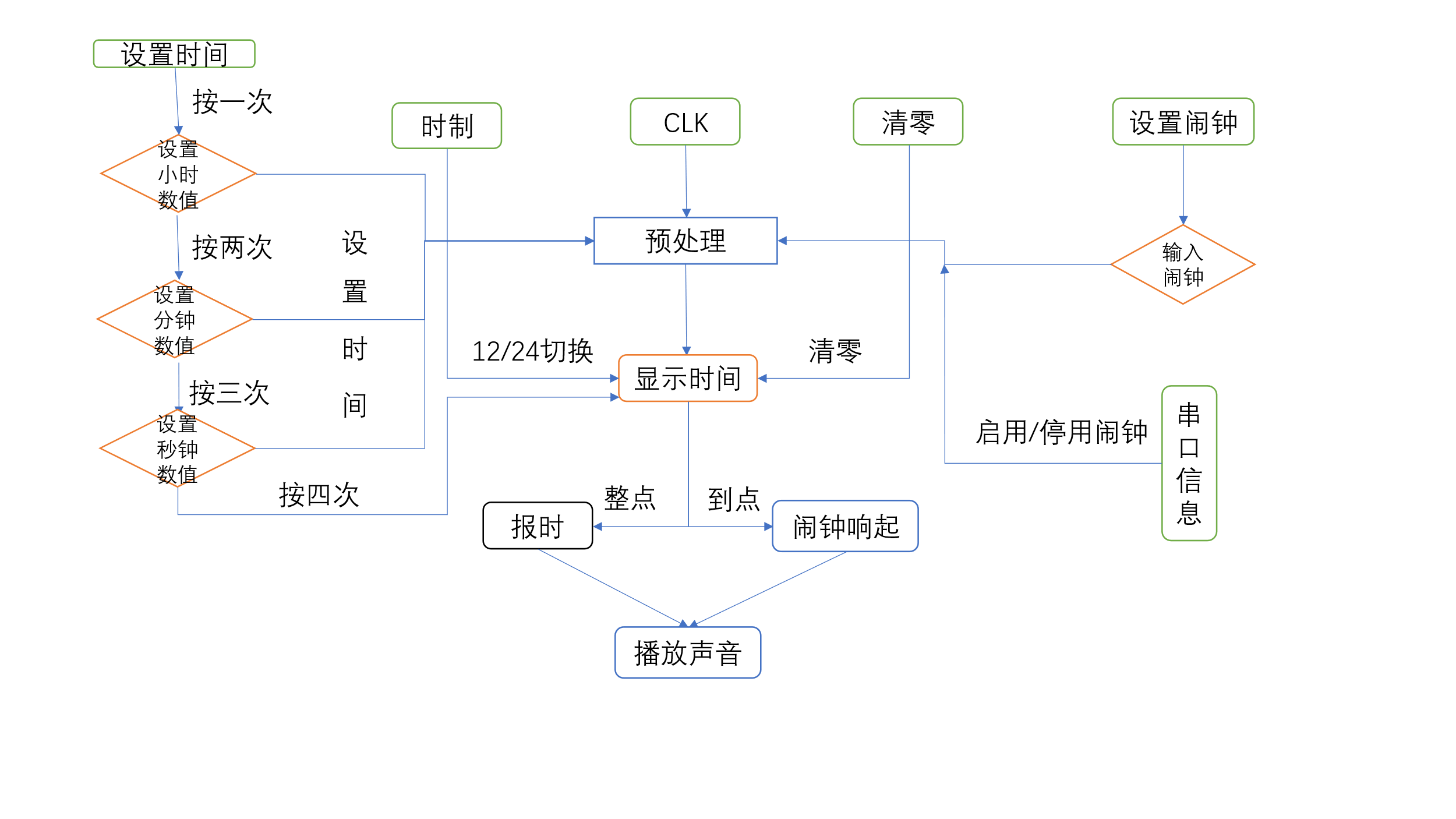
（2）设计整点报时功能，当数字钟到达整点时LED[6]亮3秒，并播放一段时长3秒的音乐。

（3）设计设置时间和设置闹钟的功能，并添加闹铃功能，当时间到达闹钟时间，LED[7]亮3秒，并播放一段时长3秒的音乐。

（4）设计时制切换功能，按钮BTN1实现12/24小时显示模式切换功能，使用LED[5]表示12小时显示模式中的上午、下午。

（5）设计闹钟停启功能，通过UART串口通信使用电脑键盘1键和2键控制。

流程图如下：



# 二、实现的功能

## 1、显示时间

能显示随时间变化的24小时制的“时、分、秒”。开机默认时间0时0分0秒。按BTN0可以将时间初始化为0时0分0秒。

实现代码：

always @(posedge clk)begin //用于产生1Hz的时钟

if(clk)

begin

cnt <= cnt + 1;

if(cnt >= 125000000)

begin

clk\_1Hz = 1;

cnt <= 0;

end

else begin

clk\_1Hz = 0;

end

end

end

always @(posedge clk or negedge clr) //时间的计算

begin

if(clr) //清零

begin

second\_l <= 4'b0000; //在实际实验中，为了方便

second\_h <= 4'b0000; //观察现象，我们将其初始化

minute\_l <= 4'b0000; //设置为16：59：55

minute\_h <= 4'b0000; //故实验视频中可能会不是00：00

hour <= 5'b00000;

end

else

begin

if(clk\_1Hz)

begin

second\_l = second\_l + 1; //计时

if(bell) //计算整点亮灯时长

begin

bell\_cnt <= bell\_cnt + 1;

if(bell\_cnt >= 3)

begin

bell\_cnt <= 0;

bell = 0;

end

end

end

if (mode1==0)begin //计算时间

if(second\_l>= 10)

begin

second\_l = second\_l - 10;

second\_h = second\_h + 1;

end

if(second\_h>=6)

begin

second\_h = second\_h - 6;

minute\_l = minute\_l + 1;

end

if(minute\_l>=10)

begin

minute\_l = minute\_l - 10;

minute\_h = minute\_h + 1;

end

if(minute\_h>=6)

begin

minute\_h = minute\_h -6;

hour = hour + 1;

bell = 1; //整点了就使bell=1

end

tmpHour = hour; //用于存放hour的临时变量

if(mode)begin //注释见下文

if(tmpHour >= 12) begin

if (hour>12)begin

noon = 1;

end else begin

noon = 0;

end

tmpHour = tmpHour - 12;

end

end

else begin

noon = 0;

if(hour >=24) begin

hour = hour - 24;

end

end

Led[4:0] = tmpHour; //以下为输出显示模块

Led[5] = noon;

Led[6] = bell;

case(cnt[14:13])

0: NUM = second\_l;

1: NUM = second\_h;

2: NUM = minute\_l;

3: NUM = minute\_h;

endcase

end

module number\_output( //七段数码管显示数字

input [3:0] NUM,

output reg[6:0] a\_to\_g

);

always @(\*)

case(NUM)

0: a\_to\_g = 7'b1000000;

1: a\_to\_g = 7'b1111001;

2: a\_to\_g = 7'b0100100;

3: a\_to\_g = 7'b0110000;

4: a\_to\_g = 7'b0011001;

5: a\_to\_g = 7'b0010010;

6: a\_to\_g = 7'b0000010;

7: a\_to\_g = 7'b1111000;

8: a\_to\_g = 7'b0000000;

9: a\_to\_g = 7'b0010000;

default a\_to\_g = 7'b1111111;

endcase

endmodule

## 2、整点报时

每到整点LED[6]会亮3s，作为整点报时

实现代码：

（在always中）

if(clk\_1Hz)

begin

second\_l = second\_l + 1;

if(bell) //当到整点

begin

bell\_cnt <= bell\_cnt + 1; //整点计时器计时

if(bell\_cnt >= 3) //计到3s

begin

bell\_cnt <= 0;

bell = 0; //灭灯

end

end

end

Led[6]=bell; //输出

## 3、时制切换

使用按钮BTN1实现12/24小时显示模式切换功能，使用LED[5]表示12小时显示模式中的上午、下午。

实现代码：

module Mode( //记录当前时制

input Flag, //btn1输入

input Clk, //时钟输入

output reg mode //输出模式

);

wire pulse;

always @(posedge pulse)

begin

if (Flag) begin //按下btn1

if(mode) //切换时制

mode <= 0;//0为24，1为12

else mode <= 1;

end

end

Debounce myDebounce(.Clk(Clk),

.pulse(pulse));

endmodule

module Debounce( //锁存器

input Clk,

output pulse

);

reg [24:0] d;//取时长

always @(posedge Clk)

begin

d <= d + 1;

end

assign pulse = d[24];

endmodule

//（在count.v中对应）

tmpHour = hour;

if(mode)begin //12h制时

if(tmpHour >= 12) begin //每到12h就归零

if (hour>12)begin //这是真正的hour数值

noon = 1; //当hour>12时,说明是下午

end else begin //反之为上午

noon = 0;

end

tmpHour = tmpHour - 12; //到12h时临时变量将归零

end //但hour的真实值不变，仍24h制

end

else begin //24h制时

noon = 0; //不用区分上下午

if(hour >=24) begin //hour满24就归零

hour = hour - 24;

end

end

## 4、设置时间

使用按钮BTN2和六个开关SW[5:0]设置当前时间。使用按钮BTN2作为时间设置按钮，按一次BTN2进入“设置时间的小时数值”模式，再按一次BTN2进入“设置时间的分钟数值”模式，按第三次BTN2进入“设置时间的秒钟数值”模式，再按一次BTN2回到“显示当前时间”状态。通过开关SW[5:0]的拨上拨下以二进制的形式输入时间的小时、分钟、秒钟，开关拨上表示1，拨下表示0，小时数值使用24小时模式输入。

实现代码：

always @(posedge clk\_1Hz)begin //按下btn2切换当前模式

if (btn2) begin

mode1 = mode1 + 1; //每按一次+1

end

end

if (mode1==0 && mode2==0)begin //mode2用于区分是设置时间还是闹钟

//这部分代码如第一题所示，故隐去 //当mode2=0时为时间的显示和设置

End

else if (mode1==1 && mode2==0) begin //设置小时

Led[5:0]=sw[5:0]; //只显示小时

Led[7:6]=2'b00;

hour=sw; //将设置的时间储存在hour中

NUM=4'b1111; //num为此时数码管不亮

end//mode1==1

else if (mode1==2 && mode2==0)begin //设置分钟

Led=8'b00000000; //led灯不亮

tmpSW = sw; //临时变量

minute\_h<=tmpSW/10; //分钟高位取整十部分

minute\_l<=tmpSW%10; //分钟低位取个位

case(cnt[14:13])

0: NUM = 4'b1111; //0和1时是秒钟的管子亮

1: NUM = 4'b1111; //故设置为1111，使其长灭

2: NUM = minute\_l;

3: NUM = minute\_h;

endcase

end//mode1=2

else if (mode1==3 && mode2==0)begin //设置秒钟，类似mode1=2

Led=8'b00000000;

tmpSW = sw;

second\_h<=tmpSW/10;

second\_l<=tmpSW%10;

case(cnt[14:13])

0: NUM = second\_l;

1: NUM = second\_h;

2: NUM = 4'b1111;

3: NUM = 4'b1111;

endcase

end//mode1==3

## 5、设置闹钟

使用按钮BTN3和六个开关SW[5:0]设置闹铃，使用LED[7]亮3秒做闹铃指示。使用按钮BTN3作为闹铃设置按钮，按一次BTN3进入“设置闹铃的小时数值”模式，再按一次BTN3进入“设置闹铃的分钟数值”模式，按第三次BTN3进入“设置闹铃的秒钟数值”模式，再按一次BTN3回到“显示当前时间”状态。通过开关SW[5:0]的拨上拨下以二进制的形式输入闹铃的小时、分钟、秒钟，开关拨上表示1，拨下表示0，小时数值使用24小时模式输入。

实现代码:

实现思路与实验四完全一致

reg [3:0] bell\_second\_l; //秒数低位闹钟

reg [3:0] bell\_second\_h; //秒数高位闹钟

reg [3:0] bell\_minute\_l; //分钟低位闹钟

reg [3:0] bell\_minute\_h; //分钟高位闹钟

reg [4:0] bell\_hour; //小时闹钟

output reg [1:0] ring\_bell\_cnt, //闹钟到了的信号

output reg ring\_bell

reg [1:0] mode2; //设置闹钟的状态

always @(posedge clk\_1Hz)begin //按btn3一次就+1

if (btn3) begin

mode2 = mode2 + 1;

end

end

//当mode2=0时代码如实验一、实验四所示，下面是其他情况

else if (mode2==1) begin //设置小时，内容和实验四一直

Led[5:0]=sw[5:0]; //只是将输入存贮到bell\_hour中

Led[7:6]=2'b00;

bell\_hour=sw;

NUM=4'b1111;

end//mode2==1

else if (mode2==2) begin //同理和实验四一致

Led=8'b00000000;

tmpSW = sw;

bell\_minute\_h<=tmpSW/10;

bell\_minute\_l<=tmpSW%10;

case(cnt[14:13])

0: NUM = 4'b1111;

1: NUM = 4'b1111;

2: NUM = bell\_minute\_l;

3: NUM = bell\_minute\_h;

endcase

end//mode2==2

else if (mode2==3) begin //同理

Led=8'b00000000;

tmpSW = sw;

bell\_second\_h<=tmpSW/10;

bell\_second\_l<=tmpSW%10;

case(cnt[14:13])

0: NUM = bell\_second\_l;

1: NUM = bell\_second\_h;

2: NUM = 4'b1111;

3: NUM = 4'b1111;

endcase

end//mode2==3

## 6、整点及闹钟响铃

整点到时以及闹铃到时，用SSM2603音频编解码器播放一段声音

实现代码:

//其实是调用了老师提供的文件，故只在top文件中有改动

Module top(change,clk,clr,btn2,sw,btn3,an,Led,

a\_to\_g,uart\_rxd,uart\_txd,AC\_BCLK, AC\_MCLK,AC\_MUTEN,AC\_PBDAT,

AC\_PBLRC,AC\_RECDAT,AC\_RECLRC,

AC\_SCL,AC\_SDA

);

inout AC\_BCLK; //一些io变量

output AC\_MCLK;

output AC\_MUTEN;

output AC\_PBDAT;

inout AC\_PBLRC;

input AC\_RECDAT;

inout AC\_RECLRC;

output AC\_SCL;

inout AC\_SDA;

wire sound; //整点和闹钟到了的信号

assign sound = (bell || ring\_bell); //当整点或闹钟到时为1

recorder recoder( //实例化一个recoder

.clk(clk),

.AC\_BCLK(AC\_BCLK),

.AC\_MCLK(AC\_MCLK),

.AC\_MUTEN(AC\_MUTEN),

.AC\_PBDAT(AC\_PBDAT),

.AC\_PBLRC(AC\_PBLRC),

.AC\_RECDAT(AC\_RECDAT),

.AC\_RECLRC(AC\_RECLRC),

.AC\_SCL(AC\_SCL),

.AC\_SDA(AC\_SDA),

.LEDREC(),

.LEDTONE(),

.LEDPLAY(),

.LEDFEEDBACK(),

.reset(clr),

.voldown(),

.volup(),

.rec\_aud(),

.play\_tone(sound), //将信号传入

.play\_aud(1), //一直置1

.feedback(),

.t\_node()

);

//具体的recoder改动几乎没有，就不放出来了

## 7、串口通讯

通过UART串口通信，使用电脑键盘1键和2键控制停用和启用设定好的闹铃。

实现代码:

module UART\_crl(

input sys\_clk , //外部125MHz时钟

input reset, //系外部复位信号，高有效

//UART端口

input uart\_rxd , //UART接收端口

output uart\_txd, //UART发送端口

output reg en //闹铃的启用信号

);

//parameter define

parameter CLK\_FREQ = 125000000; //定义系统时钟频率

parameter UART\_BPS = 115200 ; //定义串口波特率

//wire define

wire uart\_rx\_done; //UART接收完成信号

wire [7:0] uart\_rx\_data; //UART接收数据，为键盘输入的控制，通过对这个数据//的判断为使能输出赋值

always @(posedge sys\_clk)begin //根据键盘的输入对使能输出赋值

if(uart\_rx\_data==49)begin //输入1，en变为1

en=1;

end

else if(uart\_rx\_data==50)begin //按下2，en变为0

en=0;

end

end

//串口接收模块

uart\_rx #(

.CLK\_FREQ (CLK\_FREQ),

.UART\_BPS (UART\_BPS)

)

u\_uart\_rx(

.clk (sys\_clk ),

.reset (reset ),

.uart\_rxd (uart\_rxd ),

.uart\_rx\_done (uart\_rx\_done),

.uart\_rx\_data (uart\_rx\_data)

);

//串口发送模块

uart\_tx #(

.CLK\_FREQ (CLK\_FREQ),

.UART\_BPS (UART\_BPS)

)

u\_uart\_tx(

.clk (sys\_clk ),

.reset (reset ),

.uart\_tx\_en (uart\_rx\_done),

.uart\_tx\_data (uart\_rx\_data),

.uart\_txd (uart\_txd ),

.uart\_tx\_busy ( )

);

Endmodule

//在count.v中的对应

ring\_bell = ((second\_l==bell\_second\_l)&&(second\_h==bell\_second\_h)

&&(minute\_l==bell\_minute\_l)&&(minute\_h==bell\_minute\_h)

&&(bell\_hour==hour)&&(en==0)); //en在最后作为条件

//当en=0时，闹钟到了

//ring\_bell信号变为1

//当en=1时，

//ring\_bell信号永远为0

//以此来实现闹钟的启用和停用

## 8、完整的代码

### top.v

`timescale 1ns / 1ps

module top(change,clk,clr,btn2,sw,btn3,an,Led,a\_to\_g,uart\_rxd,uart\_txd,AC\_BCLK,

AC\_MCLK,AC\_MUTEN,AC\_PBDAT,AC\_PBLRC,AC\_RECDAT,AC\_RECLRC,AC\_SCL,AC\_SDA

);

input change;

input clk;

input clr;

input btn2;

input [5:0]sw;

input btn3;

output [3:0] an;

output [7:0] Led;

output [6:0] a\_to\_g;

input uart\_rxd;

output uart\_txd;

inout AC\_BCLK;

output AC\_MCLK;

output AC\_MUTEN;

output AC\_PBDAT;

inout AC\_PBLRC;

input AC\_RECDAT;

inout AC\_RECLRC;

output AC\_SCL;

inout AC\_SDA;

wire mode;

wire [1:0] mode1;

wire [1:0] bell\_cnt;

wire [1:0] ring\_bell\_cnt;

wire bell;

wire sound;

assign sound = (bell || ring\_bell);

Mode mymode(

.Flag(change),

.Clk(clk),

.mode(mode)

);

wire en;

count Count(

.en(en),

.btn2(btn2),

.sw(sw),

.mode(mode),

.clk(clk),

.clr(clr),

.btn3(btn3),

.Led(Led),

.an(an),

.a\_to\_g(a\_to\_g),

.bell(bell),

.ring\_bell(ring\_bell)

);

UART\_crl UART\_crl(

.sys\_clk(clk),

.reset(clr),

.uart\_rxd(uart\_rxd),

.uart\_txd(uart\_txd),

.en(en)

);

recorder recoder(

.clk(clk),

.AC\_BCLK(AC\_BCLK),

.AC\_MCLK(AC\_MCLK),

.AC\_MUTEN(AC\_MUTEN),

.AC\_PBDAT(AC\_PBDAT),

.AC\_PBLRC(AC\_PBLRC),

.AC\_RECDAT(AC\_RECDAT),

.AC\_RECLRC(AC\_RECLRC),

.AC\_SCL(AC\_SCL),

.AC\_SDA(AC\_SDA),

.LEDREC(),

.LEDTONE(),

.LEDPLAY(),

.LEDFEEDBACK(),

.reset(clr),

.voldown(),

.volup(),

.rec\_aud(),

.play\_tone(sound),

.play\_aud(1),

.feedback(),

.t\_node()

);

Endmodule

### count.v

`timescale 1ns / 1ps

module count(

input en,

input btn2,

input [5:0] sw,

input mode,

input clk,

input clr,

input btn3,

output reg [7:0] Led,

output [3:0] an,

output [6:0] a\_to\_g,

output reg [1:0] bell\_cnt,

output reg [1:0] ring\_bell\_cnt,

output reg bell,

output reg ring\_bell

);

reg [3:0] second\_l; //秒数低位

reg [3:0] second\_h; //秒数高位

reg [3:0] minute\_l; //分钟低位

reg [3:0] minute\_h; //分钟高位

reg [4:0] hour; //小时

reg [3:0] bell\_second\_l; //秒数低位闹钟

reg [3:0] bell\_second\_h; //秒数高位闹钟

reg [3:0] bell\_minute\_l; //分钟低位闹钟

reg [3:0] bell\_minute\_h; //分钟高位闹钟

reg [4:0] bell\_hour; //小时闹钟

reg [32:0] cnt; // 时钟计数

reg [3:0] NUM;

reg clk\_1Hz; //1hz时钟

//reg bell; //报时信号

//reg ring\_bell; //闹钟报时信号

reg ring\_flag; //亮灯信号

//reg [2:0] bell\_cnt; //整点报时3s

//reg [2:0] ring\_bell\_cnt; //闹钟报时

reg [4:0] tmpHour; //输出am

reg noon; //判断am，pm

reg [1:0] mode1; //设置时间的

reg [1:0] mode2; //设置闹钟的

reg [5:0] tmpSW;

assign an[0] = cnt[14:13]!=0; //数码管使能信号

assign an[1] = cnt[14:13]!=1;

assign an[2] = cnt[14:13]!=2;

assign an[3] = cnt[14:13]!=3;

always @(posedge clk)begin //产生1Hz时钟

if(clk)

begin

cnt <= cnt + 1;

if(cnt >= 125000000)

begin

clk\_1Hz = 1;

cnt <= 0;

end

else begin

clk\_1Hz = 0;

end

end

end

always @(posedge clk\_1Hz)begin //时间设置模式判断

if (btn2) begin

mode1 = mode1 + 1;

end

end

always @(posedge clk\_1Hz)begin //闹钟设置模式判断

if (btn3) begin

mode2 = mode2 + 1;

end

end

always @(posedge clk or negedge clr)

begin

if(clr) //清零

begin

second\_l <= 4'b0101;

second\_h <= 4'b0101;

minute\_l <= 4'b1001;

minute\_h <= 4'b0101;

hour <= 5'b10000;

end

else

begin

if(clk\_1Hz) //计时和计算时间

begin

second\_l = second\_l + 1;

ring\_bell = ((second\_l==bell\_second\_l)&&(second\_h==bell\_second\_h)

&&(minute\_l==bell\_minute\_l)&&(minute\_h==bell\_minute\_h)

&&(bell\_hour==hour)&&(en==0));

if(bell) //整点报时

begin

bell\_cnt <= bell\_cnt + 1;

if(bell\_cnt >= 3)

begin

bell\_cnt <= 0;

bell = 0;

end

end

else if(ring\_bell || ring\_flag) //闹钟报时

begin

ring\_flag = 1;

ring\_bell\_cnt <= ring\_bell\_cnt + 1;

if(ring\_bell\_cnt >= 3)

begin

ring\_bell\_cnt <= 0;

ring\_flag = 0;

end

end

end

if (mode1==0 && mode2==0)begin

if(second\_l>= 10)

begin

second\_l = second\_l - 10;

second\_h = second\_h + 1;

end

if(second\_h>=6)

begin

second\_h = second\_h - 6;

minute\_l = minute\_l + 1;

end

if(minute\_l>=10)

begin

minute\_l = minute\_l - 10;

minute\_h = minute\_h + 1;

end

if(minute\_h>=6)

begin

minute\_h = minute\_h -6;

hour = hour + 1;

bell = 1;

end

tmpHour = hour;

if(mode)begin //时制切换

if(tmpHour >= 12) begin

if (hour>12)begin

noon = 1;

end else begin

noon = 0;

end

tmpHour = tmpHour - 12;

end

end

else begin

noon = 0;

if(hour >=24) begin

hour = hour - 24;

end

end

Led[4:0] = tmpHour; //时间和其他信号的输出

Led[5] = noon;

Led[6] = bell;

Led[7] = ring\_flag;

case(cnt[14:13])

0: NUM = second\_l;

1: NUM = second\_h;

2: NUM = minute\_l;

3: NUM = minute\_h;

endcase

end //moded1=0

else if (mode1==1 && mode2==0) begin //设置小时

Led[5:0]=sw[5:0];

Led[7:6]=2'b00;

hour=sw;

NUM=4'b1111;

end//mode1==1

else if (mode1==2 && mode2==0)begin //设置分钟

Led=8'b00000000;

tmpSW = sw;

minute\_h<=tmpSW/10;

minute\_l<=tmpSW%10;

case(cnt[14:13])

0: NUM = 4'b1111;

1: NUM = 4'b1111;

2: NUM = minute\_l;

3: NUM = minute\_h;

endcase

end//mode1=2

else if (mode1==3 && mode2==0)begin //设置秒钟

Led=8'b00000000;

tmpSW = sw;

second\_h<=tmpSW/10;

second\_l<=tmpSW%10;

case(cnt[14:13])

0: NUM = second\_l;

1: NUM = second\_h;

2: NUM = 4'b1111;

3: NUM = 4'b1111;

endcase

end//mode1==3

else if (mode2==1) begin //设置闹钟小时

Led[5:0]=sw[5:0];

Led[7:6]=2'b00;

bell\_hour=sw;

NUM=4'b1111;

end//mode2==1

else if (mode2==2) begin //设置闹钟分钟

Led=8'b00000000;

tmpSW = sw;

bell\_minute\_h<=tmpSW/10;

bell\_minute\_l<=tmpSW%10;

case(cnt[14:13])

0: NUM = 4'b1111;

1: NUM = 4'b1111;

2: NUM = bell\_minute\_l;

3: NUM = bell\_minute\_h;

endcase

end//mode2==2

else if (mode2==3) begin //设置闹钟秒钟

Led=8'b00000000;

tmpSW = sw;

bell\_second\_h<=tmpSW/10;

bell\_second\_l<=tmpSW%10;

case(cnt[14:13])

0: NUM = bell\_second\_l;

1: NUM = bell\_second\_h;

2: NUM = 4'b1111;

3: NUM = 4'b1111;

endcase

end//mode2==3

end

end

number\_output A1(.NUM(NUM),.a\_to\_g(a\_to\_g));

endmodule

module number\_output( //七段数码管显示数字 //display模块

input [3:0] NUM,

output reg[6:0] a\_to\_g

);

always @(\*)

case(NUM)

0: a\_to\_g = 7'b1000000;

1: a\_to\_g = 7'b1111001;

2: a\_to\_g = 7'b0100100;

3: a\_to\_g = 7'b0110000;

4: a\_to\_g = 7'b0011001;

5: a\_to\_g = 7'b0010010;

6: a\_to\_g = 7'b0000010;

7: a\_to\_g = 7'b1111000;

8: a\_to\_g = 7'b0000000;

9: a\_to\_g = 7'b0010000;

default a\_to\_g = 7'b1111111;

endcase

endmodule

### uart\_loopback.v

`timescale 1ns / 1ps

module UART\_crl(

input sys\_clk , //外部125MHz时钟

input reset, //系外部复位信号，高有效

//UART端口

input uart\_rxd , //UART接收端口

output uart\_txd, //UART发送端口

output reg en //使能信号输出，通过这个信号控制后续闹铃的启用和停用

);

//parameter define

parameter CLK\_FREQ = 125000000; //定义系统时钟频率

parameter UART\_BPS = 115200 ; //定义串口波特率

//wire define

wire uart\_rx\_done; //UART接收完成信号

wire [7:0] uart\_rx\_data; //UART接收数据，为键盘输入的控制，通过对这个数据的判断为使能输出赋值

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\* main code

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

always @(posedge sys\_clk)begin //根据键盘的输入对使能输出赋值

if(uart\_rx\_data==49)begin //如果按下1，则使能信号变为1

en=1;

end

else if(uart\_rx\_data==50)begin //如果按下2，则使能信号变为0

en=0;

end

end

//串口接收模块

uart\_rx #(

.CLK\_FREQ (CLK\_FREQ),

.UART\_BPS (UART\_BPS)

)

u\_uart\_rx(

.clk (sys\_clk ),

.reset (reset ),

.uart\_rxd (uart\_rxd ),

.uart\_rx\_done (uart\_rx\_done),

.uart\_rx\_data (uart\_rx\_data)

);

//串口发送模块

uart\_tx #(

.CLK\_FREQ (CLK\_FREQ),

.UART\_BPS (UART\_BPS)

)

u\_uart\_tx(

.clk (sys\_clk ),

.reset (reset ),

.uart\_tx\_en (uart\_rx\_done),

.uart\_tx\_data (uart\_rx\_data),

.uart\_txd (uart\_txd ),

.uart\_tx\_busy ( )

);

Endmodule

### Debounce.v

**//虽然名字叫防抖，但内容其实是时制切换模块**

`timescale 1ns / 1ps

module Mode(

input Flag,

input Clk,

output reg mode

);

wire pulse;

always @(posedge pulse)

begin

if (Flag) begin

if(mode)

mode <= 0;//0为24，1为12

else mode <= 1;

end

end

Debounce myDebounce(.Clk(Clk),

.pulse(pulse));

endmodule

module Debounce(

input Clk,

output pulse

);

reg [24:0] d;//取时长

always @(posedge Clk)

begin

d <= d + 1;

end

assign pulse = d[24];

endmodule

### recoder.v

module recorder(clk,

AC\_BCLK, AC\_MCLK, AC\_MUTEN, AC\_PBDAT, AC\_PBLRC, AC\_RECDAT, AC\_RECLRC, AC\_SCL, AC\_SDA,

LEDREC, LEDTONE, LEDPLAY, LEDFEEDBACK,

reset, voldown, volup, rec\_aud,

play\_tone, play\_aud, feedback,

t\_node);

// Input & Output Ports

input clk;

inout AC\_BCLK;

output AC\_MCLK;

output AC\_MUTEN;

output AC\_PBDAT;

inout AC\_PBLRC;

input AC\_RECDAT;

inout AC\_RECLRC;

output AC\_SCL;

inout AC\_SDA;

output reg LEDREC, LEDTONE, LEDPLAY, LEDFEEDBACK;

input reset, voldown, volup, rec\_aud;

input play\_tone, play\_aud, feedback;

output reg [3:0] t\_node;

reg [7:0] state;

reg [1:0] volume\_control;

wire [15:0] audio\_output;

wire [1:0] sample\_end; // write to internal register

wire [1:0] sample\_req; // take the audio input

wire clk\_48kHz; // clock 48kHz

reg [15:0] mem\_in;

reg [15:0] mem\_out;

wire [18:0]max\_ram\_address = 19'd262144;

reg [18:0] blk\_addr;

reg [15:0] blk\_din;

wire [15:0] blk\_dout;

reg blk\_wr;

reg iblk\_wr;

reg iblk\_rd;

wire clk\_50MHz, clk\_11MHz;

clk\_wiz\_0 uclk\_wiz (

.clk\_out1(clk\_50MHz), //50 MHz

.clk\_out2(clk\_11MHz), //11.2896 MHz

.clk\_out3(clk\_12MHz), //12.288 MHz

.clk\_in1(clk)

);

initial begin

volume\_control <= 1;

state <= 8'h00;

end

wire volupstate, vu\_up, vu\_dn;

wire voldnstate, vd\_up, vd\_dn;

always @(posedge clk)

begin

if(vu\_dn && (volume\_control < 3))

volume\_control <= volume\_control + 1;

else if(vd\_dn && (volume\_control > 0))

volume\_control <= volume\_control - 1;

end

debounce volumeup(

.clk(clk),

.i\_btn(volup),

.o\_state(volupstate),

.o\_ondn(vu\_dn),

.o\_onup(vu\_up)

);

debounce volumedn(

.clk(clk),

.i\_btn(voldown),

.o\_state(voldnstate),

.o\_ondn(vd\_dn),

.o\_onup(vd\_up)

);

// Audio Codec Interface Instantiation

i2c\_top i2c\_top\_inst(

.clk(clk),

.main\_clk(clk\_50MHz),

.audio\_clk(clk\_11MHz),

.clk\_12MHz(clk\_12MHz),

.play\_tone(play\_tone),

.playback(play\_aud),

.volume\_control(volume\_control),

.AC\_RECLRC(AC\_RECLRC),

.AC\_RECDAT(AC\_RECDAT),

.AC\_PBLRC(AC\_PBLRC),

.AC\_PBDAT(AC\_PBDAT),

.AC\_MCLK(AC\_MCLK),

.AC\_BCLK(AC\_BCLK),

.AC\_SCL(AC\_SCL),

.AC\_SDA(AC\_SDA),

.AC\_MUTEN(AC\_MUTEN),

.KEY(1),

.FEEDBACK(feedback),

.audio\_in(mem\_out), //audio\_input

.audio\_out(audio\_output),

.sample\_end (sample\_end),

.sample\_req (sample\_req),

.clk\_48kHz(clk\_48kHz)

);

reg [2:0] bck\_buf;

reg [8:0] lrck\_buf;

localparam [2:0]

idle = 3'b000,

l\_st = 3'b001,

l\_cap = 3'b010,

r\_st = 3'b011,

r\_cap = 3'b100;

reg [2:0] i2s\_state; // = idle;

wire lr\_pos, lr\_neg, bck\_pos, bck\_neg;

// Process for ssm\_clk\_gen\_proc

always @(posedge clk\_11MHz or posedge reset) begin

if (reset) begin

bck\_buf <= 3'b000;

lrck\_buf <= 9'b000000000;

end else if (clk\_11MHz) begin

bck\_buf <= bck\_buf + 1'b1;

lrck\_buf <= lrck\_buf + 1'b1;

end

end

edge\_detect ulr\_edge\_detect (

.clk(clk\_11MHz),

.rst(reset),

.signal\_in(lrck\_buf[8]),

.pos\_edge(lr\_pos),

.neg\_edge(lr\_neg)

);

edge\_detect ubck\_edge\_detect (

.clk(clk\_11MHz),

.rst(reset),

.signal\_in(bck\_buf[2]),

.pos\_edge(bck\_pos),

.neg\_edge(bck\_neg)

);

blk\_mem\_gen\_0 ublk\_mem\_gen (

.clka(AC\_RECLRC),

.ena(1'b1),

.wea(blk\_wr),

.addra(blk\_addr),

.dina(blk\_din),

.douta(blk\_dout)

);

// Process for cap\_dat\_proc

always @(posedge AC\_RECLRC or posedge reset) begin

if (reset) begin

i2s\_state <= idle;

blk\_wr <= 1'b0;

end else if (AC\_RECLRC) begin

blk\_wr <= 1'b0;

if (iblk\_wr) begin

blk\_din <= audio\_output;

blk\_wr <= 1'b1;

end else if (iblk\_rd) begin

mem\_out <= blk\_dout;

end

end

end

// Process for blk\_mem\_proc

always @(posedge AC\_RECLRC or posedge reset) begin

if (reset) begin

iblk\_wr <= 1'b0;

iblk\_rd <= 1'b0;

blk\_addr <= 19'b0;

end else if (AC\_RECLRC) begin

if (iblk\_wr) begin

if (blk\_addr < max\_ram\_address) begin

blk\_addr <= blk\_addr + 1'b1;

end else begin

blk\_addr <= 19'b0;

iblk\_wr <= 1'b0;

end

end else if (iblk\_rd) begin

if (blk\_addr < max\_ram\_address) begin

blk\_addr <= blk\_addr + 1'b1;

end else begin

blk\_addr <= 19'b0;

iblk\_rd <= 1'b0;

end

end else if (rec\_aud) begin

iblk\_wr <= 1'b1;

blk\_addr <= 19'b0;

end else if (play\_aud) begin

iblk\_rd <= 1'b1;

blk\_addr <= 19'b0;

end

end

end

always @(\*) begin

t\_node[0] = clk\_48kHz; //JA1

t\_node[1] = sample\_end; //JB1

t\_node[2] = AC\_MCLK; //JA2

t\_node[3] = AC\_BCLK; //JB2

t\_node[4] = AC\_PBLRC; //JA3

t\_node[5] = AC\_RECLRC; //JB3

LEDREC <= iblk\_wr;

LEDTONE <= play\_tone;

LEDPLAY <= iblk\_rd;

LEDFEEDBACK <= feedback;

// t\_node[0] = AC\_PBDAT; //JA1

// t\_node[1] = AC\_RECDAT; //JB1

end

endmodule

# 实验结果

任务1：实验现象见数电大作业->实验结果->实验一到五

实验现象分析：为了方便观察整点时数字种现象，我们将数字钟的初始化时间设为16：59：55。数字钟能显示随时间变化的24小时制的“时、分、秒”，按BTN0可以将时间初始化。

任务2：实验现象见数电大作业->实验结果->实验一到五

实验现象分析：观察到，当时间到达整点时LED[6]亮3秒作整点报时（注：由于任务先后问题，此时还未加入音乐，整点报时的音乐内容见任务6）

任务3：实验现象见数电大作业->实验结果->实验一到五

实验现象分析：观察到，按按钮BTN1实现12/24小时显示模式切换功能，使用LED[5]表示12小时显示模式中的上午、下午。默认状态，使用24小时显示模式。24小时显示模式中，保持LED[5]=0。按一下BTN1，切换到12小时显示模式。12小时显示模式中，LED[5]=0表示上午。12小时显示模式中，LED[5]=1表示下午。再按一下BTN1，切换回24小时显示模式。

任务4：实验现象见数电大作业->实验结果->实验一到五

实验现象分析：观察到，使用按钮BTN2作为时间设置按钮，按一次BTN2进入“设置时间的小时数值”模式，再按一次BTN2进入“设置时间的分钟数值”模式，按第三次BTN2进入“设置时间的秒钟数值”模式，再按一次BTN2回到“显示当前时间”状态。此时当前时间已变为设定时间。

任务5：实验现象见数电大作业->实验结果->实验一到五

实验现象分析：观察到，使用按钮BTN3作为闹铃设置按钮，按一次BTN3进入“设置闹铃的小时数值”模式，再按一次BTN3进入“设置闹铃的分钟数值”模式，按第三次BTN3进入“设置闹铃的秒钟数值”模式，再按一次BTN3回到“显示当前时间”状态。此时当前时间不变，但已经设定好了闹钟。抵达闹铃设定时间，LED[7]亮3秒。（注：由于任务先后问题，此时还未加入音乐，闹钟报时的音乐内容见任务6）

任务6：实验现象见数电大作业->实验结果->实验六

实验现象分析：整点到时LED[6]亮3秒，并播放一段时长3秒的音乐。闹铃到时LED[7]亮3秒，并播放一段时长3秒的音乐。

任务7：实验现象见数电大作业->实验结果->实验七

实验现象分析：通过UART串口通信，使用电脑键盘1键和2键控制停用和启用设定好的闹铃。输入1时闹钟到时但无现象，闹钟停用；输入2时重新启用闹钟，闹钟到时出现现象。

# 遇到的问题

1. Verilog语言调试问题总结：

（1）begin与end需要细心检查，一一对应。

（2）模块例化后之间的连接应使用wire变量，不能用reg变量。

（3）同一个寄存器变量不能在多个always块中赋值。

（4）Output变量不可以再always中赋值。

1. 程序设计问题：
2. 每添加一个新模块的功能都需要先检验再进行下一步，否则容易出现复杂报错。
3. 关键代码一点要写注释，必要时可以在end等易出现错误的地方标明与之对应的前文关键词。
4. **实验总结**

在本次实验中，我们学习了使用 Verilog语言设计数字电路的基础知识，并且实现了一个具有报时、闹钟、时制切换以及串口停启闹钟功能的数字钟。通过本次实验，我们掌握了以下几点内容：Verilog的基础语法和数据类型、数字电路设计的基本流程、使用 Verilog设计数字电路的方法、模块化设计思想、了解了数字电路设计的一些常用工具和调试方法。在长时间的努力下最终成功实现了数字钟的全部功能。同时我们要感谢佘老师对本次实验的指导和帮助，老师在讲解理论知识的同时，还给予我们思路和鼓励。最后，经过半个学期的辛勤努力，数字电子技术实验课程在佘老师的悉心指导下完美结束了。感谢您的辛勤付出。